Spectral Approaches to Learning PredictiveRepresentations

Byron Boots

September 2012
CMU-ML-12-108

MACHINE LEARNING
DEPARTMENT

Carnegie Mellon

Acknowledgments

I would like to thank the many people who have have provided support and encouragement throughout my time in the Ph.D. program at CMU. They have aided the research described here and helped bring it to a successful conclusion.

First and foremost, I would like to thank my advisor Geoff Gordon for directing me toward interesting research problems, patiently teaching me when my background was lacking, listening to my crazy ideas, and generally supporting my progression through the Ph.D. program. He has proven to be an invaluable resource, a skilled mentor, and a supportive friend. I also am indebted to Drew Bagnell, Dieter Fox, and Arthur Gretton for serving on my thesis committee and providing helpful insights and suggestions over the years. I am particularly lucky to have spent time as a guest in Dieter's lab a UW and Arthur's group at UCL: both experiences have shaped my views as a researcher and contributed to the work presented in this thesis.

I am particularly fortunate to have been a graduate student in the Machine Learning Department and SELECT lab at CMU, where I have benefited immensly from the open and collaborative atmosphere. A special thanks is due to Sajid Siddiqi and Hormoz Zarnani. Sajid coauthored a number of papers with me and taught me a great deal about research in my first few years at CMU. Hormoz entered the program with me and put up with me as a roommate for several years. I am also thankful for my lab-mates who stimulated interesting discussions and provided helpful criticism: Danny Bickson, Joseph Bradley, Anton Chechetka, Kit Chen, Carlton Downey, Miroslav Dudk, Khalid El-Arini, Stanislav Funiak, Joseph Gonzalez, Arthur Gretton, Ahmed Hefny, Sue Ann Hong, Jon Huang, Adona Iosif, Shiva Kaul, Andreas Krause, Aapo Kyrola, Wooyoung Lee, Yucheng Low, Austin McDonald, Julian Ramos, Dafna Shahaf, Ajit Singh, Le Song, Gaurav Veda, Yisong Yue, and Brian Ziebart.

Finally, I would like to thank Michelle Martin and Diane Stidle who provided crucial assistence in making sure that my graduate experience was as painless as possible.

Contents

1 Introduction **1**
1.1 Main Contributions 1
1.2 Organization 2

I Spectral Learning Algorithms for Predictive Representations **5**

2 A Spectral Learning Algorithm for Constant-Covariance Kalman Filters **7**
2.1 The State Space Equations 7
2.1.1 Inference 8
2.2 The Constant-Covariance Kalman Filter 9
2.2.1 Observable Representations 10
2.3 A Spectral Learning Algorithm 15
2.3.1 Learning from Sequences of Observations 16
2.3.2 Stability 16
2.4 Conclusions 18

3 Spectral Learning Algorithms for Predictive State Representations **19**
3.1 Introduction 19
3.2 Predictive State Representations 20
3.2.1 Tests and PSR Notation 20
3.2.2 Linear Predictive State Representations 20
3.3 Observable Representations 22
3.4 Learning Predictive State Representations 26

4 Learning Predictive State Representations with Features **29**
4.1 Characteristic and Indicative Features 29
4.1.1 Characteristic Features 29
4.1.2 Indicative Features 31
4.2 Defining PSRs with Characteristic and Indicative Features 31
4.3 Observation Features and Bayes' Rule 34
4.3.1 Bayes' Rule with Discrete Observations 35
4.3.2 Bayes' Rule with Observation Features 36
4.4 Defining PSRs with Observation Features 37

4.4.1 The Moment Spectral Learning Algorithm for PSRs 41

5 Hilbert Space Embeddings of Predictive State Representations 43
5.1 Hilbert Space Embeddings 44
5.1.1 Mean Maps 44
5.1.2 Covariance Operators 45
5.1.3 Conditional Embedding Operators 45
5.1.4 Kernel Bayes' Rule 46
5.2 Predictive Representations in RKHS 47
5.2.1 Conditional Predictive Representations 48
5.2.2 The State Update 49
5.2.3 A Minimal State Space 51
5.3 Kernel Learning Algorithms for PSRs 53
5.3.1 The Gram Matrix Formulation for Hilbert Space Embeddings of PSRs . . 54
5.3.2 Gram Matrix State Updates 54
5.3.3 A Spectral Learning Algorithm 55
5.4 Experimental Results 56
5.4.1 Robot Vision 56
5.4.2 Slot Car Inertial Measurement 57
5.4.3 Audio Event Classification 58
5.5 Conclusion 59

6 Computational Efficiency in Spectral Learning Algorithms 61
6.1 Introduction 61
6.2 Batch Learning of PSRs 62
6.2.1 An Efficient Batch Learning Algorithm 62
6.3 Iterative Updates to PSR Parameters 63
6.3.1 Batch Updates 64
6.3.2 Online updates 67
6.4 Random Projections for High Dimensional Feature Spaces 67
6.5 Experimental Results 68
6.5.1 A Synthetic Example 68
6.5.2 Slot Car Inertial Measurement 69
6.5.3 Modeling Video 70
6.6 Conclusions 70

II Spectral Learning Algorithms in Practice 73

7 Stability in Linear Dynamical Systems 75
7.1 Introduction 75
7.2 Learning Stable Linear Dynamical Systems 76
7.2.1 Formulating the Objective 76
7.2.2 Generating Constraints 77

7.2.3 Computing the Solution . . . 78
7.2.4 Refinement . . . 78
7.3 Experiments . . . 79
7.3.1 Stable Dynamic Textures . . . 79
7.3.2 Stable Baseline Models for Biosurveillance . . . 82
7.3.3 Modeling Sunspot Numbers . . . 83
7.4 Related Work . . . 83

8 Reinforcement Learning: Value Iteration in a Learned Predictive State Representation 85
8.1 Introduction . . . 85
8.2 Planning in PSRs . . . 87
8.3 Experimental Results . . . 89
8.3.1 The Autonomous Robot Domain . . . 89
8.3.2 Features . . . 90
8.3.3 Learning a Model . . . 91
8.3.4 Qualitative Evaluation . . . 91
8.3.5 Planning in the Learned Model . . . 92
8.4 Conclusions . . . 92

9 Reinforcement Learning: Predictive State Temporal Difference Learning 97
9.1 Introduction . . . 97
9.2 Value Function Approximation . . . 99
9.2.1 Least Squares Temporal Difference Learning . . . 100
9.3 Predictive Features . . . 101
9.3.1 Finding Predictive Features Through a Bottleneck . . . 101
9.3.2 Predictive State Temporal Difference Learning . . . 103
9.3.3 PSRs . . . 103
9.3.4 Predictive State Temporal Difference Learning Revisited . . . 105
9.3.5 Insights from Subspace Identification . . . 106
9.4 Experimental Results . . . 107
9.4.1 Estimating the Value Function of a RR-POMDP . . . 107
9.4.2 Pricing A High-dimensional Financial Derivative . . . 108
9.5 Conclusion . . . 110

10 A Spectral Learning Approach to Range-Only SLAM 111
10.1 Introduction . . . 111
10.2 Background . . . 112
10.2.1 Likelihood-based Range-only SLAM . . . 112
10.2.2 Spectral State Space Discovery and System Identification . . . 113
10.2.3 Orthographic Structure From Motion . . . 114
10.2.4 Dimensionality-reduction-based Methods for Mapping . . . 115
10.3 State Space Discovery and Spectral SLAM . . . 115
10.3.1 Range-only SLAM as Matrix Factorization . . . 116

10.3.2 SLAM with Headings . . . 117
10.3.3 A Spectral SLAM Algorithm . . . 119
10.3.4 Extensions: Missing Data, Online SLAM, and System ID . . . 120
10.4 Experimental Results . . . 121
10.4.1 Synthetic Experiments . . . 121
10.4.2 Robotic Experiments . . . 121
10.5 Conclusion . . . 123

III Conclusions **125**

11 Discussion **127**

IV Appendix **129**

12 Appendix **131**
12.1 Predictive State Temporal Difference Learning . . . 131
12.1.1 Determining the Compression Operator . . . 131
12.1.2 Experimental Results . . . 132
12.2 A Spectral Learning Approach to Range Only SLAM . . . 133
12.2.1 Metric Upgrade for Learned Map . . . 133
12.2.2 Sample Complexity for the Measurement Model (Robot Map) . . . 136
12.2.3 The Robot as a Nonlinear Dynamical System . . . 138

Bibliography **143**

List of Figures

2.1 Graphical representation of the deterministic-stochastic linear dynamical system. See text for details. 8

2.2 Sunspot data, sampled monthly for 200 years. Each curve is a month, the x-axis is over years. Below the graph are the first two principal components of $\mathcal{O}_i$ where Y_f and Y_p each consist of 1-observation Hankel matrices and 12-observation Hankel matrices. The 1-observation Hankel matrices do not contain enough observations to recover a state which accurately reflects the temporal patterns in the data, while the 12-observation Hankel matrices do. 16

2.3 System equilibria. (A) Unstable equilibrium. The state vector will rapidly move away from the equilibrium point when perturbed. (B) Asymptotically stable equilibrium. The state vector will return to the original equilibrium point when perturbed. (C.) Stable equilibrium. No "resistance;" a perturbed state vector will oscillate forever around the equilibrium point. Note that the notion of *asymptotic stability* is stronger than *stability*. 17

3.1 A simple 3-state 2-observation HMM. An example of a set of indicative events (a mutually exclusive and exhaustive partition of histories) for this HMM is $\mathcal{H} = \{o_1o_2, o_1o_1, o_2\}$. (E.g., o_1o_2 means that the observation at time $t-1$ is o_1, and the one at time t is o_2.) An example of a set of tests for this HMM is $\mathcal{T} = \{o_1o_2, o_1o_1, o_2o_1, o_2o_2\}$. (In this case, o_1o_2 means that the observation at time $t+1$ is o_1, and the one at time $t+2$ is o_2.) 22

5.1 Robot vision data. (A) Sample images from the robot's camera. The figure below depicts the hallway environment with a central obstacle (black) and the path that the robot took through the environment (the red counter-clockwise ellipse). (B) Squared error for prediction with different estimated models and baselines. . . . 56

5.2 Slot car inertial measurement data. (A) The slot car platform and the IMU (top) and the racetrack (bottom). (B) Squared error for prediction with different estimated models and baselines. 57

5.3 Accuracies and 95% confidence intervals for Human vs. Non-human audio event classification, comparing embedded HMMs to other common sequential models at different latent state space sizes. 59

6.1 A synthetic RR-HMM. (A.) The eigenvalues of the true transition matrix. (B.) RMS error in the nonzero eigenvalues of the estimated transition matrix vs. number of training samples, averaged over 10 trials. The error steadily decreases, indicating that the PSR model is becoming more accurate, as we incorporate more training data. 68

6.2 Slot car inertial measurement data. (A) The slot car platform: the car and IMU (top) and the racetrack (bottom). (B) Squared error for prediction with different estimated models. Dash-dot shows the baseline of simply predicting the mean measurement on all frames. 69

6.3 Modeling video. (A.) Schematic of the camera's environment. (B.) The second and third dimension of the learned belief space (the first dimension contains normalization information). Points are colored red when the camera is traveling clockwise and blue when traveling counterclockwise. The learned state space separates into two manifolds, one for each direction, connected at points where the camera changes direction. (The manifolds appear on top of one another, but are separated in the fourth latent dimension.) (C.) Loop closing: estimated historical camera positions after 100, 350, and 600 steps. Red star indicates current camera position. The camera loops around the table, and the learned map "snaps" to the correct topology when the camera passes its initial position. . . . 71

7.1 (A): Conceptual depiction of the space of $n \times n$ matrices. The region of stability (S_λ) is non-convex while the smaller region of matrices with $\sigma_1 \leq 1$ (S_σ) is convex. The elliptical contours indicate level sets of the quadratic objective function of the QP. $\hat{A}$ is the unconstrained least-squares solution to this objective. $A_{\text{LB-1}}$ is the solution found by LB-1 [57]. One iteration of constraint generation yields the constraint indicated by the line labeled 'generated constraint', and (in this case) leads to a stable solution A^*. The final step of our algorithm improves on this solution by interpolating A^* with the previous solution (in this case, $\hat{A}$) to obtain A^*_{final}. (B): The actual stable and unstable regions for the space of 2×2 matrices $E_{\alpha,\beta} = [\, 0.3 \;\; \alpha \,; \beta \;\; 0.3 \,]$, with $\alpha, \beta \in [-10, 10]$. Constraint generation is able to learn a nearly optimal model from a noisy state sequence of length 7 simulated from $E_{0,10}$, with better state reconstruction error than either LB-1 or LB-2. The matrices $E_{10,0}$ and $E_{0,10}$ are stable, but their convex combination $E_{5,5} = 0.5E_{10,0} + (1 - 0.5)E_{0,10}$ is unstable. 77

7.2 Dynamic textures. A. Samples from the original `steam` sequence and the `fountain` sequence. B. State evolution of synthesized sequences over 1000 frames (`steam` top, `fountain` bottom). The least squares solutions display instability as time progresses. The solutions obtained using LB-1 remain stable for the full 1000 frame image sequence. The constraint generation solutions, however, yield state sequences that are stable over the full 1000 frame image sequence without significant damping. C. Samples drawn from a least squares synthesized sequences (top), and samples drawn from a constraint generation synthesized sequence (bottom). Here we are displaying image values that are clipped to stay within the valid pixel range $[0, 255]$. Images for LB-1 are not shown. The constraint generation synthesized `steam` sequence is qualitatively better looking than the `steam` sequence generated by LB-1, although there is little qualitative difference between the two synthesized `fountain` sequences. 80

7.3 Bar graphs illustrating decreases in objective function value relative to the least squares solution (A,B) and the running times (C,D) for different stable LDS learning algorithms on the `fountain` and `steam` textures respectively, based on the corresponding columns of Table 7.1. 82

7.4 (A): 60 days of data for 22 drug categories aggregated over all zipcodes in the city. (B): 60 days of data for a single drug category (cough/cold) for all 29 zipcodes in the city. (C): Sunspot numbers for 200 years separately for each of the 12 months. The training data (top), simulated output from constraint generation, output from the unstable least squares model, and output from the over-damped LB-1 model (bottom). 83

8.1 Learning and Planning in the Autonomous Robot Domain. (A) The robot uses visual sensing to traverse a square domain with multi-colored walls and a central obstacle. Examples of images recorded by the robot occupying two different positions in the environment are shown at the bottom of the figure. (B) A to-scale 3-dimensional view of the environment. (C) The 2nd and 3rd dimension of the learned subspace (the first dimension primarily contained normalization information). Each point is the embedding of a single history, displayed with color equal to the average RGB color in the first image in the highest probability test. (D) The same points in (C) projected onto the environment's geometric space. (E) The value function computed for each embedded point; lighter indicates higher value. (F) Policies executed in the learned subspace. The red, green, magenta, and yellow paths correspond to the policy executed by a robot with starting positions facing the red, green, magenta, and yellow walls respectively. (G) The paths taken by the robot in geometric space while executing the policy. Each of the paths corresponds to the path of the same color in (F). The darker circles indicate the starting and ending positions, and the tick-mark indicates the robot's orientation. (H) Analysis of planning from 100 randomly sampled start positions to the target image (facing blue wall). In the bar graph: the mean number of actions taken by the optimistic solution found by A* search in configuration space (left); the mean number taken by the policy found by Perseus in the learned model (center); and the mean number taken by a random policy (right). The line graph illustrates the cumulative density of the number of actions given the optimal, learned, and random policies. 95

9.1 Experimental Results. Error bars indicate standard error. (A) Estimating the value function with a small number of informative features. PSTD and PSTD2 both do well. (B) Estimating the value function with a small set of informative features and a large set of random features. LARS-TD is designed for this scenario and dramatically outperforms PSTD and LSTD, however it does not outperform PSTD2. (C) Estimating the value function with a large set of semi-informative features. PSTD is able to determine a small set of compressed features that retain the maximal amount of information about the value function, outperforming LSTD by a very large margin. (D) Pricing a high-dimensional derivative via policy iteration. The y-axis is expected reward for the current policy at each iteration. The optimal threshold strategy (sell if price is above a threshold [115]) is in black, LSTD (16 canonical features) is in blue, LSTD (on the full 220 features) is cyan, LARS-TD (feature selection from set of 220) is in green, and PSTD (16 dimensions, compressing 220 features (16 + 204)) is in red. 108

10.1 A general principle for state space discovery. We can think of state as a *statistic* of history that is minimally *sufficient* to predict future observations. If the bottleneck is a rank constraint, then we get a *spectral* method. 114

10.2 Spectral SLAM on simulated data. See Section 10.4.1 for details. A.) Randomly generated landmarks (6 of them) and robot path through the environment (500 timesteps). A SVD of the squared distance matrix recovers a linear transform of the landmark and robot positions. Given the coordinates of 4 landmarks, we can recover the landmark and robot positions in their original coordinates; or, since $500 \geq 9$, we can recover positions up to an orthogonal transform with no additional information. Despite noisy observations, the robot recovers the true path and landmark positions with very high accuracy. B.) The convergence of the observation model $\widehat{C}_{5:6}$ for the remaining two landmarks: mean Frobenius-norm error vs. number of range readings received, averaged over 1000 randomly generated pairs of robot paths and environments. Error bars indicate 95% confidence intervals. 117

10.3 The autonomous lawn mower and spectral SLAM. A.) The robotic lawn mower platform. B.) In the first experiment, the robot traveled 1.9km receiving 3,529 range measurements. This path minimizes the effect of heading error by balancing the number of left turns with an equal number of right turns in the robot's odometry (a commonly used path pattern in lawn mowing applications). The light blue path indicates the robot's true path in the environment, light purple indicates dead-reckoning path, and dark blue indicates the spectral SLAM localization result. C.) In the second experiment, the robot traveled 1.3km receiving 1,816 range measurements. This path highlights the effect of heading error on dead reckoning performance by turning in the same direction repeatedly. Again, spectral SLAM is able to accurately recover the robot's path. 122

10.4 Comparison of Range-Only SLAM Algorithms. The table shows Localization RMSE. Spectral SLAM has localization accuracy comparable to batch optimization on its own. The best results (boldface entries) are obtained by initializing nonlinear batch optimization with the spectral SLAM solution. The graph compares runtime of Gauss-Newton batch optimization with spectral SLAM. Empirically, spectral SLAM is 3-4 *orders of magnitude* faster than batch optimization on the autonomous lawnmower datasets. 124

List of Tables

3.1 Notation for Tests. 20

7.1 Quantitative results on the dynamic textures data for different numbers of states n. CG is our algorithm, LB-1and LB-2 are competing algorithms, and LB-1* is a simulation of LB-1 using our algorithm by generating constraints until we reach S_σ, since LB-1 failed for $n > 10$ due to memory limits. e_x is percent difference in squared reconstruction error. Constraint generation, in all cases, has lower error and faster runtime. 81

Chapter 1

Introduction

Many problems in machine learning and statistics involve collecting high-dimensional multivariate observations or sequences of observations, and then hypothesizing a compact model which explains these observations. An appealing representation for such a model is a *latent variable model* that relates a set of observed variables to an additional set of unobserved or hidden variables. Examples of popular latent variable models include latent tree graphical models and dynamical system models, both of which occupy a fundamental place in engineering, control theory, economics as well as the physical, biological, and social sciences. Unfortunately, to discover the right latent state representation and model parameters, we must solve difficult structural and temporal credit assignment problems.

Prior to the work described in this thesis, research on learning latent variable models and dynamical systems has predominantly relied on likelihood maximization and local search heuristics such as expectation maximization (EM); these heuristics often lead to a search space with a host of bad local optima, and may therefore require impractically many restarts to reach a prescribed training precision.

This thesis will focus on two complementary ideas: *predictive representations* and *spectral* learning algorithms. These models and algorithms hold the promise of overcoming the problems inherent in previous approaches: unlike standard latent variable models, predictive representations can be naturally written in terms of observable quantities; and unlike the EM algorithm, spectral methods for learning predictive representations are computationally efficient, statistically consistent, and have no local optima; in addition, they can be simple to implement, and have state-of-the-art practical performance for many interesting learning problems.

1.1 Main Contributions

This thesis included the following major contributions:

- We develop a novel spectral algorithm for learning constant-covariance Kalman filters. While we are not the first to develop spectral approaches to learning Kalman filters, we developed this method specifically to demonstrate how spectral algorithms for learning Kalman filters are similar to spectral algorithms for learning nonlinear dynamical system

models. We also develop a novel method for enforcing the *stability* of learned linear dynamical system models, an important consideration when learning models for practical applications.

- We develop spectral learning algorithms for discrete-observation discrete-action Predictive State Representations (PSRs) which are an expressive class of controlled dynamical system models that includes well-known models like Hidden Markov Models and Partially Observable Markov Decision Processes. One of the advantages of PSRs is that they are very *compact* models that have the ability to represent very large discrete state spaces with low-dimensional representations. We demonstrate the power and utility of our learning algorithm by proving that it can learn Hidden Markov Models efficiently, by showing that we can learn a compact model of simulated robotic agent that supports planning, and by improving least squares temporal difference learning.
- We extend spectral learning algorithms for discrete PSRs to use continuous *features* of actions and observations. We show how to generalize Bayes' rule to feature spaces, and leverage this to create a very general dynamical system model and an efficient spectral learning algorithm. This is an important advance, in that it makes learning complicated non-linear models significantly easier and more data-efficient. We link these feature-based spectral algorithms to non-parametric dynamical system models and spectral approaches to learning models embedded in reproducing kernel Hilbert spaces.
- We show how spectral learning algorithms can be leveraged to problems *beyond* dynamical system learning like range-only simultaneous localization and mapping and robot system identification. The approaches developed may be adapted to future problems in robot vision, mapping, and dynamical system learning.

The combination of predictive representations and spectral learning algorithms is an exciting new area of research within machine learning. Since original publication, many of the novel ideas and algorithms in this thesis have been embraced and extended by the machine learning research community, and many of the core principles are starting to be applied to problems outside of dynamical system learning. Much of the research includes extensions and theoretical contributions to the algorithms and theory in this thesis [4, 7, 33, 68, 106]. Additionally, several researchers have gone beyond learning models of dynamical systems to develop predictive representations and associated spectral learning algorithms for learning graphical models [1, 74, 100, 101], latent Dirichlet allocation [2], and probabilistic context free grammars [5, 25, 26].

1.2 Organization

The remainder of the thesis is organized as follows:

Part I: Spectral Learning Algorithms for Predictive Representations We develop a family of spectral learning algorithms for learning predictive representations of dynamical systems.

Chapter 2 We introduce a spectral learning algorithm for a simple class of linear dynamical systems with constant-covariance Gaussian noise. The algorithm proceeds by building covariance matrices of observable quantities and then leverages linear algebra to learn the model parameters. The goal of this chapter is to introduce the general spectral learning approach on a simple well-known dynamical system.

Chapter 3 We generalize this approach to learning dynamical systems to a much more expressive class of models called Predictive State Representations (PSRs) which include familiar models like Hidden Markov Models and Partially Observable Markov Decision Processes as special cases. Unlike Kalman filters, these models have non-linear dynamics and assume discrete action, observation, and state spaces.

Chapter 4 We proceed to expand upon the work in Chapter 3 by developing a learning algorithm based on continuous *features* of action and observation spaces. This allows us to learn models even in the presence of complicated dynamical systems with very high cardinality discrete state, action, and observation spaces. The basic learning algorithms presented in Chapter 3 and Chapter 4 are the core contributions of this thesis.

Chapter 5 We use recent developments in Hilbert space embeddings of distributions to extend our learning algorithms to cover dynamical systems with continuous action and observation spaces. The learning algorithms in this chapter are qualitatively similar to those in previous chapters but leverage infinite-dimensional feature-spaces and use the kernel trick durning learning.

Chapter 6 We provide efficient batch and online learning algorithms that use a number of linear algebra tricks to learn from massive quantities of data.

Part II: Spectral Learning Algorithms in Practice The second part of this thesis focuses on extensions to the basic algorithms described in Part I to handle practical problems in system identification, reinforcement learning, robotics, and related fields.

Chapter 7 We look at the problem of learning back parameters of *stable* Kalman filters with small quantities of training data. Although we may know that the dynamics of the system are stable *a priori*, this is not enforced in the spectral learning algorithm. We provide a way of enforcing stability.

Chapter 8 We look at the problem of *planning* in a learned Predictive State Representation. We demonstrate the advantages of this approach over previous methods and show that one can indeed learn a model of an environment and plan in the learned model.

Chapter 9 We use theory from the first part of the thesis to link temporal difference learning to spectral system identification. Using these insights we are able to build a novel temporal-difference learning algorithm that outperforms previous methods on difficult policy learning problems.

Chapter 10 We use insights from spectral system identification to develop a novel spectral learning approach to range-only simultaneous localization and mapping.

Part III: Conclusions We summarize the main contributions of this thesis. We finally conclude with a discussion of open problems and directions for future research.

Part IV: Appendices Contains the appendices for the chapters in Parts I and II.

Part I

Spectral Learning Algorithms for Predictive Representations

Chapter 2

A Spectral Learning Algorithm for Constant-Covariance Kalman Filters

Many problems in machine learning involve sequences of real-valued multivariate observations. To model the statistical properties of such data, it is often sensible to assume each observation to be correlated to the value of an underlying latent variable, or *state*, that is evolving over the course of the sequence. In the case where the state is real-valued and the noise terms are assumed to be Gaussian, the resulting model is called a *linear dynamical system* (LDS). LDSs are an important tool for modeling time series in engineering, controls and economics as well as the physical and social sciences. In this chapter we define a LDS and describe some of the inference and learning algorithms as well as review the property of stability as it relates to the LDS transition model, which will be relevant in Chapter 7. More details on LDSs and algorithms for inference and learning LDSs (including spectral learning algorithms) can be found in several canonical papers and standard references [35, 48, 51, 62, 117].

2.1 The State Space Equations

The evolution of a stochastic linear time-invariant dynamical system (LDS) can be described by the following two equations:

$$x(h_{t+1}) = Ax(h_t) + w_t \qquad w_t \sim \mathcal{N}(0, Q) \tag{2.1a}$$

$$y_t = Cx(h_t) + v_t \qquad v_t \sim \mathcal{N}(0, R) \tag{2.1b}$$

Time is indexed by the discrete[1] variable t. Here h_t is a history of observations up to time t, $x(h_t)$ denotes the hidden states in $\mathbb{R}^n$, y_t the observations in $\mathbb{R}^m$, and the parameters of the system: the dynamics matrix $A \in \mathbb{R}^{n\times n}$ and the observation model $C \in \mathbb{R}^{m\times n}$. The variables w_t and v_t describe zero-mean normally distributed process and observation noise respectively, with covariance and cross-covariance matrices

$$\mathbb{E}\left[\begin{bmatrix} w_t \\ v_t \end{bmatrix} \begin{bmatrix} w_s^{\mathrm{T}} & v_s^{\mathrm{T}} \end{bmatrix}\right] = \begin{bmatrix} Q & S \\ S^{\mathrm{T}} & R \end{bmatrix} \delta_{ts} \tag{2.2}$$

[1] In *continuous-time* dynamical systems, the derivatives are specified as functions of the current state. They can be converted to discrete-time systems.

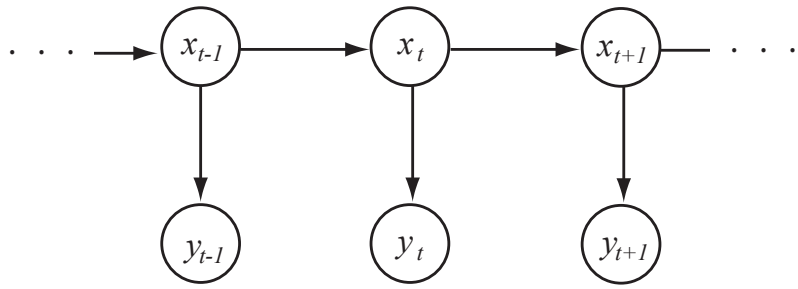

Figure 2.1: Graphical representation of the deterministic-stochastic linear dynamical system. See text for details.

where δ_{ts} is the Kronecker delta, $Q \in \mathbb{R}^{n \times n}$ is non-negative definite, and $R \in \mathbb{R}^{m \times m}$ is positive definite, and $S \in \mathbb{R}^{n \times m}$ is the cross-covariance, which must satisfy $R - SQS^\top \geq 0$. Inputs can be incorporated into the LDS model via a simple modification of Equations 2.1. See [11] for details.

2.1.1 Inference

In this section we describe the forwards and backwards inference algorithms for LDS. More details can be found in several sources [51, 62, 117].

Given a known model, the distribution over state at time t, $\mathbb{P}\left[X_t \mid y_{1:T}\right]$, can be exactly computed in two parts: a forward and a backward recursive pass. The forward pass, which is dependent on the initial state $x(h_1)$ and the observations $y_{1:t}$, is known as the *Kalman filter*, and the backward pass, which uses the observations from y_T to y_{t+1}, is known as the *Rauch-Tung-Striebel* (RTS) equations. The combined forward and backward passes are together called the *Kalman smoother*. It is worth noting that the standard LDS filtering and smoothing inference algorithms [48, 82] are instantiations of the junction tree algorithm for Bayesian Networks on the dynamic Baysian network described in Figure 2.1 (see, for example, Murphy [70]).

The Forward Pass (Kalman Filter)

Let the mean and covariance of the belief state estimate $\mathbb{P}\left[X_t \mid y_{1:t}\right]$ at time t be denoted by $\hat{x}(h_t)$ and $\hat{P}_t$ respectively. The estimates $\hat{x}(h_t)$ and $\hat{P}(h_t)$ can be predicted from the previous time step and the previous observation by Equations 2.3a–f. First we estimate the next state and next state covariance *without* correcting for an observation:

$$\bar{x}(h_{t+1}) = A\hat{x}(h_t) \tag{2.3a}$$

$$\bar{P}(h_{t+1}) = A\hat{P}(h_t)A^\top + Q \tag{2.3b}$$

Equation 2.3a can be thought of as applying the dynamics matrix A to the mean to form an initial prediction of $\hat{x}(h_t)$. Similarly, Equation 2.3b can be interpreted as using the dynamics matrix A and error covariance Q to form an initial estimate of the belief covariance $\hat{P}(h_{t+1})$. The estimates

are then adjusted:

$$\hat{x}(h_{t+1}) = \bar{x}(h_{t+1}) + K_t e_t \tag{2.3c}$$

$$\hat{P}_{t+1} = \bar{P}_{t+1} - K_t C \bar{P}_{t+1} \tag{2.3d}$$

where the error in prediction at the previous time step (the innovation) e_{t-1} and the Kalman gain matrix K_{t-1} are computed as follows:

$$e_t = y_t - C\bar{x}(h_t) \tag{2.3e}$$

$$K_t = (\bar{P}(h_t)C + S)^\top (C\bar{P}(h_t)C^{\mathrm{T}} + R)^{-1} \tag{2.3f}$$

The weighted error in Equation 2.3c corrects the predicted mean given an observation, and Equation 2.3d reduces the variance of the belief by an amount proportional to the observation covariance. Taken together, Equations 2.3a-f define a specific form of the Kalman filter known as the *forward innovation model*.

2.2 The Constant-Covariance Kalman Filter

In this chapter we will be focussing on learning the parameters A, C, and K of the *constant-covariance* Kalman filter. As mentioned in the previous section, the Kalman filter is given by the forward innovation model (Equations 2.3a-f). We make the additional assumption that the covariance of this Kalman filter, $P(h_t)$, does not vary with time t, and we assume that the linear dynamical system has a stationary covariance

$$\Sigma_{X,X} \stackrel{\text{def}}{=} \mathbb{E}\left[x(h_t)x(h_t)^\top\right] \tag{2.4}$$

where $\Sigma_{X,X}$ is independent of time t. Additionally, we define the following output covariances:

$$\begin{aligned}\Lambda &\stackrel{\text{def}}{=} \mathbb{E}\left[y_t y_t^\top\right] \\ &= \mathbb{E}\left[(Cx(h_t) + v_t)(Cx(h_t) + v_t)^\top\right] \\ &= C\mathbb{E}\left[x(h_t)x(h_t)^\top\right]C^\top + \mathbb{E}\left[v_t v_t^\top\right] \\ &= C\Sigma_{X,X}C^\top + R\end{aligned} \tag{2.5}$$

and

$$\begin{aligned}G &\stackrel{\text{def}}{=} \mathbb{E}\left[x(h_{t+1})y_t^\top\right] \\ &= \mathbb{E}\left[(Ax(h_t) + w_t)(Cx(h_t) + v_t)^\top\right] \\ &= A\mathbb{E}\left[x(h_t)x(h_t)^\top\right]C^\top + \mathbb{E}\left[w_t v_t^\top\right] \\ &= A\Sigma_{X,X}C^\top + S\end{aligned} \tag{2.6}$$

The fact that the constant state covariance $P(h_t) = \Sigma_{XX}$ implies that the Kalman gain $K = (G - AP_tC^\top)(\Lambda - CP_tC^\top)^{-1}$ [117] can be written

$$\begin{aligned}K &= (G - AP_tC^\top)(\Lambda - CP_tC^\top)^{-1} \\ &= (G - A\Sigma_{XX}C^\top)(\Lambda - C\Sigma_{XX}C^\top)^{-1} \\ &= SR^{-1}\end{aligned} \tag{2.7}$$

Learning the Kalman filter from data involves finding the parameters $\theta = \{A, C, S, R\}$ that explain the observed data. In principle, the *maximum likelihood* solution for these parameters can be found through the iterative EM algorithm; but in practice the EM algorithm often fails due to local optima [35]. An alternative approach is to use *subspace system identification* methods to compute an asymptotically unbiased solution in closed form. In practice, subspace identification is faster, more computationally robust, and easier to implement than maximum likelihood approaches [35]; so that is what we consider here.

Subspace methods calculate the parameters of an LDS by using tools from linear algebra including the oblique projection and the *singular value decomposition* (SVD) [39] to find Kalman filter estimates of the underlying state sequence in closed form. We discuss a novel subspace identification algorithm here. (See [117] for variations.)

2.2.1 Observable Representations

The key insight to spectral system identification is that the parameters of a LDS can be written solely in terms of *observable quantities*. In this section we will show how to write down an observable representation of a Kalman filter and in Section 2.3 we will design a spectral learning algorithm that leverages this observable representation.

We begin by defining infinite "matrices" of observations. Let $Y_{0|N_P}$ be defined as:

$$Y_{0|N_P} \stackrel{\text{def}}{=} \begin{bmatrix} y_0 & y_1 & \cdots \\ y_1 & y_2 & \ddots \\ \vdots & \vdots & \ddots \\ y_{N_P} & y_{N_P+1} & \cdots \end{bmatrix}_{m \cdot N_P \times \infty} \tag{2.8}$$

We will use H to denote a matrix of past observations or histories. We can think of this matrix as consisting of columns of N_P-length histories h_t. Specifically, we define

$$H \stackrel{\text{def}}{=} Y_{0|N_P} = [h_1 \; h_2 \; \ldots]$$

Similarly, let F, F^+ denote matrices of "future" observations and the one-step shifted future respectively. We can think of these matrices as consisting of columns of N_F-length future sequences of observations f_t. These are defined as

$$F \stackrel{\text{def}}{=} Y_{N_P+1|N_P+N_F} = [f_1 \; f_2 \; \ldots] \qquad F^+ \stackrel{\text{def}}{=} Y_{N_P+2|N_P+N_F+1} = [f_2 \; f_3 \; \ldots]$$

Note that the matrices H, F and F^+ all have the same form: in each matrix, each block of rows is equal to the previous block but shifted by a constant number of columns. Such matrices are called *block Hankel* matrices and play an important role in spectral system identification [62]. Finally, let

$$X = [x(h_1) \; x(h_2) \; \ldots] \in \mathbb{R}^{n \times \infty} \tag{2.9}$$

be a set of Kalman filter state estimates derived from the same set of observations. These state estimates are, by definition, an expected set of features given a history that are *sufficient* to predict future observations.

Thus, if the observations arise from an LDS, then, by the forward innovation model (Equation 2.1):

$$\mathbb{E}\left[F \mid X\right] = \begin{bmatrix} Cx(h_1) & Cx(h_2) & \cdots \\ CAx(h_1) & CAx(h_2) & \cdots \\ CA^2x(h_1) & CA^2x(h_2) & \cdots \\ \vdots & \vdots & \ddots \\ CA^{N_F-1}x(h_1) & CA^{N_F-1}x(h_2) & \cdots \end{bmatrix}_{m \cdot N_F \times \infty} \tag{2.10}$$

Next we define several matrices that will eventually let us learn the parameters of a LDS. Let Γ (sometimes called the extended observability matrix) be defined:

$$\Gamma \stackrel{\text{def}}{=} \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{N_F-1} \end{bmatrix}_{m \cdot N_F \times n} \tag{2.11}$$

Γ allows one to compute the expected future given state: e.g. $\mathbb{E}[f_1 \mid h_1] = \Gamma x(h_1)$. And, the rows of Γ allow us to compute the expectation of current and future observations: e.g. $\mathbb{E}[y_{N_P+1} \mid h_1] = \Gamma_{1:m} x(h_1)$.

Next we define the covariance of states and N_P-length histories $\Sigma_{X,\mathcal{H}} \in \mathbb{R}^{n \times N_P}$:

$$\begin{aligned} \left[\Sigma_{X,\mathcal{H}}\right]_{i,j} &\stackrel{\text{def}}{=} \mathbb{E}\left[X_{i,t} H_{j,t}^\top\right] \\ &= \mathbb{E}\left[\mathbb{E}[X_{i,t} \mid h_t]\mathbb{E}[H_{j,t}^\top \mid h_t]\right] \\ &= \mathbb{E}[x_i(h_t)\mathbb{E}[H_{j,t}^\top \mid h_t]] \\ \implies \Sigma_{X,\mathcal{H}} &= \mathbb{E}[x(h_t)\mathbb{E}[H_t^\top \mid h_t]] \end{aligned} \tag{2.12}$$

Although we cannot *directly* estimate this matrix from data, it plays a central role in our derivations below. In particular, if we define a matrix

$$\begin{aligned} \left[\Sigma_{\mathcal{H},\mathcal{H}}\right]_{i,j} &\stackrel{\text{def}}{=} \mathbb{E}\left[H_{i,t} H_{j,t}^\top\right] \\ \implies \Sigma_{\mathcal{H},\mathcal{H}} &= \mathbb{E}\left[H_t H_t^\top\right] \end{aligned} \tag{2.13}$$

it can be used in conjunction with $\Sigma_{X,\mathcal{H}}$ to find the Kalman filter state estimate at time t by orthogonal projection onto history [48]:

$$x(h_t) = \Sigma_{X,\mathcal{H}} \Sigma_{\mathcal{H},\mathcal{H}}^{-1} h_t \tag{2.14}$$

The Kalman filter state estimate is optimal linear predictor x_t from history h_t, an observation that Kalman made in his seminal paper [48]. The derivation of Equation 2.14 is somewhat complicated, but the complete proof in the context of spectral system identification can be found

in [117]. Equation 2.14 also implies that the state covariance $\Sigma_{X,X} = \mathbb{E}[x(h_t)x(h_t)^\top]$ can be found from $\Sigma_{X,\mathcal{H}}$ and $\Sigma_{\mathcal{H},\mathcal{H}}$:

$$\begin{aligned}
\Sigma_{X,X} &= \mathbb{E}[x(h_t)x(h_t)^\top] \\
&= \mathbb{E}\left[\Sigma_{X,\mathcal{H}}\Sigma_{\mathcal{H},\mathcal{H}}^{-1}h_t h_t^\top \Sigma_{\mathcal{H},\mathcal{H}}^{-1}\Sigma_{X,\mathcal{H}}^\top\right] \\
&= \Sigma_{X,\mathcal{H}}\Sigma_{\mathcal{H},\mathcal{H}}^{-1}\mathbb{E}\left[h_t h_t^\top\right]\Sigma_{\mathcal{H},\mathcal{H}}^{-1}\Sigma_{X,\mathcal{H}}^\top \\
&= \Sigma_{X,\mathcal{H}}\Sigma_{\mathcal{H},\mathcal{H}}^{-1}\Sigma_{X,\mathcal{H}}^\top
\end{aligned} \tag{2.15}$$

This fact is used in our derivations below.

The two fundamental matrices that allow us to define observable version of the parameters A and C for a LDS are $\Sigma_{\mathcal{F},\mathcal{H}}$ and $\Sigma_{\mathcal{F}^+,\mathcal{H}}$. First we define $\Sigma_{\mathcal{F},\mathcal{H}} \in \mathbb{R}^{N_F \times N_P}$, a covariance matrix of past and future sequences of observations:

$$\begin{aligned}
\left[\Sigma_{\mathcal{F},\mathcal{H}}\right]_{i,j} &\stackrel{\text{def}}{=} \mathbb{E}\left[F_{i,t}H_{j,t}^\top\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[F_{i,t}H_{j,t}^\top \mid h_t\right]\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[F_{i,t} \mid h_t\right]\mathbb{E}\left[H_{j,t} \mid h_t\right]\right] \\
&= \mathbb{E}\left[\Gamma_i A x(h_t)\mathbb{E}\left[H_{j,t} \mid h_t\right]\right] \\
&= \Gamma_i A\mathbb{E}\left[x(h_t)\mathbb{E}\left[H_{j,t} \mid h_t\right]\right] \\
&= \Gamma_i A\Sigma_{X,\mathcal{H}_j} \\
\implies \Sigma_{\mathcal{F},\mathcal{H}} &= \Gamma A\Sigma_{X,\mathcal{H}}
\end{aligned} \tag{2.16}$$

Equation 2.16 tells us that the rank of $\Sigma_{\mathcal{F},\mathcal{H}}$ is no more than n, since its factors Γ and $\Sigma_{X,\mathcal{H}}$ each have rank no more than n. At this point we can define a *sufficient set* of histories and futures: it is a set of histories for which the rank of $\Sigma_{\mathcal{F},\mathcal{H}}$ is *equal* to n.

Next we define $\Sigma_{\mathcal{F}^+,\mathcal{H}} \in \mathbb{R}^{N_F \times N_P}$, a covariance matrix of past and one-step shifted futures:

$$\begin{aligned}
\left[\Sigma_{\mathcal{F}^+,\mathcal{H}}\right]_{i,j} &\stackrel{\text{def}}{=} \mathbb{E}\left[F_{i,t}^+H_{j,t}^\top\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[F_{i,t}^+H_{j,t}^\top \mid h_t\right]\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[F_{i,t}^+ \mid h_t\right]\mathbb{E}\left[H_{j,t} \mid h_t\right]\right] \\
&= \mathbb{E}\left[\Gamma_i A^2 x(h_t)\mathbb{E}\left[H_{j,t} \mid h_t\right]\right] \\
&= \Gamma_i A^2\mathbb{E}\left[x(h_t)\mathbb{E}\left[H_{j,t} \mid h_t\right]\right] \\
&= \Gamma_i A^2\Sigma_{X,\mathcal{H}_j} \\
\implies \Sigma_{\mathcal{F}^+,\mathcal{H}} &= \Gamma A^2\Sigma_{X,\mathcal{H}}
\end{aligned} \tag{2.17}$$

Just like $\Sigma_{\mathcal{F},\mathcal{H}}$, $\Sigma_{\mathcal{F}^+,\mathcal{H}}$ has rank at most n due to Γ and $\Sigma_{\mathcal{F},\mathcal{H}}$.

Finally, we need one additional covariance matrix in order to estimate the Kalman gain:

$$
\begin{aligned}
[\Sigma_{\mathcal{F}^+,y}]_{i,j} &\stackrel{\text{def}}{=} \mathbb{E}\left[F^+_{i,t} y^\top_{j,t}\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[F^+_{i,t} y^\top_{j,t} \mid h_{t+1}\right]\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[F^+_{i,t} \mid h_{t+1}\right] \mathbb{E}\left[y_{j,t} \mid h_{t+1}\right]\right] \\
&= \mathbb{E}\left[\Gamma_i A x(h_{t+1}) \mathbb{E}\left[y_{j,t} \mid h_{t+1}\right]\right] \\
&= \Gamma_i A \mathbb{E}\left[x(h_t+1) \mathbb{E}\left[y_{j,t} \mid h_{t+1}\right]\right] \\
&= \Gamma_i A \Sigma_{X^+,Y_j} \\
\implies \Sigma_{\mathcal{F}^+,\mathcal{H}} &= \Gamma A \Sigma_{X^+,Y}
\end{aligned}
\tag{2.18}
$$

where $\Sigma_{X^+,Y} \stackrel{\text{def}}{=} \mathbb{E}\left[x(h_t+1)\mathbb{E}\left[y_t \mid h_{t+1}\right]\right]$. We are now in position to define the parameters of a Kalman filter in terms of observable covariance matrices. The key to this approach is that, for linear dynamical systems, there are many possible *equivalent* representations for the same system. That is, two LDSs are said to be equivalent if the second order statistics of the output generated by the models is the same, i.e. the covariance sequence of the output is identical [117].[2] This means that the Kalman filter is only defined up to a similarity transform: the predictions made from any of these systems is the same. For example, given a linear transform S, we can define parameters $\tilde{A} \stackrel{\text{def}}{=} SAS^{-1}$, $\tilde{C} \stackrel{\text{def}}{=} CS^{-1}$, $\tilde{K} \stackrel{\text{def}}{=} SK$, and $\tilde{x}(h_t) \stackrel{\text{def}}{=} Sx(h_t)$. Predictions from the transformed system are the same as the original system, e.g. $\tilde{C}\tilde{x}(h_t) = CS^{-1}Sx(h_t) = Cx(h_t)$. Filtering and simulating from the filter are similarly equivalent. See Equations 2.20 below for more details.

Practically, the equivalence of transformed LDSs means that we can pick an additional matrix $U \in \mathbb{R}^{N_F \times n}$ such that $U^\top \Gamma$ is invertible and we can work with parameters transformed by $U^\top \Gamma A$.[3] A natural choice for U is the leading left singular vectors of $\Sigma_{\mathcal{F},\mathcal{H}}$, although a randomly generated U will work with probability 1 (but will typically result in slower learning). We now define the parameters A and C of an LDS in terms of the observable matrices $\Sigma_{\mathcal{F},\mathcal{H}}$, $\Sigma_{\mathcal{F}^+,\mathcal{H}}$, and U, and simplify the definitions using Equations 2.19, to show that our parameters are only a

[2]The definition of equivalence is that the entire PDF of observation sequences is the same. But it turns out that all we need to check is second order stats, since the joint distribution is Gaussian.

[3]We assume without loss of generality that A is full rank, therefore, the transform $U^\top \Gamma A$ is invertible since both $U^\top \Gamma$ and A are assumed to be invertible.

similarity transform away from the original LDS parameters:

$$
\begin{aligned}
\tilde{x}(h_t) &\stackrel{\text{def}}{=} U^\top \Sigma_{\mathcal{F},\mathcal{H}} \Sigma_{\mathcal{H},\mathcal{H}}^{-1} h_t \\
&= (U^\top \Gamma A) \Sigma_{X,\mathcal{H}} \Sigma_{\mathcal{H},\mathcal{H}}^{-1} h_t \\
&= (U^\top \Gamma A) x(h_t) && (2.19a) \\
\tilde{A} &\stackrel{\text{def}}{=} U^\top \Sigma_{\mathcal{F}^+,\mathcal{H}} (U^\top \Sigma_{\mathcal{F},\mathcal{H}})^\dagger \\
&= U^\top \Gamma A^2 \Sigma_{X,\mathcal{H}} (U^\top \Sigma_{\mathcal{F},\mathcal{H}})^\dagger \\
&= (U^\top \Gamma A) A (U^\top \Gamma A)^{-1} (U^\top \Gamma A) \Sigma_{X,\mathcal{H}} (U^\top \Sigma_{\mathcal{F},\mathcal{H}})^\dagger \\
&= (U^\top \Gamma A) A (U^\top \Gamma A)^{-1} && (2.19b) \\
\tilde{C} &\stackrel{\text{def}}{=} U_{1:m} \tilde{A}^{-1} \\
&= U_{1:m} ((U^\top \Gamma A) A (U^\top \Gamma A)^{-1})^{-1} \\
&= U_{1:m} (U^\top \Gamma A) A^{-1} (U^\top \Gamma A)^{-1} \\
&= \Gamma_{1:m} A A^{-1} (U^\top \Gamma A)^{-1} \\
&= \Gamma_{1:m} (U^\top \Gamma A)^{-1} \\
&= C (U^\top \Gamma A)^{-1} && (2.19c)
\end{aligned}
$$

Here we used the fact that the first m rows of Γ are C from the definition of Γ (Equation 2.11). In order to filter the system or to simulate from the system we additionally need to find the observation covariance matrix R and the Kalman gain K. The Kalman gain can be found given R and a similarity transform of S by a modification of Equation 2.7:

$$
\begin{aligned}
R &\stackrel{\text{def}}{=} \Lambda - \tilde{C} U^\top \Sigma_{\mathcal{F},\mathcal{H}} \Sigma_{\mathcal{H},\mathcal{H}}^{-1} \Sigma_{\mathcal{F},\mathcal{H}}^\top U \tilde{C}^\top \\
&= \Lambda - \tilde{C} (U^\top \Gamma A) \Sigma_{X,\mathcal{H}} \Sigma_{\mathcal{H},\mathcal{H}}^{-1} \Sigma_{X,\mathcal{H}}^\top (U^\top \Gamma A)^\top \tilde{C}^\top \\
&= \Lambda - C (U^\top \Gamma A)^{-1} (U^\top \Gamma A) \Sigma_{X,\mathcal{H}} \Sigma_{\mathcal{H},\mathcal{H}}^{-1} \Sigma_{X,\mathcal{H}}^\top (U^\top \Gamma A)^\top (U^\top \Gamma A)^{-\mathsf{T}} C^\top \\
&= \Lambda - C \Sigma_{X,\mathcal{H}} \Sigma_{\mathcal{H},\mathcal{H}}^{-1} \Sigma_{X,\mathcal{H}}^\top C^\top \\
&= \Lambda - C \Sigma_{X,X} C^\top && (2.19d) \\
\tilde{S} &\stackrel{\text{def}}{=} U^\top \Sigma_{F^+,Y} - U^\top \Sigma_{\mathcal{F}^+,\mathcal{H}} \Sigma_{\mathcal{H},\mathcal{H}}^{-1} \Sigma_{\mathcal{F},\mathcal{H}}^\top U \tilde{C}^\top \\
&= (U^\top \Gamma A) \Sigma_{X^+,Y} - (U^\top \Gamma A) A \Sigma_{X,\mathcal{H}} \Sigma_{\mathcal{H},\mathcal{H}}^{-1} \Sigma_{\mathcal{F},\mathcal{H}}^\top U \tilde{C}^\top \\
&= (U^\top \Gamma A) \Sigma_{X^+,Y} - (U^\top \Gamma A) A \Sigma_{X,\mathcal{H}} \Sigma_{\mathcal{H},\mathcal{H}}^{-1} \Sigma_{X,\mathcal{H}}^\top C^\top \\
&= (U^\top \Gamma A) \Sigma_{X^+,Y} - (U^\top \Gamma A) A \Sigma_{X,X} C^\top \\
&= (U^\top \Gamma A) (\Sigma_{X^+,Y} - A \Sigma_{X,X} C^\top) \\
&= (U^\top \Gamma A) S && (2.19e) \\
\tilde{K} &\stackrel{\text{def}}{=} \tilde{S} R^{-1} \\
&= (U^\top \Gamma A) S R^{-1} \\
&= (U^\top \Gamma A) K && (2.19f)
\end{aligned}
$$

We have now defined a Kalman filter in terms of only observable quantities. Given Equations 2.19(a–f), the state update equations for our transformed Kalman filter are

$$\begin{aligned} y_t &= \tilde{C}\tilde{x}(h_t) \\ &= C(U^\top \Gamma A)^{-1}(U^\top \Gamma A)x(h_t)) \\ &= Cx(h_t) \end{aligned} \tag{2.20}$$

$$\begin{aligned} \tilde{x}(h_{t+1}) &= \tilde{A}\tilde{x}(h_t) + \tilde{K}(y_t - \tilde{C}\tilde{x}(h_t)) \\ &= (U^\top \Gamma A)A(U^\top \Gamma A)^{-1}(U^\top \Gamma A)x(h_t) + (U^\top \Gamma A)K(y_t - C(U^\top \Gamma A)^{-1}(U^\top \Gamma A)x(h_t)) \\ &= (U^\top \Gamma A)Ax(h_t) + (U^\top \Gamma A)K(y_t - Cx(h_t)) \\ &= (U^\top \Gamma A)(Ax(h_t) + K(y_t - Cx(h_t))) \end{aligned} \tag{2.21}$$

The fact that our Kalman filter is defined in terms of observable quantities means that developing a learning algorithm is straightforward.

2.3 A Spectral Learning Algorithm

Our learning algorithm works by building empirical estimates $\widehat{\Sigma}_{\mathcal{F},\mathcal{H}}, \widehat{\Sigma}_{\mathcal{F}^+,\mathcal{H}}, \widehat{\Sigma}_{\mathcal{H},\mathcal{H}}, \widehat{\Sigma}_{F+,Y}$, and $\widehat{\Lambda}$ of the matrices $\Sigma_{\mathcal{F},\mathcal{H}}, \Sigma_{\mathcal{T}^+,\mathcal{H}}, \Sigma_{\mathcal{H},\mathcal{H}}\Sigma_{F+,Y}$, and Λ defined above. To build these estimates, we repeatedly sample a history h_t from the distribution ω and record the resulting observations. This data gathering strategy implies that we must be able to arrange for the system to be in a state corresponding to $h_t \sim \omega$.

In practice, reset is often not available. In this case we can estimate $\widehat{\Sigma}_{\mathcal{F},\mathcal{H}}, \widehat{\Sigma}_{\mathcal{F}^+,\mathcal{H}}, \widehat{\Sigma}_{\mathcal{H},\mathcal{H}}, \widehat{\Sigma}_{F+,Y}$, and $\widehat{\Lambda}$, by dividing a single long sequence of action-observation pairs into subsequences and pretending that each subsequence started with a reset. We are forced to use an initial distribution over histories, ω, equal to the steady state distribution of the system.

Once we have computed $\widehat{\Sigma}_{\mathcal{F},\mathcal{H}}$, we can generate $\widehat{U}$ by singular value decomposition of $\widehat{\Sigma}_{\mathcal{F},\mathcal{H}}$. We can then learn the LDS parameters by plugging $\widehat{U}$, $\widehat{\Sigma}_{\mathcal{F},\mathcal{H}}, \widehat{\Sigma}_{\mathcal{F}^+,\mathcal{H}}, \widehat{\Sigma}_{\mathcal{H},\mathcal{H}}, \widehat{\Sigma}_{F+,Y}$, and $\widehat{\Lambda}$ into Equations 2.19(a–f).

As we include more data in our averages, the law of large numbers guarantees that our estimates $\widehat{\Sigma}_{\mathcal{F},\mathcal{H}}, \widehat{\Sigma}_{\mathcal{F}^+,\mathcal{H}}, \widehat{\Sigma}_{\mathcal{H},\mathcal{H}}, \widehat{\Sigma}_{F+,Y}$, and $\widehat{\Lambda}$ converge to the true matrices $\Sigma_{\mathcal{F},\mathcal{H}}, \Sigma_{\mathcal{T}^+,\mathcal{H}}, \Sigma_{\mathcal{H},\mathcal{H}}\Sigma_{F+,Y}$, and Λ. So by continuity of the formulas above, if our system is truly a LDS of finite rank, our estimates $\widehat{A}$, $\widehat{C}$, $\widehat{R}$, and $\widehat{K}$ converge to the true parameters up to a linear transform—that is, our learning algorithm is *consistent*.[4]

Below we discuss some of the important design choices and extensions to the above learning algorithm that are often important for learning dynamical system models in practice.

[4]The pseudoinverses are continuous at the true parameters, since the matrices to be pseudoinverted have full rank. The matrix of n left singular vectors $\widehat{U}$ may not be a continuous function of $\widehat{\Sigma}_{\mathcal{F},\mathcal{H}}$ (in case of repeated singular values); to deal with this possibility, we can either fix $\widehat{U}$ (say, as the left singular vectors of our estimated $\widehat{\Sigma}_{\mathcal{F},\mathcal{H}}$ after some fixed amount of data), or we can make a slightly more complex argument based on the fact that the *column span* of $\widehat{U}$ is a continuous function of $\widehat{\Sigma}_{\mathcal{F},\mathcal{H}}$ near $\Sigma_{\mathcal{F},\mathcal{H}}$ (since the nth singular value of $\Sigma_{\mathcal{F},\mathcal{H}}$ is nonzero, and is therefore separated from the $(n+1)^{\text{st}}$, which is zero).

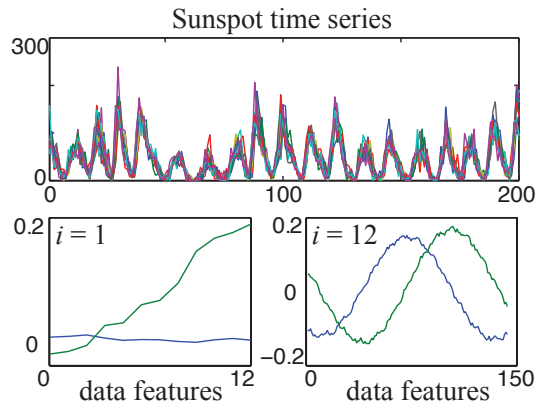

Figure 2.2: Sunspot data, sampled monthly for 200 years. Each curve is a month, the x-axis is over years. Below the graph are the first two principal components of $\mathcal{O}_i$ where Y_f and Y_p each consist of 1-observation Hankel matrices and 12-observation Hankel matrices. The 1-observation Hankel matrices do not contain enough observations to recover a state which accurately reflects the temporal patterns in the data, while the 12-observation Hankel matrices do.

2.3.1 Learning from Sequences of Observations

Having multiple observations per column in F is particularly important when the underlying dynamical system is not completely observable. For example, Figure 2.2 shows 200 years of sunspot numbers, with each month modeled as a separate variable. Sunspots are known to have two periods, the longer of which is 11 years. When spectral learning is performed using Hankel matrices with $i = 12$, the first two principal components of $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$ resemble the sine and cosine bases, and the corresponding state variables are the coefficients needed to combine these bases so as to predict 12 years of data. This is in contrast to the bases obtained by SVD on a 1-observation Y_f and Y_p, which reconstruct just the variation within a single year. Thus, with $i = 1$ the state estimate *will not* converge to the true Kalman filter state estimate even if $j \to \infty$.

2.3.2 Stability

Stability is a property of dynamical systems defined in terms of *equilibrium points*. If all solutions of a dynamical system that start out near an equilibrium state x_e stay near or converge to x_e, then the state x_e is stable or asymptotically stable respectively (see Figure 2.3.2). A linear system $x(h_t) = Ax(h_t)$ is *internally stable* if the matrix A obeys the Lyapunov stability criterion (see below). The standard algorithms for learning linear Gaussian systems *do not enforce stability*; when learning from finite data samples, the spectral learning solution may be unstable even if the true system is stable, due to the sampling constraints, modeling errors, and measurement noise. A dynamics matrix A is said to be asymptotically stable in the sense of Lyapunov if, for any given positive semi-definite symmetric matrix Q there exists a positive-definite symmetric

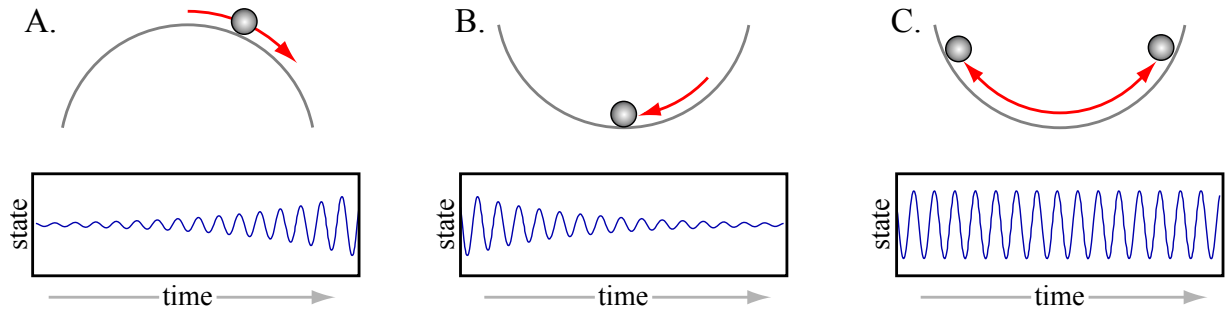

Figure 2.3: System equilibria. (A) Unstable equilibrium. The state vector will rapidly move away from the equilibrium point when perturbed. (B) Asymptotically stable equilibrium. The state vector will return to the original equilibrium point when perturbed. (C.) Stable equilibrium. No "resistance;" a perturbed state vector will oscillate forever around the equilibrium point. Note that the notion of *asymptotic stability* is stronger than *stability*.

matrix P that satisfies the following *Lyapunov criterion*:

$$P - APA^{\mathrm{T}} = Q \tag{2.22}$$

An LDS is said to be asymptotically stable in the sense of Lyapunov if its dynamics matrix A is. Thus, the Lyapunov criterion can be interpreted as holding for an LDS if, for a given covariance matrix, there exists a belief distribution where the predicted belief over state is equivalent to the previous belief over state, that is, if there exists an equilibrium point. Below we will relate stable in the sense of Lyapunov to stability and asymptotic stability

It is interesting to note that the Lyapunov criterion holds *iff* the spectral radius $\rho(A) \leq 1$. Recall that a matrix M is positive semi-definite *iff* $zMz^{\mathrm{T}} \geq 0$ for all non-zero vectors z. Let λ be an left eigenvalue of A and ν be a corresponding eigenvector, giving us $\nu A = \nu\lambda$, then

$$\nu Q \nu^{\mathrm{T}} = \nu(P - A^{\mathrm{T}}PA)\nu^{\mathrm{T}} = \nu P \nu^{\mathrm{T}} - \nu\lambda P \lambda \nu^{\mathrm{T}} = \nu P \nu^{\mathrm{T}}(1 - |\lambda|^2) \tag{2.23}$$

Since $\nu P \nu^{\mathrm{T}} \geq 0$, it follows that $|\lambda| \leq 1$ is equivalent to $\nu Q \nu^{T} \geq 0$, and therefore to Equation 2.23. When $\rho(A) < 1$, the system is asymptotically stable. To see this, suppose $D\Lambda D^{-1}$ is the eigen-decomposition of A, where Λ has the eigenvalues of A along the diagonal and D contains the eigenvectors. Then,

$$\lim_{k\to\infty} A^k = \lim_{k\to\infty} D\Lambda^k D^{-1} = D\left(\lim_{k\to\infty} \Lambda^k\right) D^{-1} = 0 \tag{2.24}$$

since it is clear that $\lim_{k\to\infty} \Lambda^k = 0$. If $\Lambda = I$, then A is stable but not asymptotically stable and the state oscillates around x_e indefinitely. However, this is true only under the assumption of zero noise. In the case of an LDS with Gaussian noise, a dynamics matrix with unit spectral radius would cause the state estimate to move steadily away from x_e. Hence such an LDS is asymptotically stable only when $\rho(A)$ is strictly less than 1.

Stability is a desirable characteristic for linear dynamical systems, but it is often ignored by algorithms that learn these systems from data. When the amount of training data is small, the

learned system is often unstable $\rho(A) > 1$. In Chapter 7 and [88] we propose a novel method for learning guaranteed stable linear dynamical systems: we formulate an approximation of the problem as a convex program, start with a solution to a relaxed version of the program, and incrementally add constraints to improve stability. We show that this approach can drastically increase the performance and utility of learned Kalman filters in practice.

2.4 Conclusions

In this chapter we presented a spectral learning algorithm for constant-covariance Kalman filters. This algorithm is novel, though quite similar to previous *subspace identification* algorithms for linear dynamical systems [51, 117]. Additional experimental results using this algorithm (in the context of learning stable Kalman filters) can be found in Chapter 7.

Several ideas that are featured in this chapter feature prominently in our contributions to learning the more general algorithms discussed in the next several chapters. In particular, we exploit the fact that dynamical systems can often be written in terms of *observable* covariance matrices and states can be viewed as *predictive* statistics. When this is the case, the state space of the dynamical system can be found by a spectral decomposition of the covariance of past and future observations. Finally, the state update can be found via application of a (potentially generalized) version of Bayes' rule.

Chapter 3

Spectral Learning Algorithms for Predictive State Representations

3.1 Introduction

In the previous chapter we looked the problem of learning a Kalman filter directly from data. In this case we assumed a latent variable model with linear transition and observation functions and Gaussian noise. For many systems, these assumptions are overly restrictive. In this chapter we will consider learning models of dynamical systems where the transitions are nonlinear, the states are discrete, and the noise model is assumed to be *multinomial*. With a large number of states, these dynamical systems can be very expressive.

Two well-known multinomial dynamical system models are Hidden Markov Models (HMMs) [79] and Partially Observable Markov Decision Processes (POMDPs) [22, 97]. Here we will focus on a general multinomial-distributed dynamical system model called a *Predictive State Representation (PSR)* [61] which inlcudes HMMs and POMDPs as special cases. PSRs and the closely related *Observable Operator Models (OOMs)* [44] are compact and complete descriptions of a dynamical system. PSRs can be viewed as generalizations of POMDPs that have attracted interest because they both have greater representational capacity than POMDPs and yield representations that are *at least* as compact [31, 93]. In contrast to the latent-variable representations of POMDPs, PSRs and OOMs represent the state of a dynamical system by tracking occurrence probabilities of a set of future events (called *tests* or *characteristic events*) conditioned on past events (called *histories* or *indicative events*). Because tests and histories are observable quantities, it has been suggested that learning PSRs and OOMs should be easier than learning POMDPs.

In this chapter we develop a spectral learning algorithm for PSRs (and thus POMDPs) that shares the advantages of spectral learning algorithms for linear dynamical systems: the algorithm is statistically consistent and easy to implement with simple linear algebra operations. The chapter is organized as follows. In Section 3.2 we will present the PSR model for dynamical systems. Then in Section 3.3 we will describe the observable representation of PSRs. Finally, in Section 3.4 we will leverage the observable representation to develop a simple spectral learning algorithm for PSRs.

a set of tests	$\mathcal{T}$
a single test in $\mathcal{T}$	τ_i
the observations in test τ_i	$\tau_i^{\mathcal{O}}$
the actions in test τ_i	$\tau_i^{\mathcal{A}}$
the prediction of a test τ_i at time t	$\mathbb{P}\left[\tau_{i,t}^{\mathcal{O}} \mid \tau_{i,t}^{\mathcal{A}}, h_t\right]$
the prediction vector of all tests τ at time t	$\mathbb{P}\left[\tau_t^{\mathcal{O}} \mid \tau_t^{\mathcal{A}}, h_t\right]$
shorthand for the prediction of test τ_i at time t	$\tau_i(h_t)$
shorthand for the prediction of all test outcomes at time t	$\tau(h_t)$

Table 3.1: Notation for Tests.

3.2 Predictive State Representations

3.2.1 Tests and PSR Notation

PSRs represent state as a set of predictions of observable experiments or *tests* that one could perform in the system. The notation for tests is summarized in Table 3.1. Specifically, a test of length N_F is an ordered sequence of future action-observations pairs $\tau_i = a_1, o_1, \ldots a_{N_F}, o_{N_F}$ that can be executed and observed at any time t. Likewise, a *history* is an ordered sequence of actions and observations $h_t = a_1, o_1, \ldots, a_{t-1}, o_{t-1}$ that has been executed and observed prior to a given time t. We will often work with *sets* of tests; therefore, let $\mathcal{T} = \{\tau_i\}$ be a set of d tests. A test τ_i is said to be *executed* at time t if we intervene to take the sequence of actions specified by the test $\tau_i^{\mathcal{A}} = a_1, \ldots, a_{N_F}$. It is said to *succeed* at time t if it is taken and if the sequence of observations in the test $\tau_i^{\mathcal{O}} = o_1, \ldots, o_{N_F}$ matches the observations generated by the system. The *prediction* for test i at time t is the probability of the test succeeding given a history h_t and given that we take it:

$$\mathbb{P}\left[\tau_{i,t}^{\mathcal{O}} \mid \text{do}\left(\tau_{i,t}^{\mathcal{A}}\right), h_t\right] = \mathbb{P}[o_1 \mid a_1, h_t] \prod_{i=2}^{N_F} \mathbb{P}\left[o_i \mid a_{1:i}, o_{1:i-1}, h_t\right] \tag{3.1}$$

We write $\tau(h_t)$ for the *prediction vector* of success probabilities for the tests $\tau_i \in \mathcal{T}$ given a history h_t, with elements:

$$\tau_i(h_t) = \mathbb{P}\left[\tau_{i,t}^{\mathcal{O}} \mid \text{do}\left(\tau_{i,t}^{\mathcal{A}}\right), h_t\right]$$

The key idea behind a PSR is that if we know the expected outcomes of executing all possible tests, then we also know everything there is to know about the state of a dynamical system [93].

Knowing the success probabilities of some tests may allow us to compute the success probabilities of other tests. That is, given a test τ_l and a prediction vector $\tau(h_t)$, there may exist a *prediction function* f_{τ_l} such that $\mathbb{P}\left[\tau_l^{\mathcal{O}} \mid \text{do}\left(\tau_l^{\mathcal{A}}\right), h_t\right] = f_{\tau_l}\left(\tau(h_t)\right)$. In this case, we say $\tau(h_t)$ is a *sufficient statistic* for $\mathbb{P}\left[\tau_l^{\mathcal{O}} \mid \text{do}\left(\tau_l^{\mathcal{A}}\right), h_t\right]$.

3.2.2 Linear Predictive State Representations

Formally, a PSR is a tuple $\langle \mathcal{O}, \mathcal{A}, \mathcal{Q}, \mathcal{F}, x_1, \rangle$. $\mathcal{O}$ is the set of possible observations and $\mathcal{A}$ is the set of possible actions. $\mathcal{Q}$ is a *core* set of tests, $x_i \in \mathcal{Q}$ i.e., a set whose prediction vector $x(h_t)$ is

a sufficient statistic for the prediction of *all* tests τ_l at time t. $\mathcal{F}$ is the set of prediction functions f_{τ_l} for all tests τ_l (which must exist since $\mathcal{Q}$ is a core set), and $x_1 = x(h_1)$ is the initial prediction vector after seeing the empty history h_1. In this work we will restrict ourselves to *linear* PSRs, in which all prediction functions are linear: $f_{\tau_l}(x(h_t)) = g_{\tau_l}^\top x(h_t)$ for some vector $g_{\tau_l} \in \mathbb{R}^d$.

Since $x(h_t)$ is a sufficient statistic for *all* tests, it is a *state* for our PSR: i.e., for each time t we can remember just $x(h_t)$ instead of the history h_t itself. After taking action a and seeing observation o, we can update $x(h_t)$ recursively by *extending* to a joint probability distribution of the current observation and the next state and then *conditioning* on the probability of the observation.

To see how this works, consider the following example. In a linear PSR we can predict the success of any test τ_l at time $t+1$ extended at the beginning by an action a and an observation o, which we call $ao\tau_l$, as a linear function of our core test predictions $x(h_t)$:

$$\mathbb{P}\left[\tau_{l,t+1}^{\mathcal{O}}, o_t = o \mid \text{do}\left(\tau_{l,t+1}^{\mathcal{A}}, a_t = a\right), h_t\right] = g_{ao\tau_l}^\top x(h_t) \tag{3.2}$$

If we write M_{ao} for the matrix with rows $g_{aox_l}^\top$ for each test $x_l \in \mathcal{Q}$, then we can write the joint probability of the next state and current observation as:

$$\mathbb{P}\left[x_{t+1}^{\mathcal{O}}, o_t = o \mid \text{do}\left(x_{t+1}^{\mathcal{A}}, a_t = a\right), h_t\right] = M_{ao} x(h_t) \tag{3.3}$$

Additionally, given a normalization vector $x_\infty \in \mathbb{R}^d$, defined by

$$\begin{aligned} x_\infty^\top \mathbb{P}\left[x_t^{\mathcal{O}} \mid \text{do}\left(x_t^{\mathcal{A}}\right), h_t\right] &= 1 \; (\forall t) \\ \implies x_\infty^\top x(h_t) &= 1 \; (\forall t) \end{aligned} \tag{3.4}$$

the PSR predicts that observation o will occur at time t with probability

$$\begin{aligned} \mathbb{P}\left[o_t = o \mid \text{do}\left(a_t = a\right), h_t\right] &= x_\infty^\top \mathbb{P}\left[x_{t+1}^{\mathcal{O}}, o_t = o \mid \text{do}\left(x_{t+1}^{\mathcal{A}}, a_t = a\right), h_t\right] \\ &= x_\infty^\top M_{ao} x(h_t) \end{aligned} \tag{3.5}$$

Note that here the normalization vector is marginalizing out the probability of the state from the joint probability $\mathbb{P}\left[x_{t+1}^{\mathcal{O}}, o_t = o \mid \text{do}\left(x_{t+1}^{\mathcal{A}}, a_t = a\right), h_t\right]$. After seeing observation o, Equations 3.3 and 3.5 allow us to implement the conditioning step and update $x(h_t)$ recursively using Bayes' Rule:

$$\begin{aligned} x(h_{t+1}) &= \mathbb{P}\left[x_{t+1}^{\mathcal{O}} \mid \text{do}\left(x_{t+1}^{\mathcal{A}}\right), h_{t+1}\right] \\ &= \mathbb{P}\left[x_{t+1}^{\mathcal{O}} \mid \text{do}\left(x_{t+1}^{\mathcal{A}}, a_t = a\right), o_t = o, h_t\right] \\ &= \frac{\mathbb{P}\left[x_{t+1}^{\mathcal{O}}, o_t = o \mid \text{do}\left(x_{t+1}^{\mathcal{A}}, a_t = a\right), h_t\right]}{\mathbb{P}\left[o_t = o \mid h_t, a_t = a\right]} \\ &= \frac{M_{ao} x(h_t)}{x_\infty^\top M_{ao} x(h_t)} \end{aligned} \tag{3.6}$$

We note that PSR parameters are only determined up to a similarity transform: let $S \in \mathbb{R}^{d \times d}$ be invertible. Then, if we replace $M_{ao} \to S M_{ao} S^{-1}$, $x(h_1) \to S x(h_1)$, and $x_\infty \to S^{-\top} x_\infty$, the

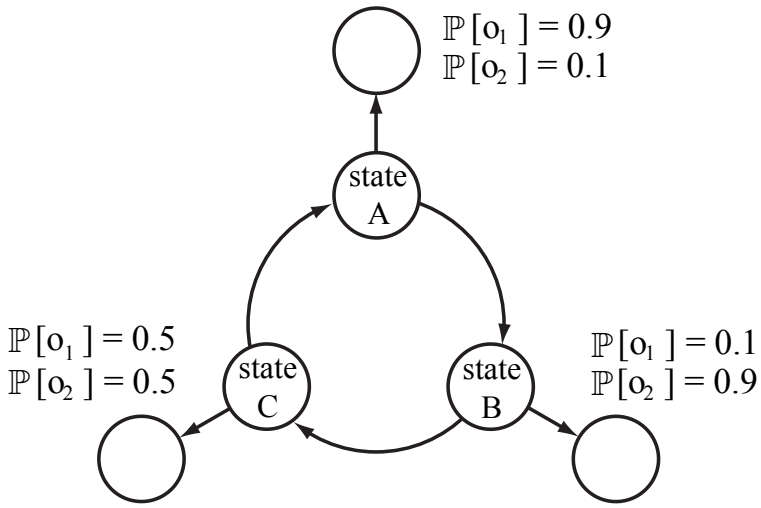

Figure 3.1: A simple 3-state 2-observation HMM. An example of a set of indicative events (a mutually exclusive and exhaustive partition of histories) for this HMM is $\mathcal{H} = \{o_1o_2, o_1o_1, o_2\}$. (E.g., o_1o_2 means that the observation at time $t-1$ is o_1, and the one at time t is o_2.) An example of a set of tests for this HMM is $\mathcal{T} = \{o_1o_2, o_1o_1, o_2o_1, o_2o_2\}$. (In this case, o_1o_2 means that the observation at time $t+1$ is o_1, and the one at time $t+2$ is o_2.)

resulting PSR makes exactly the same predictions as the original one. (The pairs $S^{-1}S$ cancel in Equations 3.5 and 3.6)

An important advantage of PSRs is that they are a natural match for fast, statistically consistent, spectral learning algorithms. In particular, we can relate a PSR's parameters to moments of directly-observable statistics, and prove bounds on the ranks of matrices of these moments. These facts suggest a simple algorithm: estimate the moments from a sample, enforce the rank constraints by projection, and then solve for estimates of the PSR parameters. Based on this intuition, we give here a spectral algorithm for learning PSRs based on an *observable representation* of a PSR, which we now define.

3.3 Observable Representations

Specifying a PSR involves first finding a core set of tests $\mathcal{Q}$, called the *discovery problem*, and then finding the parameters M_{ao}, x_∞, and x_1 for these tests, called the *learning problem*. A core set $\mathcal{Q}$ for a linear PSR is said to be *minimal* if the tests in $\mathcal{Q}$ are linearly independent [44, 93], i.e., no one test's prediction is a linear function of the other tests' predictions. Historically, the discovery problem was usually solved by searching for linearly independent tests by repeatedly performing Singular Value Decompositions (SVDs) on matrices of observations from collections of tests [123]. The learning problem was then solved by regression.

We introduce a new learning algorithm based on an *observable representation* of a PSR, that is, one where each parameter corresponds directly to observable quantities. This representation depends on a core set of tests $\mathcal{T}$ (not necessarily minimal). It also depends on a set $\mathcal{H}$ of *indicative*

events, that is, a mutually exclusive and exhaustive partition of the set of all possible histories. We will assume $\mathcal{H}$ is *sufficient* (defined below). Both $|\mathcal{T}|$ and $|\mathcal{H}|$ may be arbitrarily larger than $|\mathcal{Q}|$; this property makes it easier to pick $\mathcal{T}$ and $\mathcal{H}$ satisfying our conditions, since we are free to choose sets that we believe to be large enough and varied enough to exhibit the types of behavior that we wish to model. Figure 3.1 illustrates an example of tests and histories in a simple HMM.

For purposes of gathering data, we assume that we can sample from some sufficiently diverse distribution ω over histories; our observable representation depends on ω as well. We will define "sufficiently diverse" below, but for example, ω might be the steady-state distribution of some exploration policy. Note that this assumption means that we *cannot* estimate x_1, since we don't have multiple samples of trajectories starting from x_1. So, instead, we will estimate x_*, an arbitrary feasible state, which is enough information to enable prediction (assuming the process mixes at a reasonable rate). If we make the stronger assumption that we can repeatedly reset our PSR to its starting distribution, a straightforward modification of our algorithm will allow us to estimate x_1 as well.

We define several observable matrices in terms of $\mathcal{T}$, $\mathcal{H}$, and ω. After each definition we show how these matrices relate to the parameters of the underlying PSR. These relationships will be key tools in designing our learning algorithm and showing its consistency. The first observable matrix is $P_{\mathcal{H}} \in \mathbb{R}^{|\mathcal{H}|}$, containing the probabilities of every event $H_i \in \mathcal{H}$ when we sample a history h according to ω:

$$\begin{aligned} [P_{\mathcal{H}}]_i &\stackrel{\text{def}}{=} \mathbb{P}\left[H_i\right] \\ &\stackrel{\text{def}}{=} \mathbb{P}_\omega\left[h \in H_i\right] \\ \implies P_{\mathcal{H}} &= \mathbb{P}\left[H\right] \end{aligned} \tag{3.7a}$$

Here we have defined $\mathbb{P}[H]$ to mean to mean the vector whose elements are $\mathbb{P}[H_i]$ for $H_i \in \mathcal{H}$. Next we define $P_{X,\mathcal{H}}$, the joint probability matrix of states and indicative events:

$$\begin{aligned} [P_{X,\mathcal{H}}]_{i,j} &\stackrel{\text{def}}{=} \mathbb{E}_\omega\left[x(h_t)\mathbb{P}\left[H_{j,t} \mid h_t\right]\right] \\ \implies P_{X,\mathcal{H}} &= \mathbb{E}_\omega\left[x(h_t)\mathbb{P}\left[H_t^\top \mid h_t\right]\right] \end{aligned} \tag{3.7b}$$

We cannot directly estimate $P_{X,\mathcal{H}}$ from data, but this matrix plays a fundamental role in our derivations and will appear as a factor in several of the matrices that we define below. One of the key functions of $P_{X,\mathcal{H}}$ is that it can be simply related to x_* and $P_{\mathcal{H}}$ when $h \sim \omega$. The following relations will be important for some of our derivations below:

$$\begin{aligned} P_{\mathcal{H}}^\top &= \mathbb{E}\left[H^\top \mid h_t\right] \\ &= \mathbb{E}\left[x_\infty^\top x(h_t) H^\top \mid h_t\right] \\ &= x_\infty^\top \mathbb{E}\left[x(h_t) H^\top \mid h_t\right] \\ &= x_\infty^\top P_{X,\mathcal{H}} \end{aligned}$$

and

$$x_* = P_{X,\mathcal{H}}\mathbf{1}$$

where $\mathbf{1} \in \mathbb{R}^{|\mathcal{Q}|}$ is the vector of 1s that marginalizes out indicative events. This definition works because it sets x_* to be the mean prediction given any history.

The next observable matrix is $P_{\mathcal{T},\mathcal{H}} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{H}|}$, a covariance matrix whose entries are joint probabilities of tests $\tau_i \in \mathcal{T}$ and indicative events $H_j \in \mathcal{H}$ when we sample $h_t \sim \omega$ and take actions $\tau_{i,t}^{\mathcal{A}}$:

$$\begin{aligned}
[P_{\mathcal{T},\mathcal{H}}]_{i,j} &\stackrel{\text{def}}{=} \mathbb{E}\left[\tau_{i,t}^{\mathcal{O}} H_{j,t} \,\middle|\, \text{do}\left(\tau_{i,t}^{\mathcal{A}}\right)\right] \\
&= \mathbb{E}\left[\mathbb{P}\left[\tau_{i,t}^{\mathcal{O}}, H_{j,t} \mid h_t, \text{do}\left(\tau_{i,t}^{\mathcal{A}}\right)\right]\right] \\
&= \mathbb{E}\left[\mathbb{P}\left[\tau_{i,t}^{\mathcal{O}} \mid h_t, \text{do}\left(\tau_{i,t}^{\mathcal{A}}\right)\right] \mathbb{P}\left[H_{j,t}^{\top} \mid h_t\right]\right] \\
&= \mathbb{E}\left[\tau_i(h_t)\mathbb{P}\left[H_{j,t} \mid h_t\right]\right] \\
&= \mathbb{E}\left[g_{\tau_i}^{\top} x(h_t)\mathbb{P}\left[H_{j,t} \mid h_t\right]\right] \\
&= g_{\tau_i}^{\top}\mathbb{E}\left[x(h_t)\mathbb{P}\left[H_{j,t} \mid h_t\right]\right] \\
&= g_{\tau_i}^{\top} P_{X,\mathcal{H}_j} \\
\implies P_{\mathcal{T},\mathcal{H}} &= \Gamma P_{X,\mathcal{H}}
\end{aligned} \tag{3.7c}$$

Here the vector g_{τ_i} lets us compute the probability of test τ_i given the state, and $\Gamma \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{Q}|}$ is the matrix with rows $g_{\tau_i}^{\top}$. Equation (3.7c) tells us that the rank of $P_{\mathcal{T},\mathcal{H}}$ is no more than $|\mathcal{Q}|$, since its factors Γ and $P_{X,H}$ each have rank at most $|\mathcal{Q}|$. At this point we can define a *sufficient* set of indicative events and a *sufficiently diverse* sampling distribution as promised: they are a set of indicative events and a sampling distribution for which the rank of $P_{\mathcal{T},\mathcal{H}}$ is *equal* to $|\mathcal{Q}|$.

The final observable matrices are $P_{\mathcal{T}^+,ao,\mathcal{H}} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{H}|}$, one matrix for each action-observation pair. Entries of $P_{\mathcal{T}^+,ao,\mathcal{H}}$ are probabilities of *triples* of an indicative event, the next action-observation pair, and a subsequent test, if we execute a:

$$\begin{aligned}
[P_{\mathcal{T}^+,ao,\mathcal{H}}]_{i,j} &\stackrel{\text{def}}{=} \mathbb{E}\left[\tau_{i,t+1}^{\mathcal{O}}\mathbb{I}(o_t = o) H_{j,t} \,\middle|\, \text{do}\left(a_t, \tau_{i,t+1}^{\mathcal{A}}\right)\right] \\
&= \mathbb{E}\left[\mathbb{P}\left[\tau_{i,t+1}^{\mathcal{O}}, o_t, H_{j,t} \mid h_t, \text{do}\left(a_t, \tau_{i,t+1}^{\mathcal{A}}\right)\right]\right] \\
&= \mathbb{E}\left[\mathbb{P}\left[\tau_{i,t+1}^{\mathcal{O}}, o_t \mid h_t, \text{do}\left(a_t, \tau_{i,t+1}^{\mathcal{A}}\right)\right] \mathbb{P}\left[H_{j,t} \mid h_t\right]\right] \\
&= \mathbb{E}\left[\mathbb{P}\left[\tau_{i,t+1}^{\mathcal{O}} \mid h_t, o_t, \text{do}\left(a_t, \tau_{i,t+1}^{\mathcal{A}}\right)\right] \mathbb{P}\left[o_t \mid h_t, a_t\right] \mathbb{P}\left[H_{j,t} \mid h_t\right]\right] \\
&= \mathbb{E}\left[\tau_i(h_{t+1})\mathbb{P}[o_t \mid h_t, a_t]\mathbb{P}\left[H_{j,t} \mid h_t\right]\right] \\
&= \mathbb{E}\left[g_{\tau_i}^{\top} x(h_{t+1})\mathbb{P}[o_t \mid h_t, a_t]\mathbb{P}\left[H_{j,t} \mid h_t\right]\right] \\
&= g_{\tau_i}^{\top}\mathbb{E}\left[\frac{M_{ao}x(h_t)}{x_{\infty}^{\top} M_{ao} x(h_t)}\mathbb{P}[o_t \mid h_t, a_t]\mathbb{P}\left[H_{j,t} \mid h_t\right]\right] \\
&= g_{\tau_i}^{\top}\mathbb{E}\left[M_{ao}x(h_t)\mathbb{P}\left[H_{j,t} \mid h_t\right]\right] \\
&= g_{\tau_i}^{\top} M_{ao}\mathbb{E}\left[x(h_t)\mathbb{P}\left[H_{j,t} \mid h_t\right]\right] \\
&= g_{\tau_i}^{\top} M_{ao} P_{X,\mathcal{H}_j} \\
\implies P_{\mathcal{T}^+,ao,\mathcal{H}} &= \Gamma M_{ao} P_{X,\mathcal{H}}
\end{aligned} \tag{3.7d}$$

Here we use Equation 3.6 and the fact that $\mathbb{P}\left[o_t \mid h_t, a_t\right] = x_{\infty}^{\top} M_{ao} x(h_t)$. Just like $P_{\mathcal{T},\mathcal{H}}$, the matrices $P_{\mathcal{T}^+,ao,\mathcal{H}}$ have rank at most $|\mathcal{Q}|$ due to their factors Γ and $P_{X,\mathcal{H}}$.

Recall from Section 3.2.2 that PSRs are only defined up to a similarity transform. This means that we can learn *any* similarity transform of the PSR parameters. To exploit this fact we assume that we are given an additional matrix $U \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{Q}|}$ such that $U^\top\Gamma$ is invertible. (Or, equivalently given our assumptions above, such that $U^\top P_{\mathcal{T},\mathcal{H}}$ has full row rank.) A natural choice for U is the leading left singular vectors of $P_{\mathcal{T},\mathcal{H}}$, although a randomly-generated U will work with probability 1 (but will typically result in slower learning). We can think of the columns of U as specifying a core set $\mathcal{Q}'$ of *linear combinations* of core tests $\mathcal{Q}$ which define a state vector for our PSR.

We now define a PSR in terms of the observable matrices $P_{\mathcal{H}}$, $P_{\mathcal{T},\mathcal{H}}$, $P_{\mathcal{T}^+,ao,\mathcal{H}}$ and U, and simplify the definitions using Equations 3.7(a–c) to show that our parameters are only a similarity transform away from the original PSR parameters:

$$\begin{aligned} b_* &\stackrel{\text{def}}{=} U^\top P_{\mathcal{T},\mathcal{H}}\mathbf{1} \\ &= U^\top\Gamma P_{X,\mathcal{H}}\mathbf{1} \\ &= (U^\top\Gamma)x_* \end{aligned} \tag{3.8a}$$

$$\begin{aligned} b_\infty^\top &\stackrel{\text{def}}{=} P_{\mathcal{H}}^\top(U^\top P_{\mathcal{T},\mathcal{H}})^\dagger \\ &= x_\infty^\top P_{X,\mathcal{H}}(U^\top P_{\mathcal{T},\mathcal{H}})^\dagger \\ &= x_\infty^\top(U^\top\Gamma)^{-1}(U^\top\Gamma)P_{X,\mathcal{H}}(U^\top P_{\mathcal{T},\mathcal{H}})^\dagger \\ &= x_\infty^\top(U^\top\Gamma)^{-1} \end{aligned} \tag{3.8b}$$

$$\begin{aligned} B_{ao} &\stackrel{\text{def}}{=} U^\top P_{\mathcal{T}^+,ao,\mathcal{H}}(U^\top P_{\mathcal{T},\mathcal{H}})^\dagger \\ &= U^\top\Gamma M_{ao}P_{X,\mathcal{H}}(U^\top P_{\mathcal{T},\mathcal{H}})^\dagger \\ &= U^\top\Gamma M_{ao}(U^\top\Gamma)^{-1}(U^\top\Gamma)P_{X,\mathcal{H}}(U^\top P_{\mathcal{T},\mathcal{H}})^\dagger \\ &= (U^\top\Gamma)M_{ao}(U^\top\Gamma)^{-1} \end{aligned} \tag{3.8c}$$

To get $b_1 = (U^\top\Gamma)x_1$ instead of b_* in Equation 3.8a, replace $P_{\mathcal{T},\mathcal{H}}\mathbf{1}$, the vector of expected probabilities of tests for $h_t \sim \omega$, with the vector of probabilities of tests for $h_t = h_1$.

Given the parameters in Equations 3.8a–c, we can calculate the probability of observations $o_{1:t}$, given that we start from state x_1 and intervene with actions $a_{1:t}$. Here we write the product of a sequence of transition matrices as $M_{ao_{1:t}} = M_{a_1o_1}M_{a_2o_2}\dots M_{a_to_t}$.

$$\begin{aligned} \mathbb{P}[o_{1:t} \mid \text{do}(a_{1:t})] &= x_\infty^\top M_{ao_{1:t}}x_1 \\ &= x_\infty^\top(U^\top\Gamma)^{-1}(U^\top\Gamma)M_{ao_{1:t}}(U^\top\Gamma)^{-1}(U^\top\Gamma)x_1 \\ &= b_\infty^\top B_{ao_{1:t}}b_1 \end{aligned} \tag{3.9}$$

In addition to the initial PSR state b_1, we define normalized conditional *internal states* b_t. We define the PSR state at time $t+1$ as:

$$b_{t+1} \stackrel{\text{def}}{=} \frac{B_{ao_{1:t}}b_1}{b_\infty^\top B_{ao_{1:t}}b_1} \tag{3.10}$$

We can define a *recursive* state update from time $t \geq 1$ to $t+1$ as follows (using the above

definition of b_1 as the base case):

$$
\begin{aligned}
b_{t+1} &\stackrel{\text{def}}{=} \frac{B_{ao_{1:t}} b_1}{b_\infty^\top B_{ao_{1:t}} b_1} \\
&= \frac{B_{a_t o_t} B_{ao_{1:t-1}} b_1}{b_\infty^\top B_{a_t o_t} B_{ao_{1:t-1}} b_1} \\
&= \frac{B_{a_t o_t} b_t}{b_\infty^\top B_{a_t o_t} b_t}
\end{aligned} \tag{3.11}
$$

If we only have b_* instead of b_1, then initial probability estimates will be approximate, but the difference in predictions will disappear over time as our process mixes. The linear transform from b_t to a PSR internal state $x(h_t)$ is given by $x(h_t) = (U^\top \Gamma)^{-1} b_t$. If we chose U by SVD, then the prediction of tests $\tau(h_t)$ at time t is given by $U b_t = U U^\top \Gamma x(h_t) = \Gamma x(h_t)$ (since by the definition of SVD, U is orthonormal and the row spaces of U and Γ are the same).

3.4 Learning Predictive State Representations

Our learning algorithm works by building empirical estimates $\widehat{P}_{\mathcal{H}}$, $\widehat{P}_{\mathcal{T},\mathcal{H}}$, and $\widehat{P}_{\mathcal{T}^+,ao,\mathcal{H}}$ of the matrices $P_{\mathcal{H}}$, $P_{\mathcal{T},\mathcal{H}}$, and $P_{\mathcal{T}^+,ao,\mathcal{H}}$ defined above. To build these estimates, we repeatedly sample a history h_t from the distribution ω, execute a sequence of actions from one of our tests, and record the resulting observations. This data gathering strategy implies that we must be able to arrange for the system to be in a state corresponding to $h_t \sim \omega$; for example, if our system has a reset, we can take ω to be the distribution resulting from executing a fixed exploration policy for a few steps after reset.

In practice, reset is often not available. In this case we can estimate $\widehat{P}_{\mathcal{H}}$, $\widehat{P}_{\mathcal{T},\mathcal{H}}$, and $\widehat{P}_{\mathcal{T}^+,ao,\mathcal{H}}$ by dividing a single long sequence of action-observation pairs into subsequences and pretending that each subsequence started with a reset. We are forced to use an initial distribution over histories, ω, equal to the steady state distribution of the policy which generated the data. This approach is called the *suffix-history* algorithm [123]. With this method, the estimated matrices will be only approximately correct, since interventions that we take at one time will affect the distribution over histories at future times; however, the approximation is often a good one in practice, especially if we allow the process to mix by executing several steps of the exploration policy in between interventions.

If we know or can estimate the exploration policy, we can avoid making any interventions, and instead use importance weighting [16] to produce samples whose expectations are the true test outcome probabilities. We summarize this trick here for the special case of an open-loop randomized exploration policy. Write $\tau_1^{\mathcal{A}}, \tau_2^{\mathcal{A}}, \ldots$ for the action sequences mentioned in our tests. For simplicity, assume these sequences are distinct (if not, we can merge duplicates). Let $\zeta_1, \zeta_2, \ldots > 0$ be the probabilities of $\tau_1^{\mathcal{A}}, \tau_2^{\mathcal{A}}, \ldots$ under the exploration policy. Starting at some time, suppose the exploration policy actually executes the action sequence $\tau_i^{\mathcal{A}}$. Write δ for the random variable which is 1 if the corresponding observations match $\tau_i^{\mathcal{O}}$, and 0 if not. Now construct a sample vector which is zero everywhere except that the ith component is δ/ζ_i. It is easy to see that the expected value of our sample vector is correct, i.e., that the expectation of

the ith component is the true success probability of τ_i: the probability of selection ζ_i and the weighting factor $1/\zeta_i$ cancel out.

Once we have computed $\widehat{P}_{\mathcal{H}}$, $\widehat{P}_{\mathcal{T},\mathcal{H}}$, and $\widehat{P}_{\mathcal{T}^+,ao,\mathcal{H}}$, we can generate $\widehat{U}$ by singular value decomposition of $\widehat{P}_{\mathcal{T},\mathcal{H}}$. We can then learn the PSR parameters by plugging $\widehat{U}$, $\widehat{P}_{\mathcal{H}}$, $\widehat{P}_{\mathcal{T},\mathcal{H}}$, and $\widehat{P}_{\mathcal{T}^+,ao,\mathcal{H}}$ into Equation 3.8(a–c). For reference, we summarize the above steps here:

1. Compute empirical estimates $\widehat{P}_{\mathcal{H}}$, $\widehat{P}_{\mathcal{T},\mathcal{H}}$,$\widehat{P}_{\mathcal{T}^+,ao,\mathcal{H}}$.
2. Use SVD on $\widehat{P}_{\mathcal{T},\mathcal{H}}$ to compute $\widehat{U}$, the matrix of left singular vectors corresponding to the n largest singular values.
3. Compute model parameter estimates:
   (a) $\widehat{b}_* = \widehat{U}^\top \widehat{P}_{\mathcal{T},\mathcal{H}} \mathbf{1}$,
   (b) $\widehat{b}_\infty = (\widehat{P}_{\mathcal{T},\mathcal{H}}^\top \widehat{U})^\dagger \widehat{P}_{\mathcal{H}}$,
   (c) $\widehat{B}_{ao} = \widehat{U}^\top \widehat{P}_{\mathcal{T}^+,ao,\mathcal{H}} (\widehat{U}^\top \widehat{P}_{\mathcal{T},\mathcal{H}})^\dagger$

As we include more data in our averages, the law of large numbers guarantees that our estimates $\widehat{P}_{\mathcal{H}}$, $\widehat{P}_{\mathcal{T},\mathcal{H}}$, and $\widehat{P}_{\mathcal{T}^+,ao,\mathcal{H}}$ converge to the true matrices $P_{\mathcal{H}}$, $P_{\mathcal{T},\mathcal{H}}$, and $P_{\mathcal{T}^+,ao,\mathcal{H}}$ (defined in Equation 3.7). So by continuity of the formulas above, if our system is truly a PSR of finite rank, our estimates $\widehat{b}_*$, $\widehat{b}_\infty$, and $\widehat{B}_{ao}$ converge to the true parameters up to a linear transform—that is, our learning algorithm is *consistent*.[1] Although parameters estimated with finite data can sometimes lead to negative probability estimates when filtering or predicting, this problem can be avoided in practice by thresholding the predicted probabilities by some small positive probability.

The learning strategy employed here may be seen as a generalization of Hsu et al.'s spectral algorithm for learning HMMs [42] to PSRs. Since HMMs and POMDPs are a proper subset of PSRs, we can use the algorithm in this chapter to learn back both HMMs and POMDPs in PSR form. However, the problem of learning HMMs and POMDPs in general is hard under cryptographic assumptions [52, 109]. Therefore, some models will require a large amount of data (and thus a large amount of computation) to learn exactly. The learning algorithm presented here embodies a tradeoff: it relinquishes the ability to learn very difficult HMMs with little data and a lot of computation, but is a very effective learning algorithm for easier HMMs.

Since uncontrolled PSRs are equivalent to OOMs, the learning algorithm presented here can also be used to efficiently learn OOMs. In fact, the naïve OOM learning algorithm [44] is similar to our PSR learning algorithm, but uses a fixed subspace rather than employing SVD to choose a subspace. The more sophisticated efficiency sharpening algorithm [38] is an iterative way to choose a subspace for OOMs that results in more statistically efficient estimates than the naïve algorithm.

Finally, note that the learning algorithm presented here is distinct from the TPSR learning algorithm of [84]. In addition to the differences mentioned above, a key difference between the

[1]The pseudoinverses are continuous at the true parameters, since the matrices to be pseudoinverted have full rank. The matrix of n left singular vectors $\widehat{U}$ may not be a continuous function of $\widehat{P}_{\mathcal{T},\mathcal{H}}$ (in case of repeated singular values); to deal with this possibility, we can either fix $\widehat{U}$ (say, as the left singular vectors of our estimated $\widehat{P}_{\mathcal{T},\mathcal{H}}$ after some fixed amount of data), or we can make a slightly more complex argument based on the fact that the *column span* of $\widehat{U}$ is a continuous function of $\widehat{P}_{\mathcal{T},\mathcal{H}}$ near $P_{\mathcal{T},\mathcal{H}}$ (since the nth singular value of $P_{\mathcal{T},\mathcal{H}}$ is nonzero, and is therefore separated from the $(n+1)^{\text{st}}$, which is zero).

two algorithms is that here we estimate the *joint* probability of a past event, a current observation, and a future event in the matrix $\widehat{P}_{\mathcal{T}^+,ao,\mathcal{H}}$, whereas [84] instead estimate the probability of a future event, *conditioned* on a past event and a current observation. To compensate, Rosencrantz et al. later multiply this estimate by an approximation of the probability of the current observation, conditioned on the past event. Rosencrantz et al. also derive the approximate probability of the current observation differently: as the result of a regression instead of directly from empirical counts. Finally, Rosencrantz et al. do not make any attempt to multiply by the marginal probability of the past event, although this term cancels in the current work. In the absence of estimation errors, both algorithms would arrive at the same answer, but taking errors into account, they will typically make different predictions. The difficulty of extending the Rosencrantz et al. algorithm to handle real-valued features stems from the difference between joint and conditional probabilities: the observable matrices in Rosencrantz et al. are conditional expectations, so their algorithm depends on being able to condition on discrete indicative events or observations.

Chapter 4

Learning Predictive State Representations with Features

In this chapter we generalize our observable representation of PSRs in Chapter 3, which were built from joint probabilities of discrete observations, to *expectations* of continuous features. In data gathered from complex real-world dynamical systems, it may not be possible to find a reasonably-sized core set of discrete tests $\mathcal{T}$ or sufficient set of indicative events $\mathcal{H}$. When this is the case, we can generalize the PSR learning algorithm and work with *features* of test outcomes and histories, which we call *characteristic features* and *indicative features* respectively. Similarly, if we have a large number of discrete observations and actions, the number of parameters in the PSR can become large enough to make learning practically impossible. When this is the case, we can generalize the PSR to recursively update state based on observation features through a matrix representation of Bayes' rule.

We will discuss features in two steps. First we will define PSRs in terms of *moments* of observable features of tests and histories. This extension has important practical consequences: we often have domain knowledge that allows us to choose features that "make sense" for the system we are trying to model, and, choosing a small set of features usually means that the learning algorithm converges at a faster rate in practice. Second, we generalize the PSR Bayes' rule update to features of observations. In this chapter we will focus on learning PSRs that contain a large but finite number of discrete actions, observations, and features. We will generalize to continuous actions and observations, and infinite feature spaces, in Chapter 5.

4.1 Characteristic and Indicative Features

4.1.1 Characteristic Features

We can generalize the notion of a test from Section 3.2.1 to a *characteristic feature*, a feature of the future that is a linear combination of several tests sharing a common action sequence. These features are called *characteristic* because the expectation of the features fully characterizes the distribution of the future. For example of a characteristic feature: if τ_1 and τ_2 are two tests with $\tau_1^{\mathcal{A}} = \tau_2^{\mathcal{A}} \stackrel{\text{def}}{=} \tau^{\mathcal{A}}$, then we can make a feature $\phi = 3\tau_1 + \tau_2$. This feature is *executed* if we

intervene to $\text{do}(\tau^{\mathcal{A}})$, and, if it is executed, its *value* is $3\mathbb{I}(\tau_1^{\mathcal{O}}) + \mathbb{I}(\tau_2^{\mathcal{O}})$, where $\mathbb{I}(o_1 \ldots o_i)$ stands for an indicator random variable, taking the value 0 or 1 depending on whether we observe the sequence of observations $o_1 \ldots o_i$. The *prediction* of ϕ given h_t is $\phi(h_t) \stackrel{\text{def}}{=} \mathbb{E}\left[\phi \mid h_t, \text{do}\left(\tau^{\mathcal{A}}\right)\right] = 3\tau_1(h_t) + \tau_2(h_t)$.

While linear combinations of tests may seem restrictive, our definition is actually very expressive: we can represent an arbitrary function of a finite sequence of future observations. We could also allow convergent limits of features (i.e., take the closure of the set of features), meaning that we could represent many functions of the entire infinite sequence of future observations. Another view of this is that we could approximate a feature of the infinite sequence arbitrarily closely. To build a feature, we take a collection of tests, each of which picks out one possible realization of the sequence, and weight each test by the value of the function conditioned on that realization. For example, if our observations are integers $1, 2, \ldots, 10$, we can write the square of the next observation as $\sum_{o=1}^{10} o^2 \mathbb{I}(o)$, and the mean of the next two observations as $\sum_{o=1}^{10} \sum_{o'=1}^{10} \frac{1}{2}(o + o')\mathbb{I}(o, o')$.

The restriction to a common action sequence is necessary: without this restriction, all the tests making up a feature could never be executed at once. Once we move to feature *predictions*, however, it makes sense to lift this restriction: we will say that any linear combination of feature predictions is also a feature prediction, even if the features involved have different action sequences.

Action sequences raise some problems with obtaining empirical estimates of means and covariances of features of the future: e.g., it is not always possible to get a sample of a particular feature's value on every time step, and the feature we choose to sample at one step can restrict which features we can sample at subsequent steps. In order to carry out our derivations without running into these problems repeatedly, we will assume for the rest of the chapter that we can reset our system after every sample, and get a new history independently distributed as $h_t \sim \omega$ for some distribution ω. (With some additional bookkeeping we could remove this assumption [16], but this bookkeeping would unnecessarily complicate our derivations.)

Furthermore, we will introduce some new language, again to keep derivations simple: if we have a vector of features of the future $\phi^{\mathcal{T}}$, we will pretend that we can get a sample $\phi_t^{\mathcal{T}}$ in which we evaluate all of our features starting from a single history h_t, even if the different elements of $\phi^{\mathcal{T}}$ require us to execute different action sequences. When our algorithms call for such a sample, we will instead use the following trick to get a random vector with the correct expectation (and somewhat higher variance, which doesn't matter for any of our arguments): write $\tau_1^{\mathcal{A}}, \tau_2^{\mathcal{A}}, \ldots$ for the different action sequences, and let $\zeta_1, \zeta_2, \ldots > 0$ be a probability distribution over these sequences. We pick a single action sequence $\tau_a^{\mathcal{A}}$ according to ζ, and execute $\tau_a^{\mathcal{A}}$ to get a sample $\hat{\phi}^{\mathcal{T}}$ of the features which depend on $\tau_a^{\mathcal{A}}$. We then enter $\hat{\phi}^{\mathcal{T}}/\zeta_a$ into the corresponding coordinates of $\phi_t^{\mathcal{T}}$, and fill in zeros everywhere else. It is easy to see that the expected value of our sample vector is then correct: the probability of selection ζ_a and the weighting factor $1/\zeta_a$ cancel out. We will write $\mathbb{E}\left[\phi^{\mathcal{T}} \mid h_t, \text{do}(\zeta)\right]$ to stand for this expectation.

4.1.2 Indicative Features

Just as we generalized the notion of test to characteristic features, we can generalize the notion of a history to *indicative features*. Such features are called indicative because they *indicate* what has already happened. An indicative feature is a function of the history h_t. We have already seen a specific type of indicative feature in Section 3.3, an indicative event, which is one of a set of mutually exclusive and exhaustive partitions of history. Other indicative features might reference the number of times we saw o_1 in the past three steps; or, in a domain with actions, indicative features can reference past actions: e.g., the number of times we did action a_1 in the past three steps.

4.2 Defining PSRs with Characteristic and Indicative Features

Let $\mathcal{Q}$ be a minimal core set of tests for a dynamical system, with cardinality $d = |\mathcal{Q}|$ equal to the linear dimension of the system. Then, let $\mathcal{T}$ be a larger core set of tests (not necessarily minimal, and possibly even with $|\mathcal{T}|$ countably infinite). And, let $\mathcal{H}$ be the set of all possible histories. ($|\mathcal{H}|$ is finite or countably infinite, depending on whether our system is finite-horizon or infinite-horizon.)

We write $\phi_t^{\mathcal{H}} \in \mathbb{R}^{\ell_{\mathcal{H}}}$ for a vector of indicative features of history at time t, and write $\phi_t^{\mathcal{T}} \in \mathbb{R}^{\ell_{\mathcal{T}}}$ for a vector of characteristic features of the future at time t. Since $\mathcal{T}$ is a core set of tests, by definition we can compute any test prediction $\tau_i(h_t)$ as a linear function of the tests in $\mathcal{T}$. And, since feature predictions are linear combinations of test predictions, we can also compute any feature prediction $\mathbb{E}\left[\phi_t^{\mathcal{T}} \mid \text{do}(\zeta), h_t\right]$ as a linear function of the tests in $\mathcal{T}$. We define the matrix $\Phi^{\mathcal{T}} \in \mathbb{R}^{\ell \times |\mathcal{T}|}$ to embody our predictions of future features: that is, an entry of $\Phi^{\mathcal{T}}$ is the weight of one of the tests in $\mathcal{T}$ for calculating the prediction of one of the features in $\phi^{\mathcal{T}}$.

Below we define several matrices, $\mu_{\mathcal{H}}$, $\Sigma_{\mathcal{T},\mathcal{H}}$, and $\Sigma_{\mathcal{T}^+,ao,\mathcal{H}}$, in terms of characteristic and inidicative features $\phi_t^{\mathcal{T}}$ and $\phi_t^{\mathcal{H}}$ and discrete actions and observations a_t, and o_t, and show how these matrices relate to the parameters of the underlying PSR.

These matrices are the analog of the matrices $P_{\mathcal{H}}$, $P_{\mathcal{T},\mathcal{H}}$, and $P_{\mathcal{T}^+,ao,\mathcal{H}}$ in Equations 3.7, but will no longer contain probabilities, but rather *expected values* of features or products of features. For the special case of features that are *indicator functions* of test outcomes and sets of histories, we recover the probability matrices from Section 3.3. And, just as in Section 3.3, we can estimate expected values from repeated trials, or from one long sequence; and, we can use importance sampling to avoid having to make any interventions.

We need to require a version of *sufficiency* for our sets of features, as we did for tests and indicative events above. The updated definition of sufficiency is analogous to our earlier definitions of core tests and sufficient indicative events: we require that the rank of $\Sigma_{\mathcal{T},\mathcal{H}}$ (defined below in Equation 4.1c) is *equal* to the linear dimension of the system. The advantage of working with features instead of events is that, in practice, it seems to be easier to ensure sufficiency with a moderate number of features.

We begin by defining the analog of $P_{\mathcal{H}}$ from Equation 3.7a:

$$
\begin{aligned}
[\mu_{\mathcal{H}}]_i &\stackrel{\text{def}}{=} \mathbb{E}\left[\phi_{i,t}^{\mathcal{H}}\right] \\
\implies \mu_{\mathcal{H}} &= \mathbb{E}\left[\phi_t^{\mathcal{H}}\right] \qquad (4.1a)
\end{aligned}
$$

Next we define $\Sigma_{X,\mathcal{H}}$, the cross covariance of states and features of histories. Let

$$
\begin{aligned}
\Sigma_{X,\mathcal{H}} &\stackrel{\text{def}}{=} \mathbb{E}\left[x_t {\phi_t^{\mathcal{H}}}^\top\right] \\
&= \mathbb{E}\left[x(h_t)\mathbb{E}\left[{\phi_t^{\mathcal{H}}}^\top \mid h_t\right]\right] \qquad (4.1b)
\end{aligned}
$$

Like $P_{X,\mathcal{H}}$ in Equation 3.7b, we cannot directly estimate $\Sigma_{X,\mathcal{H}}$ from data, but this matrix will appear as a factor in several of the matrices that we define below. We can see that $\Sigma_{X,\mathcal{H}}$ is related to $\mu_{\mathcal{H}}$:

$$
\begin{aligned}
x_\infty^\top \Sigma_{X,\mathcal{H}} &= x_\infty^\top \mathbb{E}\left[x(h_t)\mathbb{E}\left[{\phi_t^{\mathcal{H}}}^\top \mid h_t\right]\right] \\
&= \mathbb{E}\left[x_\infty^\top x(h_t)\mathbb{E}\left[{\phi_t^{\mathcal{H}}}^\top \mid h_t\right]\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[{\phi_t^{\mathcal{H}}}^\top \mid h_t\right]\right] \\
&= \mathbb{E}\left[{\phi_t^{\mathcal{H}}}^\top\right] \\
&= {\mu_{\mathcal{H}}}^\top
\end{aligned}
$$

Next we define $\Sigma_{\mathcal{T},\mathcal{H}}$, the cross covariance matrix of the features of tests and histories:

$$
\begin{aligned}
[\Sigma_{\mathcal{T},\mathcal{H}}]_{i,j} &\stackrel{\text{def}}{=} \mathbb{E}\left[\phi_{i,t}^{\mathcal{T}}\phi_{j,t}^{\mathcal{H}} \,\middle|\, \text{do}(\zeta)\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\phi_{i,t}^{\mathcal{T}}\phi_{j,t}^{\mathcal{H}} \mid \text{do}\left(\zeta\right), h_t\right]\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\phi_{i,t}^{\mathcal{T}} \mid \text{do}\left(\zeta\right), h_t\right]\mathbb{E}\left[\phi_{j,t}^{\mathcal{H}} \mid h_t\right]\right] \\
&= \mathbb{E}\left[\sum_{l=1}^{|\mathcal{T}|}\Phi_{i,l}^{\mathcal{T}}\tau_l(h_t)\mathbb{E}\left[\phi_{j,t}^{\mathcal{H}} \mid h_t\right]\right] \\
&= \sum_{l=1}^{|\mathcal{T}|}\Phi_{i,l}^{\mathcal{T}}\mathbb{E}\left[\tau_l(h_t)\mathbb{E}\left[\phi_{j,t}^{\mathcal{H}} \mid h_t\right]\right] \\
&= \sum_{l=1}^{|\mathcal{T}|}\Phi_{i,l}^{\mathcal{T}}\mathbb{E}\left[g_{\tau_l}^\top x(h_t)\mathbb{E}\left[\phi_{j,t}^{\mathcal{H}} \mid h_t\right]\right] \\
&= \sum_{l=1}^{|\mathcal{T}|}\Phi_{i,l}^{\mathcal{T}}g_{\tau_l}^\top\mathbb{E}\left[x(h_t)\mathbb{E}\left[\phi_{j,t}^{\mathcal{H}} \mid h_t\right]\right] \\
&= \Phi_i^{\mathcal{T}}\Gamma\mathbb{E}\left[x(h_t)\mathbb{E}\left[\phi_{j,t}^{\mathcal{H}} \mid h_t\right]\right] \\
&= \Phi_i^{\mathcal{T}}\Gamma\Sigma_{X,\mathcal{H}_j} \\
\implies \Sigma_{\mathcal{T},\mathcal{H}} &= \Phi^{\mathcal{T}}\Gamma\Sigma_{X,\mathcal{H}} \qquad (4.1c)
\end{aligned}
$$

As in Chapter 3, the vector g_{τ_l} is the linear function that specifies the probability of the test τ_l given the probabilities of tests in the core set $\mathcal{Q}$, and the matrix Γ has all of the g_{τ_l} vectors as rows. The state vector $x(h_t)$ contains the probabilities of all tests in $\mathcal{Q}$ given history h_t. The above derivation shows that, because of our assumptions about the linear dimension of the system, the matrix $\Sigma_{\mathcal{T},\mathcal{H}}$ has factors $\Phi^{\mathcal{T}}\Gamma \in \mathbb{R}^{|\mathcal{T}|\times n}$ and $\Sigma_{X,\mathcal{H}} \in \mathbb{R}^{n\times \ell}$. Therefore, the *rank* of $\Sigma_{\mathcal{T},\mathcal{H}}$ is no more than d, the linear dimension of the system.

Finally, we define $\Sigma_{\mathcal{T}^+,ao,\mathcal{H}}$, a *set* of matrices, one for each action-observation pair, that represent the covariance between features of history before and after taking action a and observing o. In the following, $\mathbb{I}(o_t = o)$ is an indicator variable for whether we see observation o at step t.

$$
\begin{aligned}
[\Sigma_{\mathcal{T}^+,ao,\mathcal{H}}]_{i,j} &\stackrel{\text{def}}{=} \mathbb{E}\left[\phi^{\mathcal{T}}_{i,t+1}\mathbb{I}(o_t = o)\phi^{\mathcal{H}}_{j,t} \,\middle|\, \text{do}(a_t, \zeta^+)\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\phi^{\mathcal{T}}_{i,t+1}\mathbb{I}(o_t = o)\phi^{\mathcal{H}}_{j,t} \mid \text{do}\left(a_t, \zeta^+\right), h_t\right]\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\phi^{\mathcal{T}}_{i,t+1}\mathbb{I}(o_t = o) \mid \text{do}\left(a_t, \zeta^+\right), h_t\right]\mathbb{E}\left[\phi^{\mathcal{H}}_{j,t} \mid h_t\right]\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\phi^{\mathcal{T}}_{i,t+1} \mid \text{do}\left(a_t, \zeta^+\right), h_t, o_t\right]\mathbb{P}\left[o_t \mid h_t, \text{do}\left(a_t\right)\right]\mathbb{E}\left[\phi^{\mathcal{H}}_{j,t} \mid h_t\right]\right] \\
&= \mathbb{E}\left[\sum_{l=1}^{|\mathcal{T}|}\Phi^{\mathcal{T}}_{i,l}\tau_l(h_{t+1})\mathbb{P}\left[o_t \mid h_t, \text{do}(a_t)\right]\mathbb{E}\left[\phi^{\mathcal{H}}_{j,t} \mid h_t\right]\right] \\
&= \sum_{l=1}^{|\mathcal{T}|}\Phi^{\mathcal{T}}_{i,l}\mathbb{E}\left[\tau_l(h_{t+1})\mathbb{P}\left[o_t \mid h_t, \text{do}(a_t)\right]\mathbb{E}\left[\phi^{\mathcal{H}}_{j,t} \mid h_t\right]\right] \\
&= \sum_{l=1}^{|\mathcal{T}|}\Phi^{\mathcal{T}}_{i,l}\mathbb{E}\left[g_{\tau_l}^{\top}x(h_{t+1})\mathbb{P}\left[o_t \mid h_t, \text{do}(a_t)\right]\mathbb{E}\left[\phi^{\mathcal{H}}_{j,t} \mid h_t\right]\right] \\
&= \sum_{l=1}^{|\mathcal{T}|}\Phi^{\mathcal{T}}_{i,l}g_{\tau_l}^{\top}\mathbb{E}\left[x(h_{t+1})\mathbb{P}\left[o_t \mid h_t, \text{do}(a_t)\right]\mathbb{E}\left[\phi^{\mathcal{H}}_{j,t} \mid h_t\right]\right] \\
&= \Phi^{\mathcal{T}}_{i}\Gamma\mathbb{E}\left[x(h_{t+1})\mathbb{P}\left[o_t \mid h_t, \text{do}(a_t)\right]\mathbb{E}\left[\phi^{\mathcal{H}}_{j,t} \mid h_t\right]\right] \\
&= \Phi^{\mathcal{T}}_{i}\Gamma\mathbb{E}\left[\frac{M_{ao}x(h_t)}{x_{\infty}^{\top}M_{ao}x(h_t)}\mathbb{P}\left[o_t \mid h_t, \text{do}(a_t)\right]\mathbb{E}\left[\phi^{\mathcal{H}}_{j,t} \mid h_t\right]\right] \\
&= \Phi^{\mathcal{T}}_{i}\Gamma\mathbb{E}\left[M_{ao}x(h_t)\mathbb{E}\left[\phi^{\mathcal{H}}_{j,t} \mid h_t\right]\right] \\
&= \Phi^{\mathcal{T}}_{i}\Gamma M_{ao}\mathbb{E}\left[x(h_t)\mathbb{E}\left[\phi^{\mathcal{H}}_{j,t} \mid h_t\right]\right] \\
&= \Phi^{\mathcal{T}}_{i}\Gamma M_{ao}\Sigma_{X,\mathcal{H}_j} \\
\implies \Sigma_{\mathcal{T}^+,ao,\mathcal{H}} &= \Phi^{\mathcal{T}}\Gamma M_{ao}\Sigma_{X,\mathcal{H}} \qquad (4.1\text{d})
\end{aligned}
$$

Our goal is now to define a PSR in terms of the above moments of characteristic and indicative features. Similar to our approach in Chapter 3, we will need a matrix U such that $U^{\top}\Phi^{\mathcal{T}}\Gamma$ is invertible (i.e. $U^{\top}\Sigma_{\mathcal{T},\mathcal{H}}$ has full row rank); we can take U to be the left singular values of $\Sigma_{\mathcal{T},\mathcal{H}}$. We also assume that we have a vector e s.t. $\phi_t^{\mathcal{H}\top}e = \mathbf{1}\,(\forall\,\mathbf{t})$. The existence of e means that the ones vector $\mathbf{1}^{\top}$ must be in the row space of $\phi^{\mathcal{H}}$. Since $\phi^{\mathcal{H}}$ is a matrix of features, we can always ensure that this is the case by requiring one of our features to be a constant.

The parameters of the PSR (b_*, b_∞, and B_{ao}) are now defined as follows in terms of the matrices $\mu_{\mathcal{H}}$, $\Sigma_{\mathcal{T},\mathcal{H}}$, $\Sigma_{\mathcal{T}^+,ao,\mathcal{H}}$, and U. After each definition we simplify the expressions using

Equations 4.1a–d to show that our parameters are only a similarity transform away from the original PSR parameters:

$$
\begin{aligned}
b_* &\stackrel{\text{def}}{=} U^\top \Sigma_{\mathcal{T},\mathcal{H}} e \\
&= U^\top \Phi^{\mathcal{T}} \Gamma \Sigma_{X,\mathcal{H}} e \\
&= (U^\top \Phi^{\mathcal{T}} \Gamma) x_* && \text{(4.2a)} \\
b_\infty^\top &\stackrel{\text{def}}{=} \mu_{\mathcal{H}}^\top (U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger \\
&= x_\infty^\top \Sigma_{X,\mathcal{H}} (U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger \\
&= x_\infty^\top (U^\top \Phi^{\mathcal{T}} \Gamma)^{-1} (U^\top \Phi^{\mathcal{T}} \Gamma) \Sigma_{X,\mathcal{H}} (U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger \\
&= x_\infty^\top (U^\top \Phi^{\mathcal{T}} \Gamma)^{-1} && \text{(4.2b)} \\
B_{ao} &\stackrel{\text{def}}{=} U^\top \Sigma_{\mathcal{T}^+,ao,\mathcal{H}} (U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger \\
&= U^\top \Phi^{\mathcal{T}} \Gamma M_{ao} \Sigma_{X,\mathcal{H}} (U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger \\
&= U^\top \Phi^{\mathcal{T}} \Gamma M_{ao} (U^\top \Phi^{\mathcal{T}} \Gamma)^{-1} (U^\top \Phi^{\mathcal{T}} \Gamma) \Sigma_{X,\mathcal{H}} (U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger \\
&= (U^\top \Phi^{\mathcal{T}} \Gamma) M_{ao} (U^\top \Phi^{\mathcal{T}} \Gamma)^{-1} && \text{(4.2c)}
\end{aligned}
$$

Just as in Section 3.4 where we estimate $\widehat{U}$, $\widehat{P}_{\mathcal{H}}$, $\widehat{P}_{\mathcal{T},\mathcal{H}}$, and $\widehat{P}_{\mathcal{T}^+,ao,\mathcal{H}}$, we can estimate $\widehat{U}$, $\widehat{\Sigma}_{\mathcal{H}}$, $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$, and $\widehat{\Sigma}_{\mathcal{T}^+,ao,\mathcal{H}}$ from data, and then plug the estimates into Equations 4.2(a–c). And, just as before, our estimation equations are continuous near the true values of U, $\Sigma_{\mathcal{H}}$, $\Sigma_{\mathcal{T},\mathcal{H}}$, and $\Sigma_{\mathcal{T}^+,ao,\mathcal{H}}$.[1] Thus we see that if we work with characteristic and indicative features, and if our system is truly a PSR of finite rank, our estimates $\widehat{b}_*$, $\widehat{b}_\infty$, and $\widehat{B}_{ao}$ again converge to the true PSR parameters up to a linear transform.

If we are trying to model a dynamical system with a small number of discrete actions and observations, then the above approach to learning a PSR can be very effective. Characteristic and indicative features are especially useful when we want to summarize a large set of tests and history (and, therefore, increase the likelihood that this set is *core*). However, if the number of possible observations and actions are very large, and we only receive a few or no training points for each action-observation pair, then the transition parameters M_{ao} can be very difficult to learn. When this is the case we use *observation features*.

4.3 Observation Features and Bayes' Rule

In this section, we extend the above definitions and algorithms to handle a large observation set $\mathcal{O}$ by extending our use of features to features of observations conditioned on actions. Suppose that $|\mathcal{O}|$ is finite, but large enough that we cannot hope to see each observation more than a small number of times. Then the representations of $\Sigma_{\mathcal{T},\mathcal{H}}$ and $\Sigma_{\mathcal{T}^+,ao,\mathcal{H}}$ in terms of PSR parameters (Equations 4.1(a–d)) are still valid, but we cannot gather enough data to estimate $\Sigma_{\mathcal{T}^+,ao,\mathcal{H}}$ accurately unless we make additional assumptions.

Therefore we generalize observations to *observation features*, a feature of the present that is a linear combination of observations conditioned on taking a common action. In this way,

[1] Again, with the same caveat about the SVD.

observation features are very similar to characteristic features: they must be designed so that the expectation of observation features fully characterizes the probability distribution of observations at time t for each action. When obtaining empirical estimates of expectations of observation features we must pay close attention to actions in a manner similar to estimating expectations of characteristic features. We incorporate actions into the observation features by parameterizing the features with actions. That is, we assume that we know m features of observations conditioned on actions, $\phi_{k,t}^{\mathcal{AO}}$ for $k = 1 \ldots m$ at each time t. We can decompose

$$\phi_{k,t}^{\mathcal{AO}} = \Phi_{k,:}^{\mathcal{AO}} e_{a_t o_t} \tag{4.3}$$

where each element of the matrix $\Phi^{\mathcal{AO}} \in \mathbb{R}^{k \times |\mathcal{A}| \cdot |\mathcal{O}|}$ contains the value of a feature given that we take the action and observe the observation that indexes that element. The indicator vector e_{ao} picks out the column of $\Phi^{\mathcal{AO}}$ corresponding to the particular action and observation at time t. Entries of $\Phi^{\mathcal{AO}}$ that require a particular action are set to zero for other actions. This implies that $\Phi^{\mathcal{AO}}$ is *block diagonal*.

To estimate observation features from data, we will pretend we get a sample $\phi_t^{\mathcal{AO}}$ in which we evaluate all of our observation features, even if different elements of $\phi^{\mathcal{AO}}$ depend on different actions. To get such a sample, we can pick a single action a according to our policy given h_t and receive a sample observation o. We then compute the coordinates that we can, multiply by an importance weight, and fill in zeros everywhere else:

$$\hat{\phi}_{k,t}^{\mathcal{AO}} = \frac{\Phi_{k,:}^{\mathcal{AO}} e_{a_t o_t}}{\mathbb{P}[a_t = a \mid h_t]} \tag{4.4}$$

The expected value of our sample vector is correct, since the probability of selecting our action $\mathbb{P}[a_t = a \mid h_t]$ cancels with the weighting factor $\frac{1}{\mathbb{P}[a_t = a | h_t]}$.

Defining PSRs in terms of observation features is more involved than extending the definition to characteristic and indicative features. In particular we have to be careful that extending to features does not interfere with the Bayes' rule update for the PSR state. In Section 4.3.1 we review how PSRs implement Bayes' rule and we provide a matrix algebra equation for implementing Bayes' rule. In Section 4.3.2 we show how to implement Bayes' rule when we are dealing with observation features. Finally, in Section 4.4 we define a PSR in terms of characteristic, indicative, and observation features.

4.3.1 Bayes' Rule with Discrete Observations

Recall from Section 3.2.2 that at each time step $t = 1, 2, \ldots$, a PSR predicts the joint probability of observation o and next state conditioned on action a

$$\mathbb{P}\left[x(h_{t+1}), o_t = o \mid h_t, a_t = a\right] = M_{ao} x(h_t) \tag{4.5}$$

Then, the normalization vector m_∞ marginalizes out the next state, giving just the probability of observation o_t

$$\mathbb{P}\left[o_t \mid h_t, a_t\right] = m_\infty^\top M_{ao} x(h_t) \tag{4.6}$$

and combining these two equations, the next PSR state is found by Bayes' Rule

$$x(h_{t+1}) = M_{ao}x(h_t)/m_\infty^\top M_{ao}x(h_t) \tag{4.7}$$

As we will see shortly, it can be useful to write the Bayes' rule update as a set of matrix operations. If we define:

$$P_{X^+,\mathcal{AO}|h_t} = \begin{bmatrix} M_{a_1o_1}x(h_t) & \dots M_{a_{|\mathcal{A}|}o_{|\mathcal{O}|}}x(h_t) \end{bmatrix} \tag{4.8}$$

$$P_{\mathcal{AO},\mathcal{AO}|h_t} = \begin{bmatrix} m_\infty^\top M_{a_1o_1}x(h_t) & & 0 \\ & \ddots & \\ 0 & & m_\infty^\top M_{a_{|\mathcal{A}|}o_{|\mathcal{O}|}}x(h_t) \end{bmatrix} \tag{4.9}$$

then we can write the next state as a linear function of an indicator vector of the current observation:

$$P_{X^+,\mathcal{AO}|h_t}P^{-1}_{\mathcal{AO},\mathcal{AO}|h_t}e_{ao} \tag{4.10}$$

where e_{ao} is an indicator vector that when multiplied by $P_{X^+,\mathcal{AO}|h_t}P^{-1}_{\mathcal{AO},\mathcal{AO}|h_t}$, selects the column containing the correct Bayes' rule update $x(h_{t+1}) = M_{ao}x(h_t)/m_\infty^\top M_{ao}x(h_t)$ at time t.

In Section 4.3.2 below, we generalize Equations 4.8, 4.9, and 4.10 to work with *features* of observations while still preserving the underlying Bayes' rule for updating the state of a PSR.

4.3.2 Bayes' Rule with Observation Features

In this section we generalize $P_{X^+,\mathcal{AO}|h_t}$ to the cross covariance of next state and features of observations conditioned on history $\Sigma_{\mathcal{AO},\mathcal{AO}|h_t}$ and we generalize $P_{\mathcal{AO},\mathcal{AO}|h_t}$ to the covariance of features of observations conditioned on history $\Sigma_{\mathcal{AO},\mathcal{AO}|h_t}$. More specifically:

$$\begin{aligned} \left[\Sigma_{\mathcal{AO},\mathcal{AO}|h_t}\right]_{i,j} &\stackrel{\text{def}}{=} \sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \Phi^{\mathcal{AO}}_{i,ao}\mathbb{P}[o_t = o \mid a_t = a, h_t]\Phi^{\mathcal{AO}\top}_{j,ao} \\ &= \Phi^{\mathcal{AO}}P_{\mathcal{AO},\mathcal{AO}|h_t}\Phi^{\mathcal{AO}\top} \end{aligned} \tag{4.11}$$

Next we define the covariance matrix of observations and expected next state conditioned on history:

$$
\begin{aligned}
\left[\Sigma_{X^+,\mathcal{AO}|h_t}\right]_{i,j} &\stackrel{\text{def}}{=} \sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \mathbb{P}\left[x^{\mathcal{O}}_{i,t+1}, o_t = o \mid h_t, a_t = a,\, \text{do}(x^{\mathcal{A}}_{t+1})\right] \Phi^{\mathcal{AO}\top}_{j,ao} \\
&= \sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} x_i(h_t, o_t = o, a_t = a)\mathbb{P}\left[o_t = o \mid h_t, a_t = a\right] \Phi^{\mathcal{AO}\top}_{j,ao} \\
&= \sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \frac{[M_{ao}]_{i,:}x(h_t)}{m_\infty^\top M_{ao}x(h_t)}\mathbb{P}\left[o_t = o \mid h_t, a_t = a\right] \Phi^{\mathcal{AO}\top}_{j,ao} \\
&= \sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} [M_{ao}]_{i,:}x(h_t)\Phi^{\mathcal{AO}\top}_{j,ao} \\
&= [P_{X^+,\mathcal{AO}|h_t}]_{i,:}\Phi^{\mathcal{AO}\top}_{j,:}
\end{aligned} \tag{4.12}
$$

If rank $\left(\Phi^{\mathcal{AO}}\right) \geq |\mathcal{A}| \times |\mathcal{O}|$, then $\Phi^{\mathcal{AO}\dagger}\Phi^{\mathcal{AO}} = I$: the features *uniquely characterize* the observations conditioned on actions. This implies that that the Bayes' rule update from Equation 4.10 is preserved:

$$
\begin{aligned}
\Sigma_{X^+,\mathcal{AO}|h_t}\Sigma^{-1}_{\mathcal{AO},\mathcal{AO}|h_t}\phi^{\mathcal{AO}}_t &= P_{X^+,\mathcal{AO}|h_t}\Phi^{\mathcal{AO}\top}\left(\Phi^{\mathcal{AO}\top}\right)^\dagger P^{-1}_{\mathcal{AO},\mathcal{AO}|h_t}\Phi^{\mathcal{AO}\dagger}\Phi^{\mathcal{AO}}e_{ao} \\
&= P_{X^+,\mathcal{AO}|h_t}P^{-1}_{\mathcal{AO},\mathcal{AO}|h_t}e_{ao}
\end{aligned} \tag{4.13}
$$

Conversely, if rank $\left(\Phi^{\mathcal{AO}}\right) < |\mathcal{A}| \times |\mathcal{O}|$, then then Equation 4.13 is only *approximately* the Bayes' rule update. However, even when Bayes' rule is approximated, using features can be a huge advantage in practice: the fact that we never need to explicitly estimate or represent a probability density function of observations can outweigh small inaccuracies in the Bayes rule update. We explore feature representations of Bayes' rule, and the practical advantages of this strategy, in more detail in Chapter 5 where we discuss and use the recent work on kernel Bayes' rule for learning non-parametric dynamical system models.

Given the equations for computing Bayes' rule with feature covariances, we are now in a position to define PSRs with observation features in such a way that we can perform Bayes' rule updates for filtering based only on observable quantities.

4.4 Defining PSRs with Observation Features

Defining PSRs with observation features requires combining Bayes' rule from the previous section with our derivations for discovering a state space with characteristic and indicative features from Section 4.2. Specifically, we use the fact that $\Sigma_{\mathcal{T},\mathcal{H}} = \Phi^{\mathcal{T}}\Gamma\Sigma_{X,\mathcal{H}}$ from Equation 4.1c to argue that the PSR state space can be found by a spectral decomposition of an observable covariance matrix and we use Bayes' rule update to state at each time step t using only moments of observable quantities.

Instead of defining one set of moments per action-observation pair $\Sigma_{\mathcal{T}^+,ao,\mathcal{H}}$, we define one set of moments per *feature*: let index $k = 1 \dots m$ range over features, and define a 3-mode *tensor* $\Sigma_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}}$:

$$[\Sigma_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}}]_{ikj} \stackrel{\text{def}}{=} \mathbb{E}\left[\phi^{\mathcal{T}}_{i,t+1} \cdot \hat{\phi}^{\mathcal{AO}}_{k,t} \cdot \phi^{\mathcal{H}}_{j,t} \,\middle|\, \text{do}(\zeta^+)\right] \tag{4.14}$$

Also, instead of working with tensors directly, in the following derivations it will often be convenient to project a tensor like $\Sigma_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}}$ to a *matrix* as follows:

$$\Sigma_{\mathcal{T}^+,\mathcal{AO}}(\eta) \stackrel{\text{def}}{=} \mathbb{E}\left[\phi^{\mathcal{T}}_{i,t+1}\langle \eta, \phi^{\mathcal{H}}_{j,t}\rangle \hat{\phi}^{\mathcal{AO}\top}_{t} \,\middle|\, \text{do}(\zeta^+)\right] \tag{4.15}$$

We can think of the matrix $\Sigma_{\mathcal{T}^+,\mathcal{AO}}(\eta)$ as re-weighting a cross covariance of characteristic and indicative features $\Sigma_{\mathcal{T}^+,\mathcal{AO}} = \mathbb{E}\left[\phi^{\mathcal{T}}_{t+1}\hat{\phi}^{\mathcal{AO}\top}_{t} \,\middle|\, \text{do}(\zeta^+)\right]$ by a scalar $\langle \eta, \phi^{\mathcal{H}}_{t}\rangle$. The inner product

$\langle \eta, \phi_t^{\mathcal{H}} \rangle$ is contracting $\phi_t^{\mathcal{H}}$ down to a single dimension. In particular:

$$
\begin{aligned}
\left[\Sigma_{\mathcal{T}^+,\mathcal{AO}}(\eta)\right]_{i,j} &\stackrel{\text{def}}{=} \mathbb{E}\left[\phi^{\mathcal{T}}_{i,t+1}\langle \eta, \phi_t^{\mathcal{H}}\rangle \hat{\phi}^{\mathcal{AO}\top}_{t,j} \,\middle|\, \text{do}(\zeta^+)\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\phi^{\mathcal{T}}_{i,t+1}\langle \eta, \phi_t^{\mathcal{H}}\rangle \hat{\phi}^{\mathcal{AO}\top}_{t,j} \,\middle|\, \text{do}(\zeta^+), h_t\right]\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\phi^{\mathcal{T}}_{i,t+1}\hat{\phi}^{\mathcal{AO}\top}_{t,j} \,\middle|\, \text{do}(\zeta^+), h_t\right] \langle \eta, \mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t\right]\rangle\right] \\
&= \mathbb{E}\left[\left(\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \mathbb{E}\left[\phi^{\mathcal{T}}_{i,t+1} \mid \text{do}(\zeta^+), h_t, o_t = o, a_t = a\right] \mathbb{P}[o_t = o, a_t = a \mid h_t] \frac{\Phi^{\mathcal{AO}\top}_{j,ao}}{\mathbb{P}[a_t = a \mid h_t]}\right) \langle \eta, \mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t\right]\rangle\right] \\
&= \mathbb{E}\left[\left(\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \mathbb{E}\left[\phi^{\mathcal{T}}_{i,t+1} \mid \text{do}(\zeta^+), h_t, o_t = o, a_t = a\right] \mathbb{P}[o_t = o \mid h_t, a_t = a]\mathbb{P}[a_t = a \mid h_t] \frac{\Phi^{\mathcal{AO}\top}_{j,ao}}{\mathbb{P}[a_t = a \mid h_t]}\right) \langle \eta, \mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t\right]\rangle\right] \\
&= \mathbb{E}\left[\left(\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \mathbb{E}\left[\phi^{\mathcal{T}}_{i,t+1} \mid \text{do}(\zeta^+), h_t, o_t = o, a_t = a\right] \mathbb{P}[o_t = o \mid h_t, a_t = a]\Phi^{\mathcal{AO}\top}_{j,ao}\right) \langle \eta, \mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t\right]\rangle\right] \\
&= \mathbb{E}\left[\left(\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \Phi^{\mathcal{T}}_{i,:}\Gamma x(h_t, o_t = o, a_t = a)\mathbb{P}[o_t = o \mid h_t, a_t = a]\Phi^{\mathcal{AO}\top}_{j,ao}\right) \langle \eta, \mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t\right]\rangle\right] \\
&= \Phi^{\mathcal{T}}_{i,:}\Gamma\mathbb{E}\left[\left(\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} x(h_t, o_t = o, a_t = a)\mathbb{P}[o_t = o \mid h_t, a_t = a]\Phi^{\mathcal{AO}\top}_{j,ao}\right) \langle \eta, \mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t\right]\rangle\right] \\
&= \Phi^{\mathcal{T}}_{i,:}\Gamma\mathbb{E}\left[\left(\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \frac{M_{ao}x(h_t)}{x_\infty^\top M_{ao}x(h_t)}\mathbb{P}[o_t = o \mid h_t, a_t = a]\Phi^{\mathcal{AO}\top}_{j,ao}\right) \langle \eta, \mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t\right]\rangle\right] \\
&= \Phi^{\mathcal{T}}_{i,:}\Gamma\mathbb{E}\left[\left(\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} M_{ao}x(h_t)\Phi^{\mathcal{AO}\top}_{j,ao}\right) \langle \eta, \mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t\right]\rangle\right] \\
&= \Phi^{\mathcal{T}}_{i,:}\Gamma\mathbb{E}\left[\left(\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|}\sum_{l=1}^{|\mathcal{Q}|} M_{aol}x_l(h_t)\Phi^{\mathcal{AO}\top}_{j,ao}\right) \langle \eta, \mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t\right]\rangle\right] \\
&= \Phi^{\mathcal{T}}_{i,:}\Gamma\mathbb{E}\left[\sum_{l=1}^{|\mathcal{Q}|}\left(\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} M_{aol}\Phi^{\mathcal{AO}\top}_{j,ao}\right) \langle \eta, \mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t\right] x_l(h_t)\rangle\right] \\
&= \Phi^{\mathcal{T}}_{i,:}\Gamma\sum_{l=1}^{|\mathcal{Q}|}\left(\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} M_{aol}\Phi^{\mathcal{AO}\top}_{j,ao}\right) \left\langle \eta, \mathbb{E}\left[\mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t\right] x_l(h_t)\right]\right\rangle \\
&= \Phi^{\mathcal{T}}_{i,:}\Gamma\sum_{l=1}^{|\mathcal{Q}|}\left(\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} M_{aol}\Phi^{\mathcal{AO}\top}_{j,ao}\right) \left\langle \eta, \Sigma_{X,\mathcal{H}_{l,:}}\right\rangle \\
&= \Phi^{\mathcal{T}}_{i,:}\Gamma\sum_{l=1}^{|\mathcal{Q}|}\Sigma_{X^+,\mathcal{AO}|X_l}{}_{:,j} \left\langle \eta, \Sigma_{X,\mathcal{H}_{l,:}}\right\rangle \\
\implies \Sigma_{\mathcal{T}^+,\mathcal{AO}}(\eta) &= \Phi^{\mathcal{T}}\Gamma\sum_{l=1}^{|\mathcal{Q}|}\Sigma_{X^+,\mathcal{AO}|X_l} \left\langle \eta, \Sigma_{X,\mathcal{H}_{l,:}}\right\rangle
\end{aligned}
\tag{4.16}
$$

where the covariance matrix $\Sigma_{X^+,\mathcal{AO}|X_l} \stackrel{\text{def}}{=} \left(\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} M_{aol}\Phi^{\mathcal{AO}\top}_{j,ao}\right)$. From $\Sigma_{\mathcal{T}^+,\mathcal{AO}}(\eta)$ we can

compute a linear transform of $\Sigma_{X^+,\mathcal{AO}|h_t}$:

$$
\begin{aligned}
U^\top \Sigma_{\mathcal{T}^+,\mathcal{AO}}((U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger b_t) &= U^\top \Sigma_{\mathcal{T}^+,\mathcal{AO}}((U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger (U^\top \Phi^{\mathcal{T}} \Gamma) x(h_t)) \\
&= U^\top \Phi^{\mathcal{T}} \Gamma \sum_{l=1}^{|\mathcal{Q}|} \left(\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} M_{aol} \Phi_{:,ao}^{\mathcal{AO}\top} \right) \left\langle ((U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger (U^\top \Phi^{\mathcal{T}} \Gamma) x(h_t), \Sigma_{X,\mathcal{H}_{l,:}} \right\rangle \\
&= U^\top \Phi^{\mathcal{T}} \Gamma \sum_{l=1}^{|\mathcal{Q}|} \left(\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} M_{aol} \Phi_{:,ao}^{\mathcal{AO}\top} \right) x_l(h_t) \\
&= U^\top \Phi^{\mathcal{T}} \Gamma \left(\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} M_{ao} x(h_t) \Phi_{:,ao}^{\mathcal{AO}\top} \right) \\
&= (U^\top \Phi^{\mathcal{T}} \Gamma) \Sigma_{X^+,\mathcal{AO}|h_t}
\end{aligned} \tag{4.17}
$$

Next we need to find $\Sigma_{\mathcal{AO},\mathcal{AO}|h_t}$. We start by defining a 3-mode tensor $\Sigma_{\mathcal{AO},\mathcal{H},\mathcal{AO}}$:

$$
[\Sigma_{\mathcal{AO},\mathcal{H},\mathcal{AO}}]_{ilj} \stackrel{\text{def}}{=} \mathbb{E}\left[\hat{\phi}_{i,t}^{\mathcal{AO}} \cdot \phi_{l,t}^{\mathcal{H}} \cdot \hat{\phi}_{j,t}^{\mathcal{AO}} \right] \tag{4.18}
$$

and its projection $\Sigma_{\mathcal{AO},\mathcal{AO}}(\eta)$:

$$
\begin{aligned}
[\Sigma_{\mathcal{AO},\mathcal{AO}}(\eta)]_{i,j} &\stackrel{\text{def}}{=} \mathbb{E}\left[\hat{\phi}_{i,t}^{\mathcal{AO}} \langle \eta, \phi_t^{\mathcal{H}} \rangle \hat{\phi}_{j,t}^{\mathcal{AO}\top} \right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\hat{\phi}_{i,t}^{\mathcal{AO}} \langle \eta, \phi_t^{\mathcal{H}} \rangle \hat{\phi}_{j,t}^{\mathcal{AO}\top} \mid h_t \right] \right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\hat{\phi}_{i,t}^{\mathcal{AO}} \hat{\phi}_{j,t}^{\mathcal{AO}\top} \mid h_t \right] \mathbb{E}\left[\langle \eta, \phi_t^{\mathcal{H}} \rangle \mid h_t \right] \right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\hat{\phi}_{i,t}^{\mathcal{AO}} \hat{\phi}_{j,t}^{\mathcal{AO}\top} \mid h_t \right] \left\langle \eta, \mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t \right] \right\rangle \right] \\
&= \mathbb{E}\left[\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \frac{\Phi_{i,ao}^{\mathcal{AO}} \Phi_{j,ao}^{\mathcal{AO}\top}}{\mathbb{P}[a_t = a \mid h_t]} \mathbb{P}\left[o_t = o, a_t = a \mid h_t \right] \left\langle \eta, \mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t \right] \right\rangle \right] \\
&= \mathbb{E}\left[\sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \frac{\Phi_{i,ao}^{\mathcal{AO}} \Phi_{j,ao}^{\mathcal{AO}\top}}{\mathbb{P}[a_t = a \mid h_t]} \mathbb{P}\left[o_t = o \mid a_t = a, h_t \right] \mathbb{P}[a_t = a \mid h_t] \left\langle \eta, \mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t \right] \right\rangle \right] \\
&= \sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \Phi_{i,ao}^{\mathcal{AO}} \Phi_{j,ao}^{\mathcal{AO}\top} \mathbb{E}\left[\mathbb{P}\left[o_t = o \mid a_t = a, h_t \right] \left\langle \eta, \mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t \right] \right\rangle \right] \\
&= \sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \Phi_{i,ao}^{\mathcal{AO}} \Phi_{j,ao}^{\mathcal{AO}\top} \left\langle \eta, \mathbb{E}\left[\mathbb{P}\left[o_t = o \mid a_t = a, h_t \right] \mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t \right] \right] \right\rangle \\
&= \sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \Phi_{i,ao}^{\mathcal{AO}} \Phi_{j,ao}^{\mathcal{AO}\top} \left\langle \eta, \mathbb{E}\left[g_{ao}^\top x(h_t) \mathbb{E}\left[\phi_t^{\mathcal{H}} \mid h_t \right] \right] \right\rangle \\
&= \sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \Phi_{i,ao}^{\mathcal{AO}} \Phi_{j,ao}^{\mathcal{AO}\top} \langle \eta, \Sigma_{X,\mathcal{H}}^\top g_{ao} \rangle \\
\implies \Sigma_{\mathcal{AO},\mathcal{AO}}(\eta) &= \sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \Phi_{ao}^{\mathcal{AO}} \Phi_{ao}^{\mathcal{AO}\top} \langle \eta, \Sigma_{X,\mathcal{H}}^\top g_{ao} \rangle
\end{aligned} \tag{4.19}
$$

Given $\Sigma_{\mathcal{AO},\mathcal{AO}|h_t}$ and the current state b_t, we can calculate $\Sigma_{\mathcal{AO},\mathcal{AO}|h_t}$ as follows. We see that we can compute the probability $\mathbb{P}[o_t = o \mid a_t = a, h_t]$ from the current state b_t:

$$\begin{aligned}
g_{ao}^\top \Sigma_{X,\mathcal{H}}(U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger b_t &= g_{ao}^\top \Sigma_{X,\mathcal{H}}(U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger (U^\top \Phi^{\mathcal{T}} \Gamma) x(h_t) \\
&= g_{ao}^\top (U^\top \Phi^{\mathcal{T}} \Gamma)^{-1} (U^\top \Phi^{\mathcal{T}} \Gamma) \Sigma_{X,\mathcal{H}}(U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger (U^\top \Phi^{\mathcal{T}} \Gamma) x(h_t) \\
&= g_{ao}^\top (U^\top \Phi^{\mathcal{T}} \Gamma)^{-1} (U^\top \Phi^{\mathcal{T}} \Gamma) x(h_t) \\
&= g_{ao}^\top x(h_t) \\
&= \mathbb{P}[o_t = o \mid a_t = a, h_t] \qquad (4.20)
\end{aligned}$$

Combining Equation 4.19 and Equation 4.20, we can find the conditional embedding of the covariance $\Sigma_{\mathcal{AO},\mathcal{AO}|h_t}$:

$$\begin{aligned}
\Sigma_{\mathcal{AO},\mathcal{AO}}\left((U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger b_t\right) &= \sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \Phi_{ao}^{\mathcal{AO}} \Phi_{ao}^{\mathcal{AO}\top} \langle (U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger b_t, \Sigma_{X,\mathcal{H}}^\top g_{ao} \rangle \\
&= \sum_{ao=1}^{|\mathcal{A}|\times|\mathcal{O}|} \Phi_{ao}^{\mathcal{AO}} \mathbb{P}[o_t = o \mid a_t = a, h_t] \Phi_{ao}^{\mathcal{AO}\top} \\
&= \Sigma_{\mathcal{AO},\mathcal{AO}|h_t} \qquad (4.21)
\end{aligned}$$

Finally, we can implement the recursive Bayes' rule for PSRs with only observable features:

$$\begin{aligned}
b_{t+1} &= (U^\top \Phi^{\mathcal{T}} \Gamma) x_{t+1} \\
&= (U^\top \Phi^{\mathcal{T}} \Gamma) \Sigma_{X^+,\mathcal{AO}|h_t} \Sigma_{\mathcal{AO},\mathcal{AO}|h_t}^{-1} \phi_t^{\mathcal{AO}} \\
&= U^\top \Sigma_{\mathcal{T}^+,\mathcal{AO}|h_t} \Sigma_{\mathcal{AO},\mathcal{AO}|h_t}^{-1} \phi_t^{\mathcal{AO}} \\
&= U^\top \Sigma_{\mathcal{T}^+,\mathcal{AO}|h_t} \left(\Sigma_{\mathcal{AO},\mathcal{AO}}\left((U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger b_t\right)\right)^{-1} \phi_t^{\mathcal{AO}} \\
&= U^\top \Sigma_{\mathcal{T}^+,\mathcal{AO}}\left((U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger b_t\right) \left(\Sigma_{\mathcal{AO},\mathcal{AO}}\left((U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger b_t\right)\right)^{-1} \phi_t^{\mathcal{AO}} \qquad (4.22)
\end{aligned}$$

Importantly, note that in the final line above *all quantities are observable*.

We can now define our spectral learning algorithm for PSRs with many observations. Unlike the few-observation case, to save space and time, we do not precompute and store $\widehat{B}_{ao}$ for each action-observation pair. Instead, we compute the Bayes' rule state update as needed at tracking time via Equation 4.22.

4.4.1 The Moment Spectral Learning Algorithm for PSRs

The equations from the previous section yield a simple spectral learning algorithm. Our algorithm will estimate the moments $\Sigma_{\mathcal{T},\mathcal{H}}, \Sigma_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}}$, and $\Sigma_{\mathcal{AO},\mathcal{H},\mathcal{AO}}$ from data and then use Equation 4.22 to update state. Just as before, once we have computed the estimated moments $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}, \widehat{\Sigma}_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}}$ and $\widehat{\Sigma}_{\mathcal{AO},\mathcal{H},\mathcal{AO}}$, we can generate $\widehat{U}$ by singular value decomposition of $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$.

For reference, we summarize the learning algorithm here:

1. Compute empirical estimates of the moments: $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}, \widehat{\Sigma}_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}}$ and $\widehat{\Sigma}_{\mathcal{AO},\mathcal{H},\mathcal{AO}}$.

2. Compute a SVD of $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$ to find $\widehat{U}$, the matrix of left singular vectors corresponding to the d largest singular values.
3. Compute the initial state: $\widehat{b}_* = \widehat{U}^\top \widehat{P}_{\mathcal{T},\mathcal{H}} e$.
4. At each time step update state with generalized Bayes' rule:
$\hat{b}_{t+1} = \widehat{U}^\top \widehat{\Sigma}_{\mathcal{T}^+,\mathcal{AO}} \left((\widehat{U}^\top \widehat{\Sigma}_{\mathcal{T},\mathcal{H}})^\dagger \hat{b}_t \right) \left(\widehat{\Sigma}_{\mathcal{AO},\mathcal{AO}} \left((\widehat{U}^\top \widehat{\Sigma}_{\mathcal{T},\mathcal{H}})^\dagger \hat{b}_t \right) \right)^{-1} \phi_t^{\mathcal{AO}}$.

As we include more data in our averages, the law of large numbers guarantees that our estimates $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$, $\widehat{\Sigma}_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}}$ and $\widehat{\Sigma}_{\mathcal{AO},\mathcal{H},\mathcal{AO}}$ converge to the true matrices $\Sigma_{\mathcal{T},\mathcal{H}}$, $\Sigma_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}}$, and $\Sigma_{\mathcal{AO},\mathcal{H},\mathcal{AO}}$. So by continuity of the formulas above, if our system is truly a PSR of finite rank, and our features are sufficiently expressive, our estimates $\widehat{b}_*$ and Bayes' rule converge to the true parameters up to a linear transform—that is, our learning algorithm is *consistent*. [2]

[2] Again, with the same caveat about the SVD.

Chapter 5

Hilbert Space Embeddings of Predictive State Representations

The standard spectral algorithm for PSRs derived in Chapter 3 is formulated for discrete random variables, and, in Chapter 4, an efficient spectral learning algorithm was derived for *large* sets of actions and observations. The approach in Chapter 4 utilized finite sets of *features* of observations, tests, and histories. In this chapter we will explore this concept further and examine how to extend these ideas to general cases with continuous and structured variables.

In this chapter, we fully generalize PSRs to (potentially) continuous observation and action sets using a recent concept called Hilbert space embeddings of distributions [94, 103]. The essence of our method is to represent distributions of tests, histories, observations, and actions, as points in (possibly) infinite-dimensional Hilbert spaces. During filtering we update these points entirely in the Hilbert spaces using a kernel version of Bayes' rule.

The proposed method is similar to recent work that applies kernel methods to dynamical system modeling and reinforcement learning, which we summarize here. Song et al. [99] proposed a nonparametric approach to learning HMM representations in RKHS. The resulting nonparametric dynamical system model, called Hilbert Space Embeddings of Hidden Markov Models (HSE-HMMs), proved to be an impressive alternative to competing dynamical system models on a number of experimental benchmarks. Despite these successes, the HSE-HMM has two major limitations: first, the update rule for the HMM only *approximates* the state update (up to a fixed multiplicative scalar) instead of performing direct Bayesian inference in Hilbert space. Second, the model lacks the capacity to reason about actions, which limits the scope of systems that can be modeled. Our model can be viewed as an extension of HSE-HMMs that adds control inputs and updates state using a kernelized version of Bayes' rule.

Grünewälder et al. [37] proposed a nonparametric approach to learning transition dynamics in Markov decision processes (MDPs) by representing the MDP transitions as conditional distributions in RKHS. This work was extended by Nishiyama et al. [73] who developed a nonparametric approach for policy learning in POMDPs that represents distributions over the states, observations, and actions as embeddings in a RKHSs. The resulting model is called Hilbert Space Embeddings of POMDPs (HSE-POMDPs). Distributions over states given the observations are obtained by applying the kernel Bayes rule to these distribution embeddings. Policies and value functions are then defined on the feature space over states. Critically, the authors only provided

results for fully observable models, where the training data includes labels for the true latent states. By contrast, our algorithm only requires access to an (unlabeled) sequence of actions and observations.

5.1 Hilbert Space Embeddings

At a high level, we will be extending the idea of feature representations from Chapter 4 from finite feature spaces to *function spaces*. This requires some additional concepts and tools for reasoning about statistics in reproducing kernel Hilbert spaces (RKHSs), which we review here. However, once these concepts are introduced, the observable representation of PSRs in RKHSs can be seen as *identical* to the representation in Chapter 3 or Chapter 4. (For example, if we use the discrete/delta kernel on a finite set of observations, we recover the algorithm of Chapter 3; and if we use a finite-dimensional RKHS, we recover the algorithm of Chapter 4.) The sole practical difference in the learning algorithm is the use of the *kernel trick* to contend with the fact that the characteristic, indicative, and observation features are no longer required to be finite-dimensional. We will derive the algorithm using infinite-dimensional operators first, since that representation best shows the analogy to Chapters 3–4. Then, starting in Section 5.3 we will show how to implement the algorithm using only finite-dimensional Gram matrices.

5.1.1 Mean Maps

Let $\mathcal{F}$ be a reproducing kernel Hilbert space (RKHS) associated with kernel $K_X(x, x') \stackrel{\text{def}}{=} \left\langle \phi^X(x), \phi^X(x') \right\rangle_{\mathcal{F}}$. Then for all functions $f \in \mathcal{F}$ and $x \in \mathcal{X}$ we have the reproducing property: $\left\langle f, \phi^X(x) \right\rangle_{\mathcal{F}} = f(x)$, *i.e.* the evaluation of function f at x can be written as an inner product. Examples of kernels include the Gaussian RBF kernel $K_X(x, x') = \exp(-s \left\| x - x' \right\|^2)$, however kernel functions have also been defined on strings, graphs, and other structured objects.

Let $\mathcal{P}$ be the set of probability distributions on $\mathcal{X}$, and X the random variable with distribution $\mathbb{P} \in \mathcal{P}$. Following Smola et al. [94], we define the mean map of $\mathbb{P} \in \mathcal{P}$ into RKHS $\mathcal{F}$ to be $\mu_X \stackrel{\text{def}}{=} \mathbb{E}\left[\phi^X(X)\right]$, also called the Hilbert space embedding of $\mathbb{P}$ or simply mean map. For all $f \in \mathcal{F}$, $\mathbb{E}[f(X)] = \langle f, \mu_X \rangle_{\mathcal{F}}$ by the reproducing property and linearity of expectations. We may think of the mean map by analogy with a mean vector in a finite dimensional space: if $\mathcal{F} = \mathbb{R}^d$, then $f \in \mathbb{R}^d$ is some fixed vector, X is a random vector defined on $\mathbb{R}^d$ with mean μ_X, and $\mathbb{E}\left\langle f, X \right\rangle_{\mathcal{F}} = f^\top \mu_X$.

A *characteristic* RKHS is one for which the mean map is injective: that is, each distribution has a unique embedding [103]. This property holds for many commonly used kernels (eg. the Gaussian and Laplace kernels when $\mathcal{X} = \mathbb{R}^d$).

As a special case of the mean map, the marginal probability vector of a discrete variable X is a Hilbert space embedding, *i.e.* $(\mathbb{P}[X = i])_{i=1}^M = \mu_X$. Here the kernel is the delta function $K_X(x, x') = \mathbb{I}(x = x)$, and the feature map is the 1-of-M representation for discrete variables.

Given t *i.i.d.* observations $\{x_t\}_{t=1}^T$, an estimate of the mean map is straightforward:

$$\hat{\mu}_X \stackrel{\text{def}}{=} \frac{1}{T}\sum_{t=1}^{T}\phi^X(x_t) = \frac{1}{T}\Upsilon^X \mathbf{1}_T \tag{5.1}$$

where $\Upsilon^X \stackrel{\text{def}}{=} (\phi^X(x_1), \ldots, \phi^X(x_T))$ is the operator which maps the tth unit vector of $\mathbb{R}^T$ to $\phi^X(x_t)$, which we can think of as an arrangement of feature maps into columns. Assume $K_X(x, x')$ bounded; then with probability $1-\delta$, this estimate computes an approximation with an error of $\|\hat{\mu}_X - \mu_X\|_{\mathcal{F}} = O(T^{-1/2}(\log(1/\delta))^{1/2})$ [94].

5.1.2 Covariance Operators

The covariance operator is a generalization of the covariance matrix. Given a joint distribution $\mathbb{P}[X, Y]$ over two variables X on $\mathcal{X}$ and Y on $\mathcal{Y}$[1]the *uncentered* covariance operator $\mathcal{C}_{XY}$ is [6]

$$\mathcal{C}_{XY} \stackrel{\text{def}}{=} \mathbb{E}_{XY}\left[\phi^X(X) \otimes \phi^Y(Y)\right], \tag{5.2}$$

where $\otimes$ denotes tensor product. Alternatively, $\mathcal{C}_{XY}$ can simply be viewed as an embedding of joint distribution $\mathbb{P}[X, Y]$ using joint feature map $\psi(x, y) \stackrel{\text{def}}{=} \phi^X(x) \otimes \phi^Y(y)$ (in tensor product RKHS $\mathcal{F} \otimes \mathcal{G}$). Given two functions, $f \in \mathcal{F}$ and $g \in \mathcal{G}$, their cross-covariance is computed as:

$$\langle f, \mathcal{C}_{XY} g\rangle_{\mathcal{F}} \text{ or equivalently } \langle f \otimes g, \mathcal{C}_{XY}\rangle_{\mathcal{F}\otimes\mathcal{G}}. \tag{5.3}$$

For discrete variables X and Y with delta kernels on both domains, the covariance operator will coincide with the joint probability table, *i.e.* $\mathbb{P}[X = i, Y = j]_{i,j=1}^M = \mathcal{C}_{XY}$.

Given T pairs of *i.i.d.* observations $\{(x_t, y_t)\}_{t=1}^T$, we denote $\Upsilon^X = \left(\phi^X(x_1), \ldots, \phi^X(x_T)\right)$ and $\Upsilon^Y = \left(\phi^Y(y_1), \ldots, \phi^Y(y_T)\right)$. The covariance operator $\mathcal{C}_{XY}$ can then be estimated as

$$\widehat{\mathcal{C}}_{XY} = \frac{1}{T}\Upsilon^X \Upsilon^{Y*} \tag{5.4}$$

where Υ^* denotes the adjoint of Υ. Assume $K_X(x, x')$ and $K_Y(y, y')$ are bounded. With probability $1-\delta$, this estimate computes an approximation with an error of $\|\widehat{\mathcal{C}}_{XY} - \mathcal{C}_{XY}\|_{\mathcal{F}\otimes\mathcal{G}} = O(T^{-1/2}(\log(1/\delta))^{1/2})$ [94].

5.1.3 Conditional Embedding Operators

Based on covariance operators, Song et al. [98] define a conditional embedding operator which allows us to compute conditional expectations $\mathbb{E}\left[\phi^Y(Y) \mid X\right]$ as linear operators in RKHS. We define the conditional embedding operator $\mathcal{W}_{Y|X} : \mathcal{F} \mapsto \mathcal{G}$

$$\mathcal{W}_{Y|X} \stackrel{\text{def}}{=} \mathcal{C}_{YX}\mathcal{C}_{XX}^{-1} \tag{5.5}$$

[1]We assume a kernel $K_y(y, y') = \left\langle \phi^Y(y), \phi^Y(y')\right\rangle_{\mathcal{G}}$ is defined on $\mathcal{Y}$ with associated RKHS $\mathcal{G}$.

such that for all $g \in \mathcal{G}$

$$\mathbb{E}[g(Y) \mid x] = \langle g, \mathcal{W}_{Y|X}\phi^X(x)\rangle_{\mathcal{G}}$$

given several assumptions. First we assume $\mathbb{E}[g(Y) \mid X = \cdot] \in \mathcal{F}$ for all $g \in \mathcal{G}$. Next we define the operator $S : \mathcal{G} \mapsto \mathcal{F}$ such that $Sg = \mathbb{E}[g(Y) \mid X = \cdot] \in \mathcal{F}$, which implies $\mathcal{C}_{XY}g = \mathcal{C}_{XX}\mathbb{E}[g(Y) \mid X = \cdot]$ [34]. If we further assume that $\mathcal{C}_{XX}$ is injective, we can write $\mathcal{C}_{XX}^{-1}\mathcal{C}_{XY}g = \mathbb{E}[g(Y) \mid X = \cdot]$ meaning that $S \stackrel{\text{def}}{=} \mathcal{C}_{XX}^{-1}\mathcal{C}_{XY}$.[2] Finally we assume that S is bounded which implies it has an adjoint S^*. By the reproducing theorem,

$$\begin{aligned}
\mathbb{E}[g(Y) \mid X = x] &= \langle \mathbb{E}[g(Y) \mid X = \cdot], \phi^X(x)\rangle_{\mathcal{F}} \\
&= \langle Sg, \phi^X(x)\rangle_{\mathcal{F}} \\
&= \langle g, S^*\phi^X(x)\rangle_{\mathcal{F}} \\
&= \langle g, \mathcal{C}_{YX}\mathcal{C}_{XX}^{-1}\phi^X(x)\rangle_{\mathcal{F}}
\end{aligned}$$

Therefore, there exists a $\mathcal{W}_{Y|X} = \mathcal{C}_{YX}\mathcal{C}_{XX}^{-1}$.

For discrete variables with delta kernels, conditional embedding operators correspond exactly to conditional probability tables (CPT), *i.e.* $(\mathbb{P}[Y = i \mid X = j])_{i,j=1}^M = \mathcal{C}_{Y|X}$, and each individual conditional embedding corresponds to one column of the CPT, *i.e.* $(\mathbb{P}[Y = i \mid X = x])_{i=1}^M = \mu_{Y|x}$.

Given T *i.i.d.* pairs $\{(x_t, y_t)\}_{t=1}^T$ from $\mathbb{P}[X, Y]$, the conditional embedding operator can be estimated as

$$\widehat{\mathcal{W}}_{Y|X} = \frac{\Upsilon^Y\Upsilon^{X*}}{T}\left(\frac{\Upsilon^X\Upsilon^{X*}}{T} + \lambda I\right)^{-1} = \Upsilon^Y(G_{X,X} + \lambda T I)^{-1}\Upsilon^{X*} \tag{5.6}$$

where we have defined the Gram matrix $G_{X,X} \stackrel{\text{def}}{=} \Upsilon^{X*}\Upsilon^X$ with (i, j)th entry $K_X(x_i, x_j)$. The regularization parameter λ helps to avoid overfitting and to ensure invertibility (and thus that the resulting operator is well defined). Under strong assumptions Song et al. [100] show that $\widehat{\mathcal{W}}_{Y|X}$ converges to its population counterpart in probability. However, convergence should also be guaranteed under weaker assumptions, see [34] for details.

5.1.4 Kernel Bayes' Rule

We are now in a position to define the kernel mean map implementation of Bayes' rule (which we call Kernel Bayes' Rule (KBR)). In deriving the kernel realization of Bayes' rule we use conditional embedding operators to obtain the kernel mean representation of a conditional *joint* probability $\mathbb{P}[X, Y \mid Z = z]$. Given Hilbert spaces $\mathcal{F}$, $\mathcal{G}$, and $\mathcal{H}$ corresponding to the embedding of x, y, and z respectively, this can be represented as an RKHS operator $\mathcal{C}_{X,Y|z} \stackrel{\text{def}}{=}$

[2]Note that we do not require $\mathcal{C}_{XX}^{-1}$ to be bounded: in fact, it wont be bounded if the eigenspectrum is countable, which will be the case for a Gaussian kernel. But the composite operator $\mathcal{C}_{XX}^{-1}\mathcal{C}_{XY}$ can still be bounded if the singular values of $\mathcal{C}_{XY}$ decay fast enough, and its singular vectors are somewhat aligned with those of $\mathcal{C}_{XX}$.

$\mathbb{E}\left[\phi^X(X) \otimes \phi^Y(Y) \mid z\right]$. We define the conditional covariance operator $\mathcal{H} \mapsto \mathcal{F} \otimes \mathcal{G}$ given several assumptions [34]:

$$\mathcal{C}_{XY|z} \stackrel{\text{def}}{=} \mathcal{C}_{(XY)Z}\mathcal{C}_{ZZ}^{-1}\phi(z) \tag{5.7}$$

Here the covariance operator $\mathcal{C}_{(XY)Z}$ is defined by the random variable $((X, Y), Z)$.

Our goal is to perform a Bayes rule update $\mathbb{P}\left[X \mid Y = y, Z = z\right] = \frac{\mathbb{P}[X,Y=y|Z=z]}{\mathbb{P}[Y=y|Z=z]}$ in RKHS. We accomplish this by defining kernel Bayes' rule in terms of conditional covariance operators [34]:

$$\mu_{X|y,z} = \mathcal{C}_{XY|z}\mathcal{C}_{YY|z}^{-1}\phi(y) \tag{5.8}$$

For discrete variables with delta kernels, KBR corresponds exactly to the discrete Bayes' rule update *i.e.* $\left(\frac{\mathbb{P}[X=i,Y=j|Z=k]}{\mathbb{P}[Y=j|Z=k]}\right)_{i,j,k=1}^{M} = \mathcal{W}_{X|Y,Z}$, and each individual conditional embedding corresponds to one column of the CPT, *i.e.* $(\mathbb{P}\left[X = i \mid Y = y, Z = z\right])_{i=1}^{M} = \mu_{X|y,z}$.

Given T *i.i.d.* triples $\{(x_t, y_t, z_t)\}_{t=1}^{T}$ from $\mathbb{P}\left[X, Y, Z\right]$, we denote $\Upsilon^X = \left(\phi^X(x_1), \ldots, \phi^X(x_T)\right)$, $\Upsilon^Y = \left(\phi^Y(y_1), \ldots, \phi^Y(y_T)\right)$, and $\Upsilon^Z = \left(\phi^Z(z_1), \ldots, \phi^Z(z_T)\right)$. The Bayes' rule update can be estimated by first estimating the conditional embedding of the covariance operators $\widehat{\mathcal{C}}_{XY|z}$ and $\widehat{\mathcal{C}}_{YY|z}$ via the Equations in Section 5.1.3, and then estimating $\widehat{\mathcal{C}}_{X|y,z} = \widehat{\mathcal{C}}_{XY|z}\left(\widehat{\mathcal{C}}_{YY|z} + \lambda I\right)^{-1}\phi^Y(y)$. We can express this equation with Gram matrices as follows [34]:

$$\Lambda_z = \text{diag}((G_{Z,Z} + \lambda T I)^{-1}\Upsilon^{Z*}\phi^Z(z)) \tag{5.9}$$

$$\widehat{\mathcal{C}}_{X|y,z} = \Upsilon^X\Lambda_z G_{Y,Y}((\Lambda_z G_{Y,Y})^2 + \lambda T I)^{-1}\Lambda_z\Upsilon^{Y*}\phi^Y(y) \tag{5.10}$$

where we have defined the Gram matrix $G_{Y,Y} \stackrel{\text{def}}{=} \Upsilon^{Y*}\Upsilon^Y$ with (i, j)th entry $K_Y(y_i, y_j)$ and Gram matrix $G_{Z,Z} \stackrel{\text{def}}{=} \Upsilon^{Z*}\Upsilon^Z$ with (i, j)th entry $K_Z(z_i, z_j)$. The function diag$(\cdot)$ takes a vector as input and returns a diagonal matrix. The diagonal elements of Λ_z weight the samples thereby encoding the conditioning information from z. Since the weights may be negative, we use a different type of regularization than the standard Tikhonov regularization used in Equation 5.6 [34].

5.2 Predictive Representations in RKHS

We will focus on learning the conditional embedding operator $\mathcal{W}_{\mathcal{T}^{\mathcal{O}}|\mathcal{T}^{\mathcal{A}},h_t}$ for the predictive density of a core set of tests $\mathbb{P}\left[\tau_t^{\mathcal{O}} \mid \tau_t^{\mathcal{A}}, h_t\right]$ of a PSR and updating this conditional embedding operator given a new action and observation using kernel Bayes' rule. In this chapter we assume that our data was generated by a *blind* policy, where future actions do not rely on future observations. This means that the PSR state is the *conditional* probability of observation sequences given action sequences (we do not need to worry about interventions). This is an expressive representation: we can model near-arbitrary continuous-valued dynamical systems, limited only by the existence of the conditional embedding operator $\mathcal{W}_{\mathcal{T}^{\mathcal{O}}|\mathcal{T}^{\mathcal{A}},h_t}$ (and therefore the assumptions in given in Section 5.1.3).

5.2.1 Conditional Predictive Representations

We begin by defining kernels on core test observations $\tau^{\mathcal{O}}$, core test actions $\tau^{\mathcal{A}}$, observations o, actions a, and histories h:

$$K_{\mathcal{T}^{\mathcal{O}}}(\tau^{\mathcal{O}}, \tau'^{\mathcal{O}}) = \langle \phi^{\mathcal{T}^{\mathcal{O}}}(\tau^{\mathcal{O}}), \phi^{\mathcal{T}^{\mathcal{O}}}(\tau'^{\mathcal{O}}) \rangle_{\mathcal{F}} \tag{5.11}$$

$$K_{\mathcal{T}^{\mathcal{A}}}(\tau^{\mathcal{A}}, \tau'^{\mathcal{A}}) = \langle \phi^{\mathcal{T}^{\mathcal{A}}}(\tau^{\mathcal{A}}), \phi^{\mathcal{T}^{\mathcal{A}}}(\tau'^{\mathcal{A}}) \rangle_{\mathcal{G}} \tag{5.12}$$

$$K_{\mathcal{O}}(o, o') = \left\langle \phi^{\mathcal{O}}(o), \phi^{\mathcal{O}}(o') \right\rangle_{\mathcal{J}} \tag{5.13}$$

$$K_{\mathcal{A}}(a, a') = \left\langle \phi^{\mathcal{A}}(a), \phi^{\mathcal{A}}(a') \right\rangle_{\mathcal{K}} \tag{5.14}$$

$$K_{\mathcal{H}}(h, h') = \left\langle \phi^{\mathcal{H}}(h), \phi^{\mathcal{H}}(h') \right\rangle_{\mathcal{L}} \tag{5.15}$$

Predictive State in RKHS Given the covariance operator for embedded action sequences $\mathcal{C}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}} \stackrel{\text{def}}{=} \mathbb{E}\left[\phi^{\mathcal{T}^{\mathcal{A}}}(\tau_t^{\mathcal{A}}) \otimes \phi^{\mathcal{T}^{\mathcal{A}}}(\tau_t^{\mathcal{A}})\right]$ and the cross covariance operator between embedded observation sequences and action sequences $\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}} \stackrel{\text{def}}{=} \mathbb{E}\left[\phi^{\mathcal{T}^{\mathcal{O}}}(\tau_t^{\mathcal{O}}) \otimes \phi^{\mathcal{T}^{\mathcal{A}}}(\tau_t^{\mathcal{A}})\right]$, the Hilbert space embedding of tests conditioned on a *single* action sequence is the conditional embedding detailed in Section 5.1.3:

$$\mu_{\mathcal{T}^{\mathcal{O}}|\tau_t^{\mathcal{A}}} = \mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}} \mathcal{C}^{-1}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}} \phi^{\mathcal{T}^{\mathcal{A}}}(\tau_t^{\mathcal{A}}) \tag{5.16}$$

However, a predictive representation of the future is a *collection* of distributions of tests, one per action sequence. We can represent the PSR state as a conditional embedding *operator* $\mathcal{W}_{\mathcal{T}^{\mathcal{O}}|\mathcal{T}^{\mathcal{A}}}$:

$$\mathcal{W}_{\mathcal{T}^{\mathcal{O}}|\mathcal{T}^{\mathcal{A}}} = \mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}} \mathcal{C}^{-1}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}} \tag{5.17}$$

From this operator we can find the mean embedding for any action sequence as $\mu_{\mathcal{T}^{\mathcal{O}}|\tau_t^{\mathcal{A}}} = \mathcal{W}_{\mathcal{T}^{\mathcal{O}}|\mathcal{T}^{\mathcal{A}}} \phi^{\mathcal{T}^{\mathcal{A}}}(\tau_t^{\mathcal{A}})$ by Equation 5.16.

If we want to compute the PSR state given a particular history h_t, we define several new covariance operators and apply KBR. We first define $\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}},\mathcal{H}} \stackrel{\text{def}}{=} \mathbb{E}\left[\phi^{\mathcal{T}^{\mathcal{O}}}(\tau_t^{\mathcal{O}}) \otimes \phi^{\mathcal{T}^{\mathcal{A}}}(\tau_t^{\mathcal{A}}) \otimes \phi^{\mathcal{H}}(h_t)\right]$, $\mathcal{C}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}},\mathcal{H}} \stackrel{\text{def}}{=} \mathbb{E}\left[\phi^{\mathcal{T}^{\mathcal{A}}}(\tau_t^{\mathcal{A}}) \otimes \phi^{\mathcal{T}^{\mathcal{A}}}(\tau_t^{\mathcal{A}}) \otimes \phi^{\mathcal{H}}(h_t)\right]$, and $\mathcal{C}_{\mathcal{H},\mathcal{H}} \stackrel{\text{def}}{=} \mathbb{E}\left[\phi^{\mathcal{H}}(h_t) \otimes \phi^{\mathcal{H}}(h_t)\right]$. We then compute the *conditional* covariance operators

$$\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t} = \mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} \mathcal{C}^{-1}_{\mathcal{H},\mathcal{H}} \phi^{\mathcal{H}}(h_t) \tag{5.18}$$

$$\mathcal{C}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|h_t} = \mathcal{C}_{(\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} \mathcal{C}^{-1}_{\mathcal{H},\mathcal{H}} \phi^{\mathcal{H}}(h_t) \tag{5.19}$$

Finally, we can compute the conditional embedding operator $\mathcal{W}_{\mathcal{T}^{\mathcal{O}}|\mathcal{T}^{\mathcal{A}},h_t}$:

$$\mathcal{W}_{\mathcal{T}^{\mathcal{O}}|\mathcal{T}^{\mathcal{A}},h_t} = \mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t} \mathcal{C}^{-1}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|h_t} \tag{5.20}$$

The conditional embedding operator $\mathcal{W}_{\mathcal{T}^{\mathcal{O}}|\mathcal{T}^{\mathcal{A}},h_t}$ is the kernel mean embedding of $\mathbb{P}\left[\tau_t^{\mathcal{O}} \mid \tau_t^{\mathcal{A}}, h_t\right]$, i.e. it is the Hilbert space embedding of a PSR state. It is a predictive *state* since it uniquely encodes the predictive density of future observations in the dynamical system given future action sequences. In Section 5.2.2 we detail how to find a PSR state in RKHS given a previous state and an action-observation pair, instead of needing to use the entire history.

In Sections 5.2.2 and 5.3 below we break the PSR state $\mathcal{W}_{\mathcal{T}^{\mathcal{O}}|\mathcal{T}^{\mathcal{A}},h_t}$ down into its constituent parts, the conditional embedding operators $\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t}$ and $\mathcal{C}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|h_t}$. We then update these two operators instead of $\mathcal{W}_{\mathcal{T}^{\mathcal{O}}|\mathcal{T}^{\mathcal{A}},h_t}$ directly. This representation is at least as expressive since we can always reconstruct $\mathcal{W}_{\mathcal{T}^{\mathcal{O}}|\mathcal{T}^{\mathcal{A}},h_t} = \mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t}\mathcal{C}^{-1}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|h_t}$.

Conditional Observations Here we use the rules for conditional embedding from Section 5.1.3 to show how to calculate the conditional embedding of an observation distribution given a state and an action. We start by showing how to embed the observation distribution of a stateless process. We define $\mathcal{C}_{\mathcal{A},\mathcal{A}} \stackrel{\text{def}}{=} \mathbb{E}\left[\phi^{\mathcal{A}}(a_t) \otimes \phi^{\mathcal{A}}(a_t)\right]$ and $\mathcal{C}_{\mathcal{O},\mathcal{A}} \stackrel{\text{def}}{=} \mathbb{E}\left[\phi^{\mathcal{O}}(o_t) \otimes \phi^{\mathcal{A}}(a_t)\right]$. The conditional embedding of the observation distribution is therefore:

$$\mu_{\mathcal{O}|a_t} = \mathcal{C}_{\mathcal{O},\mathcal{A}}\mathcal{C}^{-1}_{\mathcal{A},\mathcal{A}}\phi^{\mathcal{A}}(a_t) \tag{5.21}$$

As with the Hilbert space predictive state, we actually want to calculate the embedding of the PDF of the observation conditioned on the current action *and* history h_t. Therefore we calculate the following conditional embeddings of covariance operators

$$\mathcal{C}_{\mathcal{O},\mathcal{A}|h_t} = \mathcal{C}_{(\mathcal{O},\mathcal{A})\mathcal{H}}\mathcal{C}^{-1}_{\mathcal{H},\mathcal{H}}\phi^{\mathcal{H}}(h_t) \tag{5.22}$$

$$\mathcal{C}_{\mathcal{A},\mathcal{A}|h_t} = \mathcal{C}_{(\mathcal{A},\mathcal{A})\mathcal{H}}\mathcal{C}^{-1}_{\mathcal{H},\mathcal{H}}\phi^{\mathcal{H}}(h_t) \tag{5.23}$$

The embedding of the conditional observation given an action and history is given by KBR:

$$\mu_{\mathcal{O}|h_t,a_t} = \mathcal{C}_{\mathcal{O},\mathcal{A}|h_t}\mathcal{C}^{-1}_{\mathcal{A},\mathcal{A}|h_t}\phi^{\mathcal{A}}(a_t) \tag{5.24}$$

5.2.2 The State Update

To implement the Bayes' rule state update we start with a PSR state at time t, then take an action, receive an observation, and incorporate this information into the PSR to get the state at time $t+1$. In what follows, we need to be careful about independence between different random variables. For example, if we evaluate $\phi^{\mathcal{T}^{\mathcal{O}}}(\tau^{\mathcal{O}}_t)$ and $\phi^{\mathcal{T}^{\mathcal{O}}}(\tau^{\mathcal{O}}_{t+1})$ at the same time step, the realizations will not be independent, even conditioned on the state of the process—if we wanted independent realizations of $\phi^{\mathcal{T}^{\mathcal{O}}}(\tau^{\mathcal{O}}_t)$ and $\phi^{\mathcal{T}^{\mathcal{O}}}(\tau^{\mathcal{O}}_{t+1})$, we'd have to assume the ability to reset the system to a desired state. Below we will take care that we only ask for operators which we can estimate without resets. (If we didn't have to obey this constraint, the algorithm would become somewhat simpler, but the need for resets would constrain applicability.) Therefore, in addition to the covariance operators $\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}},\mathcal{H}}$ and $\mathcal{C}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}},\mathcal{H}}$ defined above we define several additional covariance operators $\mathcal{C}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O},\mathcal{A},\mathcal{H}}$, $\mathcal{C}_{\mathcal{O},\mathcal{O},\mathcal{A},\mathcal{H}}$, and $\mathcal{C}_{\mathcal{A},\mathcal{A},\mathcal{H}}$, which are needed to update the state.

$$\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}},\mathcal{H}} \stackrel{\text{def}}{=} \mathbb{E}\left[\phi^{\mathcal{T}^{\mathcal{O}}}(\tau^{\mathcal{O}}_t) \otimes \phi^{\mathcal{T}^{\mathcal{A}}}(\tau^{\mathcal{A}}_t) \otimes \phi^{\mathcal{H}}(h_t)\right] \tag{5.25}$$

$$\mathcal{C}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}},\mathcal{H}} \stackrel{\text{def}}{=} \mathbb{E}\left[\phi^{\mathcal{T}^{\mathcal{A}}}(\tau^{\mathcal{A}}_t) \otimes \phi^{\mathcal{T}^{\mathcal{A}}}(\tau^{\mathcal{A}}_t) \otimes \phi^{\mathcal{H}}(h_t)\right] \tag{5.26}$$

$$\mathcal{C}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O},\mathcal{A},\mathcal{H}} \stackrel{\text{def}}{=} \mathbb{E}\left[\phi^{\mathcal{T}^{\mathcal{O}}}(\tau^{\mathcal{O}}_{t+1}) \otimes \phi^{\mathcal{T}^{\mathcal{A}}}(\tau^{\mathcal{A}}_{t+1}) \otimes \phi^{\mathcal{O}}(o_t) \otimes \phi^{\mathcal{A}}(a_t) \otimes \phi^{\mathcal{H}}(h_t)\right] \tag{5.27}$$

$$\mathcal{C}_{\mathcal{O},\mathcal{O},\mathcal{A},\mathcal{H}} \stackrel{\text{def}}{=} \mathbb{E}\left[\phi^{\mathcal{O}}(o_t) \otimes \phi^{\mathcal{O}}(o_t) \otimes \phi^{\mathcal{A}}(a_t) \otimes \phi^{\mathcal{H}}(h_t)\right] \tag{5.28}$$

$$\mathcal{C}_{\mathcal{A},\mathcal{A},\mathcal{H}} \stackrel{\text{def}}{=} \mathbb{E}\left[\phi^{\mathcal{A}}(a_t) \otimes \phi^{\mathcal{A}}(a_t) \otimes \phi^{\mathcal{H}}(h_t)\right] \tag{5.29}$$

Unlike Equations 5.18, 5.19, 5.22, and 5.23 we don't want to compute the conditional embeddings of covariance operators $\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t}$, $\mathcal{C}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|h_t}$, $\mathcal{C}_{\mathcal{O},\mathcal{A}|h_t}$, and $\mathcal{C}_{\mathcal{A},\mathcal{A}|h_t}$ directly from histories, but rather from the previous state. Since $\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t}$ is the characteristic embedding of the probability distribution of *all* tests, we assume that we can compute the embedding of the probability distribution of *any* subset of observations, actions, or tests with a conditional covariance operator from this embedding. For example,

$$\begin{aligned}
\mathcal{C}_{\mathcal{A},\mathcal{A}|h_t} &= \mathbb{E}[\phi^{\mathcal{A}}(a_t) \otimes \phi^{\mathcal{A}}(a_t) \mid h_t] \\
&= \mathbb{E}[\mathbb{E}[\phi^{\mathcal{A}}(a_t) \otimes \phi^{\mathcal{A}}(a_t) \mid \phi^{\mathcal{T}^{\mathcal{O}}}(\tau_t^{\mathcal{O}}) \otimes \phi^{\mathcal{T}^{\mathcal{A}}}(\tau_t^{\mathcal{A}})] \mid h_t] \\
&= \mathbb{E}\left[\mathcal{W}_{\mathcal{A}\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\left(\phi^{\mathcal{T}^{\mathcal{O}}}(\tau_t^{\mathcal{O}}) \otimes \phi^{\mathcal{T}^{\mathcal{A}}}(\tau_t^{\mathcal{A}})\right) \mid h_t\right] \\
&= \mathcal{W}_{\mathcal{A}\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathbb{E}[\phi^{\mathcal{T}^{\mathcal{O}}}(\tau_t^{\mathcal{O}}) \otimes \phi^{\mathcal{T}^{\mathcal{A}}}(\tau_t^{\mathcal{A}}) \mid h_t] \\
&= \mathcal{W}_{\mathcal{A}\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t}
\end{aligned}$$

Specifically, we assume that the operators $\mathcal{W}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}$, $\mathcal{W}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}$, $\mathcal{W}_{\mathcal{O},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}$, and $\mathcal{W}_{\mathcal{A},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}$ exist and that we can pseudo-invert $\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}}$ such that $\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} = I$. We see that we can then compute these operators from the covariances given in Equations 5.25–5.29:

$$\begin{aligned}
\mathcal{C}_{(\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O},\mathcal{A})\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} &= \mathcal{W}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O},\mathcal{A}|\mathcal{H}}\mathcal{C}_{\mathcal{H},\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} \\
&= \mathcal{W}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{W}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|\mathcal{H}}\mathcal{C}_{\mathcal{H},\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} \\
&= \mathcal{W}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} \\
&= \mathcal{W}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}} && (5.30) \\
\mathcal{C}_{(\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}})\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} &= \mathcal{W}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|\mathcal{H}}\mathcal{C}_{\mathcal{H},\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} \\
&= \mathcal{W}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{W}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|\mathcal{H}}\mathcal{C}_{\mathcal{H},\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} \\
&= \mathcal{W}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} \\
&= \mathcal{W}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}} && (5.31) \\
\mathcal{C}_{(\mathcal{O},\mathcal{O},\mathcal{A})\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} &= \mathcal{W}_{\mathcal{O},\mathcal{O},\mathcal{A}|\mathcal{H}}\mathcal{C}_{\mathcal{H},\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} \\
&= \mathcal{W}_{\mathcal{O},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{W}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|\mathcal{H}}\mathcal{C}_{\mathcal{H},\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} \\
&= \mathcal{W}_{\mathcal{O},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} \\
&= \mathcal{W}_{\mathcal{O},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}} && (5.32) \\
\mathcal{C}_{(\mathcal{A},\mathcal{A})\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} &= \mathcal{W}_{\mathcal{A},\mathcal{A}|\mathcal{H}}\mathcal{C}_{\mathcal{H},\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} \\
&= \mathcal{W}_{\mathcal{A},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}}\mathcal{C}_{\mathcal{H},\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} \\
&= \mathcal{W}_{\mathcal{A},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}}\mathcal{C}^{\dagger}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}} \\
&= \mathcal{W}_{\mathcal{A},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}} && (5.33)
\end{aligned}$$

Updating State with Bayes' Rule

Using the above equations, we can find covariance operators conditioned on history at time t, based on the covariance $\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t}$ at time t:

$$\mathcal{C}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O},\mathcal{A}|h_t} = \mathcal{W}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t} \tag{5.34}$$

$$\mathcal{C}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|h_t} = \mathcal{W}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t} \tag{5.35}$$

$$\mathcal{C}_{\mathcal{O},\mathcal{O},\mathcal{A}|h_t} = \mathcal{W}_{\mathcal{O},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t} \tag{5.36}$$

$$\mathcal{C}_{\mathcal{A},\mathcal{A}|h_t} = \mathcal{W}_{\mathcal{A},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t} \tag{5.37}$$

Finally, in order to update our state, we execute two instances of KBR. First, when we choose an action a_t, we update:

$$\mathcal{C}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O}|h_t,a_t} = \mathcal{C}_{(\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O})\mathcal{A}|h_t}\mathcal{C}^{-1}_{\mathcal{A},\mathcal{A}|h_t}\phi^{\mathcal{A}}(a_t) \tag{5.38}$$

$$\mathcal{C}_{\mathcal{O},\mathcal{O}|h_t,a_t} = \mathcal{C}_{(\mathcal{O},\mathcal{O})\mathcal{A}|h_t}\mathcal{C}^{-1}_{\mathcal{A},\mathcal{A}|h_t}\phi^{\mathcal{A}}(a_t) \tag{5.39}$$

Next, when we receive the observation generated by the system, o_t, we incorporate it to calculate the joint conditional covariance:

$$\mathcal{C}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+}|h_t,a_t,o_t} = \mathcal{C}_{(\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+})\mathcal{O}|h_t,a_t}\mathcal{C}^{-1}_{\mathcal{O},\mathcal{O}|h_t,a_t}\phi^{\mathcal{O}}(o_t) \tag{5.40}$$

Finally, the joint conditional covariance at time $t+1$ is identified as $\mathcal{C}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+}|h_t,a_t,o_t}$:

$$\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_{t+1}} \equiv \mathcal{C}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+}|h_t,a_t,o_t} \tag{5.41}$$

The PSR state can now be computed from Equation 5.35 and Equation 5.20:

$$\mathcal{C}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|h_{t+1}} = \mathcal{W}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_{t+1}} \tag{5.42}$$

$$\mathcal{W}_{\mathcal{T}^{\mathcal{O}}|\mathcal{T}^{\mathcal{A}},h_{t+1}} = \mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_{t+1}}\mathcal{C}^{-1}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|h_{t+1}} \tag{5.43}$$

5.2.3 A Minimal State Space

Instead of working with mean embeddings in generic RKHSs, it is sometimes possible to embed distributions in a finite-dimensional *subspace* of the RKHS. For discrete action-observation PSRs with delta kernels, such a subspace corresponds to a *core set* of tests (see Chapter 3 for details). In the more general case, we can *factor* the conditional embedding of the covariance operator $\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t}$ into a finite dimensional operator $\mathcal{C}_{X^{\mathcal{O}},X^{\mathcal{A}}|h_t}$ and a conditional covariance operator $\mathcal{U}$. Here $X = (X^{\mathcal{O}}, X^{\mathcal{A}})$ is a finite set of linear combinations of tests, which we choose to make the factorization possible.

$$\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t} = \mathcal{U}\mathcal{C}_{X^{\mathcal{O}},X^{\mathcal{A}}|h_t} \tag{5.44}$$

Analogous to the discrete case we can find $\mathcal{U}$ by performing a 'thin' SVD of the covariance operator $\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}}$ and taking the top d singular vectors as $\mathcal{U}$. (So we are choosing X so that $\mathcal{U}^*\mathcal{U} = I$.) Instead of using the infinite-dimensional operators $\mathcal{W}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}$, $\mathcal{W}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}$,

$\mathcal{W}_{\mathcal{O},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}$, and $\mathcal{W}_{\mathcal{A},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}$ in conjunction with KBR to update state, we use finite dimensional operators $\mathcal{W}_{X^{\mathcal{O}+},X^{\mathcal{A}+},\mathcal{O},\mathcal{A}|X^{\mathcal{O}},X^{\mathcal{A}}}$, $\mathcal{W}_{X^{\mathcal{A}},X^{\mathcal{A}}|X^{\mathcal{O}},X^{\mathcal{A}}}$, $\mathcal{W}_{\mathcal{O},\mathcal{O},\mathcal{A}|X^{\mathcal{O}},X^{\mathcal{A}}}$, and $\mathcal{W}_{\mathcal{A},\mathcal{A}|X^{\mathcal{O}},X^{\mathcal{A}}}$, which we can find as follows:

$$\begin{aligned}
\mathcal{U}^*\mathcal{C}_{(\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O},\mathcal{A})\mathcal{H}}(\mathcal{U}^*\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}})^\dagger &= \mathcal{U}^*\mathcal{W}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}}(\mathcal{U}^*\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}})^\dagger \\
&= \mathcal{U}^*\mathcal{W}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{U}\mathcal{C}_{(X^{\mathcal{O}},X^{\mathcal{A}})\mathcal{H}}(\mathcal{C}_{(X^{\mathcal{O}},X^{\mathcal{A}})\mathcal{H}})^\dagger \\
&= \mathcal{U}^*\mathcal{W}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{U} \\
&= \mathcal{W}_{X^{\mathcal{O}+},X^{\mathcal{A}+},\mathcal{O},\mathcal{A}|X^{\mathcal{O}},X^{\mathcal{A}}} \qquad (5.45)
\end{aligned}$$

The last line follows from the fact that

$$\begin{aligned}
\mathcal{U}^*\mathcal{C}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+},\mathcal{O},\mathcal{A},\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{C}^{-1}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{U} &= \mathcal{W}_{X^{\mathcal{O}+},X^{\mathcal{A}+},\mathcal{O},\mathcal{A},X^{\mathcal{O}},X^{\mathcal{A}}}\mathcal{U}^*(\mathcal{U}\mathcal{C}_{X^{\mathcal{O}},X^{\mathcal{A}}}\mathcal{U}^*)^{-1}\mathcal{U} \\
&= \mathcal{W}_{X^{\mathcal{O}+},X^{\mathcal{A}+},\mathcal{O},\mathcal{A},X^{\mathcal{O}},X^{\mathcal{A}}}\mathcal{C}^{-1}_{X^{\mathcal{O}},X^{\mathcal{A}}} \\
&= \mathcal{W}_{X^{\mathcal{O}+},X^{\mathcal{A}+},\mathcal{O},\mathcal{A}|X^{\mathcal{O}},X^{\mathcal{A}}}
\end{aligned}$$

Continuing, we see that:

$$\begin{aligned}
\mathcal{C}_{(\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}})\mathcal{H}}(\mathcal{U}^*\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}})^\dagger &= \mathcal{W}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}}(\mathcal{U}^*\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}})^\dagger \\
&= \mathcal{W}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{U}\mathcal{C}_{(X^{\mathcal{O}},X^{\mathcal{A}})\mathcal{H}}(\mathcal{C}_{(X^{\mathcal{O}},X^{\mathcal{A}})\mathcal{H}})^\dagger \\
&= \mathcal{W}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{U} \\
&= \mathcal{W}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|X^{\mathcal{O}},X^{\mathcal{A}}} \qquad (5.46) \\
\mathcal{C}_{(\mathcal{O},\mathcal{O},\mathcal{A})\mathcal{H}}(\mathcal{U}^*\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}})^\dagger &= \mathcal{W}_{\mathcal{O},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}}(\mathcal{U}^*\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}})^\dagger \\
&= \mathcal{W}_{\mathcal{O},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{U}\mathcal{C}_{(X^{\mathcal{O}},X^{\mathcal{A}})\mathcal{H}}(\mathcal{C}_{(X^{\mathcal{O}},X^{\mathcal{A}})\mathcal{H}})^\dagger \\
&= \mathcal{W}_{\mathcal{O},\mathcal{O},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{U} \\
&= \mathcal{W}_{\mathcal{O},\mathcal{O},\mathcal{A}|X^{\mathcal{O}},X^{\mathcal{A}}} \qquad (5.47) \\
\mathcal{C}_{(\mathcal{A},\mathcal{A})\mathcal{H}}(\mathcal{U}^*\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}})^\dagger &= \mathcal{W}_{\mathcal{A},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}}(\mathcal{U}^*\mathcal{C}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}})\mathcal{H}})^\dagger \\
&= \mathcal{W}_{\mathcal{A},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{U}\mathcal{C}_{(X^{\mathcal{O}},X^{\mathcal{A}})\mathcal{H}}(\mathcal{C}_{(X^{\mathcal{O}},X^{\mathcal{A}})\mathcal{H}})^\dagger \\
&= \mathcal{W}_{\mathcal{A},\mathcal{A}|\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}}\mathcal{U} \\
&= \mathcal{W}_{\mathcal{A},\mathcal{A}|X^{\mathcal{O}},X^{\mathcal{A}}} \qquad (5.48)
\end{aligned}$$

Similar to Equations 5.34–5.37 we can find covariance operators conditioned on history at time t:

$$\mathcal{C}_{X^{\mathcal{O}+},X^{\mathcal{A}+},\mathcal{O},\mathcal{A}|h_t} = \mathcal{W}_{X^{\mathcal{O}+},X^{\mathcal{A}+},\mathcal{O},\mathcal{A}|X^{\mathcal{O}},X^{\mathcal{A}}}\mathcal{C}_{X^{\mathcal{O}},X^{\mathcal{A}}|h_t} \qquad (5.49)$$

$$\mathcal{C}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|h_t} = \mathcal{W}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|X^{\mathcal{O}},X^{\mathcal{A}}}\mathcal{C}_{X^{\mathcal{O}},X^{\mathcal{A}}|h_t} \qquad (5.50)$$

$$\mathcal{C}_{\mathcal{O},\mathcal{O},\mathcal{A}|h_t} = \mathcal{W}_{\mathcal{O},\mathcal{O},\mathcal{A}|X^{\mathcal{O}},X^{\mathcal{A}}}\mathcal{C}_{X^{\mathcal{O}},X^{\mathcal{A}}|h_t} \qquad (5.51)$$

$$\mathcal{C}_{\mathcal{A},\mathcal{A}|h_t} = \mathcal{W}_{\mathcal{A},\mathcal{A}|X^{\mathcal{O}},X^{\mathcal{A}}}\mathcal{C}_{X^{\mathcal{O}},X^{\mathcal{A}}|h_t} \qquad (5.52)$$

Finally, we update our state with action a_t and observation o_t by applying KBR twice:

$$\mathcal{C}_{X^{\mathcal{O}+},X^{\mathcal{A}+},\mathcal{O}|h_t,a_t} = \mathcal{C}_{(X^{\mathcal{O}+},X^{\mathcal{A}+},\mathcal{O})\mathcal{A}|h_t}\mathcal{C}^{-1}_{\mathcal{A},\mathcal{A}|h_t}\phi^{\mathcal{A}}(a_t) \qquad (5.53)$$

$$\mathcal{C}_{\mathcal{O},\mathcal{O}|h_t,a_t} = \mathcal{C}_{(\mathcal{O},\mathcal{O})\mathcal{A}|h_t}\mathcal{C}^{-1}_{\mathcal{A},\mathcal{A}|h_t}\phi^{\mathcal{A}}(a_t) \qquad (5.54)$$

$$\mathcal{C}_{X^{\mathcal{O}+},X^{\mathcal{A}+}|h_t,a_t,o_t} = \mathcal{C}_{(X^{\mathcal{O}+},X^{\mathcal{A}+})\mathcal{O}|h_t,a_t}\mathcal{C}^{-1}_{\mathcal{O},\mathcal{O}|h_t,a_t}\phi^{\mathcal{O}}(o_t) \qquad (5.55)$$

The PSR state can now be computed:

$$\mathcal{C}_{X^{\mathcal{O}},X^{\mathcal{A}}|h_{t+1}} \equiv \mathcal{C}_{X^{\mathcal{O}+},X^{\mathcal{A}+}|h_t,a_t,o_t} \tag{5.56}$$

$$\mathcal{C}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|h_{t+1}} = \mathcal{W}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|X^{\mathcal{O}},X^{\mathcal{A}}}\mathcal{C}_{X^{\mathcal{O}},X^{\mathcal{A}}|h_{t+1}} \tag{5.57}$$

$$\mathcal{W}_{\mathcal{T}^{\mathcal{O}}|\mathcal{T}^{\mathcal{A}},h_{t+1}} = \mathcal{U}\mathcal{C}_{X^{\mathcal{O}},X^{\mathcal{A}}|h_{t+1}}\mathcal{C}^{-1}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|h_{t+1}} \tag{5.58}$$

5.3 Kernel Learning Algorithms for PSRs

The equations in Section 5.2 allow us to estimate the appropriate covariance operators and update the Hilbert space embedding of the PSR state. If the RKHS embeddings are *finite*, then the learning algorithm and state update are *the same* as Chapter 4. However, if the RKHS is *infinite*, which is often the case, then it is not possible to store or manipulate the above covariances directly. Instead, we use the "kernel trick" and represent all of the covariances and compute state updates with Gram matrices.

Given T *i.i.d.* tuples $\left\{(\tau_t^{\mathcal{O}}, \tau_t^{\mathcal{A}}, o_t, a_t, h_t)\right\}_{t=1}^{T}$ from a PSR, we denote:

$$\begin{aligned}
\Upsilon^{\mathcal{T}^{\mathcal{O}}} &= \left(\phi^{\mathcal{T}^{\mathcal{O}}}(\tau_1^{\mathcal{O}}), \ldots, \phi^{\mathcal{T}^{\mathcal{O}}}(\tau_{T-1}^{\mathcal{O}})\right) \\
\Upsilon^{\mathcal{T}^{\mathcal{A}}} &= \left(\phi^{\mathcal{T}^{\mathcal{A}}}(\tau_1^{\mathcal{A}}), \ldots, \phi^{\mathcal{T}^{\mathcal{A}}}(\tau_{T-1}^{\mathcal{A}})\right) \\
\Upsilon^{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}} &= \left(\phi^{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}(\tau_1^{\mathcal{O}}, \tau_1^{\mathcal{A}}), \ldots, \phi^{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}(\tau_{T-1}^{\mathcal{O}}, \tau_{T-1}^{\mathcal{A}})\right) \\
\Upsilon^{\mathcal{T}^{\mathcal{O}+}} &= \left(\phi^{\mathcal{T}^{\mathcal{O}}}(\tau_2^{\mathcal{O}}), \ldots, \phi^{\mathcal{T}^{\mathcal{O}}}(\tau_{T}^{\mathcal{O}})\right) \\
\Upsilon^{\mathcal{T}^{\mathcal{A}+}} &= \left(\phi^{\mathcal{T}^{\mathcal{A}}}(\tau_2^{\mathcal{A}}), \ldots, \phi^{\mathcal{T}^{\mathcal{A}}}(\tau_{T}^{\mathcal{A}})\right) \\
\Upsilon^{\mathcal{T}^{\mathcal{O}+}\otimes\mathcal{T}^{\mathcal{A}+}} &= \left(\phi^{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}(\tau_2^{\mathcal{O}}, \tau_2^{\mathcal{A}}), \ldots, \phi^{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}(\tau_{T}^{\mathcal{O}}, \tau_{T}^{\mathcal{A}})\right) \\
\Upsilon^{\mathcal{O}} &= \left(\phi^{\mathcal{O}}(o_1), \ldots, \phi^{\mathcal{O}}(o_{T-1})\right) \\
\Upsilon^{\mathcal{A}} &= \left(\phi^{\mathcal{A}}(a_1), \ldots, \phi^{\mathcal{A}}(a_{T-1})\right) \\
\Upsilon^{\mathcal{H}} &= \left(\phi^{\mathcal{H}}(h_1), \ldots, \phi^{\mathcal{H}}(h_{T-1})\right)
\end{aligned} \tag{5.59}$$

The learning algorithm for HSE-PSRs proceeds to estimate covariance operators by Gram matrices. The following Gram matrices are used below:

$$G_{\mathcal{H},\mathcal{H}} = \Upsilon^{\mathcal{H}*}\Upsilon^{\mathcal{H}} \tag{5.60}$$

$$G_{\mathcal{A},\mathcal{A}} = \Upsilon^{\mathcal{A}*}\Upsilon^{\mathcal{A}} \tag{5.61}$$

$$G_{\mathcal{O},\mathcal{O}} = \Upsilon^{\mathcal{O}*}\Upsilon^{\mathcal{O}} \tag{5.62}$$

$$G_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}} = \Upsilon^{\mathcal{T}^{\mathcal{A}}*}\Upsilon^{\mathcal{T}^{\mathcal{A}}} \tag{5.63}$$

$$G_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{O}}} = \Upsilon^{\mathcal{T}^{\mathcal{O}}*}\Upsilon^{\mathcal{T}^{\mathcal{O}}} \tag{5.64}$$

$$G_{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}} = G_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{O}}} \circ G_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}} \tag{5.65}$$

$$G_{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{O}+}\otimes\mathcal{T}^{\mathcal{A}+}} = G_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{O}+}} \circ G_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}+}} \tag{5.66}$$

where $\circ$ is the Hadamard (element-wise matrix) product.

5.3.1 The Gram Matrix Formulation for Hilbert Space Embeddings of PSRs

The PSR state is encoded as a set of weights α_{h_t} on samples. In order to recover the predictive state, we need to compute the conditional covariance $\mathcal{W}_{\mathcal{T}^{\mathcal{O}}|\mathcal{T}^{\mathcal{A}},h_t}$, which involves first computing $\mathcal{C}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|h_t}$ and $\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t}$.

$$\mathcal{C}_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}|h_t} = \Upsilon^{\mathcal{T}^{\mathcal{A}}}\text{diag}(\alpha_{h_t})\Upsilon^{\mathcal{T}^{\mathcal{A}}*} \tag{5.67}$$

$$\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t} = \Upsilon^{\mathcal{T}^{\mathcal{O}}}\text{diag}(\alpha_{h_t})\Upsilon^{\mathcal{T}^{\mathcal{A}}*} \tag{5.68}$$

Let Λ_{h_t} be a diagonal matrix with the set of weights α_{h_t} along the diagonal: $\Lambda_{h_t} = \text{diag}\left(\alpha_{h_t}\right)$. To write the predictive state at time t in terms of Gram matrices we apply Equation 5.10:

$$\mathcal{W}_{\mathcal{T}^{\mathcal{O}}|\mathcal{T}^{\mathcal{A}},h_t} = \Upsilon^{\mathcal{T}^{\mathcal{O}}}\Lambda_{h_t}G_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}}((\Lambda_{h_t}G_{\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{A}}})^2 + \lambda TI)^{-1}\Lambda_{h_t}\Upsilon^{\mathcal{T}^{\mathcal{A}}*} \tag{5.69}$$

5.3.2 Gram Matrix State Updates

Given the Gram matrix representation of state, we recursively apply kernel Bayes' rule to *update* state given a new action and observation. We start with a set of weights α_{h_t} on samples and the corresponding diagonal matrix Λ_{h_t}. After choosing action a_t, we condition to find another set of weights (again via Equation 5.10):

$$\alpha_{a_t,h_t} = \Lambda_{h_t}G_{\mathcal{A},\mathcal{A}}\left((\Lambda_{h_t}G_{\mathcal{A},\mathcal{A}})^2 + \lambda TI\right)^{-1}\Lambda_{h_t}\Upsilon^{\mathcal{A}^*}\phi^{\mathcal{A}}(a_t) \tag{5.70}$$

Equation 5.70 is the Gram matrix analogue of Equation 5.38. This vector can be used to find an estimate of the conditional embedding $\mathcal{C}_{\mathcal{O}|h_t,a_t}$ by weighting the samples $\Upsilon^{\mathcal{O}}$:

$$\mathcal{C}_{\mathcal{O}|h_t,a_t} = \Upsilon^{\mathcal{O}}\alpha_{h_t,a_t} \tag{5.71}$$

Given a new observation o_t, we apply KBR again

$$\alpha_{h_t,a_t,o_t} = \Lambda_{h_t,a_t}G_{\mathcal{O},\mathcal{O}}((\Lambda_{h_t,a_t}G_{\mathcal{O},\mathcal{O}})^2 + \lambda TI)^{-1}\Lambda_{h_t,a_t}\Upsilon^{\mathcal{O}^*}\phi^{\mathcal{O}}(o_t) \tag{5.72}$$

Equation 5.72 is the Gram matrix analogue of Equation 5.40. Given the coefficients $\hat{\alpha}_{h_{t+1}}$, we can calculate the embedding of the conditional joint probability of future observation sequences and future action sequences as

$$\mathcal{C}_{\mathcal{T}^{\mathcal{O}+},\mathcal{T}^{\mathcal{A}+}|h_t,a_t,o_t} = \Upsilon^{\mathcal{T}^{\mathcal{O}+}\otimes\mathcal{T}^{\mathcal{A}+}}\alpha_{h_t,a_t,o_t} \tag{5.73}$$

To finish updating the state we map these coefficients on samples $\Upsilon^{\mathcal{T}^{\mathcal{O}+}\otimes\mathcal{T}^{\mathcal{A}+}}$ to coefficients on samples $\Upsilon^{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}$:

$$\alpha_{h_{t+1}} = G_{\mathcal{H},\mathcal{H}}(G_{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}G_{\mathcal{H},\mathcal{H}} + \lambda TI)^{-1}G_{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{O}+}\otimes\mathcal{T}^{\mathcal{A}+}}\alpha_{h_t,a_t,o_t} \tag{5.74}$$

Equation 5.74 can be viewed as the *first* step of our recursive filtering algorithm, that is, we are taking the state at time $t+1$ and applying Equation 5.9 to begin the process of finding the conditional embedding weights $\alpha_{a_{t+1},h_{t+1}}$ etc. Equation 5.70. To continue updating the state, we recursively apply Equations 5.70–5.74.

5.3.3 A Spectral Learning Algorithm

The kernel spectral algorithm for PSRs proceeds by first performing a 'thin' SVD of the sample covariance $\widehat{\mathcal{C}}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}),\mathcal{H}} = \frac{1}{T}\Upsilon^{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}\Upsilon^{\mathcal{H}*}$. Then the left singular vector $v = \Upsilon^{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}\alpha$ ($\alpha \in \mathbb{R}^T$) can be estimated as follows

$$\begin{aligned}
&\widehat{\mathcal{C}}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}),\mathcal{H}}\widehat{\mathcal{C}}_{(\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}),\mathcal{H}*}v = \omega v \\
\Leftrightarrow\ &\Upsilon^{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}G_{\mathcal{H},\mathcal{H}}\Upsilon^{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}*}v = \omega v \\
\Leftrightarrow\ &\Upsilon^{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}G_{\mathcal{H},\mathcal{H}}G_{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}\alpha = \omega\Upsilon^{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}\alpha \\
\Leftrightarrow\ &G_{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}G_{\mathcal{H},\mathcal{H}}G_{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}\alpha = \omega G_{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}\alpha,\ (\alpha \in \mathbb{R}^T,\ \omega \in \mathbb{R})
\end{aligned} \tag{5.75}$$

where α is the generalized eigenvector. After normalization, we have

$$v = \frac{1}{\sqrt{\alpha^\top G_{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}\alpha}}\Upsilon^{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}\alpha \tag{5.76}$$

Then the $\mathcal{U}$ operator is the column concatenation of the d top left singular vectors, *i.e.* $\hat{\mathcal{U}} = (v_1, \ldots, v_d)$. If we let $A \stackrel{\text{def}}{=} (\alpha_1, \ldots, \alpha_d) \in \mathbb{R}^{T\times d}$ be the column concatenation of the d top α_i, and $D \stackrel{\text{def}}{=} \text{diag}\left((\alpha_1^\top G_{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}\alpha_1)^{-1/2}, \ldots, (\alpha_d^\top G_{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}\alpha_d)^{-1/2}\right) \in \mathbb{R}^{d\times d}$, we can concisely express $\hat{\mathcal{U}} = \Upsilon^{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}AD$. Therefore, we can compute the low-dimensional embedding:

$$\begin{aligned}
\mathcal{C}_{X^{\mathcal{O}},X^{\mathcal{A}}|h_t} &= \mathcal{U}^*\mathcal{C}_{\mathcal{T}^{\mathcal{O}},\mathcal{T}^{\mathcal{A}}|h_t} \\
&= \mathcal{U}^*\Upsilon^{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}\alpha_{h_t} \\
&= D^\top A^\top G_{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}\alpha_{h_t}
\end{aligned} \tag{5.77}$$

Again, let $\Lambda_{h_t} = \text{diag}\,(\alpha_{h_t})$. Updating state given a new action and observation proceeds in exactly the same way as the non-minimal case:

$$\begin{aligned}
\alpha_{a_t,h_t} &= \Lambda_{h_t}G_{\mathcal{A},\mathcal{A}}\left((\Lambda_{h_t}G_{\mathcal{A},\mathcal{A}})^2 + \lambda TI\right)^{-1}\Lambda_{h_t}\Upsilon^{\mathcal{A}*}\phi^{\mathcal{A}}(a_t) \\
\Lambda_{h_t,a_t} &= \text{diag}(\alpha_{a_t,h_t}) \\
\hat{\alpha}_{h_{t+1}} &= \alpha_{h_t,a_t,o_t} = \Lambda_{h_t,a_t}G_{\mathcal{O},\mathcal{O}}((\Lambda_{h_t,a_t}G_{\mathcal{O},\mathcal{O}})^2 + \lambda TI)^{-1}\Lambda_{h_t,a_t}\Upsilon^{\mathcal{O}*}\phi^{\mathcal{O}}(o_t)
\end{aligned}$$

To finish updating the state

$$\begin{aligned}
&\alpha_{h_t+1} \\
&= G_{\mathcal{H},\mathcal{H}}G_{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}AD(D^\top A^\top G_{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}G_{\mathcal{H},\mathcal{H}}G_{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}}}AD)^{-1}D^\top A^\top G_{\mathcal{T}^{\mathcal{O}}\otimes\mathcal{T}^{\mathcal{A}},\mathcal{T}^{\mathcal{O}+}\otimes\mathcal{T}^{\mathcal{A}+}}\hat{\alpha}_{h_{t+1}}
\end{aligned} \tag{5.78}$$

Although we have not supplied any sample complexity results for this algorithm yet, we believe that we can place bounds on each individual empirically estimated covariance operator, and then chain those bounds together to get a bound on the result: a similar approach was used to obtain a bound on a previous version of this algorithm [99].

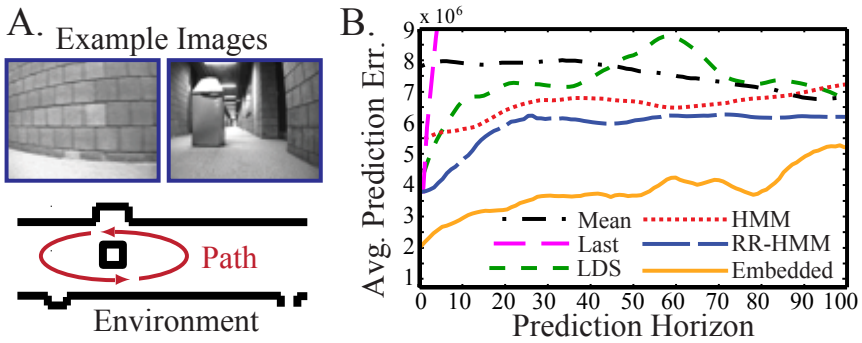

Figure 5.1: Robot vision data. (A) Sample images from the robot's camera. The figure below depicts the hallway environment with a central obstacle (black) and the path that the robot took through the environment (the red counter-clockwise ellipse). (B) Squared error for prediction with different estimated models and baselines.

5.4 Experimental Results

For our experimental results, we implemented a slight variation on the algorithm described above for a special subclass of HSE-PSRs called Hilbert Space Embeddings of Hidden Markov Models (HSE-HMMs). HSE-HMMs are something of a misnomer: the model is a "hidden Markov model" with a potentially infinite number of states, it is therefore equivalent to an Observable Operator Model or an uncontrolled PSR. As such we can just use the learning algorithm described above, but ignore actions. This makes the math and learning considerably easier in practice.

In the experimental results described here, we used an older variation of this algorithm with some differences in the observation update [99]. In particular, Song et al. [99] update the embedding of the belief state of the HMM given a new observation o_t with an estimated operator $\mathcal{B}_{o_t}$, which is a conditional mean update in RKHS multiplied by a conditional density estimate (which must be estimated separately). This is a less direct option than performing the update completely in the embedding space. We hypothesize that using kernel Bayes' rule, as described in the previous section will is not just more direct, but should work better in practice. We will directly compare the two approaches in future work.

We designed 3 sets of experiments to evaluate the effectiveness of learning embedded HMMs for difficult real-world filtering and prediction tasks. In each case we compare the learned embedded HMM to several alternative time series models including (I) linear dynamical systems (LDS) learned by spectral methods (Chapter 2) with stability constraints (Chapter 7), (II) discrete HMMs learned by EM, and (III) the Reduced-rank HMM (RR-HMM) learned by spectral methods [90]. In these experiments we demonstrate that the kernel spectral learning algorithm for embedded HMMs achieves the state-of-the-art performance.

5.4.1 Robot Vision

In this experiment, a video of 2000 frames was collected at 6 Hz from a Point Grey Bumblebee2 stereo camera mounted on a Botrics Obot d100 mobile robot platform circling a stationary obstacle (under imperfect human control) (Figure 5.1(A)) and 1500 frames were used as training

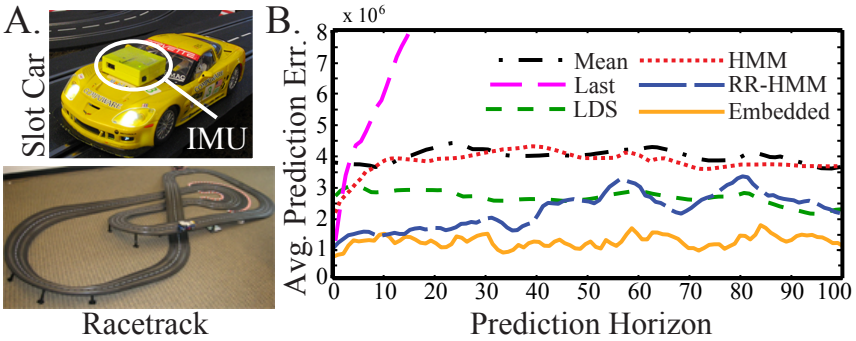

Figure 5.2: Slot car inertial measurement data. (A) The slot car platform and the IMU (top) and the racetrack (bottom). (B) Squared error for prediction with different estimated models and baselines.

data for each model. Each frame from the training data was reduced to 100 dimensions via SVD on single observations. The goal of this experiment was to learn a model of the noisy video, and, after filtering, to predict future image observations.

We trained a 50-dimensional embedded HMM with tests consisting of sequences of 20 consecutive observations. Gaussian RBF kernels are used and the bandwidth parameter is set with the median of squared distance between training points (median trick). The regularization parameter λ is set of 10^{-4}. For comparison, a 50-dimensional RR-HMM with Parzen windows is also learned with sequences of 20 observations [90]; a 50-dimensional LDS is learned using Subspace ID with Hankel matrices of 20 time steps; and finally a 50-state discrete HMM and axis-aligned Gaussian observation models is learned using EM algorithm run until convergence.

For each model, we performed filtering[3] for different extents $t_1 = 100, 101, \ldots, 250$, then predicted an image which was a further t_2 steps in the future, for $t_2 = 1, 2..., 100$. The squared error of this prediction in pixel space was recorded, and averaged over all the different filtering extents t_1 to obtain means which are plotted in Figure 5.1(B). As baselines, we also plot the error obtained by using the mean of filtered data as a predictor (Mean), and the error obtained by using the last filtered observation (Last).

Any of the more complex algorithms perform better than the baselines (though as expected, the 'Last' predictor is a good one-step predictor), indicating that this is a nontrivial prediction problem. The embedded HMM learned by the kernel spectral algorithm yields significantly lower prediction error compared to each of the alternatives (including the RR-HMM) consistently for the duration of the prediction horizon (100 timesteps, *i.e.* 16 seconds).

5.4.2 Slot Car Inertial Measurement

In a second experiment, the setup consisted of a track and a miniature car (1:32 scale model) guided by a slot cut into the track. Figure 6.2(A) shows the car and the attached IMU (an Intel Inertiadot) in the upper panel, and the 14m track which contains elevation changes and banked curves. At each time step we extracted the estimated 3-D acceleration of the car and the

[3]Update models online with incoming observations.

estimated difference between the 3-D orientation of the car from the previous time step at a rate of 10Hz. We collected 3000 successive measurements of this data while the slot car circled the track controlled by a constant policy. The goal was to learn a model of the noisy IMU data, and, after filtering, to predict future readings.

We trained a 20-dimensional embedded HMM with tests consisting of sequences of 150 consecutive observations. The bandwidth parameter of the Gaussian RBF kernels is set with 'median trick'. The regularization parameter λ is 10^{-4}. For comparison, a 20-dimensional RR-HMM with Parzen windows is learned also with sequences of 150 observations; a 20-dimensional LDS is learned using Subspace ID with Hankel matrices of 150 time steps; and finally, a 20-state discrete HMM (with 400 level of discretization for observations) is learned using EM algorithm.

For each model, we performed filtering for different extents $t_1 = 100, 101, \ldots, 250$, then predicted an image which was a further t_2 steps in the future, for $t_2 = 1, 2..., 100$. The squared error of this prediction in the IMU's measurement space was recorded, and averaged over all the different filtering extents t_1 to obtain means which are plotted in Figure 6.2(B). Again the embedded HMM yields lower prediction error compared to each of the alternatives consistently for the duration of the prediction horizon.

5.4.3 Audio Event Classification

Our final experiment concerns an audio classification task. The data, recently presented in [81], consisted of sequences of 13-dimensional Mel-Frequency Cepstral Coefficients (MFCC) obtained from short clips of raw audio data recorded using a portable sensor device. Six classes of labeled audio clips were present in the data, one being Human speech. For this experiment we grouped the latter five classes into a single class of Non-human sounds to formulate a binary Human vs. Non-human classification task. Since the original data had a disproportionately large amount of Human Speech samples, this grouping resulted in a more balanced dataset with 40 minutes 11 seconds of Human and 28 minutes 43 seconds of Non-human audio data. To reduce noise and training time we averaged the data every 100 timesteps (equivalent to 1 second).

For each of the two classes, we trained embedded HMMs with $10, 20, \ldots, 50$ latent dimensions using spectral learning and Gaussian RBF kernels with bandwidth set with the 'median trick'. The regularization parameter λ is 10^{-1}. For comparison, regular HMMs with axis-aligned Gaussian observation models, LDSs and RR-HMMs were trained using multi-restart EM (to avoid local minima), stable Subspace ID and the spectral algorithm of [90] respectively, also with $10, \ldots, 50$ latent dimensions.

For RR-HMMs, regular HMMs and LDSs, the class-conditional data sequence likelihood is the scoring function for classification. For embedded HMMs, the scoring function for a test sequence $x_{1:t}$ is the log of the product of the compatibility scores for each observation, *i.e.* $\sum_{\tau=1}^{t} \log\left(\left\langle \varphi(x_\tau), \hat{\mu}_{X_\tau|x_{1:\tau-1}} \right\rangle_{\mathcal{F}}\right)$.

For each model size, we performed 50 random 2:1 partitions of data from each class and used the resulting datasets for training and testing respectively. The mean accuracy and 95% confidence intervals over these 50 randomizations are reported in Figure 5.3. The graph indicates that embedded HMMs have higher accuracy and lower variance than other standard alternatives at every model size. Though other learning algorithms for HMMs and LDSs exist, our experiment shows this to be a non-trivial sequence classification problem where embedded HMMs signif-

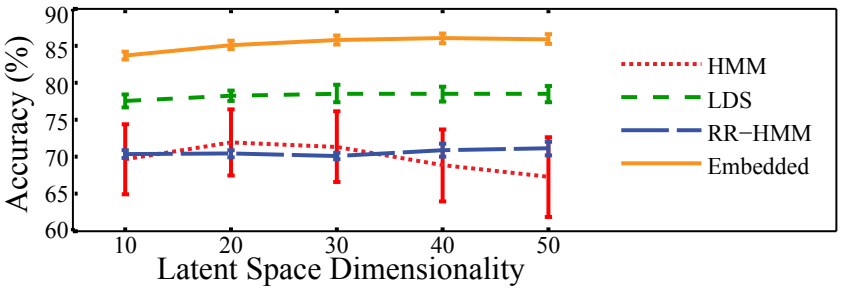

Figure 5.3: Accuracies and 95% confidence intervals for Human vs. Non-human audio event classification, comparing embedded HMMs to other common sequential models at different latent state space sizes.

icantly outperform commonly used sequential models trained using typical learning and model selection methods.

5.5 Conclusion

In this chapter we proposed an extension of the feature-based observable representation of PSRs presented in Chapter 4. We extended the finite features of that chapter to Hilbert spaces, resulting in Hilbert space embeddings of PSRs. The essence of this new approach is to represent distributions over tests as elements in Hilbert spaces, and update these elements entirely in the Hilbert spaces using kernel Bayes' rule. This allows us to derive a local-minimum-free kernel spectral algorithm for learning the embedded PSRs. In our experimental results we show that a variation of this algorithm [99] exceeds previous state-of-the-art in real world challenging problems.

We briefly note that it is possible to extend the algorithms presented in this chapter to a *manifold dynamical system learning algorithm*. If we are interested in modeling a dynamical system whose state space lies on a low-dimensional manifold, then we can use this additional knowledge to constrain the learning algorithm and produce a more accurate model for a given amount of training data. For details see [13].

Chapter 6

Computational Efficiency in Spectral Learning Algorithms

6.1 Introduction

In the previous chapters, we have described several novel spectral learning algorithms that can be used to learn models of partially observable nonlinear dynamical systems such as HMMs [42, 90] and PSRs [12, 14, 84]. These algorithms are *statistically consistent*, unlike the popular expectation maximization (EM) algorithm, which is subject to local optima. Furthermore, we have seen that spectral learning algorithms are easy to implement with a series of linear algebra operations. Despite these attractive features, these algorithms have so far had an important drawback: they are *batch* methods (needing to store their entire training data set in memory at once) instead of *online* ones (with space complexity independent of the number of training examples and time complexity linear in the number of training examples).

To remedy this drawback, we propose a fast, online spectral algorithm for PSRs. Since PSRs *subsume* HMMs and POMDPs [84, 93], the algorithm described in this chapter also improves on past algorithms for these other models. Our method leverages fast, low-rank modifications of the thin singular value decomposition [20], and uses tricks such as random projections to scale to extremely large numbers of examples and features per example. Consequently, the new method can handle orders of magnitude larger data sets than previous methods, and can therefore scale to learn models of systems that are too complex for previous methods.

Experiments show that our online spectral learning algorithm does a good job recovering the parameters of a nonlinear dynamical system in two partially observable domains. In our first experiment we empirically demonstrate that our online spectral learning algorithm is unbiased by recovering the parameters of a small but difficult synthetic Reduced-Rank HMM. In our second experiment we demonstrate the performance of the new method on a difficult, high-bandwidth video understanding task.

6.2 Batch Learning of PSRs

In Chapter 4, we presented a straightforward learning algorithm for PSRs: we build empirical estimates of observable features $\widehat{\Sigma}_{\mathcal{AO},\mathcal{H},\mathcal{AO}}$, $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$, and $\widehat{\Sigma}_{\mathcal{T}+,\mathcal{AO},\mathcal{H}}$ and compute $\widehat{U}$ as the matrix of d leading left singular vectors of $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$. Finally, we use the estimated covariances and $\widehat{U}$ to compute estimated PSR parameters. One of the advantages of subspace identification is that the complexity of the model can be tuned by selecting the number of singular vectors in $\widehat{U}$, at the risk of losing prediction quality.

As we include more data in our averages, the law of large numbers guarantees that our estimates converge to the true matrices $\Sigma_{\mathcal{AO},\mathcal{H},\mathcal{AO}}$, $\Sigma_{\mathcal{T},\mathcal{H}}$, and $\Sigma_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}}$. So by continuity of the formulas in Chapter 4, if our system is truly a PSR of finite rank, our estimated parameters converge, with probability 1, to the true parameters up to a linear transform—that is, our learning algorithm is *consistent*.[1]

Unfortunately, it is difficult to implement the naïve algorithm in practice. For example, if there are a very large number of features of tests or features of histories, we may not be able even to *store* the full parameters $\Sigma_{\mathcal{AO},\mathcal{H},\mathcal{AO}}$, $\Sigma_{\mathcal{T},\mathcal{H}}$, and $\Sigma_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}}$ in memory. Therefore, we want to use a more efficient algorithm, one that does not explicitly build these parameters. We will start by improving the batch algorithm, then make it online in Section 6.3.

6.2.1 An Efficient Batch Learning Algorithm

The key idea is to compute a set of smaller-sized intermediate quantities from realizations of characteristic features $\phi^{\mathcal{T}}$, indicative features $\phi^{\mathcal{H}}$, and observation features $\phi^{\mathcal{AO}}$, and then compute PSR parameters from these quantities.

In the batch setting we can store $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$ and compute its rank-d singular value decomposition $\widehat{U}$, $\widehat{S}$, $\widehat{V}^\top$. Then, instead of computing $\Sigma_{\mathcal{AO},\mathcal{H},\mathcal{AO}}$, and $\Sigma_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}}$ directly, we use the factors $\widehat{U}$, $\widehat{S}$, $\widehat{V}^\top$ to make storing the other matrices and calculating the ultimate PSR parameters much more efficient. (When we discuss iterative updating in Section 6.3 below we don't even have to store $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$ or compute its SVD directly, potentially increasing our computational and memory savings by a substantial amount.)

In more detail, we begin by estimating $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$. Recall that each element of our estimate $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$ is an unnormalized empirical expectation of the *product* of one indicative feature and one characteristic feature, if we sample a history from ω and then follow an appropriate sequence of actions. We can compute all elements of $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$ from a single sample of trajectories if we sample histories from ω, follow an appropriate exploratory policy, and then importance-weight each sample [16]: $[\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}]_{ij}$ is $\sum_{t=1}^{w} \lambda_t \phi_{it}^{\mathcal{T}} \phi_{jt}^{\mathcal{H}}$, where λ_t is an importance weight.

Next we compute $\widehat{U}\widehat{S}\widehat{V}^\top$, the rank-$d$ thin singular value decomposition of $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$:

$$\langle \widehat{U}, \widehat{S}, \widehat{V} \rangle = \text{SVD}\left(\sum_{t=1}^{w} \lambda_t \phi_t^{\mathcal{T}} \phi_t^{\mathcal{H}} \right) \tag{6.1}$$

[1]Continuity holds if we fix $\widehat{U}$; a similar but more involved argument works if we estimate $\widehat{U}$ as well.

The left singular vectors $\widehat{U}$ define the state space of the PSR and therefore play a direct role in the PSR learning algorithm. However, the right singular vectors $\widehat{V}$ and singular values $\widehat{S}$ can also be used to make computation of the other PSR parameters more efficient.

First, and most obviously, we can directly compute the estimate of an initial feasible state b_* from Equation 4.2a using $\widehat{S}$ and $\widehat{V}$. The key is noting that $U^\top \Sigma_{\mathcal{T},\mathcal{H}} = SV^\top$. Then

$$\hat{b}_* = \widehat{S}\widehat{V}^\top e \tag{6.2a}$$

More importantly, we can compute the Bayes' rule state update without computing the full tensors $\Sigma_{\mathcal{AO},\mathcal{H},\mathcal{AO}}$, and $\Sigma_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}}$. To see that, recall that the Bayes rule state update is given by (Equation 4.22):

$$b_{t+1} = U^\top \Sigma_{\mathcal{T}^+,\mathcal{AO}} \left((U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger b_t\right) \ \left(\Sigma_{\mathcal{AO},\mathcal{AO}} \left((U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger b_t\right)\right)^{-1} \phi_t^{\mathcal{AO}}$$

Therefore, instead of computing $\widehat{\Sigma}_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}} = \sum_{t=1}^{w} \lambda_t \phi_t^{\mathcal{T}} \otimes \phi_t^{\mathcal{AO}} \otimes \phi_t^{\mathcal{H}}$ we can instead compute the much smaller tensor $\widehat{\Sigma}_{B^+,\mathcal{AO}|B} = \widehat{U}^\top \widehat{\Sigma}_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}} (U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger$ directly:

$$\widehat{\Sigma}_{B^+,\mathcal{AO}|B} = \sum_{t=1}^{w} \lambda_t \left(\widehat{U}^\top \phi_t^{\mathcal{T}}\right) \otimes \left(\phi_t^{\mathcal{AO}}\right) \otimes \left(\widehat{S}^{-1}\widehat{V}^\top \phi_t^{\mathcal{H}}\right) \tag{6.2b}$$

Similarly, instead of estimating $\Sigma_{\mathcal{AO},\mathcal{H},\mathcal{AO}} = \sum_{t=1}^{w} \lambda_t \hat{\phi}_t^{\mathcal{AO}} \otimes \phi_t^{\mathcal{H}} \otimes \hat{\phi}_t^{\mathcal{AO}}$ we compute the smaller tensor

$$\widehat{\Sigma}_{\mathcal{AO},\mathcal{AO}|B} = \sum_{t=1}^{w} \lambda_t \left(\hat{\phi}_t^{\mathcal{AO}}\right) \otimes \left(\widehat{S}^{-1}\widehat{V}^\top \phi_t^{\mathcal{H}}\right) \otimes \left(\hat{\phi}_t^{\mathcal{AO}}\right) \tag{6.2c}$$

Finally, we can compute Bayes' rule as

$$\begin{aligned} b_{t+1} &= U^\top \Sigma_{\mathcal{T}^+,\mathcal{AO}} \left((U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger b_t\right) \ \left(\Sigma_{\mathcal{AO},\mathcal{AO}} \left((U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger b_t\right)\right)^{-1} \phi_t^{\mathcal{AO}} \\ &= U^\top \Sigma_{\mathcal{T}^+,\mathcal{AO}} \left((U^\top \Sigma_{\mathcal{T},\mathcal{H}})^\dagger b_t\right) \ \left(\widehat{\Sigma}_{\mathcal{AO},\mathcal{AO}|B}\left(b_t\right)\right)^{-1} \phi_t^{\mathcal{AO}} \\ &= \widehat{\Sigma}_{B^+,\mathcal{AO}|B}\left(b_t\right) \ \left(\widehat{\Sigma}_{\mathcal{AO},\mathcal{AO}|B}\left(b_t\right)\right)^{-1} \phi_t^{\mathcal{AO}} \end{aligned} \tag{6.2d}$$

In summary, this learning algorithm works well when the number of features of tests, histories, and action-observation pairs is relatively small, and in cases where data is collected in batch. These restrictions can be limiting for many real-world data sets. In practice, the number of features may need to be quite large in order to accurately estimate the parameters of the PSR. Additionally, we are often interested in estimating PSRs from massive datasets, updating PSR parameters given a new batch of data, or learning PSRs online from a data stream. Below we develop several computationally efficient extensions to overcome these practical obstacles to learning in real-world situations.

6.3 Iterative Updates to PSR Parameters

We first attack the problem of updating existing PSR parameters given a batch of new information. Next, we look at the special case of updating PSR parameters in an online setting (batch size 1), and develop additional optimizations for this situation.

6.3.1 Batch Updates

We first present an algorithm for updating existing PSR parameters given a new batch of characteristic features $\phi^{\mathcal{T}}_{\text{new}}$, one-step removed characteristic features $\phi^{\mathcal{T}}_{\text{new}+}$, indicative features $\phi^{\mathcal{H}}_{\text{new}}$, and observation features $\phi^{\mathcal{AO}}_{\text{new}}$. Naïvely, we can just store empirical estimates and update them from each new batch of data: $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}} + \phi^{\mathcal{T}}_{\text{new}} {\phi^{\mathcal{H}}_{\text{new}}}^{\top}$, $\widehat{\Sigma}_{\mathcal{AO},\mathcal{H},\mathcal{AO}} + \phi^{\mathcal{AO}}_{\text{new}} \otimes \phi^{\mathcal{H}}_{\text{new}} \otimes \phi^{\mathcal{AO}}_{\text{new}}$,and $\widehat{\Sigma}_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}} + \sum_{t=1}^{w} \phi^{\mathcal{T}}_{\text{new}+} \otimes \phi^{\mathcal{AO}}_{\text{new}} \otimes \phi^{\mathcal{H}}_{\text{new}}$. Then, after each batch, we can learn new PSR parameters.

This naïve algorithm is very inefficient: it requires storing $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$, $\widehat{\Sigma}_{\mathcal{AO},\mathcal{H},\mathcal{AO}}$,and $\widehat{\Sigma}_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}}$, updating these tensors given new information, and recomputing the PSR parameters. However, as we have seen, it is also possible to write the PSR parameters in terms of a set of lower-dimensional memory-efficient matrices and tensors (Equations 6.2a–d), made possible by the singular value decomposition of $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$. The key idea is to update these lower-dimensional matrices directly, instead of the naïve updates suggested above, by taking advantage of numerical algorithms for updating singular value decompositions efficiently [20].

The main computational savings come from using incremental SVD to update $\widehat{U}$, $\widehat{S}$, $\widehat{V}$, $\widehat{\Sigma}_{B^+,\mathcal{AO}|B}$, and $\widehat{\Sigma}_{\mathcal{AO},\mathcal{AO}|B}$ directly. The incremental update for $\widehat{U}$, $\widehat{S}$, $\widehat{V}$ is much more efficient than the naïve additive update when the number of new data points is much smaller than the number of features in $\phi^{\mathcal{T}}$ and $\phi^{\mathcal{H}}$. The incremental updates for $\widehat{\Sigma}_{B^+,\mathcal{AO}|B}$ and $\widehat{\Sigma}_{\mathcal{AO},\mathcal{AO}|B}$ save time and space when the latent dimension d is much smaller than the number of features in $\phi^{\mathcal{T}}$ and $\phi^{\mathcal{H}}$.

Our goal is therefore to compute the updated SVD,

$$\langle \widehat{U}_{\text{new}}, \widehat{S}_{\text{new}}, \widehat{V}_{\text{new}} \rangle = \text{SVD}\left(\widehat{\Sigma}_{\mathcal{T},\mathcal{H}} + \phi^{\mathcal{T}}_{\text{new}} \Lambda {\phi^{\mathcal{H}}_{\text{new}}}^{\top} \right),$$

where Λ is a diagonal matrix of importance weights $\Lambda = \text{diag}(\lambda_{1:t})$. We will derive the incremental SVD update in two steps. First, if the new data $\phi^{\mathcal{T}}_{\text{new}}$ and $\phi^{\mathcal{H}}_{\text{new}}$ lies entirely within the column spaces of $\widehat{U}$ and $\widehat{V}$ respectively, then we can find $\widehat{S}_{\text{new}}$ by projecting both the new and old data onto the subspaces defined by $\widehat{U}$ and $\widehat{V}$, and diagonalizing the resulting small $(n \times n)$ matrix:

$$\begin{aligned}
\langle \widehat{\mathcal{U}}, \widehat{S}_{\text{new}}, \widehat{\mathcal{V}} \rangle &= \text{SVD}\left(\widehat{U}^{\top} \left(\widehat{\Sigma}_{\mathcal{T},\mathcal{H}} + \phi^{\mathcal{T}}_{\text{new}} \Lambda {\phi^{\mathcal{H}}_{\text{new}}}^{\top} \right) \widehat{V} \right) \\
&= \text{SVD}\left(\widehat{S} + (\widehat{U}^{\top} \phi^{\mathcal{T}}_{\text{new}}) \Lambda (\widehat{V}^{\top} \phi^{\mathcal{H}}_{\text{new}})^{\top} \right)
\end{aligned}$$

We can then compute $\widehat{U}_{\text{new}} = \widehat{U}\widehat{\mathcal{U}}^{\top}$ and $\widehat{V}_{\text{new}} = \widehat{V}\widehat{\mathcal{V}}^{\top}$, the rotations of $\widehat{U}$ and $\widehat{V}$ induced by the new data.

If the new data does *not* lie entirely within the column space of $\widehat{U}$ and $\widehat{V}$, we can update the SVD efficiently (and optionally approximately) following Brand [20]. The idea is to split the new data into a part within the column span of $\widehat{U}$ and $\widehat{V}$ and a remainder, and use this decomposition to construct a small matrix to diagonalize as above.

Let C and D be orthonormal bases for the component of the column space of $\phi^{\mathcal{T}}_{\text{new}}$ orthogonal to $\widehat{U}$ and the component of the column space of $\phi^{\mathcal{H}}_{\text{new}}$ orthogonal to $\widehat{V}$:

$$C = \text{orth}\left((I - \widehat{U}\widehat{U}^{\top}) \phi^{\mathcal{T}}_{\text{new}} \right) \tag{6.3a}$$

$$D = \text{orth}\left((I - \widehat{V}\widehat{V}^{\top}) \phi^{\mathcal{H}}_{\text{new}} \right) \tag{6.3b}$$

The dimension of C and D is upper-bounded by the number of data points in our new batch, or the number of features of tests and histories, whichever is smaller. (If the dimension is large, the orthogonalization step above (as well as other steps below) may be too expensive; we can accommodate this case by splitting a large batch of examples into several smaller batches.) Let

$$K = \begin{bmatrix} \widehat{S} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} \widehat{U}^\top \\ C^\top \end{bmatrix} \phi^{\mathcal{T}}_{\text{new}} \Lambda \phi^{\mathcal{H}}_{\text{new}}{}^\top [\, \widehat{V} \; D \,], \tag{6.3c}$$

and diagonalize K to get the update to $\widehat{S}$:

$$\langle \widehat{\mathcal{U}}, \widehat{S}_{\text{new}}, \widehat{\mathcal{V}} \rangle = \text{SVD}\,(K) \tag{6.3d}$$

Finally, as above, $\widehat{\mathcal{U}}$ and $\widehat{\mathcal{V}}$ rotate the extended subspaces $[\, \widehat{U} \; C \,]$ and $[\, \widehat{V} \; D \,]$:

$$\widehat{U}_{\text{new}} = [\, \widehat{U} \; C \,] \widehat{\mathcal{U}} \tag{6.3e}$$

$$\widehat{V}_{\text{new}} = [\, \widehat{V} \; D \,] \widehat{\mathcal{V}} \tag{6.3f}$$

Note that if there are components orthogonal to $\widehat{U}$ and $\widehat{V}$ in the new data (i.e., if C and D are nonempty), the size of the thin SVD will grow. So, during this step, we may choose to tune the complexity of our estimated model by restricting the dimensionality of the SVD. If we do so, we may *lose information* compared to a batch SVD: if future data causes our estimated leading singular vectors to change, the dropped singular vectors may become relevant again. However, empirically, this information loss can be minimal, especially if we keep extra "buffer" singular vectors beyond what we expect to need.

Also note that the above updates do not necessarily preserve orthonormality of $\widehat{U}_{\text{new}}$ and $\widehat{V}_{\text{new}}$, due to discarded nonzero singular values and the accumulation of numerical errors. To correct for this, every few hundred iterations, we re-orthonormalize using a QR decomposition and a SVD:

$$\begin{aligned}
\langle U_Q, U_R \rangle &= \text{QR}\left(\widehat{U}_{\text{new}}\right) \\
\langle V_Q, V_R \rangle &= \text{QR}\left(\widehat{V}_{\text{new}}\right) \\
\langle U_{QR}, S_{QR}, V_{QR} \rangle &= \text{SVD}\left(U_R \widehat{S}_{\text{new}} V_R^\top\right) \\
\widehat{U}_{\text{new}} &= U_Q U_{QR} \\
\widehat{V}_{\text{new}} &= V_Q V_{QR} \\
\widehat{S}_{\text{new}} &= S_{QR}
\end{aligned}$$

The updated SVD now gives us enough information to compute the updates to $\widehat{\Sigma}_{B^+,\mathcal{AO}|B}$ and $\widehat{\Sigma}_{\mathcal{AO},\mathcal{AO}|B}$. Let Λ be the diagonal tensor of importance weights $\Lambda_{i,i,i} = \lambda_i (i \in 1, 2, \ldots, t)$. Using the newly computed subspaces $\widehat{U}_{\text{new}}, \widehat{S}_{\text{new}}$, and $\widehat{V}_{\text{new}}$, we can compute additive updates from the

new data. First, we compute the updated tensor $\widehat{\Sigma}_{B^+,\mathcal{AO}|B_{\text{new}}}$:

$$\begin{aligned}\widehat{\Sigma}_{B^+,\mathcal{AO}|B_{\text{new}}} &= \left(\widehat{\Sigma}_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}} + \Lambda \times_1 \phi^{\mathcal{T}}_{\text{new}+} \times_2 \phi^{\mathcal{AO}}_{\text{new}} \times_3 \phi^{\mathcal{H}}_{\text{new}}\right) \\ &\quad \times_1 \widehat{U}^{\top}_{\text{new}} \times_3 \widehat{S}^{-T}_{\text{new}} \widehat{V}^{\top}_{\text{new}} \\ &= \widehat{\Sigma}^{B^+,\mathcal{AO}}_{\text{update}} + \widehat{\Sigma}^{B^+,\mathcal{AO}}_{\text{new}} \end{aligned} \tag{6.4}$$

where $\widehat{\Sigma}^{B^+,\mathcal{AO}}_{\text{update}}$ can be viewed as the projection of $\widehat{\Sigma}_{B^+,\mathcal{AO}|B}$ onto the new subspace:

$$\begin{aligned}\widehat{\Sigma}^{B^+,\mathcal{AO}}_{\text{update}} &= \widehat{\Sigma}_{\mathcal{T}^+,\mathcal{AO},\mathcal{H}} \times_1 \widehat{U}^{\top}_{\text{new}} \times_3 \widehat{S}^{-T}_{\text{new}} \widehat{V}^{\top}_{\text{new}} \\ &= \begin{bmatrix} \widehat{\Sigma}_{B^+,\mathcal{AO}|B} & 0 \\ 0 & 0 \end{bmatrix} \times_1 \left(\widehat{U}^{\top}_{\text{new}} \begin{bmatrix} \widehat{U} & 0 \end{bmatrix}\right) \\ &\quad \times_3 \left(\widehat{S}^{-1}_{\text{new}} \widehat{V}^{\top}_{\text{new}} \begin{bmatrix} \widehat{V}\widehat{S} & 0 \end{bmatrix}\right)\end{aligned}$$

and $\widehat{\Sigma}^{B^+,\mathcal{AO}}_{\text{new}}$ is the projection of the additive update onto the new subspace:

$$\widehat{\Sigma}^{B^+,\mathcal{AO}}_{\text{new}} = \Lambda \times_1 (\widehat{U}^{\top}_{\text{new}} \phi^{\mathcal{T}}_{\text{new}+}) \times_2 (\phi^{\mathcal{AO}}_{\text{new}}) \times_3 (\widehat{S}^{-1}_{\text{new}} \widehat{V}^{\top}_{\text{new}} \phi^{\mathcal{H}}_{\text{new}})$$

Next we compute the updated tensor $\widehat{\Sigma}_{\mathcal{AO},\mathcal{AO}|B_{\text{new}}}$:

$$\begin{aligned}\widehat{\Sigma}_{\mathcal{AO},\mathcal{AO}|B_{\text{new}}} &= \left(\widehat{\Sigma}_{\mathcal{AO},\mathcal{H}.\mathcal{AO}} + \Lambda \times_1 \phi^{\mathcal{AO}}_{\text{new}} \times_2 \phi^{\mathcal{H}}_{\text{new}} \times_3 \phi^{\mathcal{AO}}_{\text{new}}\right) \\ &\quad \times_2 \widehat{S}^{-T}_{\text{new}} \widehat{V}^{\top}_{\text{new}} \\ &= \widehat{\Sigma}^{\mathcal{AO},\mathcal{AO}}_{\text{update}} + \widehat{\Sigma}^{\mathcal{AO},\mathcal{AO}}_{\text{new}} \end{aligned} \tag{6.5}$$

where $\widehat{\Sigma}^{\mathcal{AO},\mathcal{AO}}_{\text{update}}$ can be viewed as the projection of $\widehat{\Sigma}_{\mathcal{AO},\mathcal{AO}|B}$ onto the new subspace:

$$\begin{aligned}\widehat{\Sigma}^{\mathcal{AO},\mathcal{AO}}_{\text{update}} &= \widehat{\Sigma}_{\mathcal{AO},\mathcal{H},\mathcal{AO}} \times_2 \widehat{S}^{-T}_{\text{new}} \widehat{V}^{\top}_{\text{new}} \\ &= \begin{bmatrix} \widehat{\Sigma}_{B^+,\mathcal{AO}|B} & 0 \end{bmatrix} \times_2 \left(\widehat{S}^{-1}_{\text{new}} \widehat{V}^{\top}_{\text{new}} \begin{bmatrix} \widehat{V}\widehat{S} & 0 \end{bmatrix}\right)\end{aligned}$$

and $\widehat{\Sigma}^{\mathcal{AO},\mathcal{AO}}_{\text{new}}$ is the projection of the additive update onto the new subspace:

$$\widehat{\Sigma}^{\mathcal{AO},\mathcal{AO}}_{\text{new}} = \Lambda \times_1 (\phi^{\mathcal{AO}}_{\text{new}}) \times_2 (\widehat{S}^{-1}_{\text{new}} \widehat{V}^{\top}_{\text{new}} \phi^{\mathcal{H}}_{\text{new}}) \times_3 (\phi^{\mathcal{AO}}_{\text{new}})$$

Once the updated estimates in Equation 6.4 and Equation 6.5 have been calculated, we can compute the new PSR Bayes' rule update as

$$b_{t+1} = \widehat{\Sigma}_{B^+,\mathcal{AO}|B_{\text{new}}}(b_t) \left(\Sigma_{\mathcal{AO},\mathcal{AO}|B_{\text{new}}}(b_t)\right)^{-1} \phi^{\mathcal{AO}}_t \tag{6.6}$$

6.3.2 Online updates

In the online setting (with just one sample per batch), the updates to $\widehat{S}$ are rank-1, allowing some additional efficiencies. We compute the rank-1 update to the matrices $\widehat{U}, \widehat{S}$, and $\widehat{V}^\top$. We can compute C and D efficiently via a simplified Gram-Schmidt step [20]:

$$\begin{aligned}
\widetilde{C} &= (I - \widehat{U}\widehat{U}^\top)\phi_1^{\mathcal{T}} \\
C &= \widetilde{C}/\|\widetilde{C}\| \\
\widetilde{D} &= (I - \widehat{V}\widehat{V}^\top)\phi_1^{\mathcal{H}} \\
D &= \widetilde{D}/\|\widetilde{D}\|
\end{aligned}$$

Finally, we can compute K by adding a rank-1 matrix to $\widehat{S}$:

$$K = \begin{bmatrix} \widehat{S} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} \widehat{U}^\top \\ C^\top \end{bmatrix} \phi_1^{\mathcal{T}} \lambda_1 \phi_1^{\mathcal{H}^\top} [\, \widehat{V} \; D \,],$$

We then compute the updated parameters using Eqs. 6.3d–f.

The online update incurs significant computational cost due to the fact that we must compute a SVD at each time step. Therefore, the online updates are not worth the computation time unless new parameters are truly needed after each observation. By contrast, updating the parameters with small batches of new information provides a good tradeoff between batch and online updates when the efficient batch algorithm is computationally intractable.

6.4 Random Projections for High Dimensional Feature Spaces

Despite their simplicity and wide applicability, HMMs, POMDPs, and PSRs are limited in that they are usually restricted to discrete observations, and the state is usually restricted to have only moderate cardinality. In Chapter 4, we described a feature-based representation for PSRs that relaxes this restriction. In Chapter 5 and Song et al. [99] we proposed a spectral learning algorithm for PSRs and HMMs with continuous observations by representing distributions over these observations and continuous latent states as embeddings in an infinite dimensional Hilbert space. These Hilbert Space Embeddings of PSRs (HSE-PSRs) and HMMs (HSE-HMMs) use essentially the same framework as other spectral learning algorithms for HMMs and PSRs, but avoid working in the infinite-dimensional Hilbert space by the well-known "kernel trick."

HSE-HMMs have been shown to perform well on several real-world datasets, often beating the next best method by a substantial margin. However, they scale poorly due to the need to work with the kernel matrix, whose size is quadratic in the number of training points.

We can overcome this scaling problem and learn PSRs that approximate HSE-HMMs using *random features* for kernel machines [80]: we construct a large but finite set of random features which let us *approximate* a desired kernel using ordinary dot products. (Rahimi and Recht show how to approximate several popular kernels, including radial basis function (RBF) kernels and Laplacian kernels.) The benefit of random features is that we can use fast linear methods that do not depend on the number of data points to approximate the original kernel machine.

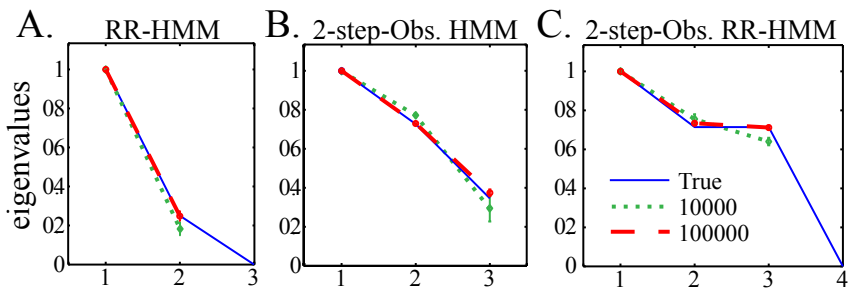

Figure 6.1: A synthetic RR-HMM. (A.) The eigenvalues of the true transition matrix. (B.) RMS error in the nonzero eigenvalues of the estimated transition matrix vs. number of training samples, averaged over 10 trials. The error steadily decreases, indicating that the PSR model is becoming more accurate, as we incorporate more training data.

HSE-HMMs are no exception: using random features of tests and histories, we can *approximate* a HSE-HMM with a PSR. If we combine random features with the above online learning algorithm, we can approximate an HSE-HMM very closely by using an extremely large number of random features. Such a large set of features would overwhelm batch spectral learning algorithms, but our online method allows us to approximate an HSE-HMM very closely, and scale HSE-HMMs to orders of magnitude larger training sets or even to *streaming* datasets with an inexhaustible supply of training data.

6.5 Experimental Results

We designed 3 sets of experiments to evaluate the statistical properties and practical potential of our online spectral learning algorithm. In the first experiment we show the convergence behavior of the algorithm. In the second experiment we show how online spectral learning combined with random projections can be used to learn a PSR that closely approximates the performance of a HSE-HMM. In the third experiment we demonstrate how this combination allows us to model a high-bandwidth, high-dimensional video, where the amount of training data would overwhelm a kernel-based method like HSE-HMMs and the number of features would overwhelm a PSR batch learning algorithm.

6.5.1 A Synthetic Example

First we demonstrate the convergence behavior of our algorithm on a difficult synthetic HMM from Siddiqi et al. [90]. This HMM is 2-step observable, with 4 states, 2 observations, and a rank-3 transition matrix. (So, the HMM is reduced rank (an "RR-HMM") and features of multiple observations are required to disambiguate state.) The transition matrix T and the observation

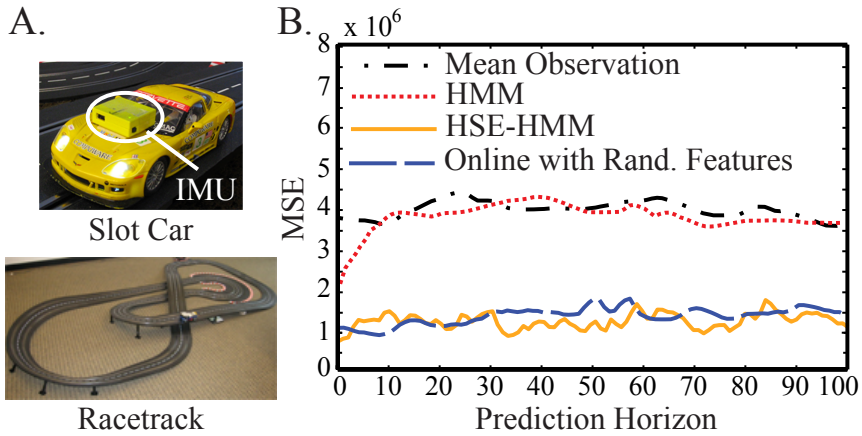

Figure 6.2: Slot car inertial measurement data. (A) The slot car platform: the car and IMU (top) and the racetrack (bottom). (B) Squared error for prediction with different estimated models. Dash-dot shows the baseline of simply predicting the mean measurement on all frames.

matrix O are:

$$T = \begin{bmatrix} 0.7829 & 0.1036 & 0.0399 & 0.0736 \\ 0.1036 & 0.4237 & 0.4262 & 0.0465 \\ 0.0399 & 0.4262 & 0.4380 & 0.0959 \\ 0.0736 & 0.0465 & 0.0959 & 0.7840 \end{bmatrix}$$

$$O = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

We sample observations from the true model and then estimate the model using the algorithm of Section 6.3.2. Since we only expect to recover the transition matrix up to a similarity transform, we compare the eigenvalues of $\widehat{B} = \sum_o \widehat{B}_o$ in the learned model to the eigenvalues of the transition matrix T of the true model. Fig. 6.1 shows that the learned eigenvalues converge to the true ones as the amount of data increases.

6.5.2 Slot Car Inertial Measurement

In a second experiment, we compare the online spectral algorithm with random features to HSE-HMMs with Gaussian RBF kernels. The setup consisted of a track and a miniature car (1:32 scale) guided by a slot cut into the track [99]. Figure 6.2(A) shows the car and the attached IMU (an Intel Inertiadot), as well as the 14m track, which contains elevation changes and banked curves. We collected the estimated 3D acceleration and velocity of the car at 10Hz. The data consisted of 3000 successive measurements while the slot car circled the track controlled by a constant policy. The goal was to learn a model of the noisy IMU data, and, after filtering, to predict future readings.

We trained a 20-dimensional HSE-HMM using the algorithm of Song et al., with tests and histories consisting of 150 consecutive observations. We set the bandwidth parameter of the Gaussian RBF kernels with the "median trick," and the regularization (ridge) parameter was 10^{-4}. For details see Song et al. (2010).

Next we trained a 20-dimensional PSR with random Fourier features to *approximate* the Gaussian RBF kernel. We generated 25000 features for the tests and histories and 400 features for current observations, and then used the online spectral algorithm to learn a model. Finally, to provide some context, we learned a 20-state discrete HMM (with 400 levels of discretization for observations) using the Baum-Welch EM algorithm run until convergence.

For each model, we performed filtering for different extents $t_1 = 100, 101, \ldots, 250$, then predicted an image which was a further $t_2 = 1, 2, \ldots, 100$ steps in the future. The squared error of this prediction in the IMU's measurement space was recorded, and averaged over all the different filtering extents t_1 to obtain means which are plotted in Figure 6.2(B).

The results demonstrate that the online spectral learning algorithm with a large number of random Fourier features does an excellent job matching the performance of the HSE-HMM, and suggest that the online spectral learning algorithm is a viable alternative to HSE-HMMs when the amount of training data grows large.

6.5.3 Modeling Video

Next we look at the problem of mapping from video: we collected a sequence of 11,000 160×120 grayscale frames at 24 fps in an indoor environment (a camera circling a conference room, occasionally switching directions; each full circuit took about 400 frames). This data was collected by hand, so the camera's trajectory is quite noisy. The high frame rate and complexity of the video mean that learning an accurate model requires a very large dataset. Unfortunately, a dataset of this magnitude makes learning an HSE-HMM difficult or impossible: e.g., the similar but less complex example of Song et al. used only 1500 frames.

Instead, we used random Fourier features and an online PSR to approximate a HSE-HMM with Gaussian RBF kernels. We used tests and histories based on 400 sequential frames from the video, generated 100,000 random features, and learned a 50-dimensional PSR. To duplicate this setup, the batch PSR algorithm would have to find the SVD of a 100,000$\times$100,000 matrix; by contrast, we can efficiently update our parameters by incorporating 100,000-element feature vectors one at a time and maintaining 50×50 and $50\times$100,000 matrices.

Figure 6.3 shows our results. The final learned model does a surprisingly good job at capturing the major features of this environment, including both the continuous location of the camera and the discrete direction of motion (either clockwise or counterclockwise). Furthermore, the fact that a general-purpose online algorithm learns these manifolds is a powerful result: we are essentially performing simultaneous localization and mapping in a difficult loop closing scenario, *without* any prior knowledge (even, say, that the environment is three-dimensional, or whether the sensor is a camera, a laser rangefinder, or something else).

6.6 Conclusions

We presented spectral learning algorithms for PSR models of partially-observable nonlinear dynamical systems. In particular, we showed how to *update* the parameters of a PSR given new batches of data, and built on these updates to develop an efficient *online* spectral learning algorithm. We also showed how to use random projections in conjunction with PSRs to efficiently

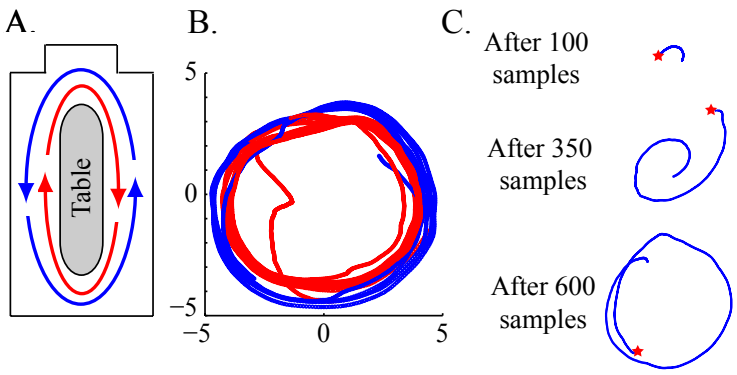

Figure 6.3: Modeling video. (A.) Schematic of the camera's environment. (B.) The second and third dimension of the learned belief space (the first dimension contains normalization information). Points are colored red when the camera is traveling clockwise and blue when traveling counterclockwise. The learned state space separates into two manifolds, one for each direction, connected at points where the camera changes direction. (The manifolds appear on top of one another, but are separated in the fourth latent dimension.) (C.) Loop closing: estimated historical camera positions after 100, 350, and 600 steps. Red star indicates current camera position. The camera loops around the table, and the learned map "snaps" to the correct topology when the camera passes its initial position.

approximate HSE-HMMs. The combination of random projections and online updates allows us to take advantage of powerful Hilbert space embeddings while handling training data sets that are orders of magnitude larger than previous methods, and therefore, to learn models that are too complex for previous methods.

Part II

Spectral Learning Algorithms in Practice

Chapter 7

Stability in Linear Dynamical Systems

A major difficulty in learning LDSs is that standard learning algorithms, both spectral and likelihood based methods, can result in models with unstable dynamics, which causes them to be ill-suited for several important tasks such as simulation and long-term prediction. This problem can arise even when the underlying dynamical system emitting the data is stable, particularly if insufficient training data is available, which is often the case for high-dimensional temporal sequences. In this chapter we propose an extension to Subspace ID that enforces the estimated parameters to be stable. Though stability is a non-convex constraint, we will see how a constraint-generation-based optimization approach yields approximations to the optimal solution that are more efficient and more accurate than previous state-of-the-art stabilizing methods.

7.1 Introduction

We propose an optimization algorithm for learning the dynamics matrix of an LDS while guaranteeing stability. We first obtain an estimate of the underlying state sequence using subspace identification. We then formulate the least-squares minimization problem for the dynamics matrix as a quadratic program (QP) [18], initially without constraints. When we solve this QP, the estimate $\widehat{A}$ we obtain may be unstable. However, any unstable solution allows us to derive a linear constraint which we then add to our original QP and re-solve. This constraint is a conservative approximation to the true feasible region. The above two steps are iterated until we reach a stable solution, which is then refined by a simple interpolation to obtain the best possible stable estimate. The overall algorithm is illustrated in Figure 7.1.

Our method can be viewed as constraint generation [40] for an underlying convex program with a feasible set of all matrices with singular values at most 1, similar to work in control systems such as [57]. This convex set approximates the true, non-convex feasible region. So, we terminate before reaching feasibility in the convex program, by checking for matrix stability after each new constraint. This makes our algorithm less conservative than previous methods for enforcing stability since it chooses the best of a larger set of stable dynamics matrices. The difference in the resulting stable systems is noticeable when simulating data. The constraint generation approach also results in much greater efficiency than previous methods in nearly all cases.

One application of LDSs in computer vision is learning dynamic textures from video data [96]. An advantage of learning dynamic textures is the ability to play back a realistic looking generated sequence of any desired duration. In practice, however, videos synthesized from dynamic texture models can quickly become degenerate because of instability in the underlying LDS. In contrast, sequences generated from dynamic textures learned by our method remain sane even after arbitrarily long durations. We also apply our algorithm to learning baseline dynamic models of over-the-counter (OTC) drug sales for biosurveillance, and sunspot numbers from the UCR archive [54]. Comparison to the best alternative methods [57] on these problems yields positive results.

7.2 Learning Stable Linear Dynamical Systems

The spectral algorithm in Section 3.4 does not enforce stability in $\hat{A}$ which can cause problems when predicting and simulating from an LDS learned from data. To account for stability, we first formulate the dynamics matrix learning problem as a quadratic program with a feasible set that includes the set of stable dynamics matrices. Then we demonstrate how instability in its solutions can be used to generate linear constraints that are added to the QP to restrict this feasible set appropriately. As a final step, the solution is refined to be as close as possible to the least-squares estimate while remaining stable. We compare our results algorithm to two previous approaches from control theory, LB-1 and LB-2, in our experimental results [57, 58].[1] The overall algorithm is illustrated in Figure 7.1(A). We now explain the algorithm in more detail.

7.2.1 Formulating the Objective

In our spectral learning algorithm for Kalman filters, it is possible to write the objective function for $\hat{A}$ as a *quadratic function*. For subspace ID we define a quadratic objective function:

$$\begin{aligned}
\hat{A} &= \arg\min_{A} \left\| A\Sigma_{\mathcal{F},\mathcal{H}} - \Sigma_{\mathcal{F}^+,\mathcal{H}} \right\|_F^2 \\
&= \arg\min_{A} \Big\{ \text{tr}\Big[\left(A\Sigma_{\mathcal{F},\mathcal{H}} - \Sigma_{\mathcal{F}^+,\mathcal{H}}\right)^{\mathrm{T}} \left(A\Sigma_{\mathcal{F},\mathcal{H}} - \Sigma_{\mathcal{F}^+,\mathcal{H}}\right) \Big] \Big\} \\
&= \arg\min_{A} \left\{ \text{tr}\left(A\Sigma_{\mathcal{F},\mathcal{H}}\Sigma_{\mathcal{F},\mathcal{H}}^{\mathrm{T}}A^{\mathrm{T}}\right) - 2\text{tr}\left(\Sigma_{\mathcal{F},\mathcal{H}}\Sigma_{\mathcal{F}^+,\mathcal{H}}^{\mathrm{T}}A\right) + \text{tr}\left(\Sigma_{\mathcal{F}^+,\mathcal{H}}^{\mathrm{T}}\Sigma_{\mathcal{F}^+,\mathcal{H}}\right) \right\} \\
&= \arg\min_{a} \left\{ a^{\mathrm{T}}Pa - 2\, q^{\mathrm{T}}a + r \right\} \qquad (7.1a)
\end{aligned}$$

where $a \in \mathbb{R}^{n^2\times 1}$, $q \in \mathbb{R}^{n^2\times 1}$, $P \in \mathbb{R}^{n^2\times n^2}$ and $r \in \mathbb{R}$ are defined as:

$$a = \text{vec}(A) = [A_{11}\ A_{21}\ A_{31}\ \cdots\ A_{nn}]^{\mathrm{T}} \qquad (7.1b)$$

$$P = I_n \otimes \left(\Sigma_{\mathcal{F},\mathcal{H}}\Sigma_{\mathcal{F},\mathcal{H}}^{\mathrm{T}}\right) \qquad (7.1c)$$

$$q = \text{vec}(\Sigma_{\mathcal{F},\mathcal{H}}\Sigma_{\mathcal{F}^+,\mathcal{H}}^{\mathrm{T}}) \qquad (7.1d)$$

$$r = \text{tr}\left(\Sigma_{\mathcal{F}^+,\mathcal{H}}^{\mathrm{T}}\Sigma_{\mathcal{F}^+,\mathcal{H}}\right) \qquad (7.1e)$$

[1] For detailed comparisons between our algorithm and these approaches, see [89].

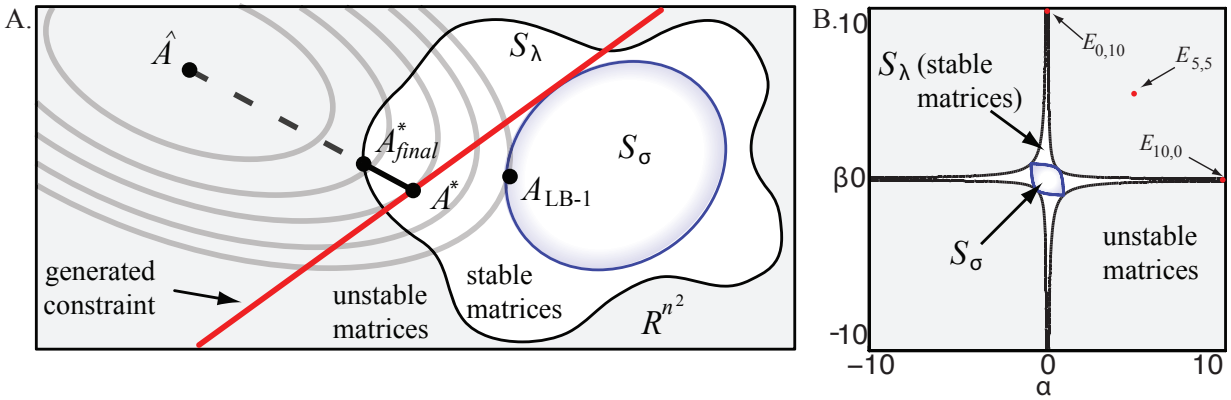

Figure 7.1: (A): Conceptual depiction of the space of $n \times n$ matrices. The region of stability (S_λ) is non-convex while the smaller region of matrices with $\sigma_1 \leq 1$ (S_σ) is convex. The elliptical contours indicate level sets of the quadratic objective function of the QP. $\hat{A}$ is the unconstrained least-squares solution to this objective. $A_{\text{LB-1}}$ is the solution found by LB-1 [57]. One iteration of constraint generation yields the constraint indicated by the line labeled 'generated constraint', and (in this case) leads to a stable solution A^*. The final step of our algorithm improves on this solution by interpolating A^* with the previous solution (in this case, $\hat{A}$) to obtain A^*_{final}. (B): The actual stable and unstable regions for the space of 2×2 matrices $E_{\alpha,\beta} = [\, 0.3 \;\; \alpha \,;\, \beta \;\; 0.3 \,]$, with $\alpha, \beta \in [-10, 10]$. Constraint generation is able to learn a nearly optimal model from a noisy state sequence of length 7 simulated from $E_{0,10}$, with better state reconstruction error than either LB-1 or LB-2. The matrices $E_{10,0}$ and $E_{0,10}$ are stable, but their convex combination $E_{5,5} = 0.5E_{10,0} + (1 - 0.5)E_{0,10}$ is unstable.

I_n is the $n \times n$ identity matrix and $\otimes$ denotes the Kronecker product. Note that P (which is not related to the P in Section 2.3.2) is a symmetric positive semi-definite matrix and the objective function in Equation 7.1a is a quadratic function of a.

7.2.2 Generating Constraints

The feasible set of the quadratic objective function is the space of all $n \times n$ matrices, regardless of their stability. When its solution yields an unstable matrix, the spectral radius of $\hat{A}$ (i.e. $|\lambda_1(\hat{A})|$) is greater than 1. Ideally we would like to use $\hat{A}$ to calculate a convex constraint on the spectral radius. However, consider the class of 2×2 matrices: $E_{\alpha,\beta} = [\, 0.3 \;\; \alpha \,;\, \beta \;\; 0.3 \,]$ [71]. The matrices $E_{10,0}$ and $E_{0,10}$ are stable with $\lambda_1 = 0.3$, but their convex combination $\gamma E_{10,0} + (1 - \gamma)E_{0,10}$ is unstable for (e.g.) $\gamma = 0.5$ (Figure 7.1(B)). This shows that the set of stable matrices is non-convex for $n = 2$, and in fact this is true for all $n > 1$. We turn instead to the largest *singular value*, which is a closely related quantity since

$$\sigma_{min}(\hat{A}) \leq |\lambda_i(\hat{A})| \leq \sigma_{max}(\hat{A}) \quad \forall i = 1, \ldots, n \qquad [39]$$

Therefore every unstable matrix has a singular value greater than one, but the converse is not necessarily true. Moreover, the set of matrices with $\sigma_1 \leq 1$ *is* convex[2]. Figure 7.1(A) conceptually depicts the non-convex region of stability S_λ and the convex region S_σ with $\sigma_1 \leq 1$ in the space of all $n \times n$ matrices for some fixed n. The difference between S_σ and S_λ can be significant. Figure 7.1(B) depicts these regions for $E_{\alpha,\beta}$ with $\alpha, \beta \in [-10, 10]$. The stable matrices $E_{10,0}$ and $E_{0,10}$ reside at the edges of the figure. While results for this class of matrices vary based on the instance used, the constraint generation algorithm described below is able to learn a nearly optimal model from a noisy state sequence of $\tau = 7$ simulated from $E_{0,10}$, with better state reconstruction error than LB-1 and LB-2.

Let $\hat{A} = \tilde{U}\tilde{\Sigma}\tilde{V}^{\mathrm{T}}$ by SVD, where $\tilde{U} = [\tilde{u}_i]_{i=1}^n$ and $\tilde{V} = [\tilde{v}_i]_{i=1}^n$ and $\tilde{\Sigma} = \mathrm{diag}\{\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n\}$. Then:

$$\hat{A} = \tilde{U}\tilde{\Sigma}\tilde{V}^{\mathrm{T}} \Rightarrow \quad \tilde{\Sigma} = \tilde{U}^{\mathrm{T}}\hat{A}\tilde{V} \Rightarrow \quad \tilde{\sigma}_1(\hat{A}) = \tilde{u}_1^{\mathrm{T}}\hat{A}\tilde{v}_1 = \mathrm{tr}(\tilde{u}_1^{\mathrm{T}}\hat{A}\tilde{v}_1) \tag{7.2}$$

Therefore, instability of $\hat{A}$ implies that:

$$\tilde{\sigma}_1 > 1 \Rightarrow \quad \mathrm{tr}\left(\tilde{u}_1^{\mathrm{T}}\hat{A}\tilde{v}_1\right) > 1 \Rightarrow \quad \mathrm{tr}\left(\tilde{v}_1\tilde{u}_1^{\mathrm{T}}\hat{A}\right) > 1 \Rightarrow \quad g^{\mathrm{T}}\hat{a} > 1 \tag{7.3}$$

Here $g = \mathrm{vec}(\tilde{u}_1\tilde{v}_1^{\mathrm{T}})$. Since Eq. (7.3) arose from an unstable solution of Eq. (7.1a), g is a hyperplane separating $\hat{a}$ from the space of matrices with $\sigma_1 \leq 1$. We use the negation of Eq. (7.3) as a constraint:

$$g^{\mathrm{T}}\hat{a} \leq 1 \tag{7.4}$$

7.2.3 Computing the Solution

The overall quadratic program can be stated as:

$$\begin{array}{ll} \text{minimize} & a^{\mathrm{T}}Pa - 2\,q^{T}a + r \\ \text{subject to} & Ga \leq h \end{array} \tag{7.5}$$

with a, P, q and r as defined in Eqs. (7.1e). $\{G, h\}$ define the set of constraints, and are initially empty. The QP is invoked repeatedly until the stable region, i.e. S_λ, is reached. At each iteration, we calculate a linear constraint of the form in Eq. (7.4), add the corresponding g^{T} as a row in G, and augment h with 1. Note that we will almost always stop *before* reaching the feasible region S_σ.

7.2.4 Refinement

Once a stable matrix is obtained, it is possible to refine this solution. We know that the last constraint caused our solution to cross the boundary of S_λ, so we interpolate between the last solution and the previous iteration's solution using binary search to look for a boundary of the

[2]Since $\sigma_1(M) \stackrel{\text{def}}{=} \max_{u,v:\|u\|_2=1,\|v\|_2=1} u^{\mathrm{T}}Mv$, so if $\sigma_1(M_1) \leq 1$ and $\sigma_1(M_2) \leq 1$, then for all convex combinations, $\sigma_1(\gamma M_1 + (1-\gamma)M_2) = \max_{u,v:\|u\|_2=1,\|v\|_2=1} \gamma u^{\mathrm{T}}M_1v + (1-\gamma)u^{\mathrm{T}}M_2v \leq 1$.

stable region, in order to obtain a better objective value while remaining stable. This results in a stable matrix with top eigenvalue slightly less than 1. In principle, such an interpolation could be attempted between the least squares solution and any stable solution. However, the stable region can be highly complex, and there may be several folds and boundaries of the stable region in the interpolated area. In our experiments (not shown), interpolating from the solutions given by LB-1 and LB-2 yielded worse results.

7.3 Experiments

For learning the dynamics matrix, we implemented subspace identification, constraint generation (using `quadprog`), LB-1 [57] and LB-2 [58] (using `CVX` with `SeDuMi`) in Matlab on a 3.2 GHz Pentium with 2 GB RAM. Note that the algorithms that constrain the solution to be stable give a different result from the basic subspace ID algorithm only in situations when the learned $\hat{A}$ is unstable. However, LDSs learned in scarce-data scenarios are unstable for almost any domain, and some domains lead to unstable models up to the limit of available data (e.g. the `steam` dynamic textures in Section 7.3.1). The goals of our experiments are to: (1) examine the state evolution and simulated observations of models learned using constraint generation, and compare them to previous work on learning stable dynamical systems; and (2) compare the algorithms in terms of predictive accuracy and computational efficiency. We apply these algorithms to learning dynamic textures from the vision domain (Section 7.3.1) as well as OTC drug sales counts (Section 7.3.2) and sunspot numbers (Section 7.3.3).

7.3.1 Stable Dynamic Textures

Dynamic textures in vision can intuitively be described as models for sequences of images that exhibit some form of low-dimensional structure and recurrent (though not necessarily repeating) characteristics, e.g., fixed-background videos of rising smoke or flowing water. Treating each frame of a video as an observation vector of pixel values y_t, we learned dynamic texture models of two video sequences by subspace identification: the `steam` sequence, composed of 120×170 pixel images, and the `fountain` sequence, composed of 150×90 pixel images, both of which originated from the MIT temporal texture database (Figure 7.2(A)). We use the following parameters: training data size $\tau = 80$, number of latent state dimensions $n = 15$, and number of past and future observations in the Hankel matrix $i = 5$. Note that, while the observations are the raw pixel values, the underlying state sequence we learn has no *a priori* interpretation. We can, however, interpret the underlying state space as a set of predicted observations; that is, states are coefficients for the learned observation basis.

An LDS model of a dynamic texture may *synthesize* an arbitrarily long sequence of images by driving the model with zero mean Gaussian noise. Each of our two models uses an 80 frame training sequence to generate 1000 sequential images in this way. To better visualize the difference between image sequences generated by least-squares, LB-1, and constraint generation, the evolution of each method's state is plotted over the course of the synthesized sequences (Figure 7.2(B)). Sequences generated by the least squares models appear to be unstable, and this was in fact the case; both the `steam` and the `fountain` sequences resulted in unstable dy-

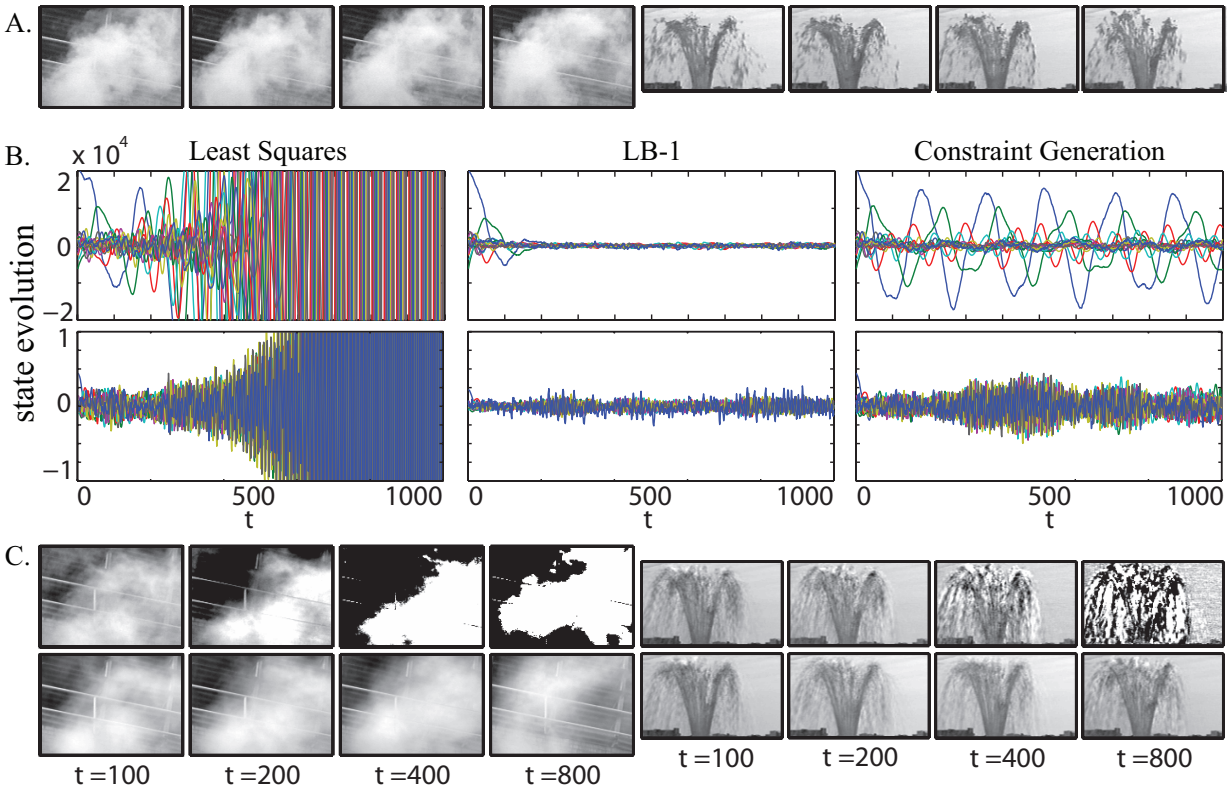

Figure 7.2: Dynamic textures. A. Samples from the original `steam` sequence and the `fountain` sequence. B. State evolution of synthesized sequences over 1000 frames (`steam` top, `fountain` bottom). The least squares solutions display instability as time progresses. The solutions obtained using LB-1 remain stable for the full 1000 frame image sequence. The constraint generation solutions, however, yield state sequences that are stable over the full 1000 frame image sequence without significant damping. C. Samples drawn from a least squares synthesized sequences (top), and samples drawn from a constraint generation synthesized sequence (bottom). Here we are displaying image values that are clipped to stay within the valid pixel range $[0, 255]$. Images for LB-1 are not shown. The constraint generation synthesized `steam` sequence is qualitatively better looking than the `steam` sequence generated by LB-1, although there is little qualitative difference between the two synthesized `fountain` sequences.

namics matrices. Conversely, the constrained subspace identification algorithms all produced well-behaved sequences of states and stable dynamics matrices (Table 7.1), although constraint generation demonstrates the fastest runtime, best scalability, and lowest error of any stability-enforcing approach.

A qualitative comparison of images generated by constraint generation and least squares (Figure 7.2(C)) indicates the effect of instability in synthesized sequences generated from dynamic texture models. While the unstable least-squares model demonstrates a dramatic increase in image contrast over time, the constraint generation model continues to generate qualitatively reasonable images. Qualitative comparisons between constraint generation and LB-1 indicate

	CG	LB-1	LB-1*	LB-2	CG	LB-1	LB-1*	LB-2
	`steam` ($n = 10$)				`fountain` ($n = 10$)			
$\lvert\lambda_1\rvert$	1.000	0.993	0.993	1.000	0.999	0.987	0.987	0.997
σ_1	1.036	1.000	1.000	1.034	1.051	1.000	1.000	1.054
$e_x(\%)$	**45.2**	103.3	103.3	546.9	**0.1**	4.1	4.1	3.0
time	**0.45**	95.87	3.77	0.50	**0.15**	15.43	1.09	0.49
	`steam` ($n = 20$)				`fountain` ($n = 20$)			
$\lvert\lambda_1\rvert$	0.999	—	0.990	0.999	0.999	—	0.988	0.996
σ_1	1.037	—	1.000	1.062	1.054	—	1.000	1.056
$e_x(\%)$	**58.4**	—	154.7	294.8	**1.2**	—	5.0	22.3
time	**2.37**	—	1259.6	33.55	**1.63**	—	159.85	5.13
	`steam` ($n = 30$)				`fountain` ($n = 30$)			
$\lvert\lambda_1\rvert$	1.000	—	0.988	1.000	1.000	—	0.993	0.998
σ_1	1.054	—	1.000	1.130	1.030	—	1.000	1.179
$e_x(\%)$	**63.0**	—	341.3	631.5	**13.3**	—	14.9	104.8
time	**8.72**	—	23978.9	62.44	**12.68**	—	5038.94	48.55
	`steam` ($n = 40$)				`fountain` ($n = 40$)			
$\lvert\lambda_1\rvert$	1.000	—	0.989	1.000	1.000	—	0.991	1.000
σ_1	1.120	—	1.000	1.128	1.034	—	1.000	1.172
$e_x(\%)$	**20.24**	—	282.7	768.5	**3.3**	—	4.8	21.5
time	**5.85**	—	79516.98	289.79	**61.9**	—	43457.77	239.53

Table 7.1: Quantitative results on the dynamic textures data for different numbers of states n. CG is our algorithm, LB-1and LB-2 are competing algorithms, and LB-1* is a simulation of LB-1 using our algorithm by generating constraints until we reach S_σ, since LB-1 failed for $n > 10$ due to memory limits. e_x is percent difference in squared reconstruction error. Constraint generation, in all cases, has lower error and faster runtime.

that constraint generation learns models that generate more natural-looking video sequences[3] than LB-1.

Given the paucity of data available when modeling dynamic textures, it is not possible to test the long-range predictive power of the learned dynamical systems quantitatively. Instead, the error metric used for the quantitative experiments when evaluating matrix A^* is

$$e_x(A^*) = 100\% \times \left(J^2(A^*) - J^2(\hat{A})\right) / J^2(\hat{A}) \tag{7.6}$$

i.e. percent increase in squared reconstruction error compared to least squares, with $J(\cdot)$ as defined in Eq. (7.1a). Table 7.1 demonstrates that constraint generation always has the lowest error as well as the fastest runtime. The running time of constraint generation depends on the number of constraints needed to reach a stable solution. Note that LB-1 is more efficient and scalable when simulated using constraint generation (by adding constraints until S_σ is reached) than it is in its original SDP formulation.

[3] See videos at http://www.select.cs.cmu.edu/projects/stableLDS

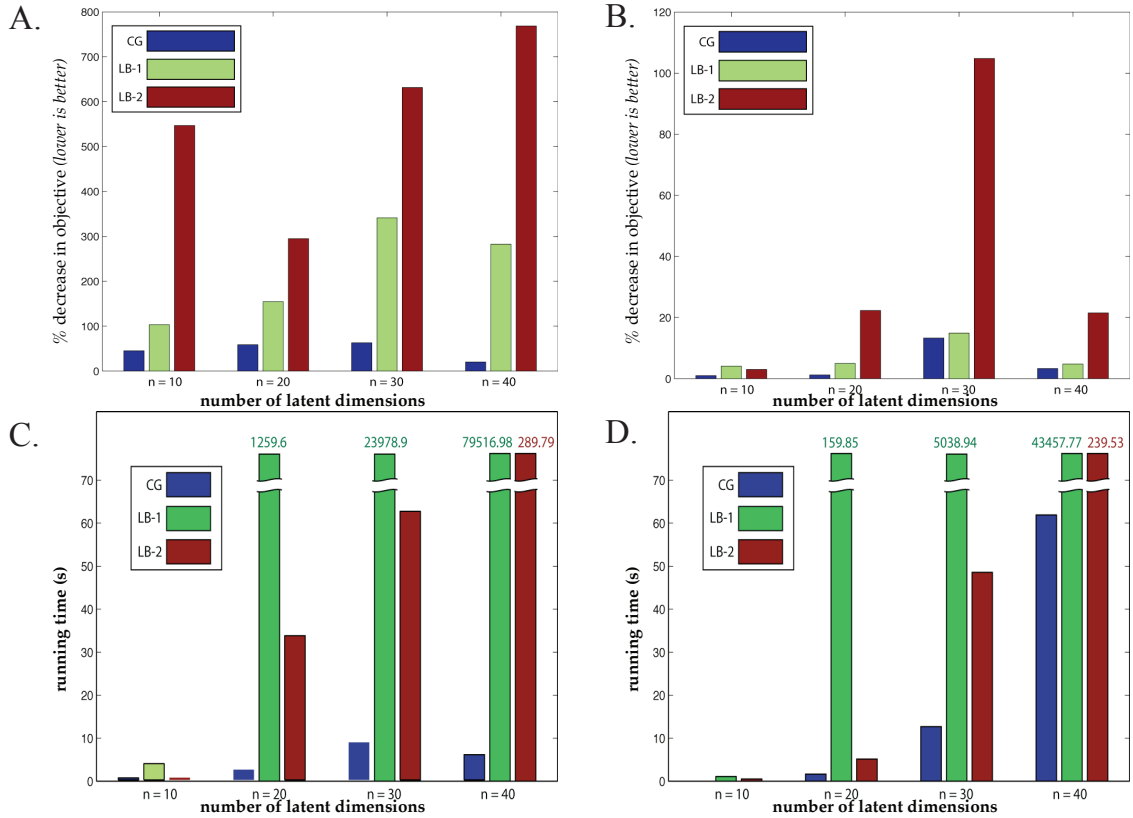

Figure 7.3: Bar graphs illustrating decreases in objective function value relative to the least squares solution (A,B) and the running times (C,D) for different stable LDS learning algorithms on the `fountain` and `steam` textures respectively, based on the corresponding columns of Table 7.1.

7.3.2 Stable Baseline Models for Biosurveillance

We examine daily counts of OTC drug sales in pharmacies, obtained from the National Data Retail Monitor (NDRM) collection [118]. The counts are divided into 23 different categories and are tracked separately for each zipcode in the country. We focus on zipcodes from a particular American city (not identified here due to data privacy restrictions). The data exhibits 7-day periodicity due to differential buying patterns during weekdays and weekends. We isolate a 60-day subsequence where the data dynamics remain relatively stationary, and attempt to learn LDS parameters to be able to simulate sequences of baseline values for use in detecting anomalies.

We perform two experiments on different aggregations of the OTC data, with parameter values We use the following parameters: number of latent state dimensions $n = 7$, number of past and future observations in the Hankel matrix $i = 4$, and training data size $\tau = 14$. Figure 7.4(A) plots 22 different drug categories aggregated over all zipcodes, and Figure 7.4(B) plots a single drug category (cough/cold) in 29 different zipcodes separately. In both cases, constraint generation is able to use very little training data to learn a stable model that captures the periodicity in the data, while the least squares model is unstable and its predictions diverge

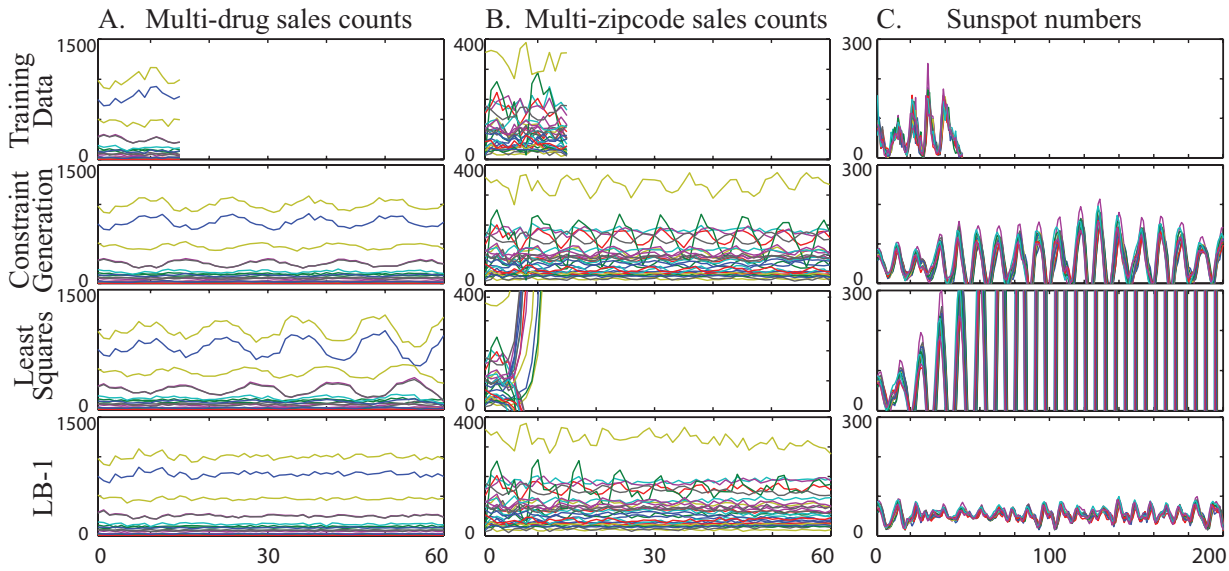

Figure 7.4: (A): 60 days of data for 22 drug categories aggregated over all zipcodes in the city. (B): 60 days of data for a single drug category (cough/cold) for all 29 zipcodes in the city. (C): Sunspot numbers for 200 years separately for each of the 12 months. The training data (top), simulated output from constraint generation, output from the unstable least squares model, and output from the over-damped LB-1 model (bottom).

over time. LB-1 learns a model that is stable but overconstrained, and the simulated observations quickly drift from the correct magnitudes. Further details can be found in [89].

7.3.3 Modeling Sunspot Numbers

We compared least squares and constraint generation on learning LDS models for the sunspot data discussed earlier. We use the following parameters: number of latent state dimensions $n = 7$, number of past and future observations in the Hankel matrix $i = 9$, and training data size $\tau = 50$. Figure 7.4(C) represents a data-poor training scenario where we train a least-squares model on 18 timesteps, yielding an unstable model whose simulated observations increase in amplitude steadily over time. Again, constraint generation is able to use very little training data to learn a stable model that seems to capture the periodicity in the data if not the magnitude, while the least squares model is unstable. The model learned by LB-1 attenuates more noticeably, capturing the periodicity to a smaller extent. Quantitative results on both these domains exhibit similar trends as those in Table 7.1.

7.4 Related Work

Linear system identification is a well-studied subject [62]. Within this area, subspace identification methods ([117], Chapter 2) have been very successful. These techniques first estimate the

model dimensionality and the underlying state sequence, and then derive parameter estimates using least squares. Within subspace methods, techniques have been developed to enforce stability by augmenting the extended observability matrix with zeros [24] or adding a regularization term to the least squares objective [116].

All previous methods were outperformed by LB-1 [57]. They formulate the problem as a semidefinite program (SDP) whose objective minimizes the state sequence reconstruction error, and whose constraint bounds the largest singular value by 1. This convex constraint is obtained by rewriting the nonlinear matrix inequality $I_n - AA^{\mathrm{T}} \succeq 0$ as a linear matrix inequality[4], where I_n is the $n \times n$ identity matrix. Here, $\succ 0$ ($\succeq 0$) denotes positive (semi-) definiteness. The existence of this constraint also proves the convexity of the $\sigma_1 \leq 1$ region. This condition is *sufficient* but not *necessary*, since a matrix that violates this condition may still be stable.

A follow-up to this work by the same authors [58], which we call LB-2, attempts to overcome the conservativeness of LB-1 by approximating the Lyapunov inequalities $P - APA^{\mathrm{T}} \succ 0$, $P \succ 0$ with the inequalities $P - APA^{\mathrm{T}} - \delta I_n \succeq 0$, $P - \delta I_n \succeq 0$, $\delta > 0$. These inequalities hold *iff* the spectral radius is less than 1.[5] However, the approximation is achieved only at the cost of inducing a nonlinear distortion of the objective function by a problem-dependent reweighting matrix involving P, which is a variable to be optimized. In our experiments, this causes LB-2 to perform worse than LB-1 (for any δ) in terms of the state sequence reconstruction error (dynamic textures) and predictive log-likelihood (robot sensor data), even while obtaining solutions outside the feasible region of LB-1. Consequently, we focus on LB-1 in our conceptual and qualitative comparisons as it is the strongest baseline available. However, LB-2 is more scalable than LB-1, so quantitative results are presented for both.

To summarize the distinction between constraint generation, LB-1 and LB-2: it is hard to have both the right objective function (reconstruction error) and the right feasible region (the set of stable matrices). LB-1 optimizes the right objective but over the wrong feasible region (the set of matrices with $\sigma_1 \leq 1$). LB-2 has a feasible region close to the right one, but at the cost of distorting its objective function to an extent that it fares worse than LB-1 in nearly all cases. In contrast, our method optimizes the right objective over a less conservative feasible region than that of any previous algorithm with the right objective, and this combination is shown to work the best in practice.

[4]This bounds the top singular value by 1 since it implies $\forall x\ x^T(I_n - AA^T)x \geq 0 \Rightarrow \forall x\ x^T AA^T x \leq x^T x \Rightarrow$ for $\nu = \nu_1(AA^{\mathrm{T}})$ and $\lambda = \lambda_1(AA^{\mathrm{T}})$, $\nu^T AA^T \nu \leq \nu^T \nu \Rightarrow \nu^{\mathrm{T}} \lambda \nu \leq 1 \Rightarrow \sigma_1^2(A) \leq 1$ since $\nu^{\mathrm{T}}\nu = 1$ and $\sigma_1^2(M) = \lambda_1(MM^T)$ for any square matrix M.

[5]For a proof sketch, see Horn and Johnson [39] pg. 410.

Chapter 8

Reinforcement Learning: Value Iteration in a Learned Predictive State Representation

8.1 Introduction

In this chapter we shift our focus from learning models of linear dynamical systems to *planning* in *predictive state representations*. Planning a sequence of actions or a policy to maximize reward has long been considered a fundamental problem for autonomous agents. In the hardest version of the problem, an agent must form a plan based solely on its own experience, without the aid of a human engineer who can design problem-specific models, features or heuristics; it is this version of the problem which we must solve to build a truly autonomous agent.

Partially Observable Markov Decision Processes (POMDPs) [22, 97] are a general framework for single-agent planning. POMDPs model the state of the world as a latent variable and explicitly reason about uncertainty in both action effects and state observability. Plans in POMDPs are expressed as *policies*, which specify the action to take given any possible probability distribution over states. Unfortunately, exact planning algorithms such as *value iteration* [97] are computationally intractable for most realistic POMDP planning problems. Furthermore, researchers have had only limited success learning POMDP models from data. There are arguably two primary reasons for these problems [77]. The first is the "curse of dimensionality": for a POMDP with n states, the optimal policy is a function of an $n-1$ dimensional distribution over latent state. The second is the "curse of history": the number of distinct policies increases exponentially in the planning horizon. We hope to mitigate the curse of dimensionality by seeking an approximate dynamical system model with *low dimensionality*, and to mitigate the curse of history by looking for a model that is susceptible to *approximate* planning.

As we demonstrated in Part I of this thesis, *Predictive State Representations (PSRs)* [61] and the closely related *Observable Operator Models (OOMs)* [44] are generalizations of POMDPs that have attracted interest because they both have greater representational capacity than POMDPs and yield representations that are *at least* as compact [31, 93]. An additional benefit of PSRs and OOMs is that many successful approximate planning techniques for POMDPs can be used to

plan in PSRs and OOMs with minimal adjustment. Accordingly, PSR and OOM models of dynamical systems have potential to overcome both the curse of dimensionality and the curse of history.

The quality of an optimized policy for a POMDP, PSR, or OOM depends strongly on the accuracy of the model: inaccurate models typically lead to useless plans. We can specify a model manually or learn one from data. A fully autonomous agent must be able to learn models from data, but due to the difficulty of learning, it is far more common to see planning algorithms applied to hand-specified models, and therefore to small systems where there is extensive and goal-relevant domain knowledge. For example, recent extensions of approximate planning techniques for PSRs have only been applied to hand-constructed models [43, 45].

Work that does learn models for planning in partially observable environments has so far met with only limited success. As a result, there have been few successful attempts at closing the loop by learning a model from an environment, planning in that model, and testing the plan in the environment. For example, Expectation-Maximization (or EM—see, e.g., [9]) does not avoid local minima or scale to large state spaces; and, although many learning algorithms have been proposed for PSRs [16, 67, 92, 119, 123] and OOMs [38, 44, 63], none have been shown to learn models that are accurate enough for planning.

Several researchers have, however, made progress in the problem of planning using a learned model. In one instance [87], researchers obtained a POMDP heuristically from the output of a model-free algorithm [66] and demonstrated planning on a small toy maze. In another instance [85], researchers used Markov Chain Monte Carlo (MCMC) inference both to learn a factored Dynamic Bayesian Network (DBN) representation of a POMDP in a small synthetic network administration domain, as well as to perform online planning. Due to the cost of the MCMC sampler used, this approach is still impractical for larger models. In a third example, researchers learned Linear-Linear Exponential Family PSRs from an agent traversing a simulated environment, and found a policy using a policy gradient technique with a parameterized function of the learned PSR state as input [120, 122]. In this case both the learning and the planning algorithm were subject to local optima. In addition, the authors determined that the learned model was too inaccurate to support value-function-based planning methods [120]. Finally, there is a successful line of research which computes closed-loop controllers from learned or partly-learned models, starting from linear subspace identification [117] and ranging to controllers for helicopters [72] and bird-like robots [107]. This line of research uses techniques similar to the ones described here; but, it focuses on control-like problems, in which accurate state estimation and dealing with continuous controls are the main sources of difficulty, in contrast to the planning-like problems we consider here, in which longer-term lookahead and discrete choices are more important.

The current work differs from these and other previous examples of planning in learned models: it both uses a principled and provably statistically consistent model-learning algorithm, which we developed in Chapter 3 and Chapter 4, and demonstrates that this algorithm is able to learn compact models of a difficult, realistic dynamical system without any prior domain knowledge built into the model or algorithm. Finally, we perform approximate point-based value iteration (PBVI) in the learned compact models, and demonstrate that the greedy policy for the resulting value function works well in the original (not the learned) system. To our knowledge this is the first research that combines all of these achievements, closing the loop from observations

to actions in an unknown nonlinear, non-Gaussian planning system with no human intervention beyond collecting the raw transition data and specifying features.

8.2 Planning in PSRs

The primary motivation for modeling a controlled dynamical system is to reason about the effects of taking a sequence of actions in the system. Although this paper is not predominantly about planning algorithms for PSRs, since PSR planning is a straightforward extension of POMDP planning [43, 45], we describe PSR planning here since it is needed for our closing-the-loop experiments. The PSR model can be augmented for this purpose by specifying a linear reward function for taking an action a_t in state b_t:

$$\mathcal{R}(b_t, a_t) \stackrel{\text{def}}{=} \eta_{a_t}^\top b_t \tag{8.1}$$

where $\eta_{a_t}^\top \in \mathbb{R}^n$ is the linear function specified by action a_t. (This extension generalizes the state-dependent rewards of (PO)MDPs.) Given this function and a discount factor γ, the planning problem for PSRs is to find a policy that maximizes the expected discounted sum of rewards, $\mathbb{E}\left[\sum_t \gamma^t \mathcal{R}(b_t, a_t)\right]$. The optimal policy can be compactly represented using the optimal value function $J^*(b_t)$, which specifies the expected sum of future rewards in each PSR state. The value function is defined recursively as:

$$J^*(b_t) \stackrel{\text{def}}{=} \max_{a \in \mathcal{A}} \left[\mathcal{R}(b_t, a_t = a) + \gamma \sum_{o \in \mathcal{O}} \mathbb{P}\left[o_t = o \,|\, b_t, a_t = a\right] V^*(b_{tao}) \right] \tag{8.2}$$

where b_{tao} is the state obtained from b_t after executing action a and observing o. We have implicitly assumed that the expected reward is a linear function of the PSR state; we can ensure that this assumption holds by including the reward as an observation when we learn the PSR dynamics. (Or, if the reward is not directly observable, by including its expectation given all observable information.) We can obtain the optimal action by taking the $\arg\max$ instead of the $\max$ in Equation 8.3:

$$\pi^*(b_t) \stackrel{\text{def}}{=} \arg\max_{a \in \mathcal{A}} \left[\mathcal{R}(b_t, a_t = a) + \gamma \sum_{o \in \mathcal{O}} \mathbb{P}\left[o_t = o \,|\, b_t, a_t = a\right] V^*(b_{tao}) \right] \tag{8.3}$$

When optimized exactly, the value function is always piecewise linear and convex (PWLC) in the state, and has finitely many pieces in finite-horizon planning problems [45]. In this case, the value function for a finite horizon t can be expressed by a set of vectors $\Gamma_t = \{\alpha_1, \ldots, \alpha_g\}$. Each α-vector represents a hyperplane, and defines a value function over a bounded region of the PSR state space:

$$J_t(b_t) = \max_{\alpha \in \Gamma_t} \alpha^\top b_t \tag{8.4}$$

The finite horizon t set Γ_t can be generated recursively from Γ_{t-1} through a process called *Exact Value Iteration*. Let

$$\Gamma_t^{a,o} \stackrel{\text{def}}{=} \{\alpha_i^{a,o} \mid \alpha_i \in \Gamma_{t-1}, a \in A, o \in \mathcal{O}\} \tag{8.5}$$

$$\alpha_i^{a,o} \stackrel{\text{def}}{=} \gamma B_{ao}^\top \alpha_i \tag{8.6}$$

Then, for each action $a \in \mathcal{A}$, the set Γ_t^a is generated by:

$$\Gamma_t^a = \mathcal{R}(b_t, a_t) + \bigoplus_{o \in O} \Gamma_t^{a,o} \tag{8.7}$$

where $\oplus$ denotes the cross-sum operator. Finally, the new set Γ_t is the union

$$\Gamma_t = \bigcup_{a \in \mathcal{A}} \Gamma_t^a \tag{8.8}$$

Exact value iteration for PSRs optimizes the value function over all possible beliefs or state vectors. However, computing the exact value function is problematic because the number of sequences of actions that must be considered grows exponentially with the planning horizon, called the "curse of history." Approximate point-based planning techniques (see below) specifically target the curse of history by attempting only to calculate the best sequence of actions at some finite set of belief points. Unfortunately, in high dimensions, approximate planning techniques have difficulty adequately sampling the space of possible beliefs. This is called the "curse of dimensionality." Because PSRs often admit a compact low-dimensional representation, they can reduce the effect of the curse of dimensionality, and so approximate point-based planning techniques can work well in these models.

Point-Based Value Iteration (PBVI) [76] is an efficient approximation of exact value iteration that performs value backup steps on a finite set of heuristically-chosen belief points rather than over the entire belief simplex. PBVI exploits the fact that the value function is PWLC. A linear lower bound on the value function at one point b can be used as a lower bound at nearby points; this insight allows the value function to be approximated with a finite set of α-vectors, one for each chosen point. Although PBVI was designed for POMDPs, the approach has been generalized to PSRs [43]. Formally, given some set of points $\mathcal{B} = \{b_1, \ldots, b_g\}$ in the PSR state space, we recursively compute the value function and linear lower bounds at only these points. The approximate value function after t iterations can be represented by a set of α-vectors Γ_t such that each α_i is maximal for least one prediction vector b_i.

As with exact value iteration, Γ_t can be generated recursively from Γ_{t-1}.

$$\Gamma_t^a = \{\alpha_b^a \mid b \in \mathcal{B}\} \qquad \alpha_b^a = \mathcal{R}(b, a) + \sum_{o \in O} \arg\max_{\alpha \in \Gamma_t^{a,o}} \alpha^\top b \tag{8.9}$$

where $\Gamma_t^{a,o}$ is as given in Equation 8.5. Because we only need to generate one α-vector α_b^a per PSR state b for each action $a \in A$, we calculate summations for only these states and do not need the cross-sum operation (Equation 8.7). Finally we find the best α-vector for each PSR state

$$\alpha_b = \arg\max_{\alpha \in \cup_a \Gamma_t^a} \alpha^\top b \quad (\forall b \in \mathcal{B}) \tag{8.10}$$

Once Γ_t has been calculated, the approximate value function at *any* PSR state b, including $b \notin \mathcal{B}$ can be found as before (Equation 8.4).

In addition to being tractable on much larger-scale planning problems than exact value iteration, PBVI comes with theoretical guarantees in the form of error bounds that are low-order polynomials in the degree of approximation, range of reward values, and discount factor γ [43, 76]. In our experiments, we use Perseus [45, 102], a variant of PBVI that updates the value function over a small randomized subset of a large set of reachable belief points at each time step. By only updating a subset of belief points, Perseus can achieve a computational advantage over plain PBVI in some domains.

8.3 Experimental Results

We have introduced a novel algorithm for learning PSRs directly from data, as well as a kernel-based extension for modeling continuous observations. We judge the quality of our PSR learning algorithm by first learning a model of a challenging non-linear, partially observable, controlled domain directly from sensor inputs and then "closing the loop" by *planning* in the learned model. Successful planning is a much stronger result than standard dynamical system evaluations such as one-step squared error or prediction log-likelihood. Unlike previous attempts to learn PSRs, which either lack planning results [84, 121], or which compare policies within the learned system [122], we compare our resulting policy to a bound on the best possible solution in the original system and demonstrate that the policy is close to optimal.

8.3.1 The Autonomous Robot Domain

Our simulated autonomous robot domain consisted of a simple 45×45 unit square arena with a central obstacle and brightly colored walls (Figure 8.1(A-B)), containing a robot of radius 2 units. The robot could move around the floor of the arena and rotate to face in any direction. The robot had a simulated 16×16 pixel color camera, with a focal plane one unit in front of the robot's center of rotation, and with a visual field of $45°$ in both azimuth and elevation, corresponding to an angular resolution of $\sim 2.8°$ per pixel. Images on the sensor matrix at any moment were simulated by a non-linear perspective transformation and projection onto the camera's focal plane, based on the robot's current position and orientation in the environment. The resulting 768-element pattern of unprocessed RGB values was the only input to the robot (images were *not* preprocessed to extract features), and each action produced a new set of pixel values. The robot was able to move forward 1 or 0 units, and simultaneously rotate $15°$, $-15°$, or $0°$, resulting in 6 unique actions. In the real world, friction, uneven surfaces, and other factors confound precisely predictable movements. To simulate this uncertainty, a small amount of Gaussian noise was added to the translation (mean 0, standard deviation .1 units) and rotation (mean 0, standard deviation $5°$) components of the actions. The robot was allowed to occupy any real-valued (x, y, θ) pose that didn't intersect a wall; in case of an attempt to drive through a wall, we interrupted the commanded motion just before contact, simulating an inelastic collision.

The autonomous robot domain was designed to be a difficult domain comparable to the most complex domains that previous PSR algorithms have attempted to model. In particular, the

domain in this paper was modeled after the autonomous robot domains found in recent PSR work [121, 122]. The proposed problem, learning a model of this domain and then planning in the learned model, is quite difficult. The autonomous robot has *no* knowledge of any of the underlying properties of the domain, e.g., the geometry of the environment or the robot motion model; it only has access to samples of the 3×256 pixel features, and how these features change as actions are executed. Writing a correct policy for a specific task in this domain by hand would be at best tedious—and in any case, as mentioned above, it is often impractical to hand-design a policy for an autonomous agent, since doing so requires guessing the particular planning problems that the agent may face in the future. Furthermore, the continuous and non-linear nature of this domain makes learning models difficult. For example, a POMDP model of this domain would require a prohibitively large number of hidden states, making learning and planning next to impossible. PSRs are able to overcome this problem by *compactly* representing state in a low-dimensional real-valued space, and the algorithm presented in this work allows us to efficiently learn the parameters of the PSR in closed form.

8.3.2 Features

In order to contend with the continuous observations generated by our experimental system, we used *features* of actions and observations when building our observable representation. In particular, we built features for each observation by using a fraction of the training observations as kernel centers. We used a multivariate Gaussian kernel with an elliptical covariance matrix, chosen by PCA: that is, we used a spherical covariance after projecting onto the eigenvectors of the covariance matrix of the observations and scaling by the square roots of the eigenvalues. We chose the bandwidth manually, by a coarse search. However, as we demonstrated in Chapter 4 the exact details of kernel choice are not an essential feature of our algorithm: any smooth kernel could suffice (and, in fact, many different types of features could be used).

One of the nice properties of this choice of features is that, the kernel density estimate (KDE) of the observation probability density function (PDF) is a convex combination of these kernels; since each kernel integrates to 1, this estimator also integrates to 1. KDE theory [91] tells us that, with the correct kernel weights, as the number of kernel centers and the number of samples go to infinity and the kernel bandwidth goes to zero (at appropriate rates), the KDE estimator converges to the observation PDF in L_1 norm. The kernel density estimator is completely determined by the normalized vector of kernel weights; therefore, if we can estimate this vector accurately, our estimate of the observation PDF will converge to the observation PDF as well. Hence our goal is to predict the correct expected value of this normalized kernel vector given all past observations.

Given these features, we can write our latent-state update in the continuous-observation case in the form of Equation 4.2c, using a matrix B_{ao}; however, rather than learning each of the uncountably-many B_{ao} matrices separately, we learn one base operator per kernel center, and use convex combinations of these base operators to compute observable operators as needed. For details on practical aspects of learning with continuous observations and these features, see Boots et al. [14].

8.3.3 Learning a Model

We learn our model from a sample of 10000 short trajectories, each containing 7 action-observation pairs. We generate each trajectory by starting from a uniformly randomly sampled position in the environment and executing a uniform random sequence of actions. In each trajectory, we consider the first 3 actions the "past," the 4th action the "present," and the last 3 actions the "future." (The initial distribution ω is, therefore, the distribution obtained by initializing uniformly and taking 3 random actions.) We used the first $l = 2000$ trajectories to generate kernel centers, and the remaining $w = 8000$ to estimate the matrices $\mu_{\mathcal{H}}$, $\Sigma_{\mathcal{T},\mathcal{H}}$, and $\Sigma_{\mathcal{T}^+,ao,\mathcal{H}}$.

To define these matrices, we need to specify a set of indicative features, a set of observation kernel centers, and a set of characteristic features. We use Gaussian kernels to define our indicative and characteristic features, in a similar manner to the Gaussian kernels described above for observations; our analysis allows us to use arbitrary indicative and characteristic features, but we found Gaussian kernels to be convenient and effective. Note that the resulting features over tests and histories are just *features*; unlike the kernel centers defined over observations, there is no need to let the kernel width approach zero, since we are not attempting to learn accurate PDFs over the histories and tests in $\mathcal{H}$ and $\mathcal{T}$.

In more detail, we define a set of 2000 *indicative kernels*, each one centered at a sequence of 3 observations from the initial segment of one of our trajectories. We choose the kernel covariance using PCA on these sequences of observations, just as described for single observations in Section 8.3.2. We then generate our indicative features for a new sequence of three observations by evaluating each indicative kernel at the new sequence, and normalizing so that the vector of features sums to one. Similarly, we define 2000 *characteristic kernels*, each one centered at a sequence of 3 observations from the end of one of our sample trajectories, choose a kernel covariance, and define our characteristic feature vector by evaluating each kernel at a new observation sequence and normalizing. Finally, we define 500 *observation kernels*, each one centered at a single observation from the middle of one of our sample trajectories, and replace each observation by its corresponding vector of normalized kernel weights. Next, we construct the matrices $\widehat{\mu}_{\mathcal{H}}$, $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$, and $\widehat{\Sigma}_{\mathcal{T}^+,ao,\mathcal{H}}$ as the empirical expectations over our 8000 training trajectories according to the equations in Section 4.2. Finally we chose $|\mathcal{Q}'| = 5$ as the dimension of our PSR, the smallest dimension that was able to produce high quality policies (see Section 8.3.5 below).

8.3.4 Qualitative Evaluation

Having learned the parameters of the PSR according to the algorithm in Section 4.2, we can use the model for prediction, filtering, and planning in the autonomous robot domain. We first evaluated the model *qualitatively* by projecting the sets of histories in the training data onto the learned PSR state space: $\widehat{U}^\top \phi^{\mathcal{H}}_{1:w}$. We colored each data point according to the average of the red, green, and blue components of the highest probability observation following the projected history. The features of the low dimensional embedding clearly capture the topology of the major features of the robot's visual environment (Figure 8.1(C–D)), and continuous paths in the environment translate into continuous paths in the latent space (Figure 8.1(F)). For example, positions near different walls are mapped to different "spines" in the star-shaped embedding of the state space (Figure 8.1(C)).

8.3.5 Planning in the Learned Model

To test the quality of the learned model, we set up a navigation problem where the robot was required to plan to reach a goal image (looking directly at the blue wall). We specified a large reward (1000) for this observation, a reward of -1 for colliding with a wall, and 0 for every other observation. We learned a reward function by linear regression from the embedded histories $\widehat{U}^\top \phi^{\mathcal{H}}_{1:w}$ to the observed immediate rewards. We used the learned reward function to compute an approximate state-action value function via the PSR extension of the Perseus variant of PBVI [43, 45, 76, 102] with discount factor $\gamma = .8$, a prediction horizon of 10 steps, and with the 8000 embedded histories as the set of belief points. The learned value function is displayed in Figure 8.1(E). When faced with a new 3-step history, we computed an initial belief by starting with b_* and tracking for 3 action-observation pairs, and from then on executed the greedy policy for our learned value function while updating our belief with each new observation. Examples of paths planned in the learned model are presented in Figure 8.1(F); the same paths are shown in geometric space in Figure 8.1(G). (Recall that the robot only has access to images, *never* its own position.)

The reward function encouraged the robot to navigate to a specific set of points in the environment, so the planning problem can be viewed as a shortest path problem. Even though we don't encode this intuition into our algorithm, we can use it to quantitatively evaluate the performance of the policy in the original system. First we randomly sampled 100 initial histories in the environment and asked the robot to plan paths for each based on its learned policy. We compared the number of actions taken both to a random policy and to the *optimistic path*, calculated by A* search in the robot's configuration space given the *true underlying position*. Note that this comparison is somewhat unfair: in order to achieve the same cost as the optimistic path, the robot would have to know its true underlying position, the dynamics would have to be deterministic, all rotations would have to be instantaneous, and the algorithm would need an unlimited amount of training data. Nonetheless, the results (Figure 8.1(H)) indicate that the performance of the PSR policy is close to this optimistic bound. We think that this finding is remarkable, especially given that previous approaches have encountered significant difficulty modeling continuous domains [47] and domains with similarly high levels of complexity [122].

8.4 Conclusions

In Chapter 3 and Chapter 4 we presented a novel *consistent* subspace identification algorithms that simultaneously solve the *discovery* and *learning* problems for PSRs (and therefore POMDPs). In this chapter we showed how point-based approximate planning techniques can be used to solve the *planning* problem in a learned model. We demonstrated the representational capacity of our model and the effectiveness of our learning algorithm by learning a compact model from simulated autonomous robot vision data. Finally, we closed the loop by successfully planning with the learned models. To our knowledge this is the first instance of learning a model and successfully planning in the learned model for a simulated robot in a nonlinear, non-Gaussian, partially observable environment of this complexity using a consistent algorithm. We compare the policy generated by our model to a bound on the best possible value, and determine that our policy is

close to optimal.

We believe that spectral PSR learning algorithms can increase the scope of planning under uncertainty for autonomous agents in previously intractable scenarios. We believe that this improvement is partly due to the greater representational power of PSRs as compared to POMDPs, and partly due to the efficient and statistically consistent nature of the learning method.

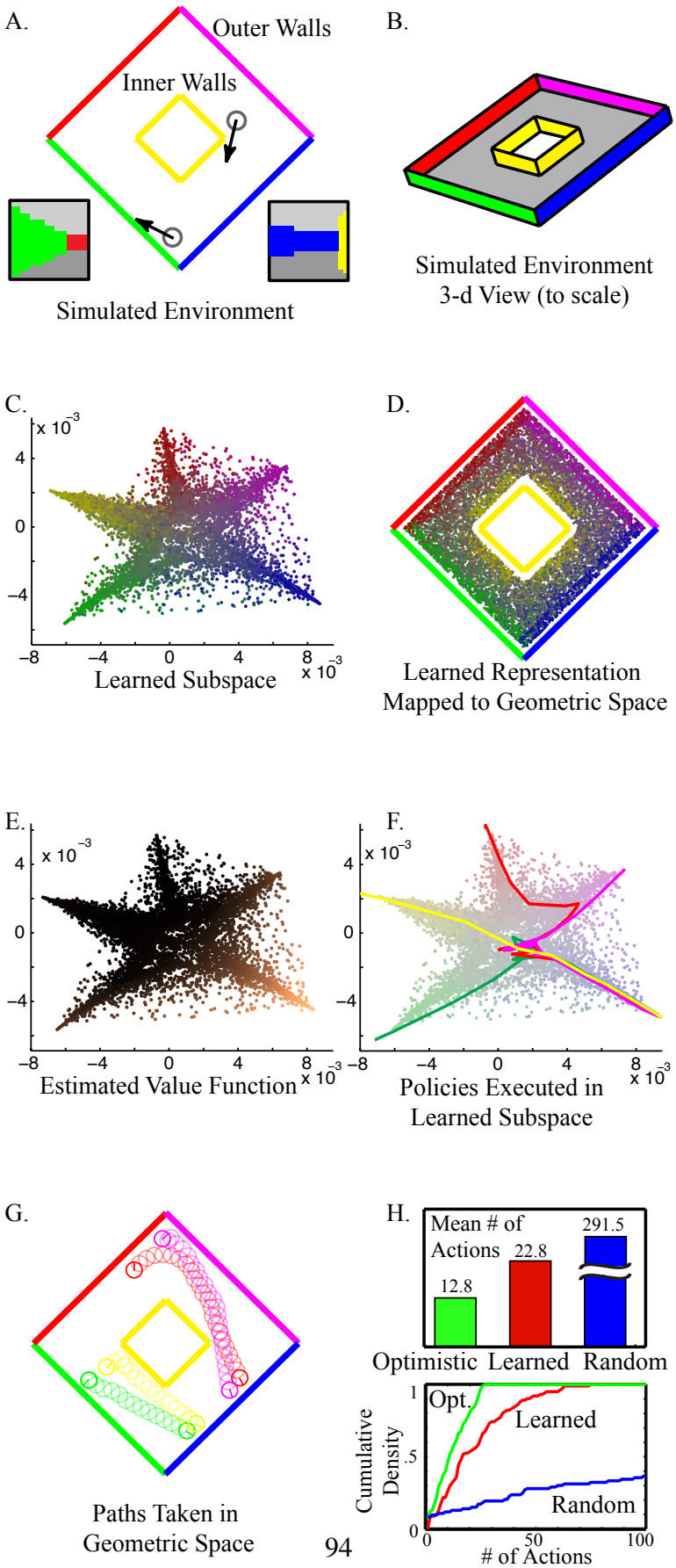
A.
Outer Walls
Inner Walls
Simulated Environment
B.
Simulated Environment
3-d View (to scale)
C.
x 10^-3
4
0
−4
−8
−4
0
4
8
x 10^-3
Learned Subspace
D.
Learned Representation
Mapped to Geometric Space
E.
x 10^-3
Estimated Value Function
F.
Policies Executed in
Learned Subspace
G.
Paths Taken in
Geometric Space
H.
Mean # of
Actions
12.8
22.8
291.5
Optimistic
Learned
Random
Cumulative
Density
Opt.
Learned
Random
1
.5
0
50
100
# of Actions

Figure 8.1: Learning and Planning in the Autonomous Robot Domain. (A) The robot uses visual sensing to traverse a square domain with multi-colored walls and a central obstacle. Examples of images recorded by the robot occupying two different positions in the environment are shown at the bottom of the figure. (B) A to-scale 3-dimensional view of the environment. (C) The 2nd and 3rd dimension of the learned subspace (the first dimension primarily contained normalization information). Each point is the embedding of a single history, displayed with color equal to the average RGB color in the first image in the highest probability test. (D) The same points in (C) projected onto the environment's geometric space. (E) The value function computed for each embedded point; lighter indicates higher value. (F) Policies executed in the learned subspace. The red, green, magenta, and yellow paths correspond to the policy executed by a robot with starting positions facing the red, green, magenta, and yellow walls respectively. (G) The paths taken by the robot in geometric space while executing the policy. Each of the paths corresponds to the path of the same color in (F). The darker circles indicate the starting and ending positions, and the tick-mark indicates the robot's orientation. (H) Analysis of planning from 100 randomly sampled start positions to the target image (facing blue wall). In the bar graph: the mean number of actions taken by the optimistic solution found by A* search in configuration space (left); the mean number taken by the policy found by Perseus in the learned model (center); and the mean number taken by a random policy (right). The line graph illustrates the cumulative density of the number of actions given the optimal, learned, and random policies.

Chapter 9

Reinforcement Learning: Predictive State Temporal Difference Learning

9.1 Introduction

In this chapter we will show how to use spectral learning algorithms to enhance least squares temporal difference learning. Specifically, we examine the problem of estimating a policy's value function within a decision process in a high dimensional and partially-observable environment, when the parameters of the process are unknown: i.e. we only have access to trajectories of observations and rewards sampled from the process. In this situation, a common strategy is to employ a *linear architecture* and represent the value function as a linear combination of *features* of (sequences of) observations. A popular family of model-free algorithms called temporal difference (TD) algorithms [105] can then be used to estimate the parameters of the value function. Least-squares TD (LSTD) algorithms [17, 19, 59] exploit the linearity of the value function to find the optimal parameters in a least-squares sense from time-adjacent samples of features.

Unfortunately, choosing a good set of features is hard. The features must be predictive of future reward, and the set of features must be small relative to the amount of training data, or TD learning will be prone to overfitting. The problem of selecting a small set of reasonable features has been approached from a number of different perspectives. In many domains, the features are selected by hand according to expert knowledge; however, this task can be difficult and time consuming in practice. Therefore, a considerable amount of research has been devoted to the problem of automatically identifying features that support value function approximation.

Much of this research is devoted to finding sets of features when the dynamical system is known, but the state space is large and difficult to work with. For example, in a large fully observable Markov decision process (MDP), it is often easier to estimate the value function from a low dimensional set of features than by using state directly. So, several approaches attempt to automatically discover a small set of features from a given larger description of an MDP, e.g., by using a spectral analysis of the state-space transition graph to discover a low-dimensional feature set that preserves the graph structure [46, 64, 65].

Partially observable Markov decision processes (POMDPs) extend MDPs to situations where the state is not directly observable [3, 22, 97]. In this circumstance, an agent can plan using a

continuous *belief state* with dimensionality equal to the number of hidden states in the POMDP. When the number of hidden states is large, dimensionality reduction in POMDPs can be achieved by projecting a high dimensional belief space to a lower dimensional one; of course, the difficulty is to find a projection which preserves decision quality. Strategies for finding good projections include value-directed compression [78] and non-negative matrix factorization [60, 110]. The resulting model after compression is a Predictive State Representation (PSR) [61, 93], an Observable Operator Model [44], or a multiplicity automaton [30]. Moving to one of these representations can often compress a POMDP by a large factor with little or no loss in accuracy: examples exist with arbitrarily large lossless compression factors, and in practice, we can often achieve large compression ratios with little loss.

The drawback of all of the approaches enumerated above is that they first assume that the dynamical system model is known, and only then give us a way of finding a compact representation and a value function. In practice, we would like to be able to find a good set of features, *without prior knowledge of the system model*. Kolter and Ng [55] contend with this problem from a sparse feature selection standpoint. Given a large set of possibly-relevant features of observations, they proposed augmenting LSTD by applying an L_1 penalty to the coefficients, forcing LSTD to select a sparse set of features for value function estimation. The resulting algorithm, LARS-TD, works well in certain situations (for example, see Section 9.4.1), but only if our original large set of features contains a small subset of highly-relevant features.

Recently, Parr et al. looked at the problem of value function estimation from the perspective of both model-free and model-based reinforcement learning [75]. The model-free approach estimates a value function directly from sample trajectories, i.e., from sequences of feature vectors of visited states. The model-based approach, by contrast, first learns a model and then computes the value function from the learned model. Parr et al. compared LSTD (a model-free method) to a model-based method in which we first learn a linear model by viewing features as a proxy for state (leading to a linear transition matrix that predicts future features from past features), and then compute a value function from this approximate model. Parr et al. demonstrated that these two approaches compute exactly the same value function [75], formalizing a fact that has been recognized to some degree before [17].

In this chapter, we build on this insight, while simultaneously finding a compact set of features using spectral system identification ideas from Chapter 3 and Chapter 4. First, we look at the problem of improving LSTD from a model-free predictive-bottleneck perspective: given a large set of features of history, we devise a new TD method called Predictive State Temporal Difference (PSTD) learning that estimates the value function through a bottleneck that preserves only *predictive* information (Section 9.3.2). Intuitively, this approach has some of the same benefits as LARS-TD: by finding a small set of predictive features, we avoid overfitting and make learning more data-efficient. However, our method differs in that we identify a small *subspace* of features instead of a sparse *subset* of features. Hence, PSTD and LARS-TD are applicable in different situations: as we show in our experiments below, PSTD is better when we have many marginally-relevant features, while LARS-TD is better when we have a few highly-relevant features hidden among many irrelevant ones.

Second, we look at the problem of value function estimation from a model-based perspective (Section 9.3.4). Instead of learning a linear transition model from features, as in [75], we use subspace identification [14, 84] to learn a PSR from our samples. Then we compute a value

function via the Bellman equations for our learned PSR. This new approach has a substantial benefit: while the linear feature-to-feature transition model of [75] does not seem to have any common uses outside that paper, PSRs have been proposed numerous times on their own merits (including being invented independently at least three times), and are a strict generalization of POMDPs.

Just as Parr et al. showed for the two simpler methods, we show that our two improved methods (model-free and model-based) are equivalent. This result yields some appealing theoretical benefits: for example, coefficients of PSTD features can be explicitly interpreted as a statistically consistent estimate of the true underlying system state. And, the feasibility of finding the true value function can be shown to depend on the *linear dimension* of the dynamical system, or equivalently, the dimensionality of the predictive state representation—*not* on the cardinality of the POMDP state space. Therefore our representation is naturally "compressed" in the sense of [78], speeding up convergence.

The improved methods also yield practical benefits; we demonstrate these benefits with several experiments. First, we compare PSTD to LSTD and LARS-TD on a synthetic example using different sets of features to illustrate the strengths and weaknesses of each algorithm. Next, we apply PSTD to a difficult optimal stopping problem for pricing high-dimensional financial derivatives. A significant amount of work has gone into hand tuning features for this problem. We show that, if we add a large number of weakly relevant features to these hand-tuned features, PSTD can find a predictive subspace which performs much better than competing approaches, improving on the best previously reported result for this particular problem by a substantial margin.

The theoretical and empirical results reported here suggest that, for many applications where LSTD is used to compute a value function, PSTD can be simply substituted to produce better results.

9.2 Value Function Approximation

The notion of a value function is of central importance in reinforcement learning: for a given policy π, the value of state s is defined as the expected discounted sum of rewards obtained when starting in state s and following policy π, $J^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t) \mid s_0 = s, \pi\right]$. It is well known that the value function must obey the Bellman equation

$$J^\pi(s) = \mathcal{R}(s) + \gamma \sum_{s'} J^\pi(s') \mathbb{P}[s' \mid s, \pi(s)] \tag{9.1}$$

If we know the transition function T, and if the set of states $\mathcal{S}$ is sufficiently small, we can use (9.1) directly to solve for the value function J^π. We can then execute the *greedy policy* for J^π, setting the action at each state to maximize the right-hand side of (9.1).

However, we consider instead the harder problem of estimating the value function when s is a *partially observable* latent variable, and when the transition function T is unknown. In this situation, we receive information about s through observations from a finite set O. Our state (i.e., the information which we can use to make decisions) is not an element of $\mathcal{S}$ but a *history* (an ordered sequence of action-observation pairs $h_t = a_1, o_1, \ldots, a_{t-1}, o_{t-1}$ that have been executed and observed prior to time t). If we knew the transition model T, we could use h_t to infer a

belief distribution over $\mathcal{S}$, and use that belief (or a compression of that belief) as a state instead; below, we will discuss how to learn a compressed belief state. Because of partial observability, we can only hope to predict expected reward conditioned on history, $\mathcal{R}(h_t) = \mathbb{E}[\mathcal{R}(s) \mid h_t]$, and we must choose actions as a function of history, $\pi(h_t)$ instead of $\pi(s)$.

Let $\mathcal{H}$ be the set of all possible histories. $\mathcal{H}$ is often very large or infinite, so instead of finding a value separately for each history, we focus on value functions that are linear in *features* of histories

$$J^\pi(s) \approx w^\top \phi^{\mathcal{H}}(h_t) \tag{9.2}$$

Here $w \in \mathbb{R}^j$ is a parameter vector and $\phi^{\mathcal{H}}(h_t) \in \mathbb{R}^j$ is a feature vector for a history h_t. So, we can rewrite the Bellman equation as

$$w^\top \phi^{\mathcal{H}}(h_t) \approx \mathcal{R}(h_t) + \gamma \sum_{o \in O} w^\top \phi^{\mathcal{H}}(h_t \pi o) \mathbb{P}[h_t \pi o \mid h_t \pi] \tag{9.3}$$

where $h\pi o$ is history h extended by taking action $\pi(h_t)$ and observing o.

9.2.1 Least Squares Temporal Difference Learning

In general we don't know the transition probabilities $\mathbb{P}[h_t \pi o \mid h_t]$, but we do have samples of state features $\phi_t^{\mathcal{H}} = \phi^{\mathcal{H}}(h_t)$, next-state features $\phi_{t+1}^{\mathcal{H}} = \phi^{\mathcal{H}}(h_{t+1})$, and immediate rewards $\mathcal{R}_t = \mathcal{R}(h_t)$. We can thus *estimate* the Bellman equation

$$w^\top \phi_{1:k}^{\mathcal{H}} \approx \mathcal{R}_{1:k} + \gamma w^\top \phi_{2:k+1}^{\mathcal{H}} \tag{9.4}$$

(Here we have used the notation $\phi_{1:k}^{\mathcal{H}}$ to mean the matrix whose columns are $\phi_t^{\mathcal{H}}$ for $t = 1 \ldots k$.) We can immediately attempt to estimate the parameter w by solving the linear system in the least squares sense: $\hat{w}^\top = \mathcal{R}_{1:k} \left(\phi_{1:k}^{\mathcal{H}} - \gamma \phi_{2:k+1}^{\mathcal{H}}\right)^\dagger$, where † indicates the Moore–Penrose pseudo-inverse. However, this solution is biased [19], since the independent variables $\phi_t^{\mathcal{H}} - \gamma \phi_{t+1}^{\mathcal{H}}$ are *noisy* samples of the expected difference $\mathbb{E}[\phi^{\mathcal{H}}(h_t) - \gamma \sum_{o \in O} \phi^{\mathcal{H}}(h_t \pi o) \mathbb{P}[h_t \pi o \mid h_t]]$. In other words, estimating the value function parameters w is an *error-in-variables* problem.

The *least squares temporal difference* (LSTD) algorithm provides a consistent estimate of the independent variables by right multiplying the approximate Bellman equation (Equation 9.4) by ${\phi_t^{\mathcal{H}}}^\top$. The quantity ${\phi_t^{\mathcal{H}}}^\top$ can be viewed as an *instrumental* variable [19], i.e., a measurement that is correlated with the true independent variables, but uncorrelated with the noise in our estimates of these variables.[1] The value function parameter w may then be estimated as follows:

$$\hat{w}^\top = \frac{1}{k} \sum_{t=1}^{k} \mathcal{R}_t {\phi_t^{\mathcal{H}}}^\top \left(\frac{1}{k} \sum_{t=1}^{k} \phi_t^{\mathcal{H}} {\phi_t^{\mathcal{H}}}^\top - \frac{\gamma}{k} \sum_{t=1}^{k} \phi_{t+1}^{\mathcal{H}} {\phi_t^{\mathcal{H}}}^\top \right)^{-1} \tag{9.5}$$

As the amount of data k increases, the empirical covariance matrices $\phi_{1:k}^{\mathcal{H}} {\phi_{1:k}^{\mathcal{H}}}^\top / k$ and $\phi_{2:k+1}^{\mathcal{H}} {\phi_{1:k}^{\mathcal{H}}}^\top / k$ converge with probability 1 to their population values, and so our estimate of the matrix to be inverted in (9.5) is consistent. Therefore, as long as this matrix is nonsingular, our estimate of the inverse is also consistent, and our estimate of w therefore converges to the true parameters with probability 1.

[1] The LSTD algorithm can also be theoretically justified as the result of an application of the Bellman operator followed by an orthogonal projection back onto the row space of $\phi^{\mathcal{H}}$ [59].

9.3 Predictive Features

Although LSTD provides a consistent estimate of the value function parameters w, in practice, the potential size of the feature vectors can be a problem. If the number of features is large relative to the number of training samples, then the estimation of w is prone to overfitting. This problem can be alleviated by choosing some small set of features that only contain information that is relevant for value function approximation. However, with the exception of LARS-TD [55], there has been little work on the problem of how to select features automatically for value function approximation when the system model is unknown; and of course, manual feature selection depends on not-always-available expert guidance.

We approach the problem of finding a good set of features from a *bottleneck* perspective. That is, given some signal from history, in this case a large set of features, we would like to find a compression that preserves only relevant information for predicting the value function J^π. This idea, finding a set of *predictive* features of the future, is directly related to spectral identification of PSRs. In particular, we think of predictions of the future as tests (Chapter 3) or, more generally *characteristic features* (Chapter 4). Here we will use the terms "characteristic features" and "features of the future" interchangeably.

9.3.1 Finding Predictive Features Through a Bottleneck

In order to find a *predictive* feature compression, we first need to determine what we would like to predict; i.e. what characteristic features to choose. Since we are interested in value function approximation, the most relevant prediction is the value function itself; so, we could simply try to predict total future discounted reward given a history. Unfortunately, total discounted reward has high variance, so unless we have a lot of data, learning will be difficult.

We can reduce variance by including other prediction tasks as well. For example, predicting *individual* rewards at future time steps, while not strictly necessary to predict total discounted reward, seems highly relevant, and gives us much more immediate feedback. Similarly, future *observations* hopefully contain information about future reward, so trying to predict observations can help us predict reward better. Finally, in any specific RL application, we may be able to add *problem-specific* prediction tasks that will help focus our attention on relevant information: for example, in a path-planning problem, we might try to predict which of several goal states we will reach (in addition to how much it will cost to get there).

We can represent all of these prediction tasks as features of the future: e.g., to predict which goal we will reach, we add a distinct observation at each goal state, or to predict individual rewards, we add individual rewards as observations.[2] We will write $\phi_t^{\mathcal{T}}$ for the vector of all features of the "future at time t," i.e., events starting at time $t+1$ and continuing forward.

So, instead of remembering a large *arbitrary* set of features of history, we want to find a small subspace of features of history that is relevant for predicting features of the future. We will call

[2]If we don't wish to reveal extra information by adding additional observations, we can instead add the corresponding feature *predictions* as observations; these predictions, by definition, reveal no additional information. To save the trouble of computing these predictions, we can use realized feature values rather than predictions in our learning algorithms below, at the cost of some extra variance: the expectation of the realized feature value is the same as the expectation of the predicted feature value.

this subspace a *predictive compression*, and we will write the value function as a linear function of only the predictive compression of features.

To find our predictive compression, we will use *reduced-rank regression* [83]. We define the following empirical covariance matrices between features of the future and features of histories:

$$\widehat{\Sigma}_{\mathcal{T},\mathcal{H}} = \frac{1}{k}\sum_{t=1}^{k} \phi_t^{\mathcal{T}} \phi_t^{\mathcal{H}\top} \qquad \widehat{\Sigma}_{\mathcal{H},\mathcal{H}} = \frac{1}{k}\sum_{t=1}^{k} \phi_t^{\mathcal{H}} \phi_t^{\mathcal{H}\top} \tag{9.6}$$

Let $L_{\mathcal{H}}$ be the lower triangular Cholesky factor of $\widehat{\Sigma}_{\mathcal{H},\mathcal{H}}$. Then we can find a predictive compression of histories by a singular value decomposition (SVD) of the weighted covariance: write

$$\mathcal{U}\mathcal{D}\mathcal{V}^\top \approx \widehat{\Sigma}_{\mathcal{T},\mathcal{H}} L_{\mathcal{H}}^{-\mathsf{T}} \tag{9.7}$$

for a truncated SVD [36] of the weighted covariance, where $\mathcal{U}$ are the left singular vectors, $\mathcal{V}^\top$ are the right singular vectors, and $\mathcal{D}$ is the diagonal matrix of singular values. The number of columns of $\mathcal{U}$, $\mathcal{V}$, or $\mathcal{D}$ is equal to the number of retained singular values.[3] Then we define

$$\widehat{U} = \mathcal{U}\mathcal{D}^{1/2} \tag{9.8}$$

to be the mapping from the low-dimensional compressed space up to the high-dimensional space of features of the future.

Given $\widehat{U}$, we would like to find a compression operator V that optimally predicts features of the future through the bottleneck defined by $\widehat{U}$. The least squares estimate can be found by minimizing the loss

$$\mathcal{L}(V) = \left\| \phi_{1:k}^{\mathcal{T}} - \widehat{U} V \phi_{1:k}^{\mathcal{H}} \right\|_F^2 \tag{9.9}$$

where $\|\cdot\|_F$ denotes the Frobenius norm. We can find the minimum by taking the derivative of this loss with respect to V, setting it to zero, and solving for V (see Appendix, Section 12.1.1 for details), giving us:

$$\widehat{V} = \arg\min_V \mathcal{L}(V) = \widehat{U}^\top \widehat{\Sigma}_{\mathcal{T},\mathcal{H}} (\widehat{\Sigma}_{\mathcal{H},\mathcal{H}})^{-1} \tag{9.10}$$

By weighting different features of the future differently, we can change the approximate compression in interesting ways. For example, as we will see in Section 9.3.4, scaling up future reward by a constant factor results in a *value-directed compression*—but, unlike previous ways

[3] If our empirical estimate $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}$ were exact, we could keep all nonzero singular values to find the smallest possible compression that does not lose any predictive power. In practice, though, there will be noise in our estimate, and $\widehat{\Sigma}_{\mathcal{T},\mathcal{H}} L_{\mathcal{H}}^{-\mathsf{T}}$ will be full rank. If we know the true rank n of $\Sigma_{\mathcal{T},\mathcal{H}}$, we can choose the first n singular values to define a subspace for compression. Or, we can choose a smaller subspace that results in an *approximate* compression: by selectively dropping columns of $\mathcal{U}$ corresponding to small singular values, we can trade off compression against predictive power. Directions of larger variance in features of the future correspond to larger singular values in the SVD, so we minimize prediction error by truncating the smallest singular values. By contrast with an SVD of the unscaled covariance, we do not attempt to minimize reconstruction error for features of history, since features of history are standardized when we multiply by the inverse Cholesky factor.

to find value-directed compressions [78], we do not need to know a model of our system ahead of time. For another example, define $L_{\mathcal{T}}$ to be the lower triangular Cholesky factor of the empirical covariance of future features $\widehat{\Sigma}_{\mathcal{T},\mathcal{T}}$. Then, if we scale features of the future by $L_{\mathcal{T}}^{-\top}$, the singular value decomposition will preserve the largest possible amount of mutual information between features of the future and features of history. This is equivalent to canonical correlation analysis [41, 95], and the matrix $\mathcal{D}$ becomes a diagonal matrix of canonical correlations between futures and histories.

9.3.2 Predictive State Temporal Difference Learning

Now that we have found a predictive compression operator $\widehat{V}$ via Equation 9.10, we can replace the features of history $\phi_t^{\mathcal{H}}$ with the compressed features $\widehat{V}\phi_t^{\mathcal{H}}$ in the Bellman recursion, Equation 9.4. Doing so results in the following approximate Bellman equation:

$$w^\top \widehat{V}\phi_{1:k}^{\mathcal{H}} \approx \mathcal{R}_{1:k} + \gamma w^\top \widehat{V}\phi_{2:k+1}^{\mathcal{H}} \tag{9.11}$$

The least squares solution for w is still prone to an error-in-variables problem. The variable $\phi^{\mathcal{H}}$ is still correlated with the true independent variables and uncorrelated with noise, and so we can again use it as an instrumental variable to unbias the estimate of w. Define the additional empirical covariance matrices:

$$\widehat{\Sigma}_{\mathcal{R},\mathcal{H}} = \frac{1}{k}\sum_{t=1}^{k} \mathcal{R}_t \phi_t^{\mathcal{H}\top} \qquad \widehat{\Sigma}_{\mathcal{H}^+,\mathcal{H}} = \frac{1}{k}\sum_{t=1}^{k} \phi_{t+1}^{\mathcal{H}} \phi_t^{\mathcal{H}\top} \tag{9.12}$$

Then, the corrected Bellman equation is:

$$\hat{w}^\top \widehat{V}\widehat{\Sigma}_{\mathcal{H},\mathcal{H}} = \widehat{\Sigma}_{\mathcal{R},\mathcal{H}} + \gamma \hat{w}^\top \widehat{V}\widehat{\Sigma}_{\mathcal{H}^+,\mathcal{H}} \tag{9.13}$$

and solving for $\hat{w}$ gives us the Predictive State Temporal Difference (PSTD) learning algorithm:

$$\hat{w}^\top = \widehat{\Sigma}_{\mathcal{R},\mathcal{H}} \left(\widehat{V}\widehat{\Sigma}_{\mathcal{H},\mathcal{H}} - \gamma \widehat{V}\widehat{\Sigma}_{\mathcal{H}^+,\mathcal{H}} \right)^\dagger \tag{9.14}$$

So far we have provided some intuition for why predictive features should be better than arbitrary features for temporal difference learning. Below we will show an additional benefit: the model-free algorithm in Equation 9.14 is, under some circumstances, equivalent to a model-based value function approximation method which uses subspace identification to learn Predictive State Representations [14, 84].

9.3.3 PSRs

We are ow almost ready to show that the model-free PSTD learning algorithm introduced in Section 9.3.2 is *equivalent* to a model-based algorithm built around PSR learning. In order to do so, we will use a number of equations from Chapter 4, specifically the definitions of moments given in Equations 4.1 and PSR parameter estimates given in Equations 4.2. In addition we define a few more moments and parameters:

First we define $\Sigma_{\mathcal{H}^+,ao,\mathcal{H}}$, a *set* of matrices, one for each action-observation pair, that represent the covariance between features of history before and after taking action a and observing o. In the following, we assume that histories $h_t \sim \omega$.

$$\begin{aligned}
\widehat{\Sigma}_{\mathcal{H}^+,ao,\mathcal{H}} &\stackrel{\text{def}}{=} \frac{1}{k}\sum_{t=1}^{k} \phi^{\mathcal{H}}_{t+1}\mathbb{I}(o_t = o){\phi^{\mathcal{H}}_t}^\top \\
\Sigma_{\mathcal{H}^+,ao,\mathcal{H}} &\stackrel{\text{def}}{=} \mathbb{E}\left[\widehat{\Sigma}_{\mathcal{H}^+,ao,\mathcal{H}} \,\middle|\, a_t = a\right] \\
&= \mathbb{E}\left[\frac{1}{k}\sum_{t=1}^{k} \phi^{\mathcal{H}}_{t+1}\mathbb{I}(o_t = o){\phi^{\mathcal{H}}_t}^\top \,\middle|\, a_t = a\right] \qquad (9.15a)
\end{aligned}$$

Since the dimensions of each $\widehat{\Sigma}_{\mathcal{H}^+,ao,\mathcal{H}}$ are fixed, as $k \to \infty$ these empirical covariances converge to the true covariances $\Sigma_{\mathcal{H}^+,ao,\mathcal{H}}$ with probability 1. Next we define $\Sigma_{\mathcal{R},\mathcal{H}} \stackrel{\text{def}}{=} \mathbb{E}[\mathcal{R}_t {\phi^{\mathcal{H}}_t}^\top \mid h_t \sim \omega]$, and approximate the covariance (in this case a vector) of reward and features of history:

$$\begin{aligned}
\widehat{\Sigma}_{\mathcal{R},\mathcal{H}} &\stackrel{\text{def}}{=} \frac{1}{k}\sum_{t=1}^{k} \mathcal{R}_t {\phi^{\mathcal{H}}_t}^\top \\
\Sigma_{\mathcal{R},\mathcal{H}} &\stackrel{\text{def}}{=} \mathbb{E}\left[\widehat{\Sigma}_{\mathcal{R},\mathcal{H}}\right] \\
&= \mathbb{E}\left[\frac{1}{k}\sum_{t=1}^{k} \mathcal{R}_t {\phi^{\mathcal{H}}_t}^\top\right] \\
&= \mathbb{E}\left[\frac{1}{k}\sum_{t=1}^{k} \eta^\top x(h_t) {\phi^{\mathcal{H}}_t}^\top\right] \\
&= \eta^\top \mathbb{E}\left[\frac{1}{k}\sum_{t=1}^{k} x(h_t) {\phi^{\mathcal{H}}_t}^\top\right] \\
&= \eta^\top \Sigma_{X,\mathcal{H}} \qquad (9.15b)
\end{aligned}$$

Again, as $k \to \infty$, $\widehat{\Sigma}_{\mathcal{R},\mathcal{H}}$ converges to $\Sigma_{\mathcal{R},\mathcal{H}}$ with probability 1.

We now wish to use the above-defined matrices to learn a PSR from data. To do so we need to make a somewhat-restrictive assumption: we assume that our features of history are rich enough to determine the state of the system, i.e., the regression from $\phi^{\mathcal{H}}$ to s is exact: $s_t = \Sigma_{X,\mathcal{H}}\Sigma^{-1}_{\mathcal{H},\mathcal{H}}\phi^{\mathcal{H}}_t$. We discuss how to relax this assumption below in Section 9.3.5. We also need a matrix U such that $U^\top \Phi^{\mathcal{T}} \Gamma$ is invertible; with probability 1 a random matrix satisfies this condition, but as we will see below, it is useful to choose U via SVD of a scaled version of $\Sigma_{\mathcal{T},\mathcal{H}}$ as described in

Sec. 9.3.1. Using our assumptions we can show a useful identity for $\Sigma_{\mathcal{H}^+,ao,\mathcal{H}}$:

$$\begin{aligned}
\Sigma_{X,\mathcal{H}}\Sigma^{-1}_{\mathcal{H},\mathcal{H}}\Sigma_{\mathcal{H}^+,ao,\mathcal{H}} &= \mathbb{E}\left[\frac{1}{k}\sum_{t=1}^{k}\Sigma_{X,\mathcal{H}}\Sigma^{-1}_{\mathcal{H},\mathcal{H}}\phi^{\mathcal{H}}_{t+1}\mathbb{I}(o_t=o)\phi^{\mathcal{H}\top}_t \,\middle|\, a_t=a\right] \\
&= \mathbb{E}\left[\frac{1}{k}\sum_{t=1}^{k}x(h_{t+1})\mathbb{P}[o_t=o \mid a_t=a,h_t]\phi^{\mathcal{H}\top}_t\right] \\
&= \mathbb{E}\left[\frac{1}{k}\sum_{t=1}^{k}M_{ao}x(h_t)\phi^{\mathcal{H}\top}_t\right] \\
&= M_{ao}\Sigma_{X,\mathcal{H}} \qquad (9.16)
\end{aligned}$$

This identity is at the heart of our learning algorithm: it shows that $\Sigma_{\mathcal{H},ao,\mathcal{H}}$ contains a hidden copy of M_{ao}, the main PSR parameter that we need to learn. We would like to recover M_{ao} via Eq. 9.16, $M_{ao} = \Sigma_{X,\mathcal{H}}\Sigma^{-1}_{\mathcal{H},\mathcal{H}}\Sigma_{\mathcal{H}^+,ao,\mathcal{H}}\Sigma^{\dagger}_{X,\mathcal{H}}$; but of course we do not know $\Sigma_{X,\mathcal{H}}$. Fortunately, though, it turns out that we can use $U^\top\Sigma_{\mathcal{T},\mathcal{H}}$ as a stand-in, since this matrix differs from $\Sigma_{X,\mathcal{H}}$ only by an invertible transform.

$$\begin{aligned}
b_t &\stackrel{\text{def}}{=} U^\top\Sigma_{\mathcal{T},\mathcal{H}}(\Sigma_{\mathcal{H},\mathcal{H}})^{-1}\phi^{\mathcal{H}}_t \\
&= U^\top\Phi^{\mathcal{T}}\Gamma\Sigma_{X,\mathcal{H}}(\Sigma_{\mathcal{H},\mathcal{H}})^{-1}\phi^{\mathcal{H}}_t \\
&= (U^\top\Phi^{\mathcal{T}}\Gamma)s_t \qquad (9.17a)
\end{aligned}$$

$$\begin{aligned}
B_{ao} &\stackrel{\text{def}}{=} U^\top\Sigma_{\mathcal{T},\mathcal{H}}(\Sigma_{\mathcal{H},\mathcal{H}})^{-1}\Sigma_{\mathcal{H}^+,ao,\mathcal{H}}(U^\top\Sigma_{\mathcal{T},\mathcal{H}})^{\dagger} \\
&= U^\top\Phi^{\mathcal{T}}\Gamma\Sigma_{X,\mathcal{H}}(\Sigma_{\mathcal{H},\mathcal{H}})^{-1}\Sigma_{\mathcal{H}^+,ao,\mathcal{H}}(U^\top\Sigma_{\mathcal{T},\mathcal{H}})^{\dagger} \\
&= (U^\top\Phi^{\mathcal{T}}\Gamma)M_{ao}\,\Sigma_{X,\mathcal{H}}(U^\top\Sigma_{\mathcal{T},\mathcal{H}})^{\dagger} \\
&= (U^\top\Phi^{\mathcal{T}}\Gamma)M_{ao}(U^\top\Phi^{\mathcal{T}}\Gamma)^{-1}(U^\top\Phi^{\mathcal{T}}R)\Sigma_{X,\mathcal{H}}(U^\top\Sigma_{\mathcal{T},\mathcal{H}})^{\dagger} \\
&= (U^\top\Phi^{\mathcal{T}}\Gamma)M_{ao}(U^\top\Phi^{\mathcal{T}}\Gamma)^{-1} \qquad (9.17b)
\end{aligned}$$

$$\begin{aligned}
b_\eta^\top &\stackrel{\text{def}}{=} \Sigma_{\mathcal{R},\mathcal{H}}(U^\top\Sigma_{\mathcal{T},\mathcal{H}})^{\dagger} \\
&= \eta^\top\Sigma_{X,\mathcal{H}}(U^\top\Sigma_{\mathcal{T},\mathcal{H}})^{\dagger} \\
&= \eta^\top(U^\top\Phi^{\mathcal{T}}\Gamma)^{-1}(U^\top\Phi^{\mathcal{T}}\Gamma)\Sigma_{X,\mathcal{H}}(U^\top\Sigma_{\mathcal{T},\mathcal{H}})^{\dagger} \\
&= \eta^\top(U^\top\Phi^{\mathcal{T}}\Gamma)^{-1} \qquad (9.17c)
\end{aligned}$$

9.3.4 Predictive State Temporal Difference Learning Revisited

With the additional parameter definitions above it is not difficult to show that model-free PSTD learning is actually leveraging ideas from system identification to get good value function estimates. For a fixed policy π, a PSR's value function is a linear function of state, $J^\pi(s) = w^\top b$, and is the solution of the PSR Bellman equation [45]: for all b, $w^\top b = b_\eta^\top b + \gamma\sum_{o\in O} w^\top B_{\pi o} b$, or equivalently,

$$w^\top = b_\eta^\top + \gamma\sum_{o\in O} w^\top B_{\pi o} \qquad (9.18)$$

If we substitute in our learned PSR parameters from Equations 9.17(a–c), we get

$$\hat{w}^\top = \widehat{\Sigma}_{\mathcal{R},\mathcal{H}}(U^\top \widehat{\Sigma}_{\mathcal{T},\mathcal{H}})^\dagger + \gamma \sum_{o \in O} \hat{w}^\top U^\top \widehat{\Sigma}_{\mathcal{T},\mathcal{H}}(\widehat{\Sigma}_{\mathcal{H},\mathcal{H}})^{-1} \widehat{\Sigma}_{\mathcal{H},\pi o,\mathcal{H}}(U^\top \widehat{\Sigma}_{\mathcal{T},\mathcal{H}})^\dagger$$

$$\hat{w}^\top U^\top \widehat{\Sigma}_{\mathcal{T},\mathcal{H}} = \widehat{\Sigma}_{\mathcal{R},\mathcal{H}} + \gamma \hat{w}^\top U^\top \widehat{\Sigma}_{\mathcal{T},\mathcal{H}}(\widehat{\Sigma}_{\mathcal{H},\mathcal{H}})^{-1} \widehat{\Sigma}_{\mathcal{H}^+,\mathcal{H}}$$

since, by comparing Eqs. 9.15a and 9.12, we can see that $\sum_{o \in O} \widehat{\Sigma}_{\mathcal{H}^+,\pi o,\mathcal{H}} = \widehat{\Sigma}_{\mathcal{H}^+,\mathcal{H}}$. Now, suppose that we define $\widehat{U}$ and $\widehat{V}$ by Eqs. 9.8 and 9.10, and let $U = \widehat{U}$ as suggested above in Sec. 4.2. Then $U^\top \widehat{\Sigma}_{\mathcal{T},\mathcal{H}} = \widehat{V}\widehat{\Sigma}_{\mathcal{H},\mathcal{H}}$, and

$$\hat{w}^\top \widehat{V}\widehat{\Sigma}_{\mathcal{H},\mathcal{H}} = \widehat{\Sigma}_{\mathcal{R},\mathcal{H}} + \gamma \hat{w}^\top \widehat{V}\widehat{\Sigma}_{\mathcal{H}^+,\mathcal{H}}$$

$$\hat{w}^\top = \widehat{\Sigma}_{\mathcal{R},\mathcal{H}} \left(\widehat{V}\widehat{\Sigma}_{\mathcal{H},\mathcal{H}} - \gamma \widehat{V}\widehat{\Sigma}_{\mathcal{H}^+,\mathcal{H}} \right)^\dagger \tag{9.19}$$

Eq. 9.19 is exactly the PSTD algorithm (Eq. 9.14). So, we have shown that, if we learn a PSR by the subspace identification algorithm of Sec. 4.2 and then compute its value function via the Bellman equation, we get the exact same answer as if we had directly learned the value function via the model-free PSTD method. In addition to adding to our understanding of both methods, an important corollary of this result is that PSTD is a *statistically consistent* algorithm for PSR value function approximation—to our knowledge, the first such result for a TD method.

PSTD learning is related to value-directed compression of POMDPs [78]. If we learn a PSR from data generated by a POMDP, then the PSR state is exactly a linear compression of the POMDP state [84, 93]. The compression can be exact or approximate, depending on whether we include enough features of the future and whether we keep all or only some nonzero singular values in our bottleneck. If we include *only* reward as a feature of the future, we get a value-directed compression in the sense of Poupart and Boutilier [78]. If desired, we can *tune* the degree of value-directedness of our compression by scaling the relative variance of our features: the higher the variance of the reward feature compared to other features, the more value-directed the resulting compression will be. Our work significantly diverges from previous work on POMDP compression in one important respect: prior work assumes access to the true POMDP model, while we make no such assumption, and learn a compressed representation directly from data.

9.3.5 Insights from Subspace Identification

The close connection to subspace identification for PSRs provides additional insight into the temporal difference learning procedure. In Equation 9.17 we made the assumption that the features of history are rich enough to completely determine the state of the dynamical system. In fact, using theory developed in Chapter 4 and [14], it is possible to relax this assumption and instead assume that state is merely *correlated* with features of history. In this case, the value function parameter w can be estimated as $\hat{w}^\top = \widehat{\Sigma}_{\mathcal{R},\mathcal{H}}(\widehat{U}^\top \widehat{\Sigma}_{\mathcal{T},\mathcal{H}})^\dagger (I - \sum_{o \in O} \widehat{U}^\top \widehat{\Sigma}_{\mathcal{T}^+,ao,\mathcal{H}}(\widehat{U}^\top \widehat{\Sigma}_{\mathcal{T},\mathcal{H}})^\dagger)^\dagger = \widehat{\Sigma}_{\mathcal{R},\mathcal{H}}(\widehat{U}^\top \widehat{\Sigma}_{\mathcal{T},\mathcal{H}} - \sum_{o \in O} \widehat{U}^\top \widehat{\Sigma}_{\mathcal{T}^+,ao,\mathcal{H}})^\dagger$. Since we no longer assume that state is completely specified by features of history, we can no longer apply the learned value function to $\widehat{U}\Sigma_{\mathcal{T},\mathcal{H}}(\Sigma_{\mathcal{H},\mathcal{H}})^{-1}\phi_t$ at each time t. Instead we need to learn a full PSR model and *filter* with the model to estimate state.

9.4 Experimental Results

We designed several experiments to evaluate the properties of the PSTD learning algorithm. In the first set of experiments we look at the comparative merits of PSTD with respect to LSTD and LARS-TD when applied to the problem of estimating the value function of a reduced-rank POMDP. In the second set of experiments, we apply PSTD to a benchmark optimal stopping problem (pricing a fictitious financial derivative), and show that PSTD outperforms competing approaches.

9.4.1 Estimating the Value Function of a RR-POMDP

We evaluate the PSTD learning algorithm on a synthetic example derived from [90]. The problem is to find the value function of a policy in a partially observable Markov decision Process (POMDP). The POMDP has 4 latent states, but the policy's transition matrix is low rank: the resulting belief distributions can be represented in a 3-dimensional subspace of the original belief simplex. A reward of 1 is given in the first and third latent state and a reward of 0 in the other two latent states (see Appendix, Section 12.1.2). The system emits 2 possible observations, conflating information about the latent states.

We perform 3 experiments, comparing the performance of LSTD, LARS-TD, PSTD, and PSTD as formulated in Section 9.3.5 (which we call PSTD2) when different sets of features are used. In each case we compare the value function estimated by each algorithm to the true value function computed by $J^\pi = \mathcal{R}(I - \gamma T^\pi)^{-1}$.

In the first experiment we execute the policy π for 1000 time steps. We split the data into overlapping histories and tests of length 5, and sample 10 of these histories and tests to serve as centers for Gaussian radial basis functions. We then evaluate each basis function at every remaining sample. Then, using these features, we learned the value function using LSTD, LARS-TD, PSTD with linear dimension 3, and PSTD2 with linear dimension 3 (Figure 9.1(A)).[4] In this experiment, PSTD and PSTD2 both had lower mean squared error than the other approaches. For the second experiment, we added 490 random features to the 10 good features and then attempted to learn the value function with each of the 3 algorithms (Figure 9.1(B)). In this case, LSTD and PSTD both had difficulty fitting the value function due to the large number of irrelevant features in both tests and histories and the relatively small amount of training data. LARS-TD, designed for precisely this scenario, was able to select the 10 relevant features and estimate the value function better by a substantial margin. Surprisingly, in this experiment PSTD2 not only outperformed PSTD but bested even LARS-TD. For the third experiment, we increased the number of sampled features from 10 to 500. In this case, each feature was somewhat relevant, but the number of features was relatively large compared to the amount of training data. This situation occurs frequently in practice: it is often easy to find a large number of features that are at least somewhat related to state. PSTD and PSTD2 both outperform LARS-TD and each of these subspace and subset selection methods outperform LSTD by a large margin by *efficiently* estimating the value function (Figure 9.1(C)).

[4]Comparing LSTD and PSTD is straightforward; the two methods differ only by the compression operator $\widehat{V}$.

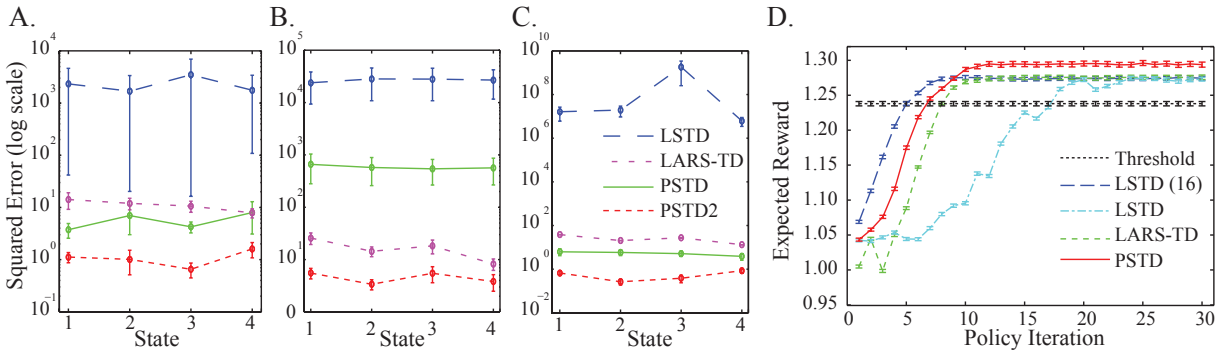

Figure 9.1: Experimental Results. Error bars indicate standard error. (A) Estimating the value function with a small number of informative features. PSTD and PSTD2 both do well. (B) Estimating the value function with a small set of informative features and a large set of random features. LARS-TD is designed for this scenario and dramatically outperforms PSTD and LSTD, however it does not outperform PSTD2. (C) Estimating the value function with a large set of semi-informative features. PSTD is able to determine a small set of compressed features that retain the maximal amount of information about the value function, outperforming LSTD by a very large margin. (D) Pricing a high-dimensional derivative via policy iteration. The y-axis is expected reward for the current policy at each iteration. The optimal threshold strategy (sell if price is above a threshold [115]) is in black, LSTD (16 canonical features) is in blue, LSTD (on the full 220 features) is cyan, LARS-TD (feature selection from set of 220) is in green, and PSTD (16 dimensions, compressing 220 features (16 + 204)) is in red.

9.4.2 Pricing A High-dimensional Financial Derivative

Derivatives are financial contracts with payoffs linked to the future prices of basic assets such as stocks, bonds and commodities. In some derivatives the contract holder has no choices, but in more complex cases, the contract owner must make decisions—e.g., with *early exercise* the contract holder can decide to terminate the contract at any time and receive payments based on prevailing market conditions. In these cases, the value of the derivative depends on how the contract holder acts. Deciding when to exercise is therefore an optimal stopping problem: at each point in time, the contract holder must decide whether to continue holding the contract or exercise. Such stopping problems provide an ideal testbed for policy evaluation methods, since we can easily collect a single data set which is sufficient to evaluate any policy: we just choose the "continue" action forever. (We can then evaluate the "stop" action easily in any of the resulting states, since the immediate reward is given by the rules of the contract, and the next state is the terminal state by definition.)

We consider the financial derivative introduced by Tsitsiklis and Van Roy [115]. The derivative generates payoffs that are contingent on the prices of a single stock. At the end of a given day, the holder may opt to exercise. At exercise the owner receives a payoff equal to the current price of the stock divided by the price 100 days beforehand. We can think of this derivative as a "psychic call": the owner gets to decide whether s/he would like to have bought an ordinary 100-day European call option, at the then-current market price, 100 days ago.

In our simulation (and unknown to the investor), the underlying stock price follows a geometric Brownian motion with volatility $\sigma = 0.02$ and continuously compounded short term growth rate $\rho = 0.0004$. Assuming stock prices fluctuate only on days when the market is open, these parameters correspond to an annual growth rate of $\sim 10\%$. In more detail, if w_t is a standard Brownian motion, then the stock price p_t evolves as $\nabla p_t = \rho p_t \nabla t + \sigma p_t \nabla w_t$, and we can summarize relevant state at the end of each day as a vector $x_t \in \mathbb{R}^{100}$, with $x_t = \left(\frac{p_{t-99}}{p_{t-100}}, \frac{p_{t-98}}{p_{t-100}}, \ldots, \frac{p_t}{p_{t-100}}\right)^\top$. The ith dimension $x_t(i)$ represents the amount a \$1 investment in a stock at time $t - 100$ would grow to at time $t - 100 + i$. This process is Markov and ergodic [23, 115]: x_t and x_{t+100} are independent and identically distributed. The immediate reward for exercising the option is $G(x) = x(100)$, and the immediate reward for continuing to hold the option is 0. The discount factor $\gamma = e^{-\rho}$ is determined by the growth rate; this corresponds to assuming that the risk-free interest rate is equal to the stock's growth rate, meaning that the investor gains nothing in expectation by holding the stock itself.

The value of the derivative, if the current state is x, is given by $V^*(x) = \sup_t \mathbb{E}[\gamma^t G(x_t) \mid x_0 = x]$. Our goal is to calculate an approximate value function $V(x) = w^\top \phi^{\mathcal{H}}(x)$, and then use this value function to generate a stopping time $\min\{t \,|\, G(x_t) \geq V(x_t)\}$. To do so, we sample a sequence of 1,000,000 states $x_t \in \mathbb{R}^{100}$ and calculate features $\phi^{\mathcal{H}}$ of each state. We then perform *policy iteration* on this sample, alternately estimating the value function under a given policy and then using this value function to define a new greedy policy "stop if $G(x_t) \geq w^\top \phi^{\mathcal{H}}(x_t)$."

Within the above strategy, we have two main choices: which features do we use, and how do we estimate the value function in terms of these features. For value function estimation, we used LSTD, LARS-TD, or PSTD. In each case we re-used our 1,000,000-state sample trajectory for all iterations: we start at the beginning and follow the trajectory as long as the policy chooses the "continue" action, with reward 0 at each step. When the policy executes the "stop" action, the reward is $G(x)$ and the next state's features are all 0; we then restart the policy 100 steps in the future, after the process has fully mixed. For feature selection, we are fortunate: previous researchers have hand-selected a "good" set of 16 features for this data set through repeated trial and error (see Appendix, Section 12.1.2 and [23, 115]). We greatly expand this set of features, then use PSTD to synthesize a small set of high-quality combined features. Specifically, we add the entire 100-step state vector, the squares of the components of the state vector, and several additional nonlinear features, increasing the total number of features from 16 to 220. We use histories of length 1, tests of length 5, and (for comparison's sake) we choose a linear dimension of 16. Tests (but not histories) were value-directed by reducing the variance of all features *except reward* by a factor of 100.

Figure 9.1D shows results. We compared PSTD (reducing 220 to 16 features) to LSTD with either the 16 hand-selected features or the full 220 features, as well as to LARS-TD (220 features) and to a simple thresholding strategy [115]. In each case we evaluated the final policy on 10,000 new random trajectories. PSTD outperformed each of its competitors, improving on the next best approach, LARS-TD, by 1.75 percentage points. In fact, PSTD performs better than the best previously reported approach [23, 115] by 1.24 percentage points. These improvements correspond to appreciable fractions of the risk-free interest rate (which is about 4 percentage points over the 100 day window of the contract), and therefore to significant arbitrage opportunities: an investor who doesn't know the best strategy will consistently undervalue the security, allowing

an informed investor to buy it for below its expected value.

9.5 Conclusion

In this chapter, we attack the feature selection problem for temporal difference learning. Although well-known temporal difference algorithms such as LSTD can provide asymptotically unbiased estimates of value function parameters in linear architectures, they can have trouble in finite samples: if the number of features is large relative to the number of training samples, then they can have high variance in their value function estimates. For this reason, in real-world problems, a substantial amount of time is spent selecting a small set of features, often by trial and error [23, 115].

To remedy this problem, we present the PSTD algorithm, a new approach to feature selection for TD methods, which demonstrates how insights from system identification can benefit reinforcement learning. PSTD automatically chooses a small set of features that are relevant for prediction and value function approximation. It approaches feature selection from a *bottleneck* perspective, by finding a small set of features that preserves only *predictive* information. Because of the focus on predictive information, the PSTD approach is closely connected to PSRs: under appropriate assumptions, PSTD's compressed set of features is asymptotically equivalent to PSR state, and PSTD is a consistent estimator of the PSR value function.

We demonstrate the merits of PSTD compared to two popular alternative algorithms, LARS-TD and LSTD, on a synthetic example, and argue that PSTD is most effective when approximating a value function from a large number of features, each of which contains at least a little information about state. Finally, we apply PSTD to a difficult optimal stopping problem, and demonstrate the practical utility of the algorithm by outperforming several alternative approaches and topping the best reported previous results.

Chapter 10

A Spectral Learning Approach to Range-Only SLAM

This chapter represents a significant shift in focus from the previous chapters. The bulk of this thesis has focused on predictive representations and spectral learning algorithms for learning the parameters of dynamical system models. This chapter uses similar tools but focuses on a specific applied problem: range-only simultaneous localization and mapping (range-only SLAM) with known correspondences and limited amounts of missing data [10].

10.1 Introduction

In range-only SLAM, we are given a sequence of range measurements from a robot to fixed landmarks, and possibly a matching sequence of odometry measurements. We then attempt to simultaneously estimate the robot's trajectory and the locations of the landmarks. Popular approaches to range-only SLAM include EKFs and EIFs [27, 28, 50, 56, 111], multiple-hypothesis trackers (including particle filters and multiple EKFs/EIFs) [29, 111], and batch optimization of a likelihood function [53].

In all the above approaches, the most popular representation for a hypothesis is a list of landmark locations $(m_{n,x}, m_{n,y})$ and a list of robot poses (x_t, y_t, θ_t). Unfortunately, both the motion and measurement models are highly nonlinear in this representation, leading to computational problems: inaccurate linearizations in EKF/EIF/MHT and local optima in batch optimization approaches (see Section 10.2 for details). Much work has attempted to remedy this problem, e.g., by changing the hypothesis representation [27] or by keeping multiple hypotheses [27, 29, 111]. While considerable progress has been made, none of these methods are ideal; common difficulties include the need for an extensive initialization phase, inability to recover from poor initialization, lack of performance guarantees, or excessive computational requirements.

We take a very different approach: we formulate range-only SLAM as a matrix factorization problem, where features of observations are linearly related to a 4- or 7-dimensional state space. This approach has several desirable properties. First, we need weaker assumptions about the measurement model and motion model than previous approaches to range-only SLAM. Second, our state space yields a *linear* measurement model, so we hope to lose less information during

tracking to approximation errors and local optima. Third, our formulation leads to a simple spectral learning algorithm, based on a fast and robust singular value decomposition (SVD)—in fact, our algorithm is an instance of a general spectral system identification framework, from which it inherits desirable guarantees including statistical consistency and no local optima. Fourth, we don't need to worry as much as previous methods about errors such as a consistent bias in odometry, or a receiver mounted at a different height from the transmitters: in general, we can learn to correct such errors automatically by expanding the dimensionality of our state space.

As we will discuss in Section 10.2, our approach to range-only SLAM has much in common with spectral algorithms for subspace identification [14, 117]; unlike these methods, our focus on SLAM makes it easy to *interpret* our state space. Our approach is also related to factorization-based structure from motion [49, 113, 114], as well as to recent dimensionality-reduction-based methods for localization and mapping [8, 32, 86, 124].

We begin in Section 10.2 by reviewing background related to our approach. In Section 10.3 we present the basic spectral learning algorithm for range-only SLAM, and discuss how it relates to state space discovery for a dynamical system. We conclude in Section 10.4 by comparing spectral SLAM to other popular methods for range-only SLAM on real world range data collected from an autonomous lawnmower with time-of-flight ranging radios.

10.2 Background

There are four main pieces of relevant background: first, the well-known solutions to range-only SLAM using variations of the extended Kalman filter and batch optimization; second, recently-discovered spectral approaches to identifying parameters of nonlinear dynamical systems; third, matrix factorization for finding structure from motion in video; and fourth, dimensionality-reduction methods for localization and mapping. Below, we will discuss the connections among these areas, and show how they can be unified within a spectral learning framework.

10.2.1 Likelihood-based Range-only SLAM

The standard probabilistic model for range-only localization [50, 56] represents robot state by a vector $s_t = [x_t, y_t, \theta_t]^\top$; the robot's (nonlinear) motion and observation models are

$$s_{t+1} = \begin{bmatrix} x_t + v_t \cos(\theta_t) \\ y_t + v_t \sin(\theta_t) \\ \theta_t + \omega_t \end{bmatrix} + \epsilon_t \qquad d_{t,n} = \sqrt{(m_{n,x} - x_t)^2 + (m_{n,y} - y_t)^2} + \eta_t \tag{10.1}$$

Here v_t is the distance traveled, ω_t is the orientation change, $d_{t,n}$ is the estimate of the range from the nth landmark location $(m_{n,x}, m_{n,y})$ to the current location of the robot (x_t, y_t), and ϵ_t and η_t are noise. Throughout this chapter we assume known correspondences, since range sensing systems such as radio beacons typically associate unique identifiers with each reading. In the case of unknown correspondences or large amounts of missing data, our method becomes a solution to a subproblem – which helps since being able to solve the subproblem reliably makes the overall search easier.

To handle SLAM rather than just localization, we can extend the state to include landmark positions:

$$s_t = [x_t, y_t, \theta_t, m_{1,x}, m_{1,y}, \ldots, m_{N,x}, m_{N,y}]^\top \tag{10.2}$$

where N is the number of landmarks. The motion and measurement models remain the same. Given this model, we can use any standard optimization algorithm (such as Gauss-Newton) to fit the unknown robot and landmark parameters by maximum likelihood. Or, we can track these parameters online using EKFs, EIFs, or MHT methods like particle filters.

EKFs and EIFs are a popular solution for localization and mapping problems: for each new odometry input $a_t = [v_t, \omega_t]^\top$ and each new measurement d_t, we propagate the estimate of the robot state and error covariance by linearizing the non-linear motion and measurement models. Unfortunately, though, range-only SLAM is notoriously difficult for EKFs/EIFs: since range-only sensors are not informative enough to completely localize a robot or a landmark from a small number of readings, nonlinearities are much worse in range-only SLAM than they are in other applications such as range-and-bearing SLAM. In particular, if we don't have a sharp prior distribution for landmark positions, then after a few steps, the exact posterior becomes highly non-Gaussian and multimodal; so, any Gaussian approximation to the posterior is necessarily inaccurate. Furthermore, an EKF will generally not even produce the best possible Gaussian approximation: a good linearization would tell us a lot about the modes of the posterior, which would be equivalent to solving the original SLAM problem. So, practical applications of the EKF to range-only SLAM attempt to *delay* linearization until enough information is available, e.g., via an extended initialization phase for each landmark. Such delays simply push the problem of finding a good hypothesis onto the initialization algorithm.

Djugash et al. proposed a polar parameterization to more accurately represent the annular and multimodal distributions typically encountered in range-only SLAM. The resulting approach is called the ROP-EKF, and is shown to outperform the ordinary (Cartesian) EKF in several real-world problems, especially in combination with multiple-hypothesis tracking [27, 28]. However, the multi-hypothesis ROP-EKF can be much more expensive than an EKF, and is still a heuristic approximation to the true posterior.

Instead of the posterior covariance of the state (as used by the EKF), the extended information filter (EIF) maintains an estimate of the *inverse* covariance. The two representations are statistically equivalent (and therefore have the same failure modes). But, the inverse covariance is often approximately sparse, leading to much more efficient approximate computation [111].

10.2.2 Spectral State Space Discovery and System Identification

System identification algorithms attempt to *learn* dynamical system parameters such as a state space, a dynamics model (motion model), and an observation model (measurement model) directly from samples of observations and actions. In the last few years, *spectral* system identification algorithms have become popular; these algorithms learn a state space via a spectral decomposition of a carefully designed matrix of observable features, then find transition and observation models by linear regressions involving the learned states. Originally, subspace identification algorithms were almost exclusively used for linear system identification [117], but re-

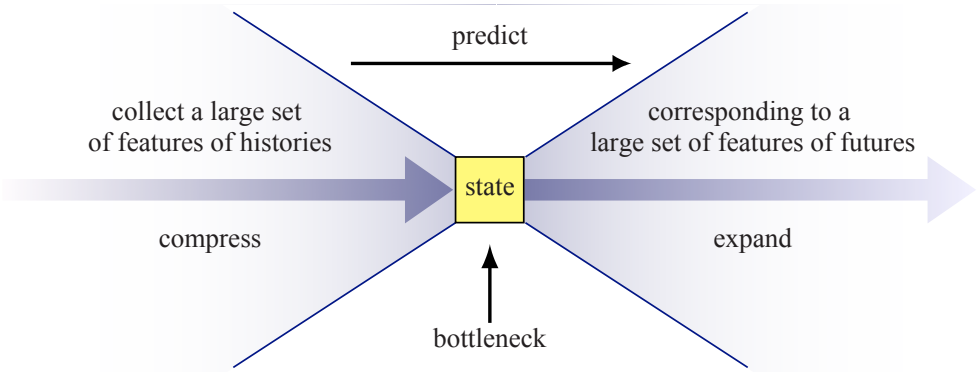

Figure 10.1: A general principle for state space discovery. We can think of state as a *statistic* of history that is minimally *sufficient* to predict future observations. If the bottleneck is a rank constraint, then we get a *spectral* method.

cently, similar spectral algorithms have been used to learn models of partially observable nonlinear dynamical systems such as HMMs [42, 90] and PSRs [12, 14, 15, 84]. All of these spectral algorithms share a strategy for state space discovery: they learn a state space via a spectral decomposition of a matrix of observations (Figure 10.1), resulting in a linear observation function, and then they learn a model of the dynamics in the learned low-dimensional state space. This is a powerful and appealing approach: the resulting algorithms are *statistically consistent*, and they are easy to implement with efficient linear algebra operations. In contrast, batch optimization of likelihood (e.g., via the popular expectation maximization (EM) algorithm) is only known to be consistent if we find the *global* optimum of the likelihood function—typically an impractical requirement.

As we will see in Section 10.3, we can view the range-only SLAM problem as an instance of spectral state space discovery. And, the Appendix (Sec. 12.2.3) discusses how to identify transition and measurement models given the learned states. The same properties that make spectral methods appealing for system identification carry over to our spectral SLAM algorithm: computational efficiency, statistical consistency, and finite-sample error bounds.

10.2.3 Orthographic Structure From Motion

In some ways the orthographic structure from motion (SfM) problem in vision [113] is very similar to the SLAM problem: the goal is to recover scene geometry and camera rotations from a sequence of images (compare with landmark geometry and robot poses from a sequence of range observations). And in fact, one popular solution for SfM is very similar to the state space discovery step in spectral state space identification. The key idea in spectral SfM is that is that an image sequence can be represented as a $2F \times P$ measurement matrix W, containing the horizontal and vertical coordinates of P points tracked through F frames. If the images are the result of an orthographic camera projection, then it is possible to show that $\text{rank}(W) = 3$. As a consequence, the measurement matrix can be factored into the product of two matrices U and V, where U contains the 3d positions of the features and V contains the camera axis rotations [113]. With respect to system identification, it is possible to interpret the matrix U as an observation model

and V as an estimate of the system state. Inspired by SfM, we reformulate range-only SLAM problem in a similar way in Section 10.3, and then similarly solve the problem with a spectral learning algorithm. Also similar to SfM, we examine the identifiability of our factorization, and give a *metric upgrade* procedure which extracts additional geometric information beyond what the factorization gives us.

10.2.4 Dimensionality-reduction-based Methods for Mapping

Dimensionality reduction methods have recently provided an alternative to more traditional likelihood-based methods for mapping. In particular, the problem of finding a good map can be viewed as finding a (possibly nonlinear) *embedding* of sensor data via methods like multidimensional scaling (MDS) and manifold learning.

For example, MDS has been used to determine a Euclidean map of sensor locations where there is no distinction between landmark positions and robot positions [86]: instead *all-to-all* range measurements are assumed for a set of landmarks. If some pairwise measurements are not available, these measurements can be approximated by some interpolation method, e.g. the geodesic distance between the landmarks [86, 108].

Our problem differs from this previous work: in contrast to MDS, we have no landmark-to-landmark measurements and only inaccurate robot-to-robot measurements (from odometry, which may not be present, and which often has significant errors when integrated over more than a short distance). Additionally, our smaller set of measurements introduces additional challenges not present in classical MDS: linear methods can recover the positions only up to a linear transformation. This ambiguity forces changes compared to the MDS algorithm: while MDS factors the all-to-all matrix of squared ranges, in Sec. 10.3.1 we factor only a block of this matrix, then use either a metric upgrade step or a few global position measurements to resolve the ambiguity.

A popular alternative to linear dimensionality reduction techniques like classical MDS is *manifold learning*: nonlinearly mapping sensor inputs to a feature space that "unfolds" the manifold on which the data lies and *then* applying dimensionality reduction. Such nonlinear dimensionality reduction has been used to learn maps of wi-fi networks and landmark locations when sensory data is thought to be nonlinearly related to the underlying Eucidean space in which the landmarks lie [8, 32, 124]. Unlike theses approaches, we show that *linear* dimensionality reduction is sufficient to solve the range-only SLAM problem. (In particular, [124] suggests solving range-only mapping using nonlinear dimensionality reduction. We not only show that this is unnecessary, but additionally show that linear dimensionality reduction is sufficient for localization as well.) This greatly simplifies the learning algorithm and allows us to provide strong statistical guarantees for the mapping portion of SLAM (Sec. 10.3.3).

10.3 State Space Discovery and Spectral SLAM

We start with SLAM from range data without odometry. For now, we assume no noise, no missing data, and batch processing. We will generalize below: Sec. 10.3.2 discusses how to recover robot orientation, Sec. 10.3.3 discusses noise, and Sec. 10.3.4 discusses missing data and

online SLAM. In the Appendix (Section 12.2.3) we discuss learning motion and measurement models.

10.3.1 Range-only SLAM as Matrix Factorization

Consider the matrix $Y \in \mathbb{R}^{N \times T}$ of squared ranges, with $N \geq 4$ landmarks and $T \geq 4$ time steps:

$$Y = \frac{1}{2} \begin{bmatrix} d_{11}^2 & d_{12}^2 & \dots & d_{1T}^2 \\ d_{21}^2 & d_{22}^2 & \dots & d_{2T}^2 \\ \vdots & \vdots & \vdots & \vdots \\ d_{N1}^2 & d_{N2}^2 & \dots & d_{NT}^2 \end{bmatrix} \tag{10.3}$$

where $d_{n,t}$ is the measured distance from the robot to landmark n at time step t.

The most basic version of our spectral SLAM method relies on the insight that Y *factors* according to robot position (x_t, y_t) and landmark position $(m_{n,x}, m_{n,y})$. To see why, note

$$d_{n,t}^2 = (m_{n,x}^2 + m_{n,y}^2) - 2m_{n,x} \cdot x_t - 2m_{n,y} \cdot y_t + (x_t^2 + y_t^2) \tag{10.4}$$

If we write $C_n = [(m_{n,x}^2 + m_{n,y}^2)/2, m_{n,x}, m_{n,y}, 1]^\top$ and $X_t = [1, -x_t, -y_t, (x_t^2 + y_t^2)/2]^\top$, it is easy to see that $d_{n,t}^2 = 2C_n^\top X_t$. So, Y factors as $Y = CX$, where $C \in \mathbb{R}^{N \times 4}$ contains the positions of landmarks,

$$C = \begin{bmatrix} (m_{1,x}^2 + m_{1,y}^2)/2 & m_{1,x} & m_{1,y} & 1 \\ (m_{2,x}^2 + m_{2,y}^2)/2 & m_{2,x} & m_{2,y} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ (m_{N,x}^2 + m_{N,y}^2)/2 & m_{N,x} & m_{N,y} & 1 \end{bmatrix} \tag{10.5}$$

and $X \in \mathbb{R}^{4 \times T}$ contains the positions of the robot over time

$$X = \begin{bmatrix} 1 & \dots & 1 \\ -x_1 & \dots & -x_T \\ -y_1 & \dots & -y_T \\ (x_1^2 + y_1^2)/2 & \dots & (x_T^2 + y_T^2)/2 \end{bmatrix} \tag{10.6}$$

If we can recover C and X, we can read off the solution to the SLAM problem. The fact that Y's rank is at most 4 suggests that we might be able to use a rank-revealing factorization of Y, such as the singular value decomposition, to find C and X. Unfortunately, such a factorization only determines C and X up to a linear transform: given an invertible matrix S, we can write $Y = CX = CS^{-1}SX$. Therefore, factorization can only hope to recover $U = CS^{-1}$ and $V = SX$.

To upgrade the factors U and V to a full metric map, we have two options. If global position estimates are available for at least four landmarks, we can *learn* the transform S via linear regression, and so recover the original C and X. This method works as long as we know at least four landmark positions. Figure 10.2A shows a simulated example.

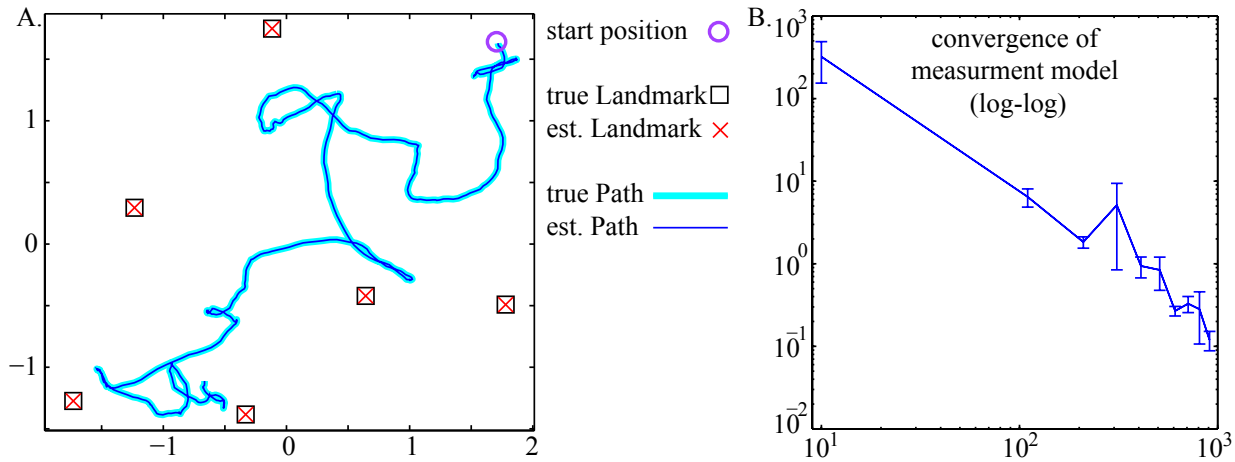

Figure 10.2: Spectral SLAM on simulated data. See Section 10.4.1 for details. A.) Randomly generated landmarks (6 of them) and robot path through the environment (500 timesteps). A SVD of the squared distance matrix recovers a linear transform of the landmark and robot positions. Given the coordinates of 4 landmarks, we can recover the landmark and robot positions in their original coordinates; or, since $500 \geq 9$, we can recover positions up to an orthogonal transform with no additional information. Despite noisy observations, the robot recovers the true path and landmark positions with very high accuracy. B.) The convergence of the observation model $\widehat{C}_{5:6}$ for the remaining two landmarks: mean Frobenius-norm error vs. number of range readings received, averaged over 1000 randomly generated pairs of robot paths and environments. Error bars indicate 95% confidence intervals.

On the other hand, if no global positions are known, the best we can hope to do is recover landmark and robot positions up to an orthogonal transform (translation, rotation, and reflection). It turns out that Eqs. (10.5–10.6) provide enough additional geometric constraints to do so: in the Appendix (Sec. 12.2.1) we show that, if we have at least 9 time steps and at least 9 landmarks, and if each of these point sets is non-singular in an appropriate sense, then we can compute the metric upgrade in closed form. The idea is to fit a quadratic surface to the rows of U, then change coordinates so that the surface becomes the function in (10.5). (By contrast, the usual metric upgrade for orthographic structure from motion [113], which uses the constraint that camera projection matrices are orthogonal, requires a nonlinear optimization.)

10.3.2 SLAM with Headings

In addition to location, we often want the robot's global heading θ. We could get headings by post-processing our learned positions, but in practice we can reduce variance by learning positions and headings simultaneously. We do so by adding more features to our measurement matrix: differences between successive pairs of squared distances, scaled by velocity (which we can estimate from odometry). Since we need pairs of time steps, we now have $Y \in \mathbb{R}^{2N \times T-1}$:

$$Y = \frac{1}{2}\begin{bmatrix} d_{11}^2 & d_{12}^2 & \dots & d_{1T-1}^2 \\ \vdots & \vdots & \ddots & \vdots \\ d_{N1}^2 & d_{N2}^2 & \dots & d_{NT-1}^2 \\ \frac{d_{12}^2 - d_{11}^2}{v_1} & \frac{d_{13}^2 - d_{12}^2}{v_2} & \dots & \frac{d_{1T}^2 - d_{1T-1}^2}{v_{T-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{d_{N2}^2 - d_{N1}^2}{v_1} & \frac{d_{N2}^2 - d_{N3}^2}{v_2} & \dots & \frac{d_{NT}^2 - d_{NT-1}^2}{v_{T-1}} \end{bmatrix} \tag{10.7}$$

As before, we can factor Y into a robot state matrix and a landmark matrix. The key new observation is that we can write the new features in terms of $\cos(\theta)$ and $\sin(\theta)$:

$$\begin{aligned} \frac{d_{n,t+1}^2 - d_{n,t}^2}{2v_t} &= -\frac{m_{n,x}(x_{t+1} - x_t)}{v_t} - \frac{m_{n,y}(y_{t+1} - y_t)}{v_t} + \frac{x_{t+1}^2 - x_t^2 + y_{t+1}^2 - y_t^2}{2v_t} \\ &= -m_{n,x}\cos(\theta_t) - m_{n,y}\sin(\theta_t) + \frac{x_{t+1}^2 - x_t^2 + y_{t+1}^2 - y_t^2}{2v_t} \end{aligned} \tag{10.8}$$

From Eq. 10.4 and Eq. 10.8 it is easy to see that Y has rank at most 7 (exactly 7 if the robot path and landmark positions are not singular): we have $Y = CX$, where $C \in \mathbb{R}^{N\times 7}$ contains functions of landmark positions and $X \in \mathbb{R}^{7\times T}$ contains functions of robot state,

$$C = \begin{bmatrix} (m_{1,x}^2 + m_{1,y}^2)/2 & m_{1,x} & m_{1,y} & 1 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (m_{N,x}^2 + m_{N,y}^2)/2 & m_{N,x} & m_{N,y} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & m_{1,x} & m_{1,y} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & m_{N,x} & m_{N,y} & 1 \end{bmatrix} \tag{10.9}$$

$$X = \begin{bmatrix} 1 & \dots & 1 \\ -x_1 & \dots & -x_{T-1} \\ -y_1 & \dots & -y_{T-1} \\ (x_1^2 + y_1^2)/2 & \dots & (x_{T-1}^2 + y_{T-1}^2)/2 \\ -\cos(\theta_1) & \dots & -\cos(\theta_{T-1}) \\ -\sin(\theta_1) & \dots & -\sin(\theta_{T-1}) \\ \frac{x_2^2 - x_1^2 + y_2^2 - y_1^2}{2v_1} & \dots & \frac{x_T^2 - x_{T-1}^2 + y_T^2 - y_{T-1}^2}{2v_{T-1}} \end{bmatrix} \tag{10.10}$$

As with the basic SLAM algorithm in Section 10.3.1, we can factor Y using SVD, this time keeping 7 singular values. To make the state space interpretable, we can then look at the top part of the learned transform of C: as long as we have at least four landmarks in non-singular position, this block will have exactly a three-dimensional nullspace (due to the three columns of zeros in the top part of C). After eliminating this nullspace, we can proceed as before to learn S and make the state space interpretable: either use the coordinates of at least 4 landmarks as regression targets, or perform a metric upgrade. (See the Appendix, Sec. 12.2.1, for details). Once we have positions, we can recover headings as angles between successive positions.

Algorithm 1 Spectral SLAM

In: *i.i.d.* pairs of observations $\{o_t, a_t\}_{t=1}^T$; optional: measurement model for ≥ 4 landmarks $C_{1:4}$
Out: measurement model (map) $\widehat{C}$, robot locations $\widehat{X}$ (the tth column is location at time t)

1: Collect observations and odometry into a matrix $\widehat{Y}$ (Eq. 10.7)
2: Find the the top 7 singular values and vectors: $\langle \widehat{U}, \widehat{\Lambda}, \widehat{V}^\top \rangle \leftarrow \text{SVD}(\widehat{Y}, 7)$
The transformed measurement matrix is $\widehat{C}S^{-1} = \widehat{U}$ and robot states are $S\widehat{X} = \widehat{\Lambda}\widehat{V}^\top$.
3: Find $\widehat{S}$ via linear regression (from $\widehat{U}$ to $C_{1:4}$) or metric upgrade (see Appendix)
and return $\widehat{C} = \widehat{U}\widehat{S}$ and $\widehat{X} = \widehat{S}^{-1}\widehat{\Lambda}\widehat{V}^\top$

10.3.3 A Spectral SLAM Algorithm

The matrix factorizations of Secs. 10.3.1 and 10.3.2 suggest a straightforward SLAM algorithm, Alg. 1: build an empirical estimate $\widehat{Y}$ of Y by sampling observations as the robot traverses its environment, then apply a rank-7 thin SVD, discarding the remaining singular values to suppress noise.

$$\langle \widehat{U}, \widehat{\Lambda}, \widehat{V}^\top \rangle \leftarrow \text{SVD}(\widehat{Y}, 7) \tag{10.11}$$

Following Section 10.3.2, the left singular vectors $\widehat{U}$ are an estimate of our transformed measurement matrix CS^{-1}, and the weighted right singular vectors $\widehat{\Lambda}\widehat{V}^\top$ are an estimate of our transformed robot state SX. We can then learn S via regression or metric upgrade.

Statistical Consistency and Sample Complexity Let $M \in \mathbb{R}^{N \times N}$ be the *true* observation covariance for a randomly sampled robot position, and let $\widehat{M} = \frac{1}{T}\widehat{Y}\widehat{Y}^\top$ be the empirical covariance estimated from T observations. Then the true and estimated measurement models are the top singular vectors of M and $\widehat{M}$. Assuming that the noise in $\widehat{M}$ is zero-mean, as we include more data in our averages, we will show below that the law of large numbers guarantees that $\widehat{M}$ converges to the true covariance M. So, our learning algorithm is *consistent* for estimating the range of M, i.e., the landmark locations. (The estimated robot positions will typically not converge, since we typically have a bounded effective number of observations relevant to each robot position. But, as we see each landmark again and again, the robot position errors will average out, and we will recover the true map.)

In more detail, we can give finite-sample bounds on the error in recovering the true factors. For simplicity of presentation we assume that noise is i.i.d., although our algorithm will work for any zero-mean noise process with a finite mixing time. (The error bounds will of course become weaker in proportion to mixing time, since we gain less new information per observation.) The argument (see the Appendix, Sec. 12.2.2, for details) has two pieces: standard concentration bounds show that each element of our estimated covariance approaches its population value; then the continuity of the SVD shows that the learned subspace also approaches its true value.

The final bound is:

$$||\sin \Psi||_2 \leq \frac{Nc\sqrt{\frac{2\log(T)}{T}}}{\gamma} \tag{10.12}$$

where Ψ is the vector of canonical angles between the learned subspace and the true one, c is a constant depending on our error distribution, and γ is the true smallest nonzero eigenvalue of the covariance. In particular, this bound means that the sample complexity is $\tilde{O}(\zeta^2)$ to achieve error ζ.

10.3.4 Extensions: Missing Data, Online SLAM, and System ID

Missing data So far we have assumed that we receive range readings to all landmarks at each time step. In practice this assumption is rarely satisfied: we may receive range readings asynchronously, some range readings may be missing entirely, and it is often the case that odometry data is sampled faster than range readings. Here we outline two methods for overcoming this practical difficulty.

First, if a relatively small number of observations are missing, we can use standard approaches for factorization with missing data. For example, probabilistic PCA [112] estimates the missing entries via an EM algorithm, and matrix completion [21] uses a trace-norm penalty to recover a low-rank factorization with high probability. However, for range-only data, often the fraction of missing data is high and the missing values are structural rather than random.

The second approach is interpolation: we divide the data into overlapping subsets and then use local odometry information to interpolate the range data within each subset. To interpolate the data, we estimate a robot path by dead reckoning. For each point in the dead reckoning path we build the feature representation $[1, -x, -y, (x^2 + y^2)/2]^\top$. We then learn a linear model that predicts a squared range reading from these features (for the data points where range is available), as in Eq. 10.4. Next we predict the squared range along the entire path. Finally we build the matrix $\widehat{Y}$ by averaging the locally interpolated range readings. This interpolation approach works much better in practice than the fully probabilistic approaches mentioned above, and was used in our experiments in Section 10.4.

Online Spectral SLAM The algorithms developed in this section so far have had an important drawback: unlike many SLAM algorithms, they are batch methods not online ones. The extension to online SLAM is straightforward: instead of first estimating $\widehat{Y}$ and then performing a SVD, we sequentially estimate our factors $\langle \widehat{U}, \widehat{\Lambda}, \widehat{V}^\top \rangle$ via online SVD [15, 20].

Robot Filtering and System Identification So far, our algorithms have not directly used (or needed) a robot motion model in the learned state space. However, an explicit motion model is required if we want to *predict* future sensor readings or *plan* a course of action. We have two choices: we can derive a motion model from our learned transformation S between latent states and physical locations, or we can learn a motion model directly from data using spectral system identification. More details about both of these approaches can be found in the Appendix, Sec. 12.2.3.

10.4 Experimental Results

We perform several SLAM and robot navigation experiments to illustrate and test the ideas proposed in this chapter. First we show how our methods work in theory with synthetic experiments where complete observations are received at each point in time and *i.i.d.* noise is sampled from a multivariate Gaussian distribution. Next we demonstrate our algorithm on data collected from a real-world robotic system with substantial amounts of missing data. Experiments were performed in Matlab, on a 2.66 GHz Intel Core i7 computer with 8 GB of RAM. In contrast to batch nonlinear optimization approaches to SLAM, the spectral learning methods described in this chapter are *very* fast, usually taking less than a second to run.

10.4.1 Synthetic Experiments

Our simulator randomly places 6 landmarks in a 2-D environment. A simulated robot then randomly moves through the environment for 500 time steps and receives a range reading to each one of the landmarks at each time step. The range readings are perturbed by noise sampled from a Gaussian distribution with variance equal to 1% of the range. Given this data, we apply the algorithm from Section 10.3.3 to solve the SLAM problem. We use the coordinates of 4 landmarks to learn the linear transform S and recover the true state space, as shown in Figure 10.2A. The results indicate that we can accurately recover both the landmark locations and the robot path.

We also investigated the empirical convergence rate of our observation model (and therefore the map) as the number of range readings increased. To do so, we generated 1000 different random pairs of environments and robot paths. For each pair, we repeatedly performed our spectral SLAM algorithm on increasingly large numbers of range readings and looked at the difference between our estimated measurement model (the robot's map) and the true measurement model, excluding the landmarks that we used for reconstruction: $\|\widehat{C}_{5:6} - C_{5:6}\|_{\mathcal{F}}$. The results are shown in Figure 10.2B, and show that our estimates steadily converge to the true model, corroborating our theoretical results (in Section 10.3.3 and the Appendix).

10.4.2 Robotic Experiments

We used two freely available range-only SLAM data sets collected from an autonomous lawn mowing robot [27], shown in Fig. 10.3A.[1] These "Plaza" datasets were collected via radio nodes from Multispectral Solutions that use time-of-flight of ultra-wide-band signals to provide inter-node ranging measurements. (Additional details on the experimental setup can be found in [27].) This system produces a time-stamped range estimate between the mobile robot and stationary nodes (landmarks) in the environment. The landmark radio nodes are placed atop traffic cones approximately 138cm above the ground throughout the environment, and one node was placed on top of the center of the robot's coordinate frame (also 138cm above the ground). The robot odometry (dead reckoning) comes from an onboard fiberoptic gyro and wheel encoders. The two environmental setups, including the locations of the landmarks, the dead reckoning paths, and

[1] http://www.frc.ri.cmu.edu/projects/emergencyresponse/RangeData/index.html

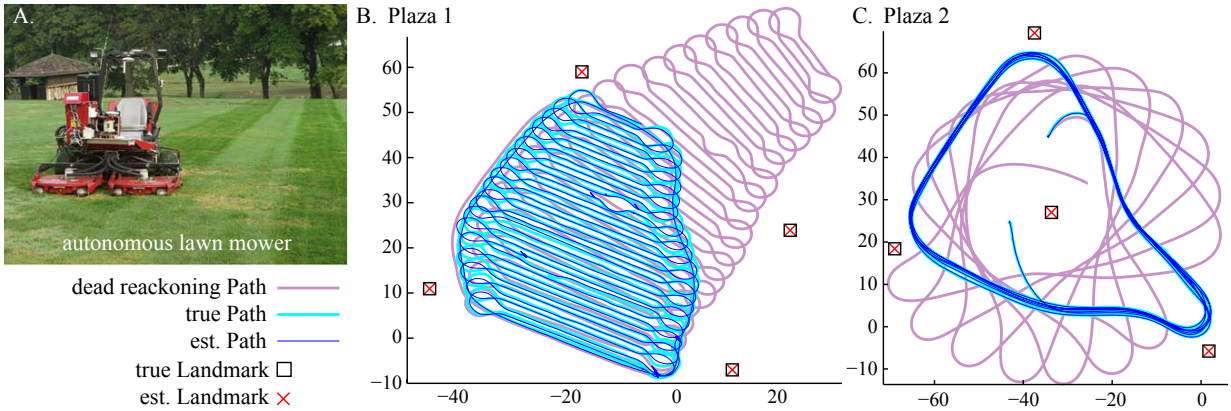

Figure 10.3: The autonomous lawn mower and spectral SLAM. A.) The robotic lawn mower platform. B.) In the first experiment, the robot traveled 1.9km receiving 3,529 range measurements. This path minimizes the effect of heading error by balancing the number of left turns with an equal number of right turns in the robot's odometry (a commonly used path pattern in lawn mowing applications). The light blue path indicates the robot's true path in the environment, light purple indicates dead-reckoning path, and dark blue indicates the spectral SLAM localization result. C.) In the second experiment, the robot traveled 1.3km receiving 1,816 range measurements. This path highlights the effect of heading error on dead reckoning performance by turning in the same direction repeatedly. Again, spectral SLAM is able to accurately recover the robot's path.

the ground truth paths, are shown in Figure 10.3B-C. The ground truth paths have 2cm accuracy according to [27].

The two Plaza datasets that we used to evaluate our algorithm have very different characteristics. In "Plaza 1," the robot travelled 1.9km, occupied 9,658 distinct poses, and received 3,529 range measurements. The path taken is a typical lawn mowing pattern that balances left turns with an equal number of right turns; this type of pattern minimizes the effect of heading error. In "Plaza 2," the robot travelled 1.3km, occupied 4,091 poses, and received 1,816 range measurements. The path taken is a loop which amplifies the effect of heading error. The two data sets were both very sparse, with approximately 11 time steps (and up to 500 steps) between range readings for the worst landmark. We first interpolated the missing range readings with the method of Section 10.3.4. Then we applied the rank-7 spectral SLAM algorithm of Section 10.3.3; the results are depicted in Figure 10.3B-C. Qualitatively, we see that the robot's localization path conforms to the true path.

In addition to the qualitative results, we quantitatively compared spectral SLAM to a number of different competing range-only SLAM algorithms. The localization root mean squared error (RMSE) in meters for each algorithm is shown in Figure 10.4. The baseline is dead reckoning (using only the robot's odometry information). Next are several standard online range-only SLAM algorithms, summarized in [27]. These algorithms included the Cartesian EKF, FastSLAM [69] with 5,000 particles, and the ROP-EKF [28]. These previous results only reported the RMSE for the last 10% of the path, which is typically the *best* 10% of the path (since it gives

the most time to recover from initialization problems). The full path localization error can be considerably worse, particularly for the initial portion of the path—see Fig. 5 (right) of [28].

We also compared to batch nonlinear optimization, via Gauss-Newton as implemented in Matlab's `fminunc` (see [53] for details). This approach to solving the range-only SLAM problem can be very data efficient, but is subject to local optima and is very computationally intensive. We followed the suggestions of [53] and initialized with the dead-reckoning estimate of the robot's path. The algorithm took roughly 2.5 hours to converge on Plaza 1, and 45 minutes to converge on Plaza 2. Under most evaluation metrics, the nonlinear batch algorithm handily beats the EKF-based alternatives.

Finally, we ran our spectral SLAM algorithm on the same data sets. In contrast to Gauss-Newton, spectral SLAM is *statistically consistent*, and much faster: the bulk of the computation is the fixed-rank SVD, so the time complexity of the algorithm is $O((2N)^2T)$ where N is the number of landmarks and T is the number of time steps. Empirically, spectral SLAM produced results that were comparable to batch optimization in 3-4 *orders of magnitude* less time (see Figure 10.4).

Spectral SLAM can also be used as an initialization procedure for nonlinear batch optimization. This strategy combines the best of both algorithms by allowing the locally optimal nonlinear optimization procedure to start from a theoretically guaranteed good starting point. Therefore, the local optimum found by nonlinear batch optimization should be *no worse* than the spectral SLAM solution and likely much better than the batch optimization seeded by dead-reckoning. Empirically, we found this to be the case (Figure 10.4). If time and computational resources are scarce, then we believe that spectral SLAM is clearly the best approach; if computation is not an issue, the best results will almost certainly be found by refining the spectral SLAM solution using a nonlinear batch optimization procedure.

10.5 Conclusion

We proposed a novel solution for the range-only SLAM problem that differs substantially from previous approaches. The essence of this new approach is to formulate SLAM as a factorization problem, which allows us to derive a local-minimum free spectral learning method that is closely related to SfM and spectral approaches to system identification. We provide theoretical guarantees for our algorithm, discuss how to derive an online algorithm, and show how to generalize to a full robot system identification algorithm. Finally, we demonstrate that our spectral approach to SLAM beats other state-of-the-art SLAM approaches on real-world range-only SLAM problems.

Method	Plaza 1	Plaza 2
Dead Reckoning (full path)	15.92m	27.28m
Cartesian EKF (last, best 10%)	0.94m	0.92m
FastSLAM (last, best 10%)	0.73m	1.14m
ROP EKF (last, best 10%)	0.65m	0.87m
Batch Opt. (worst 10%)	1.04m	0.45m
Batch Opt. (last 10%)	1.01m	0.45m
Batch Opt. (best 10%)	0.56m	0.20m
Batch Opt. (full path)	0.79m	0.33m
Spectral SLAM (worst 10%)	1.01m	0.51m
Spectral SLAM (last 10%)	0.98m	0.51m
Spectral SLAM (best 10%)	0.59m	0.22m
Spectral SLAM (full path)	0.79m	0.35m
Spectral + Batch Optimization (worst 10%)	**0.89m**	**0.40m**
Spectral + Batch Optimization (last 10%)	0.81m	**0.32m**
Spectral + Batch Optimization (best 10%)	**0.54m**	**0.18m**
Spectral + Batch Optimization (full path)	**0.69m**	**0.30m**

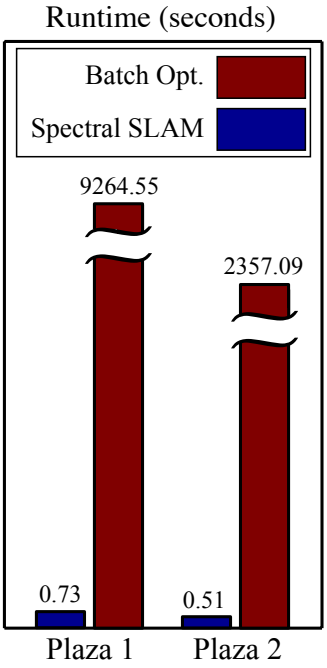

Figure 10.4: Comparison of Range-Only SLAM Algorithms. The table shows Localization RMSE. Spectral SLAM has localization accuracy comparable to batch optimization on its own. The best results (boldface entries) are obtained by initializing nonlinear batch optimization with the spectral SLAM solution. The graph compares runtime of Gauss-Newton batch optimization with spectral SLAM. Empirically, spectral SLAM is 3-4 *orders of magnitude* faster than batch optimization on the autonomous lawnmower datasets.

Part III

Conclusions

Chapter 11

Discussion

Learning models of dynamical systems is a fundamental task for predicting, classifying, and simulating observations as well as reasoning about actions in time series. In this thesis we focused on and combined two complimentary ideas: *predictive representations* that model state as an expectation of the future and *spectral algorithms* for learning a low-dimensional state representation. The key idea is that predictive representations allow dynamical system parameters to be written in terms of *observable* quantities; these observable quantities can then be leveraged to find a valid state space and model parameters.

This framework contrasts sharply with previous approaches to reasoning about and learning dynamical systems: the predominant class of probabilistic models for representing dynamical systems is *latent variable models*, which assume that observations are generated by an unobservable state. However, learning latent variable models is very difficult. The most popular approach is the EM algorithm, a local search heuristic that alternatively posits values for the latent variables and then uses these variables to estimate the system parameters. Unfortunately, EM is generally not a great approach: the algorithm is highly sensitive to initial conditions, takes a long time to converge, and is often numerically brittle in practice.

In contrast to latent variable models and EM, the algorithms presented in this thesis have excellent theoretical properties that translate into good practical performance. In particular, the spectral algorithms here are almost all *statistically consistent* and thus have no local optima. Additionally, the matrix algebra that is used to learn these models is very fast and numerically stable. We highlight some of the contributed algorithms here.

First, we provided a novel spectral algorithm for learning an observable representation of a Kalman filter that is closely linked to the spectral learning algorithms for the more expressive models that follow. We address the issue of instability in Kalman filters, proposing a constraint-generation algorithm that outperforms previous methods in both efficiency and accuracy.

Second, we developed a novel spectral learning algorithm for learning predictive state representations (PSRs) that outperforms previous approaches to learning these systems. Our approach is fast, accurate, and statistically consistent. One of the drawbacks of PSRs is that if a dynamical system has a very large number of different actions and observations, then learning the parameters of such a system can be very difficult. Therefore, we extend our representation so that it can be written in terms of *features* of actions and observations. We show that this algorithm is also consistent and has good predictive accuracy. Additionally, we demonstrate that our algorithms

can be used to learn accurate models of complex environments. In particular, we learn a representation of an agent moving through a simulated visual environment and then plan in the learned model. We show that the resulting plans were close to optima in the original environment. We also show how to extend temporal difference learning using some of the theory developed for our spectral system identification algorithm, and we show that the resulting extension outperforms many competing algorithms. Finally, we show how dynamical systems can be represented non-parameterically in reproducing kernel Hilbert spaces. We develop a spectral learning algorithm for this case, show its consistency, and demonstrate that this approach can lead to very accurate predictive models in practice.

Third, we develop a spectral learning approach to solving the range-only simultaneous localization and mapping problem. This is a different application of spectral learning compared with the rest of the thesis, but the results demonstrate the basic approach to learning a dynamical system state has applications beyond stochastic process modeling.

Together, the approaches enumerated here show how predictive representations and spectral learning algorithms can be unified into a powerful framework for learning predictive dynamical system models. The research presented in this thesis has the potential to positively impact many fields where sequential data modeling is important: robot sensing and planning, video modeling, activity recognition and user modeling, speech recognition, bioinformatics, and more.

Part IV

Appendix

Chapter 12

Appendix

12.1 Predictive State Temporal Difference Learning

12.1.1 Determining the Compression Operator

We find a compression operator V that optimally predicts test-features through the CCA bottleneck defined by $\widehat{U}$. The least squares estimate can be found by minimizing the following loss

$$\begin{aligned}\mathcal{L}(V) &= \left\|\phi^{\mathcal{T}}_{1:k} - \widehat{U}V\phi^{\mathcal{H}}_{1:k}\right\|_F^2 \\ \widehat{V} &= \arg\min_V \mathcal{L}(V)\end{aligned}$$

where $\|\cdot\|_F$ denotes the Frobenius norm. We can find $\widehat{V}$ by taking a derivative of this loss $\mathcal{L}$ with respect to V, setting it to zero, and solving for V

$$\begin{aligned}\mathcal{L} &= \frac{1}{k}^\top \left((\phi^{\mathcal{T}}_{1:k} - \widehat{U}V\phi^{\mathcal{H}}_{1:k})(\phi^{\mathcal{T}}_{1:k} - \widehat{U}V\phi^{\mathcal{H}}_{1:k})^\top\right) \\ &= \frac{1}{k}^\top \left({\phi^{\mathcal{T}}_{1:k}}^\top\phi^{\mathcal{T}}_{1:k} - 2{\phi^{\mathcal{T}}_{1:k}}^\top\widehat{U}V\phi^{\mathcal{H}}_{1:k} + {\phi^{\mathcal{H}}_{1:k}}^\top V^\top\widehat{U}^\top\widehat{U}V\phi^{\mathcal{H}}_{1:k}\right) \\ \implies \mathbf{d}\mathcal{L} &= -2\text{tr}\left({\phi^{\mathcal{H}}_{1:k}}^\top\mathbf{d}V^\top\widehat{U}^\top\phi^{\mathcal{T}}_{1:k}\right) + 2\text{tr}\left({\phi^{\mathcal{H}}_{1:k}}^\top\mathbf{d}V^\top\widehat{U}^\top\widehat{U}V\widehat{\phi}^{\mathcal{H}}_{1:k}\right) \\ \implies \mathbf{d}\mathcal{L} &= -2\text{tr}\left(\mathbf{d}V^\top\widehat{U}^\top\phi^{\mathcal{T}}_{1:k}{\phi^{\mathcal{H}}_{1:k}}^\top\right) + 2\text{tr}\left(\mathbf{d}V^\top\widehat{U}^\top\widehat{U}V\widehat{\phi}^{\mathcal{H}}_{1:k}{\phi^{\mathcal{H}}_{1:k}}^\top\right) \\ \implies \mathbf{d}\mathcal{L} &= -2\text{tr}\left(\mathbf{d}V^\top\widehat{U}^\top\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}\right) + 2\text{tr}\left(\mathbf{d}V^\top\widehat{U}^\top\widehat{U}V\widehat{\Sigma}_{\mathcal{H},\mathcal{H}}\right) \\ \implies \frac{\mathbf{d}\mathcal{L}}{\mathbf{d}V^\top} &= -2\text{tr}\left(\widehat{U}^\top\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}\right) + 2\text{tr}\left(\widehat{U}^\top\widehat{U}V\widehat{\Sigma}_{\mathcal{H},\mathcal{H}}\right) \\ \implies 0 &= -\widehat{U}^\top\widehat{\Sigma}_{\mathcal{T},\mathcal{H}} + \widehat{U}^\top\widehat{U}V\widehat{\Sigma}_{\mathcal{H},\mathcal{H}} \\ \implies \widehat{V} &= (\widehat{U}^\top\widehat{U})^{-1}\widehat{U}^\top\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}(\widehat{\Sigma}_{\mathcal{H},\mathcal{H}})^{-1} \\ &= \widehat{U}^\top\widehat{\Sigma}_{\mathcal{T},\mathcal{H}}(\widehat{\Sigma}_{\mathcal{H},\mathcal{H}})^{-1}\end{aligned}$$

12.1.2 Experimental Results

RR-POMDP

The RR-POMDP parameters are:

[$m = 4$ hidden states, $n = 2$ observations, $k = 3$ transition matrix rank].

$$T^{\pi} = \begin{bmatrix} 0.7829 & 0.1036 & 0.0399 & 0.0736 \\ 0.1036 & 0.4237 & 0.4262 & 0.0465 \\ 0.0399 & 0.4262 & 0.4380 & 0.0959 \\ 0.0736 & 0.0465 & 0.0959 & 0.7840 \end{bmatrix} \quad O = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

The discount factor is $\gamma = 0.9$.

Pricing a financial derivative

Basis functions The fist 16 are the basis functions suggested by Van Roy; for full description and justification see [23, 115]. The first functions consist of a constant, the reward, the minimal and maximal returns, and how long ago they occurred:

$$\begin{aligned}
\phi_1(x) &= 1 \\
\phi_2(x) &= G(x) \\
\phi_3(x) &= \min_{i=1,\ldots,100} x(i) - 1 \\
\phi_4(x) &= \max_{i=1,\ldots,100} x(i) - 1 \\
\phi_5(x) &= \arg\min_{i=1,\ldots,100} x(i) - 1 \\
\phi_6(x) &= \arg\max_{i=1,\ldots,100} x(i) - 1
\end{aligned}$$

The next set of basis functions summarize the characteristics of the basic shape of the 100 day sample path. They are the inner product of the path with the first four Legendre polynomial degrees. Let $j = i/50 - 1$.

$$\begin{aligned}
\phi_7(x) &= \frac{1}{100} \sum_{i=1}^{100} \frac{x(i) - 1}{\sqrt{2}} \\
\phi_8(x) &= \frac{1}{100} \sum_{i=1}^{100} x(i) \sqrt{\frac{3}{2}} j \\
\phi_9(x) &= \frac{1}{100} \sum_{i=1}^{100} x(i) \sqrt{\frac{5}{2}} \left(\frac{3j^2 - 1}{2} \right) \\
\phi_{10}(x) &= \frac{1}{100} \sum_{i=1}^{100} x(i) \sqrt{\frac{7}{2}} \left(\frac{5j^3 - 3j}{2} \right)
\end{aligned}$$

Nonlinear combinations of basis functions:

$$\begin{aligned}
\phi_{11}(x) &= \phi_2(x)\phi_3(x) \\
\phi_{12}(x) &= \phi_2(x)\phi_4(x) \\
\phi_{13}(x) &= \phi_2(x)\phi_7(x) \\
\phi_{14}(x) &= \phi_2(x)\phi_8(x) \\
\phi_{15}(x) &= \phi_2(x)\phi_9(x) \\
\phi_{16}(x) &= \phi_2(x)\phi_{10}(x)
\end{aligned}$$

In order to improve our results, we added a large number of additional basis functions to these hand-picked 16. PSTD will compress these features for us, so we can use as many additional basis functions as we would like. First we defined 4 additional basis functions consisting of the inner products of the 100 day sample path with the 5th and 6th Legende polynomials and we added the corresponding nonlinear combinations of basis functions:

$$\begin{aligned}
\phi_{17}(x) &= \frac{1}{100}\sum_{i=1}^{100} x(i)\sqrt{\frac{9}{2}}\left(\frac{35j^4 - 30x^2 + 3}{8}\right) \\
\phi_{18}(x) &= \frac{1}{100}\sum_{i=1}^{100} x(i)\sqrt{\frac{11}{2}}\left(\frac{63j^5 - 70j^3 + 15j}{8}\right) \\
\phi_{19}(x) &= \phi_2(x)\phi_{17}(x) \\
\phi_{20}(x) &= \phi_2(x)\phi_{18}(x)
\end{aligned}$$

Finally we added the the entire sample path and the squared sample path:

$$\begin{aligned}
\phi_{21:120} &= x_{1:100} \\
\phi_{121:220} &= x_{1:100}^2
\end{aligned}$$

12.2 A Spectral Learning Approach to Range Only SLAM

12.2.1 Metric Upgrade for Learned Map

In the main body of the paper, we assumed that global position estimates of at least four landmarks were known. When these landmarks are known, we can recover all of the estimated landmark positions and robot locations.

In many cases, however, no global positions are known; the best we can hope to do is recover landmark and robot positions up to an orthogonal transform (translation, rotation, and reflection). It turns out that Eqs. (10.5–10.6) provide enough geometric constraints to perform this metric upgrade, as long as we have at least 9 landmarks and at least 9 time steps, and as long as C and X are *nonsingular* in the following sense: define the matrix C_2, with the same number of rows as C but 10 columns, whose ith row has elements $c_{i,j}c_{i,k}$ for $1 \leq j \leq k \leq 4$ (in any fixed order). Note that the rank of C_2 can be at most 9: from Eq. 10.5, we know that $c_{i,2}^2 + c_{i,3}^2 - 2c_{i,4} = 0$, and each of the three terms in this function is a multiple of a column of C_2. We will say that

C is nonsingular if C_2 has rank exactly 9, i.e., is rank deficient by exactly 1 dimension. The conditions for X are analogous, swapping rows for columns.[1]

To derive the metric upgrade, suppose that we start from an $N \times 4$ matrix U of learned landmark coordinates and an $4 \times N$ matrix V of learned robot coordinates from the algorithm of Sec. 10.3.1. And, suppose that we have at least 9 nonsingular landmarks and robot positions. We would like to transform the learned coordinates into two new matrices C and X such that

$$c_1 \approx 1 \tag{12.1}$$

$$c_4 \approx \frac{1}{2}c_2^2 + \frac{1}{2}c_3^2 \tag{12.2}$$

$$x_4 \approx 1 \tag{12.3}$$

$$x_1 \approx \frac{1}{2}x_2^2 + \frac{1}{2}x_3^2 \tag{12.4}$$

where c is a row of C and x is a column of X.

At a high level, we first fit a quadratic surface to the rows of U, then transform this surface so that it satisfies Eq. 12.1–12.2, and scale the surface so that it satisfies Eq. 12.3. Our surface will then automatically also satisfy Eq. 12.4, since X must be metrically correct if C is.

In more detail, we first (step i) linearly transform each row of U into approximately the form $(1, r_{i,1}, r_{i,2}, r_{i,3})$: we use linear regression to find a coefficient vector $a \in \mathbb{R}^4$ such that $Ua \approx \mathbf{1}$, then set $R = UQ$ where $Q \in \mathbb{R}^{4\times 3}$ is an orthonormal basis for the nullspace of $a^\top$. After this step, our factorization is $(UT_1)(T_1^{-1}V)$, where $T_1 = (a\ Q)$.

Next (step ii) we fit an implicit quadratic surface to the rows of R by finding 10 coefficients b_{jk} (for $0 \le j \le k \le 3$) such that

$$\begin{aligned} 0 \ \approx\ & b_{00} + b_{01}r_{i,1} + b_{02}r_{i,2} + b_{03}r_{i,3} + \\ & b_{11}r_{i,1}^2 + b_{12}r_{i,1}r_{i,2} + b_{13}r_{i,1}r_{i,3} + b_{22}r_{i,2}^2 + b_{23}r_{i,2}r_{i,3} + b_{33}r_{i,3}^2 \end{aligned}$$

To do so, we form a matrix S that has the same number of rows as U but 10 columns. The elements of row i of S are $r_{i,j}r_{i,k}$ for $0 \le j \le k \le 3$ (in any fixed order). Here, for convenience, we define $r_{i,0} = 1$ for all i. Then we find a vector $b \in \mathbb{R}^{10}$ that is approximately in the nullspace of $S^\top$ by taking a singular value decomposition of S and selecting the right singular vector corresponding to the smallest singular value. Using this vector, we can define our quadratic as $0 \approx \frac{1}{2}r^\top Hr + \ell^\top r + b_{00}$, where r is a row of R, and the Hessian matrix H and linear part ℓ are given by:

$$H = \begin{pmatrix} \frac{1}{2}b_{11} & b_{12} & b_{13} \\ b_{21} & \frac{1}{2}b_{22} & b_{23} \\ b_{31} & b_{32} & \frac{1}{2}b_{33} \end{pmatrix} \qquad \ell = \begin{pmatrix} b_{01} \\ b_{02} \\ b_{03} \end{pmatrix}$$

Over the next few steps we will transform the coordinates in R to bring our quadratic into the form of Eq. 12.2: that is, one coordinate will be a quadratic function of the other two, there will be no linear or constant terms, and the quadratic part will be spherical with coefficient $\frac{1}{2}$.

[1]For intuition, a set of landmarks or robot positions that all lie on the same quadratic surface (line, circle, parabola, etc.) will be singular. Some higher-order constraints will also lead to singularity; e.g., a set of points will be singular if they all satisfy $\frac{1}{2}(x_i^2 + y_i^2)x_i + y_i = 0$, since each of the two terms in this function is a column of C_2.

We start (step iii) by transforming coordinates so that our quadratic has no cross-terms, i.e., so that its Hessian matrix is diagonal. Using a 3×3 singular value decomposition, we can factor $H = MH'M^\top$ so that M is orthonormal and H' is diagonal. If we set $R' = RM$ and $\ell' = M\ell$, and write r' for a row of R', we can equivalently write our quadratic as $0 = \frac{1}{2}(r')^\top H' r' + (\ell')^\top r' + b_{00}$, which has a diagonal Hessian as desired. After this step, our factorization is $(UT_1T_2)(T_2^{-1}T_1^{-1}V)$, where

$$T_2 = \begin{pmatrix} 1 & 0 \\ 0 & M \end{pmatrix}$$

Our next step (step iv) is to turn our implicit quadratic surface into an explicit quadratic function. For this purpose we pick one of the coordinates of R' and write it as a function of the other two. In order to do so, we must have zero as the corresponding diagonal element of the Hessian H'—else we cannot guarantee that we can solve for a unique value of the chosen coordinate. So, we will take the index j such that H'_{jj} is minimal, and set $H'_{jj} = 0$. Suppose that we pick the last coordinate, $j = 3$. (We can always reorder columns to make this true; SVD software will typically do so automatically.) Then our quadratic becomes

$$\begin{aligned} 0 &= \frac{1}{2}H'_{11}(r'_1)^2 + \frac{1}{2}H'_{22}(r'_2)^2 + \ell'_1 r'_1 + \ell'_2 r'_2 + \ell'_3 r'_3 + b_{00} \\ r'_3 &= -\frac{1}{\ell'_3}\left[\frac{1}{2}H'_{11}(r'_1)^2 + \frac{1}{2}H'_{22}(r'_2)^2 + \ell'_1 r'_1 + \ell'_2 r'_2 + b_{00}\right] \end{aligned}$$

Now (step v) we can shift and rescale our coordinates one more time to get our quadratic in the desired form: translate so that the linear and constant coefficients are 0, and rescale so that the quadratic coefficients are $\frac{1}{2}$. For the translation, we define new coordinates $r'' = r' + c$ for $c \in \mathbb{R}^3$, so that our quadratic becomes

$$r''_3 = c_3 - \frac{1}{\ell'_3}\left[\frac{1}{2}H'_{11}(r''_1 - c_1)^2 + \frac{1}{2}H'_{22}(r''_2 - c_2)^2 + \ell'_1(r''_1 - c_1) + \ell'_2(r''_2 - c_2) + b_{00}\right]$$

By expanding and matching coefficients, we know c must satisfy

$$\begin{aligned} 0 &= \frac{H'_{11}}{\ell'_3}c_1 - \frac{\ell'_1}{\ell'_3} && \text{(coefficient of } r''_1\text{)} \\ 0 &= \frac{H'_{22}}{\ell'_3}c_2 - \frac{\ell'_2}{\ell'_3} && \text{(coefficient of } r''_2\text{)} \\ 0 &= c_3 - \frac{H'_{11}}{2\ell'_3}c_1^2 - \frac{H'_{22}}{2\ell'_3}c_2^2 + \frac{\ell'_1}{\ell'_3}c_1 + \frac{\ell'_2}{\ell'_3}c_2 - b_{00}/\ell'_3 && \text{(constant)} \end{aligned}$$

The first two equations are linear in c_1 and c_2 (and don't contain c_3). So, we can solve directly for c_1 and c_2; then we can plug their values into the last equation to find c_3. For the scaling, the coefficient of r''_1 is now $-\frac{H'_{11}}{2\ell'_3}$, and that of r''_2 is now $-\frac{H'_{22}}{2\ell'_3}$. So, we can just scale these two coordinates separately to bring their coefficients to $\frac{1}{2}$.

After this step, our factorization is $U'V'$, where $U' = UT_1T_2T_3$ and $V' = T_3^{-1}T_2^{-1}T_1^{-1}V$, and

$$T_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ c_1 & -\frac{\ell_3'}{H_{11}'} & 0 & 0 \\ c_2 & 0 & -\frac{\ell_3'}{H_{22}'} & 0 \\ c_3 & 0 & 0 & 1 \end{pmatrix}$$

The left factor U' will now satisfy Eq. 12.1–12.2. We still have one last useful degree of freedom: if we set $C = U'T_4$, where

$$T_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \mu & 0 & 0 \\ 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & \mu^2 \end{pmatrix}$$

for any $\mu \in \mathbb{R}$, then C will still satisfy Eq. 12.1–12.2. So (step vi), we will pick μ to satisfy Eq. 12.3: in particular, we set $\mu = \sqrt{\text{mean}(V_{4,:}')}$, so that when we set $X = T_4^{-1}V'$, the last row of X will have mean 1.

If we have 7 learned coordinates in U as in Sec. 10.3.2, we need to find a subspace of 4 coordinates in order to perform metric upgrade. To do so, we take advantage of the special form of the correct answer, given in Eq. 10.9: in the upper block of C in Eq. 10.9, three coordinates are identically zero. Since U is a linear transformation of C, there will be three linear functions of the top block of U that are identically zero (or approximately zero in the presence of noise). As long as the landmark positions are nonsingular, we can use SVD on the top block of U to find and remove these linear functions (by setting the smallest three singular values to zero), then proceed as above with the four remaining coordinates.

12.2.2 Sample Complexity for the Measurement Model (Robot Map)

Here we provide the details on how our estimation error scales with the number T of training examples—that is, the scaling of the difference between the estimated measurement model $\widehat{U}$, which contains the location of the landmarks, and its population counterpart.

Our bound has two parts. First we use a standard concentration bound (the Azuma-Hoeffding inequality) to show that each element of our estimated covariance $\widehat{M} = \widehat{Y}\widehat{Y}^\top$ approaches its population value. We start by rewriting the empirical covariance matrix as a vector summed over multiple samples:

$$\text{vec}\left(\widehat{M}\right) = \frac{1}{T}\sum_{t=1}^{T} \Upsilon_{:,t}$$

where $\Upsilon = (\widehat{Y} \odot \widehat{Y})^\top$ is the matrix of column-wise Kronecker products of the observations $\widehat{Y}$. We assume that each element of Υ minus its expectation $\mathbb{E}\Upsilon_i$ is bounded by a constant

c; we can derive c from bounds on anticipated errors in distance measurements and odometry measurements.

$$|\Upsilon_{i,t} - \mathbb{E}\Upsilon_i| \leq c, \quad \forall_{i,t}$$

Then the Azuma-Hoeffding inequality bounds the probability that the empirical sum differs too much from its population value: for any $\alpha \geq 0$ and any i,

$$\mathbb{P}\left[\left|\sum_{t=1}^{T}(\Upsilon_{i,t} - \mathbb{E}\Upsilon_i)\right| \geq \alpha\right] \leq 2e^{-\alpha^2/2Tc^2}$$

If we pick $\alpha = \sqrt{2Tc^2\log(T)}$, then we can rewrite the probability in terms of T:

$$\mathbb{P}\left[\frac{1}{T}\left|\sum_{t=1}^{T}(\Upsilon_{i,t} - \mathbb{E}\Upsilon_i)\right| \geq c\sqrt{\frac{2\log(T)}{T}}\right] \leq 2e^{-\log(T)}$$

which means that the probability decreases as $O(\frac{1}{T})$ and the threshold decreases as $\tilde{O}(\frac{1}{\sqrt{T}})$.

We can then use a union bound over all $(2N)^2$ covariance elements (since $\widehat{Y} \in \mathbb{R}^{2N\times T}$):

$$\mathbb{P}\left[\forall i \left|\frac{1}{T}\sum_{t=1}^{T}\Upsilon_{i,t} - \mathbb{E}\Upsilon_i\right| \geq c\sqrt{\frac{2\log(T)}{T}}\right] \leq 8N^2/T$$

That is, with high probability, the entire empirical covariance matrix $\widehat{M}$ will be close (in max-norm) to its expectation.

Next we use the continuity of the SVD to show that the learned subspace approaches its true value. Let $\widehat{M} = M + E$, where E is the perturbation (so the largest element of E is bounded). Let $\widehat{U}$ be the output of SVD, and let U be the population value (the top singular vectors of the true M). Let Ψ be the matrix of canonical angles between range(U) and range$(\widehat{U})$. Since we know the exact rank of the true M (either 4 or 7), the last (4th or 7th) singular value of M will be positive; call it $\gamma > 0$. So, by Theorem 4.4 of Stewart and Sun [104],

$$||\sin\Psi||_2 \leq \frac{||E||_2}{\gamma}$$

This result uses a 2-norm bound on E, but the bound we showed above is in terms of the largest element of E. But, the 2-norm can be bounded in terms of the largest element:

$$||E||_2 \leq N\max_{ij}|E_{ij}|$$

Finally, the result is that we can bound the canonical angle:

$$||\sin\Psi||_2 \leq \frac{Nc\sqrt{\frac{2\log(T)}{T}}}{\gamma}$$

In other words, the canonical angle shrinks at a rate of $\tilde{O}(\frac{1}{\sqrt{T}})$, with probability at least $1 - \frac{8N^2}{T}$.

12.2.3 The Robot as a Nonlinear Dynamical System

Once we have learned an interpretable state space via the algorithm of Section 10.3.3, we can simply write down the nominal robot dynamics in this space. The accuracy of the resulting model will depend on how well our sensors and actuators follow the nominal dynamics, as well as how well we have learned the transformation S to the interpretable version of the state space.

In more detail, we model the robot as a controlled nonlinear dynamical system. The evolution is governed by the following state space equations, which generalize (10.1):

$$s_{t+1} = f(s_t, a_t) + \epsilon_t \tag{12.5}$$

$$o_t = h(s_t) + \nu_t \tag{12.6}$$

Here $s_t \in \mathbb{R}^k$ denotes the hidden state, $a_t \in \mathbb{R}^l$ denotes the control signal, $o_t \in \mathbb{R}^m$ denotes the observation, $\epsilon_t \in \mathbb{R}^k$ denotes the state noise, and $\nu_t \in \mathbb{R}^m$ denotes the observation noise. For our range-only system, following the decomposition of Section 10.3, we have:

$$s_t = \begin{bmatrix} 1 \\ -x_t \\ -y_t \\ (x_t^2 + y_t^2)/2 \\ -\cos(\theta_t) \\ -\sin(\theta_t) \\ \frac{x_{t+1}^2 - x_t^2 + y_{t+1}^2 - y_t^2}{2v_t} \end{bmatrix}, \; o_t = \begin{bmatrix} d_{1t}^2/2 \\ \vdots \\ d_{Nt}^2/2 \\ \frac{d_{1t+1}^2 - d_{1t}^2}{2v_t} \\ \vdots \\ \frac{d_{Nt+1}^2 - d_{Nt}^2}{2v_t} \end{bmatrix}, \; a_t = \begin{bmatrix} v_t \\ \cos(\omega_t) \\ \sin(\omega_t) \end{bmatrix} \tag{12.7}$$

Here v_t and ω_t are the translation and rotation calculated from the robot's odometry. A nice property of this model is that expected observations are a *linear* function of state:

$$h(s_t) = Cs_t \tag{12.8}$$

The dynamics, however, are *nonlinear*: see Eq. 12.9, which can easily be derived from the basic kinematic motion model for a wheeled robot [111].

$$f(s_t, a_t) = \begin{bmatrix} 1 \\ -x_t - v_t \cos(\theta_t) \\ -y_t - v_t \sin(\theta_t) \\ \frac{x_t^2 + y_t^2}{2} + v_t x_t \cos(\theta_t) + v_t y_t \sin(\theta_t) + \frac{v_t^2 \cos^2(\theta_t) + v_t^2 \sin^2(\theta_t)}{2} \\ -\cos(\theta_t)\cos(\omega_t) + \sin(\theta_t)\sin(\omega_t) \\ -\sin(\theta_t)\cos(\omega_t) + \cos(\theta_t)\sin(\omega_t) \\ [x_t \cos(\theta_t)\cos(\omega_t) - x_t \sin(\theta_t)\sin(\omega_t) + v_t \cos^2(\theta_t)\cos(\omega_t) + \\ y_t \sin(\theta_t)\cos(\omega_t) - y_t \sin(\omega_t)\cos(\theta_t) + v_t \sin^2(\theta_t)\cos(\omega_t) - \\ 2v_t \cos(\theta_t)\sin(\theta_t)\sin(\omega_t)] \end{bmatrix} \tag{12.9}$$

Robot System Identification

To apply the model of Section 12.2.3, it is essential that we maintain states in the physical coordinate frame, and not just the linearly transformed coordinate frame—i.e., $\widehat{C}$ and not $\widehat{U} = \widehat{C}S^{-1}$. So, to use this model, we must first learn S either by regression or by metric upgrade.

However, it is possible instead to use *system identification* to learn to filter directly in the raw state space $\widehat{U}$. We conjecture that it may be more robust to do so, since we will not be sensitive to errors in the metric upgrade process (errors in learning S), and since we can learn to compensate for some deviations from the nominal model of Section 12.2.3.

To derive our system identification algorithm, we can explicitly rewrite $f(s_t, a_t)$ as a nonlinear feature-expansion map followed by a linear projection. Our algorithm will then just be to use linear regression to learn the linear part of f.

First, let's look at the dynamics for the special case of $S = I$. Each additive term in Eq. 12.9 is the product of at most two terms in s_t and at most two terms in a_t. Therefore, we define $\phi(s_t, a_t) := s_t \otimes s_t \otimes \bar{a}_t \otimes \bar{a}_t$, where $\bar{a}_t = [1, a_t]^\top$ and $\otimes$ is the Kronecker product. (Many of the dimensions of $\phi(s_t, a_t)$ are duplicates; for efficiency we would delete these duplicates, but for simplicity of notation we keep them.) Each additive term in Eq. 12.9 is a multiple of an element of $\phi(s_t, a_t)$, so we can write the dynamics as:

$$s_{t+1} = N\phi(s_t, a_t) + \epsilon_t \tag{12.10}$$

where N is a linear function that picks out the correct entries to form Eq. 12.9.

Now, given an invertible matrix S, we can rewrite $f(s_t, a_t)$ as an *equivalent* function in the transformed state space:

$$Ss_{t+1} = \bar{f}(Ss_t, a_t) + S\epsilon_t \tag{12.11}$$

To do so, we use the identity $(Ax) \otimes (By) = (A \otimes B)(x \otimes y)$. Repeated application yields

$$\begin{aligned} \phi(Ss_t, a_t) &= Ss_t \otimes Ss_t \otimes \bar{a}_t \otimes \bar{a}_t \\ &= (S \otimes S \otimes I \otimes I)(s_t \otimes s_t \otimes \bar{a}_t \otimes \bar{a}_t) \\ &= \bar{S}\,\phi(s_t, a_t) \end{aligned} \tag{12.12}$$

where $\bar{S} = S \otimes S \otimes I \otimes I$. Note that $\bar{S}$ is invertible (since $\text{rank}(A \otimes B) = \text{rank}(A)\,\text{rank}(B)$); so, we can write

$$\bar{f}(Ss_t, a_t) = SN\bar{S}^{-1}\bar{S}\phi(s_t, a_t) = Sf(s_t, a_t) \tag{12.13}$$

Using this representation, we can *learn* the linear part of f, $SN\bar{S}^{-1}$, directly from our state estimates: we just do a linear regression from $\phi(Ss_t, a_t)$ to Ss_{t+1}.

For convenience, we summarize the entire learning algorithm (state space discovery followed by system identification) as Algorithm 2.

Filtering with the Extended Kalman Filter

Whether we learn the dynamics through system identification or simply write them down in the interpretable version of our state space, we will end up with a transition model of the form (12.10)

and an observation model of the form (12.8). Given these models, it is easy to write down an EKF which tracks the robot state. The measurement update is just a standard Kalman filter update (see, e.g., [111]), since the observation model is linear. For the motion update, we need a Taylor approximation of the expected state at time $t+1$ around the current MAP state $\hat{s}_t$, given the current action a_t:

$$s_{t+1} - s_t \approx N[\phi(\hat{s}_t, a_t) + \tfrac{d\phi}{ds}\big|_{\hat{s}_t}(s_t - \hat{s}_t)] \tag{12.14}$$

$$\tfrac{d\phi}{ds}\big|_{\hat{s}} = (\hat{s} \otimes I + I \otimes \hat{s}) \otimes \bar{a}_t \otimes \bar{a}_t \tag{12.15}$$

We simply plug this Taylor approximation into the standard Kalman filter motion update (e.g., [111]).

Algorithm 2 Robot System Identification

In: T *i.i.d.* pairs of observations $\{o_t, a_t\}_{t=1}^{T}$, measurement model for 4 landmarks $C_{1:4}$ (by e.g. GPS)

Out: measurement model $\widehat{C}$, motion model $\widehat{N}$, robot states $\widehat{X}$ (the tth column is state s_t)

1: Collect observations and odometry into a matrix $\widehat{Y}$ (Eq. 10.7)
2: Find the the top 7 singular values and vectors: $\langle \widehat{U}, \widehat{\Lambda}, \widehat{V}^\top \rangle \leftarrow \text{SVD}(\widehat{Y}, 7)$
3: Find the transformed measurement matrix $\widehat{C}S^{-1} = \widehat{U}$ and robot states $S\widehat{X} = \widehat{\Lambda}\widehat{V}^\top$
4: Compute a matrix Φ with columns $\Phi_t = \phi(Ss_t, a_t)$.
5: Compute dynamics: $S\widehat{N}\bar{S}^{-1} = S\widehat{X}_{2:T}(\Phi_{1:T-1})^\dagger$
6: Compute the partial S^{-1}: $\widehat{S}^{-1} = C_{1:4}^{-1}(\widehat{C}_{1:4}S^{-1})$ where $\widehat{C}S^{-1}$ comes from step 3. $\widehat{S}^{-1}\widehat{X}$ gives us the x, y coordinates of the states. These can be used to find $\widehat{X}$ (see Section 10.3.2)
7: Given $\widehat{X}$, we can compute the full S as $S = (S\widehat{X})\widehat{X}^\dagger$
8: Finally, from steps 3,5, and 7, we find the interpretable measurement model $(\widehat{C}S^{-1})S$ and motion model $N = S^{-1}(SN\bar{S}^{-1})\bar{S}$.

Bibliography

[1] A. Anandkumar, K. Chaudhuri, D. Hsu, S. Kakade, L. Song, and T. Zhang. Spectral methods for learning multivariate latent tree structure. In *Neural Information Processing Systems (NIPS)*, 2011. 1.1

[2] Animashree Anandkumar, Dean P. Foster, Daniel Hsu, Sham M. Kakade, and Yi-Kai Liu. Two svds suffice: Spectral decompositions for probabilistic topic modeling and latent dirichlet allocation. *CoRR*, abs/1204.6703, 2012. 1.1

[3] K. J. Aström. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965. 9.1

[4] Raphaël Bailly. Qwa: Spectral algorithm. *Journal of Machine Learning Research - Proceedings Track*, 20:147–163, 2011. 1.1

[5] Raphaël Bailly, Amaury Habrard, and François Denis. A spectral approach for probabilistic grammatical inference on trees. In *ALT*, pages 74–88, 2010. 1.1

[6] C. Baker. Joint measures and cross-covariance operators. *Transactions of the American Mathematical Society*, 186:273–289, 1973. 5.1.2

[7] Borja Balle, Ariadna Quattoni, and Xavier Carreras. Local loss optimization in operator models: A new insight into spectral learning. 2012. 1.1

[8] Michael Biggs, Ali Ghodsi, Dana Wilkinson, and Michael Bowling. Action respecting embedding. In *In Proceedings of the Twenty-Second International Conference on Machine Learning*, pages 65–72, 2005. 10.1, 10.2.4

[9] Jeff Bilmes. A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. Technical Report, ICSI-TR-97-021, 1997. 8.1

[10] Byron Boot and Geoffrey J. Gordon. A spectral learning approach to range-only slam. *http://arxiv.org/abs/1207.2491*, 2012. 10

[11] B. Boots. Learning Stable Linear Dynamical Systems. Data Analysis Project, Carnegie Mellon University, 2009. 2.1

[12] Byron Boots and Geoff Gordon. Predictive state temporal difference learning. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 271–279. 2010. 6.1, 10.2.2

[13] Byron Boots and Geoffrey Gordon. Two-manifold problems with applications to nonlinear system identification. In *Proc. 29th Intl. Conf. on Machine Learning (ICML)*, 2012. 5.5

[14] Byron Boots, Sajid M. Siddiqi, and Geoffrey J. Gordon. Closing the learning-planning loop with predictive state representations. In *Proceedings of Robotics: Science and Systems VI*, 2010. 6.1, 8.3.2, 9.1, 9.3.2, 9.3.5, 10.1, 10.2.2

[15] Byron Boots, Sajid Siddiqi, and Geoffrey Gordon. An online spectral learning algorithm for partially observable nonlinear dynamical systems. In *Proceedings of the 25th National Conference on Artificial Intelligence (AAAI-2011)*, 2011. 10.2.2, 10.3.4

[16] Michael Bowling, Peter McCracken, Michael James, James Neufeld, and Dana Wilkinson. Learning predictive state representations using non-blind policies. In *Proc. ICML*, 2006. 3.4, 4.1.1, 6.2.1, 8.1

[17] Justin Boyan. Least-squares temporal difference learning. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 49–56. Morgan Kaufmann, 1999. 9.1

[18] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. 7.1

[19] Steven J. Bradtke and Andrew G. Barto. Linear least-squares algorithms for temporal difference learning. In *Machine Learning*, pages 22–33, 1996. 9.1, 9.2.1

[20] Matthew Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and its Applications*, 415(1):20–30, 2006. 6.1, 6.3.1, 6.7, 10.3.4

[21] Emmanuel J. Candès and Yaniv Plan. Matrix completion with noise. *CoRR*, abs/0903.3131, 2009. 10.3.4

[22] Anthony R. Cassandra, Leslie P. Kaelbling, and Michael R. Littman. Acting Optimally in Partially Observable Stochastic Domains. In *Proc. AAAI*, 1994. 3.1, 8.1, 9.1

[23] David Choi and Benjamin Roy. A generalized kalman filter for fixed point approximation and efficient temporal-difference learning. *Discrete Event Dynamic Systems*, 16(2):207–239, 2006. ISSN 0924-6703. doi: http://dx.doi.org/10.1007/s10626-006-8134-8. 9.4.2, 9.5, 12.1.2

[24] N. L. C. Chui and J. M. Maciejowski. Realization of stable models with subspace methods. *Automatica*, 32(100):1587–1595, 1996. 7.4

[25] Shay B. Cohen, Karl Stratos, Michael Collins, Dean P. Foster, and Lyle H. Ungar. Spectral learning of latent-variable pcfgs. In *ACL (1)*, pages 223–231, 2012. 1.1

[26] Paramveer S. Dhillon, Jordan Rodu, Michael Collins, Dean P. Foster, and Lyle H. Ungar. Spectral dependency parsing with latent variables. In *EMNLP-CoNLL*, pages 205–213, 2012. 1.1

[27] Joseph Djugash. Geolocation with range: Robustness, efficiency and scalability. PhD. Thesis, Carnegie Mellon University, 2010. 10.1, 10.2.1, 10.4.2

[28] Joseph Djugash and Sanjiv Singh. A robust method of localization and mapping using only range. In *International Symposium on Experimental Robotics*, July 2008. 10.1, 10.2.1, 10.4.2

[29] Joseph Djugash, Sanjiv Singh, and Peter Ian Corke. Further results with localization

and mapping using range from radio. In *International Conference on Field and Service Robotics (FSR '05)*, July 2005. 10.1

[30] Eyal Even-dar. Y.: Planning in pomdps using multiplicity automata. In *In: Proceedings of 21st Conference on Uncertainty in Artificial Intelligence (UAI*, pages 185–192, 2005. 9.1

[31] Eyal Even-Dar, Sham M. Kakade, and Yishay Mansour. Planning in POMDPs Using Multiplicity Automata. In *UAI*, 2005. 3.1, 8.1

[32] Brian Ferris, Dieter Fox, and Neil Lawrence. WiFi-SLAM using Gaussian process latent variable models. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 2480–2485, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. 10.1, 10.2.4

[33] Dean P. Foster, Jordan Rodu, and Lyle H. Ungar. Spectral dimensionality reduction for hmms. *CoRR*, abs/1203.6130, 2012. 1.1

[34] Kenji Fukumizu, Le Song, and Arthur Gretton. Kernel bayes' rule. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1737–1745. 2011. 5.1.3, 5.1.3, 5.1.4, 5.1.4, 5.1.4, 5.1.4

[35] Zoubin Ghahramani and Geoffrey E. Hinton. Parameter estimation for Linear Dynamical Systems. Technical Report CRG-TR-96-2, U. of Toronto, Department of Comp. Sci., 1996. 2, 2.2

[36] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1996. 9.3.1

[37] Steffen Grunewalder, Guy Lever, Luca Baldassarre, Massimiliano Pontil, and Arthur Gretton. Modelling transition dynamics in mdps with rkhs embeddings. *CoRR*, abs/1206.4655, 2012. 5

[38] H. Jaeger, M. Zhao, A. Kolling. Efficient Training of OOMs. In *NIPS*, 2005. 3.4, 8.1

[39] Roger Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985. 2.2, 7.2.2, 5

[40] R. Horst and P. M. Pardalos, editors. *Handbook of Global Optimization*. Kluwer, 1995. 7.1

[41] Harold Hotelling. The most predictable criterion. *Journal of Educational Psychology*, 26: 139–142, 1935. 9.3.1

[42] Daniel Hsu, Sham Kakade, and Tong Zhang. A spectral algorithm for learning hidden Markov models. In *COLT*, 2009. 3.4, 6.1, 10.2.2

[43] Masoumeh T. Izadi and Doina Precup. Point-based Planning for Predictive State Representations. In *Proc. Canadian AI*, 2008. 8.1, 8.2, 8.2, 8.2, 8.3.5

[44] Herbert Jaeger. Observable operator models for discrete stochastic time series. *Neural Computation*, 12:1371–1398, 2000. 3.1, 3.3, 3.4, 8.1, 9.1

[45] Michael R. James, Ton Wessling, and Nikos A. Vlassis. Improving approximate value

iteration using memories and predictive state representations. In *AAAI*, 2006. 8.1, 8.2, 8.2, 8.2, 8.3.5, 9.3.4

[46] Jeff Johns, Sridhar Mahadevan, and Chang Wang. Compact spectral bases for value function approximation using kronecker factorization. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pages 559–564. AAAI Press, 2007. ISBN 978-1-57735-323-2. 9.1

[47] Nicholas K. Jong and Peter Stone. Towards Employing PSRs in a Continuous Domain. Technical Report UT-AI-TR-04-309, University of Texas at Austin, 2004. 8.3.5

[48] R.E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 1960. 2, 2.1.1, 2.2.1, 2.2.1

[49] T. Kanade and D.D. Morris. Factorization methods for structure from motion. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 356(1740):1153–1173, 1998. 10.1

[50] George A Kantor and Sanjiv Singh. Preliminary results in range-only localization and mapping. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA '02)*, volume 2, pages 1818 – 1823, May 2002. 10.1, 10.2.1

[51] Tohru Katayama. *Subspace Methods for System Identification: A Realization Approach*. Springer, 2005. 2, 2.1.1, 2.4

[52] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM*, 41:67–95, 1994. See also Proceedings of the 21st ACM Symposium on the Theory of Computing, pp. 433-444, 1989, ACM Press. 3.4

[53] Athanasios Kehagias, Joseph Djugash, and Sanjiv Singh. Range-only SLAM with interpolated range data. Technical Report CMU-RI-TR-06-26, Robotics Institute, May 2006. 10.1, 10.4.2

[54] E. Keogh and T. Folias. The UCR Time Series Data Mining Archive, 2002. URL http://www.cs.ucr.edu/ eamonn/TSDMA/index.html. 7.1

[55] J. Zico Kolter and Andrew Y. Ng. Regularization and feature selection in least-squares temporal difference learning. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 521–528, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: http://doi.acm.org/10.1145/1553374.1553442. 9.1, 9.3

[56] Derek Kurth, George A Kantor, and Sanjiv Singh. Experimental results in range-only localization with radio. In *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '03)*, volume 1, pages 974 – 979, October 2003. 10.1, 10.2.1

[57] Seth L. Lacy and Dennis S. Bernstein. Subspace identification with guaranteed stability using constrained optimization. In *Proc. American Control Conference*, 2002. (document), 7.1, 7.2, 7.1, 7.3, 7.4

[58] Seth L. Lacy and Dennis S. Bernstein. Subspace identification with guaranteed stability using constrained optimization. *IEEE Transactions on Automatic Control*, 48(7):1259–1263, July 2003. 7.2, 7.3, 7.4

[59] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *J. Mach. Learn.*

Res., 4:1107–1149, 2003. ISSN 1532-4435. 9.1, 1

[60] Rui Li, Rui Li, Stan Sclaroff Phd, Margrit Betke Phd, and David J. Fleet. S.: Simultaneous learning of nonlinear manifold and dynamical models for high-dimensional time series. In *In: Proc. ICCV (2007*, 2007. 9.1

[61] Michael Littman, Richard Sutton, and Satinder Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems (NIPS)*, 2002. 3.1, 8.1, 9.1

[62] L. Ljung. *System Identification: Theory for the user*. Prentice Hall, 2nd edition, 1999. 2, 2.1.1, 2.2.1, 7.4

[63] M. Zhao and H. Jaeger and M. Thon. A Bound on Modeling Error in Observable Operator Models and an Associated Learning Algorithm. *Neural Computation*. 8.1

[64] Sridhar Mahadevan. Representation policy iteration. In *Proceedings of the Proceedings of the Twenty-First Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 372–379, Arlington, Virginia, 2005. AUAI Press. 9.1

[65] Sridhar Mahadevan. Samuel meets amarel: automating value function approximation using global state space analysis. In *AAAI'05: Proceedings of the 20th national conference on Artificial intelligence*, pages 1000–1005. AAAI Press, 2005. ISBN 1-57735-236-x. 9.1

[66] A. McCallum. Reinforcement Learning with Selective Perception and Hidden State. PhD Thesis, University of Rochester, 1995. 8.1

[67] Peter McCracken and Michael Bowling. Online discovery and learning of predictive state representations. In *Proc. NIPS*, 2005. 8.1

[68] Ha Quang Minh, Marco Cristani, Alessandro Perina, and Vittorio Murino. A regularized spectral algorithm for hidden markov models with applications in computer vision. In *CVPR*, pages 2384–2391, 2012. 1.1

[69] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598. AAAI, 2002. 10.4.2

[70] Kevin Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UC Berkeley, 2002. 2.1.1

[71] Andrew Y. Ng and H. Jin Kim. Stable adaptive control with online learning. In *Proc. NIPS*, 2004. 7.2.2

[72] Andrew Y. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Inverted autonomous helicopter flight via reinforcement learning. In *In International Symposium on Experimental Robotics*, 2004. 8.1

[73] Y Nishiyama, A Boularias, A Gretton, and K Fukumizu. Hilbert space embeddings of pomdps. 2012. 5

[74] A. Parikh, L. Song, and E. P. Xing. A spectral algorithm for latent tree graphical models. In *International Conference on Machine Learning (ICML)*, 2011. 1.1

[75] Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael L.

Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 752–759, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: http://doi.acm.org/10.1145/1390156.1390251. 9.1

[76] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. IJCAI*, 2003. 8.2, 8.2, 8.3.5

[77] Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research (JAIR)*, 27:335–380, 2006. 8.1

[78] Pascal Poupart and Craig Boutilier. Value-directed compression of pomdps. In *NIPS*, pages 1547–1554, 2002. 9.1, 9.3.1, 9.3.4

[79] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE*, 77(2):257–285, 1989. 3.1

[80] Ali Rahimi and Ben Recht. Random features for large-scale kernel machines. In *Neural Infomration Processing Systems*, 2007. 6.4

[81] J. Ramos, S. Siddiqi, A. Dubrawski, G. Gordon, and A. Sharma. Automatic state discovery for unstructured audio scene classification. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2010. 5.4.3

[82] H. Rauch. Solutions to the linear smoothing problem. In *IEEE Transactions on Automatic Control*, 1963. 2.1.1

[83] Gregory C. Reinsel and Rajabather Palani Velu. *Multivariate Reduced-rank Regression: Theory and Applications*. Springer, 1998. 9.3.1

[84] Matthew Rosencrantz, Geoffrey J. Gordon, and Sebastian Thrun. Learning low dimensional predictive representations. In *Proc. ICML*, 2004. 3.4, 6.1, 8.3, 9.1, 9.3.2, 9.3.4, 10.2.2

[85] Stéphane Ross and Joelle Pineau. Model-Based Bayesian Reinforcement Learning in Large Structured Domains. In *Proc. UAI*, 2008. 8.1

[86] Yi Shang, Wheeler Ruml, Ying Zhang, and Markus P. J. Fromherz. Localization from mere connectivity. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc '03, pages 201–212, New York, NY, USA, 2003. ACM. 10.1, 10.2.4

[87] Guy Shani, Ronen I. Brafman, and Solomon E. Shimony. Model-based online learning of POMDPs. In *Proc. ECML*, 2005. 8.1

[88] Sajid Siddiqi, Byron Boots, and Geoffrey J. Gordon. A constraint generation approach to learning stable linear dynamical systems. In *Proc. NIPS*, 2007. 2.3.2

[89] Sajid Siddiqi, Byron Boots, Geoffrey J. Gordon, and Artur W. Dubrawski. Learning stable multivariate baseline models for outbreak detection. *Advances in Disease Surveillance*, 4: 266, 2007. 1, 7.3.2

[90] Sajid Siddiqi, Byron Boots, and Geoffrey J. Gordon. Reduced-rank hidden Markov mod-

els. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS-2010)*, 2010. 5.4, 5.4.1, 5.4.3, 6.1, 6.5.1, 9.4.1, 10.2.2

[91] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, 1986. 8.3.2

[92] Satinder Singh, Michael L. Littman, Nicholas K. Jong, David Pardoe, and Peter Stone. Learning predictive state representations. In *Proc. ICML*, 2003. 8.1

[93] Satinder Singh, Michael James, and Matthew Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *Proc. UAI*, 2004. 3.1, 3.2.1, 3.3, 6.1, 8.1, 9.1, 9.3.4

[94] A.J. Smola, A. Gretton, L. Song, and B. Schölkopf. A Hilbert space embedding for distributions. In E. Takimoto, editor, *Algorithmic Learning Theory*, Lecture Notes on Computer Science. Springer, 2007. 5, 5.1.1, 5.1.1, 5.1.2

[95] S. Soatto and A. Chiuso. Dynamic data factorization. Technical Report UCLA-CSD 010001, UCLA, 2001. 9.3.1

[96] S. Soatto, G. Doretto, and Y. Wu. Dynamic Textures. *Intl. Conf. on Computer Vision*, 2001. 7.1

[97] E. J. Sondik. The optimal control of partially observable Markov processes. PhD. Thesis, Stanford University, 1971. 3.1, 8.1, 9.1

[98] L. Song, J. Huang, A. Smola, and K. Fukumizu. Hilbert space embeddings of conditional distributions. In *International Conference on Machine Learning*, 2009. 5.1.3

[99] L. Song, B. Boots, S. M. Siddiqi, G. J. Gordon, and A. J. Smola. Hilbert space embeddings of hidden Markov models. In *Proc. 27th Intl. Conf. on Machine Learning (ICML)*, 2010. 5, 5.3.3, 5.4, 5.5, 6.4, 6.5.2

[100] L. Song, A. Gretton, and C. Guestrin. Nonparametric tree graphical models. In *Artificial Intelligence and Statistics (AISTATS)*, 2010. 1.1, 5.1.3

[101] L. Song, A. Parikh, and E. Xing. Kernel embeddings of latent tree graphical models. In *Neural Information Processing Systems (NIPS)*, 2011. 1.1

[102] Matthijs T. J. Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005. 8.2, 8.3.5

[103] B. Sriperumbudur, A. Gretton, K. Fukumizu, G. Lanckriet, and B. Schölkopf. Injective Hilbert space embeddings of probability measures. 2008. 5, 5.1.1

[104] G. W. Stewart and Ji-Guang Sun. *Matrix Perturbation Theory*. Academic Press, 1990. 12.2.2

[105] Richard S. Sutton. Learning to predict by the methods of temporal differences. In *Machine Learning*, pages 9–44. Kluwer Academic Publishers, 1988. 9.1

[106] Erik Talvitie and Satinder Singh. Learning to make predictions in partially observable environments without a generative model. *J. Artif. Intell. Res. (JAIR)*, 42:353–392, 2011. 1.1

[107] Russ Tedrake, Zack Jackowski, Rick Cory, John William Roberts, and Warren Hoburg.

Learning to fly like a bird. In *Under Review*, 2009. 8.1

[108] Joshua B. Tenenbaum, Vin De Silva, and John Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000. doi: 10.1126/science.290.5500.2319. 10.2.4

[109] Sebastiaan Terwijn. On the learnability of hidden Markov models. In *ICGI '02: Proceedings of the 6th International Colloquium on Grammatical Inference*, pages 261–268, London, UK, 2002. Springer-Verlag. ISBN 3-540-44239-1. 3.4

[110] Georgios Theocharous and Sridhar Mahadevan. Compressing pomdps using locality preserving non-negative matrix factorization, 2010. 9.1

[111] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623. 10.1, 10.2.1, 12.2.3, 12.2.3, 12.2.3

[112] Michael E. Tipping and Chris M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B*, 61:611–622, 1999. 10.3.4

[113] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9:137–154, 1992. 10.1, 10.2.3, 10.3.1

[114] B. Triggs. Factorization methods for projective structure and motion. In *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR'96, 1996 IEEE Computer Society Conference on*, pages 845–851. IEEE, 1996. 10.1

[115] John N. Tsitsiklis and Benjamin Van Roy. Optimal stopping of markov processes: Hilbert space theory, approximation algorithms, and an application to pricing high-dimensional financial derivatives. *IEEE Transactions on Automatic Control*, 44:1840–1851, 1997. (document), 9.1, 9.4.2, 9.5, 12.1.2

[116] T. Van Gestel, J. A. K. Suykens, P. Van Dooren, and B. De Moor. Identification of stable models in subspace identification by using regularization. *IEEE Transactions on Automatic Control*, 49(9):1416–1420, 2001. 7.4

[117] P. Van Overschee and B. De Moor. *Subspace Identification for Linear Systems: Theory, Implementation, Applications*. Kluwer, 1996. 2, 2.1.1, 2.2, 2.2, 2.2.1, 2.2.1, 2.4, 7.4, 8.1, 10.1, 10.2.2

[118] M. Wagner. A national retail data monitor for public health surveillance. *Morbidity and Mortality Weekly Report*, 53:40–42, 2004. 7.3.2

[119] Eric Wiewiora. Learning predictive representations from a history. In *Proc. ICML*, 2005. 8.1

[120] David Wingate. Exponential Family Predictive Representations of State. PhD Thesis, University of Michigan, 2008. 8.1

[121] David Wingate and Satinder Singh. On discovery and learning of models with predictive representations of state for agents with continuous actions and observations. In *Proc. AAMAS*, 2007. 8.3, 8.3.1

[122] David Wingate and Satinder Singh. Efficiently learning linear-linear exponential family predictive representations of state. In *Proc. ICML*, 2008. 8.1, 8.3, 8.3.1, 8.3.5

[123] Britton Wolfe, Michael James, and Satinder Singh. Learning predictive state representations in dynamical systems without reset. In *Proc. ICML*, 2005. 3.3, 3.4, 8.1

[124] Takehisa Yairi. Map building without localization by dimensionality reduction techniques. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 1071–1078, New York, NY, USA, 2007. ACM. 10.1, 10.2.4

MACHINE LEARNING
DEPARTMENT
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA15213

Carnegie Mellon.

Carnegie Mellon University does not discriminate in admission, employment, or administration of its programs or activities on the basis of race, color, national origin, sex, handicap or disability, age, sexual orientation, gender identity, religion, creed, ancestry, belief, veteran status, or genetic information. Futhermore, Carnegie Mellon University does not discriminate and if required not to discriminate in violation of federal, state, or local laws or executive orders.

Inquiries concerning the application of and compliance with this statement should be directed to the vice president for campus affairs, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone, 412-268-2056