Algorithms and Representations for Reinforcement Learning

Thesis submitted for the degree of
Doctor of Philosophy
by

Yaakov Engel

Submitted to the Senate of the Hebrew University
April 2005

This work was carried out under the supervision of Prof. Ron Meir and Prof. Naftali Tishby.

"If we knew what it was we were doing, it would not be called research, would it?"
– Albert Einstein.

"I may not have gone where I intended to go, but I think I have ended up where I intended to be."
– Douglas Adams

To my parents.

Acknowledgments

During the past decade or so, little has remained constant in my tumultuous life. Places of dwelling have been changed, jobs have been found and lost, life-partners have come and gone (and come), a baby was born, but through all this, my persistent assertion "I'm working on my Ph.D.!" remained a fixed constant. This thesis is the product of that long period, throughout which I benefited from the assistance and support of quite a few individuals.

I am greatly indebted to my advisor Ronny Meir, who was courageous enough to supervise a thesis not in his main field of expertise. He allowed me to follow my own research agenda and to slowly plow my way through this thesis work, thankfully not despairing of me all this time. From Ronny I learned the importance of formulating problems in exact mathematical terms, and of expressing my ideas clearly, both verbally and in writing. His somewhat unnerving ability to accurately pinpoint, almost every time, the appropriate references to answer my numerous research questions testifies to his wide and deep knowledge of science.

Shie Mannor is both a friend and a collaborator in most of my published and unpublished work. His mathematical proficiency and quick grasp of core research issues were both inspiring and useful. At any time of day, I could barge into Shie's room to discuss anything from advanced probability and game theory to C-language programming and Latex commands, usually coming out with the answer to my query.

My formal advisor at the Hebrew university, Tali Tishby, is in fact responsible for originally sparking my interest in reinforcement learning, by proposing it as a topic for a graduate student seminar in the second year of my studies. Although I did not have the opportunity to work with Tali, I did benefit greatly from his courses in machine learning, from which I acquired a solid theoretical base, as well as the enthusiasm for machine learning research.

Nahum Shimkin was helpful and insightful in discussing research matters on several occasions. Nahum's course on Kalman filters played an important role in shaping my view on learning and estimation, and consequently in forming this thesis. With Ishai Menache I shared numerous cups of coffee as well as thoughts on reinforcement learning algorithms. Ishai also helped review some of my papers.

Yair Weiss and Yishay Mansour, who are the two other members of my research committee, have been kind enough to spare me some of their time on a few occasions to discuss my research, providing me with some useful comments. Ina Krinsky and Johanan Erez from the VISL laboratory in the Technion, made me feel welcome, and did their best to accommodate my every whim, with regard to software and hardware issues. Ruthi Suchi from the ICNC was very helpful in the last stages of my thesis work, especially by assisting me in tackling the formalities of submitting the thesis book.

Finally, and most importantly, I would like to thank my parents, my sister and brother and their families, and my own little family – Mitzkin and Yoyo, for their faith in me, their love and support, without which none of this would ever have come to pass. Thank you!

Abstract

Machine Learning is a field of research aimed at constructing intelligent machines that gain and improve their skills by learning and adaptation. As such, Machine Learning research addresses several classes of learning problems, including for instance, supervised and unsupervised learning. Arguably, the most ubiquitous and realistic class of learning problems, faced by both living creatures and artificial agents, is known as *Reinforcement Learning*. Reinforcement Learning problems are characterized by a long-term interaction between the learning agent and a dynamic, unfamiliar, uncertain, possibly even hostile environment. Mathematically, this interaction is modeled as a Markov Decision Process (MDP). Probably the most significant contribution of this thesis is in the introduction of a new class of Reinforcement Learning algorithms, which leverage the power of a statistical set of tools known as Gaussian Processes. This new approach to Reinforcement Learning offers viable solutions to some of the major limitations of current Reinforcement Learning methods, such as the lack of confidence intervals for performance predictions, and the difficulty of appropriately reconciling exploration with exploitation. Analysis of these algorithms and their relationship with existing methods also provides us with new insights into the assumptions underlying some of the most popular Reinforcement Learning algorithms to date.

Contents

List of Tables

List of Figures

List of Algorithms

Preface

Reinforcement Learning is arguably the most ubiquitous and realistic class of learning problems, faced by both living creatures and artificial, intelligent agents. Reinforcement Learning problems are characterized by a long-term interaction between a learning agent and a dynamic, unfamiliar, uncertain, possibly even hostile environment. Arguably, learning to behave in such environments may be considered as actually *defining* intelligent behavior. In fact, it is my personal view that this definition is both more general and more useful than the notion of intelligence suggested by Turing's famous test (Turing, 1950), as the latter does not admit any non-human or non-verbal forms of intelligence. If one accepts this view of Reinforcement Learning, then one is led to conclude that solving the Reinforcement Learning problem is a necessary and possibly sufficient condition to solving *the* Artificial Intelligence problem, which, simply phrased, is: *How can we build intelligent machines?* It can be further argued that, the more we understand how to create machine intelligence, the better chance we have at understanding our own[1].

This thesis does not claim, nor does it attempt, to solve this difficult problem. The goal I set myself at the very outset was to expand, if only by a small measure, the scope of what is currently solvable using existing Reinforcement Learning architectures and algorithms. The remainder of this thesis book is meant to convince you, the reader, that I have.

[1]As Richard Feynman succinctly put it: *"What I cannot create, I do not understand."* (Feynman, 1988).

Chapter 1

Introduction and Background

1.1 Motivation, Rationale and Overview

Machine Learning is a field of research aimed at constructing intelligent machines that gain and improve their skills by learning and adaptation. As such, Machine Learning research focuses on one specific approach to solving the Artificial Intelligence problem – the learning approach, which is quite distinct from the conventional "good-old-fashioned AI" approach, which focuses on inductive reasoning and search algorithms. Machine Learning research addresses several classes of learning problems, including for instance, supervised and unsupervised learning. Arguably, the most ubiquitous and realistic class of learning problems, for which a relatively mature theory and efficient algorithms exit, is known as *Reinforcement Learning*. Reinforcement Learning problems are characterized by a long-term interaction between a learning *agent* and a dynamic, unfamiliar, uncertain, possibly even hostile *environment*. Mathematically, this interaction is modeled as a Markov Decision Process (MDP).

The last two decades have witnessed an explosion in RL research, resulting in a large number of new algorithms and architectures. However, there remain several fundamental obstacles hindering the widespread application of RL methodology to real-world problems. Such real-world problems are characterized by most, if not all, of the following features:

- Large or even infinite state and/or action spaces.
- Requirement for online learning. Off-line learning is adequate when the environment is completely stationary, however, this is rarely the case.
- Training data is expensive. Unlike supervised learning, where large corpora of data are freely available and immediately usable, in RL, learning data is generated by the interaction of the learning agent with the dynamic system

(real or simulated) it attempts to control. For complex systems, this results in a considerable overhead for each training sample.

- Partial observability. In many problems the state of the system is not completely or directly measurable by the agent.

Even if we leave out of consideration the last item, which will not be addressed in this thesis, it turns out that this list outrules the vast majority of RL algorithms, as viable solution methods to such problems. This is due to the extreme scarcity of provably convergent, online RL algorithms that are capable of dealing with large state-action spaces. Other pressing issues that are not sufficiently well addressed by current methods are:

- How to acquire confidence intervals for performance (i.e. value) predictions?
- How to appropriately reconcile exploration with exploitation?
- How to perform RL online, using nonparametric representations?

During the past decade, Machine Learning research at large has made significant advances, both in the theory and the practice of a special class of nonparametric learning methods known as *kernel methods*. Kernel-based architectures and algorithms have become the subject of intense research and gained much popularity in recent years (see e.g., Schölkopf & Smola, 2002; Shawe-Taylor & Cristianini, 2004), mainly due to their success in achieving state-of-the-art performance levels in several supervised and unsupervised learning domains (e.g., LeCun et al., 1995), and to their solid theoretical grounding. Another attractive feature of kernel methods is their high representational flexibility, allowing them to deal with almost any conceivable object of interest, from handwritten digits (DeCoste & Schölkopf, 2002), strings (Watkins, 1999) and text documents (Joachims, 1998) to genetic microarray data (Brown et al., 2000), sets of vectors (Kondor & Jebara, 2003) and trees (Collins & Duffy, 2001), to mention a few. Reinforcement Learning research, however, has remained largely unaffected by these advances. This seems to be due to the perceived incompatibility between the constraint of online operation imposed by RL tasks, and the typical off-line nature of most kernel algorithms.

With this in mind, our first goal was to develop kernel-based algorithms for regression, which will be capable of operating sequentially and online. The sequentiality requirement means that samples are fed as a temporal stream of data to the learning algorithm. For each sample observed, the algorithm is allowed to perform some update based on the information conveyed by it, after which that sample is discarded, never to be seen again. The online, or real-time constraint is defined as the requirement that the per-sample computational cost, be independent, at least asymptotically, of the number of previously observed samples. In order to be able to bound the per-sample computational cost of any kernel algorithm it is crucial to bound the number of parameters used by it. Typically, however, kernel machines assign at least one parameter per training sample, which is of course incompatible with the real-time requirement. In order to overcome this difficulty, some dimensionality reduction, or sparsification mechanism must be employed to prevent the increase in the number of parameters as new samples are observed. In Chapter 2 such a kernel sparsification algorithm is proposed, along with some theoretical results that illuminate some of its properties. Chapter 3 describes a kernelized form of the celebrated recursive least-squares (RLS) algorithm, which employs the online sparsification method developed in the preceding chapter.

Having completed this part of our research, our plan was to apply these online kernel algorithms to the task of value estimation (a.k.a. policy evaluation), which is a crucial algorithmic component of many RL methods. At this point, however, we have become aware of the existence of a special class of kernel methods, known as Gaussian Processes (GPs), which allow Bayesian reasoning to be applied, in both parametric and nonparametric forms, to supervised learning problems. It quickly became apparent, that using GPs for value estimation would afford us with all the advantages of the algorithms developed in the first part of the thesis, while also providing valuable variance information for value estimates. The second part of the thesis, consisting of Chapter 4, is therefore devoted to the application of GPs to value estimation, and through this, to the solution of the complete RL problem, of learning near-optimal policies. Chapter 5 concludes the thesis with a summary, discussion and suggestions for future work.

1.2 Learning Theory and Kernel Methods

The introduction and implementation of the theory of Mercer kernels to machine learning research is probably due to Aizerman et al. (1964) and Kimeldorf and Wahba (1971). However, it wasn't until three decades later, when the support vector machine (SVM) was presented by Boser et al. (1992), that the influence of the "kernel trick" (see below) was beginning to make its impact on mainstream machine learning research. Since then, kernel methods have assumed a central role in the advancement of both the theory and the practice of machine learning.

Motivated by several theoretical results and by the success of the SVM, machine learning researchers have set out to produce kernelized versions of almost every type of existing learning algorithm, as well as several completely new ones. These include a multitude of classification and regression algorithms as well as various unsupervised learning methods such as clustering, density estimation, principal component analysis (PCA) and independent component analysis (ICA), to mention a few. (see Schölkopf & Smola, 2002; Shawe-Taylor & Cristianini, 2004 and references therein). Some of these have indeed proved to be useful and competitive alternatives to conventional state-of-the-art methods.

1.2.1 Hypotheses, Loss and Risk

A learning problem is invariably defined with respect to a set of objects. In order to apply a learning algorithm to such problems we need to represent each potential object by a *pattern* – a data structure summarizing the available information pertaining to it, which we consider to be relevant to the learning task. We generically denote the set of all possible patterns for a given problem by $\mathcal{X}$, and apart from requiring it to be a set, we do not impose on it any additional structure, for now. For example, if we wished to learn some function defined on the real line, then we could choose $\mathcal{X} = \mathbb{R}$.

In supervised learning problems, it is assumed that a set, Z, of t training samples is available, each consisting of an input pattern $\mathbf{x} \in \mathcal{X}$ and a label or target value $y \in \mathcal{Y}$. In the classic, supervised setting, it is also assumed that the samples are independently and identically distributed (IID)[1] according to to a fixed probability distribution $\rho(\mathbf{x}, y) = \rho(y|\mathbf{x})\rho_{\mathbf{x}}(\mathbf{x})$. We denote the random variable distributed according to $\rho(y|\mathbf{x})$, by $Y(\mathbf{x})$. Furthermore, we assume that $\rho(\mathbf{x}, y)$ is determined by a set of parameters θ; we will therefore use the notation[2] $\rho(\mathbf{x}, y|\theta)$ to remind us of this fact. The dimensionality of θ may be finite, infinite, or even uncountably infinite[3]. For instance, θ may be a function of $\mathbf{x}$, with $Y(\mathbf{x}) = \theta(\mathbf{x}) + N(\mathbf{x})$ being a noise-corrupted measurement of $\theta(\mathbf{x})$. We call θ a *hypothesis*. In the classic, frequentist setting, the "true" hypothesis θ is deterministic and fixed.

By *learning*, we refer to the problem of constructing an estimator $\hat{\theta}_Z$, for θ, based on the training sample Z. The first step in solving such a learning problem is to define a hypothesis space, Θ, in which we intend the learning algorithm to perform its search for the optimal estimator. Obviously, we also need to specify precisely what we mean by "optimal". In other words, we must define a criterion, according to which we will measure and compare the quality of candidate estimators. In order to do so, we need to make a few definitions.

[1]A large majority of the theoretical results in this field depend on the IID assumption (but see Vidyasagar, 2003 for an exception). In learning paradigms other than supervised learning the IID assumption may have to be replaced with weaker assumptions.

[2]In the frequentist setting this does not denote conditioning on θ, as θ is not a random variable. In the Bayesian setting, discussed in the next section, this will change.

[3]In decision theoretical terms, θ defines the *true state of the world*. Conventionally, θ is assumed to consist of a finite set of parameters; however, in the subsequent discussion we wish to maintain generality, and we therefore allow for general nonparametric hypotheses.

Definition 1.2.1 (Loss). *The function (or functional, if Θ is a function space) $\ell : \Theta \times \Theta \to \mathbb{R}^+$, evaluated as $\ell(\theta, \hat{\theta}_Z)$, is called the loss of the estimator $\hat{\theta}_Z$. We require that, for all $\theta \in \Theta$, $\ell(\theta, \theta) = 0$.*

The loss, $\ell(\theta, \hat{\theta}_Z)$ quantifies our feelings concerning the degree of mismatch between the estimated hypothesis $\hat{\theta}_Z$ and the true hypothesis θ. A commonly used loss is the squared loss, defined by

$$\ell(\theta, \hat{\theta}_Z) = \left\| \hat{\theta}_Z - \theta \right\|^2, \tag{1.2.1}$$

where $\| \cdot \|$ is a norm in the space Θ. Note that, in the machine learning literature the loss is sometimes defined with respect to the observations produced by the hypotheses, rather than on the hypotheses themselves[4] (e.g., Vapnik, 1998; Cucker & Smale, 2001). In this context is useful to define the *instantaneous prediction loss.*

Definition 1.2.2 (Instantneous prediction loss). *Denote by $\hat{y}_Z(\mathbf{x})$ the (deterministic) predicted observation produced by the estimator $\hat{\theta}_Z$, at the point $\mathbf{x}$. The function $\ell_{ins} : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$, evaluated as $\ell_{ins}(\mathbf{x}, y, \hat{y}_Z(\mathbf{x}))$ is called the instantaneous prediction loss of $\hat{y}_Z(\mathbf{x})$ with respect to y. It is again required that, for all $\mathbf{x}$ and y, $\ell(\mathbf{x}, y, y) = 0$.*

The instantaneous loss may be used to define a loss function between hypotheses by defining the *average prediction loss.* The average prediction loss is the instantaneous prediction loss, averaged over all possible $(\mathbf{x}, y)$ pairs:

Definition 1.2.3 (Average prediction loss). $\ell_{av} : \Theta \times \Theta \to \mathbb{R}^+$*, evaluated as*

$$\ell_{av}(\theta, \hat{\theta}_Z) = \int_{\mathcal{X},\mathcal{Y}} d\mathbf{x}\, dy\, \rho(\mathbf{x}, y|\theta)\, \ell_{ins}(\mathbf{x}, y, \hat{y}_Z(\mathbf{x})),$$

is called the average prediction loss of the hypothesis $\hat{\theta}_Z$.

It should be noted that ℓ_{av} does not generally satisfy the condition $\ell_{av}(\theta, \theta) = 0$, since the dependence of y on θ is generally stochastic, with an inherently unpredictable component. It is therefore unreasonable to expect that $\hat{y}_Z(\mathbf{x}) = y$ always, unless y depends deterministically on θ. Moreover, $\ell_{av}(\theta, \theta')$ does not necessarily attain its minimum when $\theta' = \theta$; this is known as the *Stein effect* (Robert, 1994).

In supervised learning problems, such as regression or classification, the estimated hypothesis $\hat{\theta}_Z$ and its corresponding predicted observation $\hat{y}_Z$ may be considered as the same object, i.e. $\hat{y}_Z \equiv \hat{\theta}_Z$. However, outside of the supervised domain, things can get more complex, and $\hat{y}_Z$ may have a less direct dependence on $\hat{\theta}_Z(\mathbf{x})$.

[4]This may be useful in cases where what one cares about is the prediction of observations, rather than the estimation of hypotheses.

For instance, in reinforcement learning problems, a value function often has to be estimated from observed rewards. The performance of an algorithm will be measured according to its ability to predict values, not rewards. Here, $\hat{\theta}_Z$ will be the estimated value function, whereas $\hat{y}_Z$ will be the corresponding reward function. As far as possible, we try to maintain the discussion as general as possible, and we therefore regard $\hat{\theta}_Z$ and $\hat{y}_Z$ as distinct entities.

In the frequentist setting, the training sample Z is a random variable, which depends on the deterministic θ. In order to induce an ordering on the set of estimators, which is independent of the training sample, we define the (frequentist) *risk*. The risk $\mathcal{R}$ of an estimator $\hat{\theta}_Z$, is the expected loss accrued by $\hat{\theta}_Z$, with the expectation taken over training samples Z of fixed size t.

Definition 1.2.4 (Risk). *Let $\rho_Z(Z|\theta)$ be the probability (density) of observing the sequence of samples Z, in a model specified by θ. Then the risk of the estimator $\hat{\theta}$ is*

$$\mathcal{R}(\theta, \hat{\theta}) = \int dZ \, \rho_Z(Z|\theta) \, \ell(\theta, \hat{\theta}_Z).$$

The risk tells us how well, on average, the hypothesis $\hat{\theta}_Z$ performs. It is typically assumed that the training samples are sampled from the distribution ρ, i.e. that $\rho_Z(Z|\theta) = \rho^t(X, Y)$. The optimal hypothesis is the one minimizing the Risk.

A difficulty associated with the frequentist approach, is that the risk, as defined above, depends on the true, but unknown hypothesis θ. This means that we cannot use the frequentist risk to induce an ordering on the set of estimators θ_Z, which is independent of θ. Moreover, even for a fixed θ, the estimator minimizing the risk, is optimal in the frequentist sense. That is, if we indefinitely repeat the same experiment, each time with a new training sample Z generated by $\rho_Z(Z|\theta)$, the minimum risk estimator will incur the lowest total loss. However, typically, one is handed a specific training sample Z, and would therefore prefer to have a decision procedure that behaves optimally for *that* sample (see Robert, 1994 for a thorough discussion of these issues).

Another, general, problem is that we usually do not know what form $\rho(\mathbf{x}, y|\theta)$ takes, and therefore we cannot minimize the risk directly. This remains true if we plug-in the average prediction loss ℓ_{av} as the loss used in the risk. A simple solution is to minimize the empirical risk.

Definition 1.2.5 (Empirical risk). *The empirical risk of the estimator $\hat{\theta}_Z$, for the training sample Z is*

$$\mathcal{R}_{emp}(Z, \hat{\theta}_Z) = \frac{1}{t} \sum_{i=1}^{t} \ell_{ins}(\mathbf{x}_i, y_i, \hat{y}_Z(\mathbf{x}_i)).$$

For a parametric $\hat{\theta}$, if the number of samples is of the order of the number of parameters, this solution is prone to overfitting. If $\hat{\theta}$ is nonparametric, or if the number of samples is smaller than the number of parameters, the situation is even worse, as the problem becomes ill-posed. A generic solution, the form of which is in agreement with the solutions suggested by several different learning principles, such as regularization theory (Tikhonov & Arsenin, 1977; Girosi et al., 1995; Evgeniou et al., 2000), Bayesian inference (Gibbs & MacKay, 1997; Williams, 1999) and Vapnik's structural risk minimization (SRM) (Vapnik, 1995; Vapnik, 1998), is the following:

Minimize the sum of the empirical loss and a complexity penalty term.

In Vapnik's (frequentist) SRM framework, this is the result of minimizing an upper bound on the risk, while in the Bayesian framework, this is the result of maximizing the posterior probability of the hypothesis, conditioned on the observed data, subject to Gaussianity assumptions.

The Bayesian approach affords some advantages not available in the classical frequentist approaches, such as Vapnik's SRM, as they allow for an immediate probabilistic interpretation of the regularization (penalty) term and of the resulting estimators. Let us now consider the Bayesian alternative.

1.2.2 The Bayesian View

In the frequentist setting it is assumed that there is one "true" hypothesis, θ, from which samples are generated, by the distribution $\rho(\mathbf{x}, y|\theta)$. In a sense, this corresponds to the belief that *nature* (i.e. the source of the hypothesis generating the data) is deterministic. In the Bayesian setting, it is assumed that nature randomly selects the data-generating hypothesis according to some prior distribution $\pi(\theta)$. This is typically used as a way to code one's subjective uncertainty and prior knowledge concerning what hypotheses are more or less likely, *a priori*. The existence of this prior allows us to employ Bayes' rule to derive a *posterior distribution*, over hypotheses, conditioned on the observed data, i.e.

$$\pi(\theta|Z) = \frac{\rho_Z(Z|\theta)\pi(\theta)}{\int_\Theta d\theta' \, \rho_Z(Z|\theta')\pi(\theta')}.$$

The Bayesian solution is aimed at minimizing the loss, averaged using the posterior $\pi(\theta|Z)$. Accordingly, we define the *posterior Bayesian risk* as the expected loss of a hypothesis, where the expectation is taken with respect to θ, conditioned on Z.

Definition 1.2.6 (Posterior Bayesian risk). *The posterior Bayesian risk for the estimator $\hat{\theta}_Z$, given the training sample Z, is*

$$\mathcal{R}_{post}(\hat{\theta}_Z, Z) = \int_\Theta d\theta \, \pi(\theta|Z) \, \ell(\theta, \hat{\theta}_Z)$$

The Bayes estimator is $\hat{\theta}_Z$ for which $\mathcal{R}_{post}(\hat{\theta}_Z, Z)$ is minimized. Note that, contrary to the frequentist risk, the posterior risk does not depend on θ, since the averaging over the posterior has removed that dependence. However, $\mathcal{R}_{post}$ does depend on the specific training sample observed. This implies that $\mathcal{R}_{post}$ may be used to derive optimal Bayesian estimators, whose optimality is independent of the unknown hypothesis θ, but is specific to the known training sample Z.

It is a well known (and easily proved) result that, for the squared loss, the Bayes estimator is the posterior mean of θ, conditioned on the sample Z (Scharf, 1991):

$$\hat{\theta}_Z^B = \mathbf{E}_\pi[\theta|Z] = \int_\Theta d\theta' \theta' \pi(\theta'|Z),$$

The choice of hypothesis space and solution quality criterion are probably *the* two central problems of statistical learning theory. We have briefly discussed the latter, in both the frequentist setting and the Bayesian setting. Let us now return to the former.

1.2.3 Function Spaces, Reproducing Kernels and Regularization

Up to this point we allowed the space of hypotheses Θ to be general. Now we will focus our attention on Θ, which is a function space.

One way to define a space of functions is the parametric way. As an example, let us assume that the function $\hat{\theta}$ we are looking for is located in the span of a set of n basis functions $\{\phi_1(\mathbf{x}), \ldots, \phi_n(\mathbf{x})\}$. These functions, evaluated at a given $\mathbf{x}$, may be aggregated into a vector $\boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_n(\mathbf{x}))^\top$. We refer to $\boldsymbol{\phi}(\mathbf{x})$ as a *feature vector*. Learning is now reduced to finding a vector of coefficients $\mathbf{w}$ such that, the estimator $\hat{\theta}(\mathbf{x}) = \sum_{i=1}^n w_i \phi_i(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{w}$, minimizes one of the types of risk defined in the preceding section. This brings us back to the parametric decision theoretical setting.

An alternative approach is to define a space of functions nonparametrically, for instance, as the space of all functions possessing certain smoothness properties. How can such smoothness properties be enforced? A well established method involves endowing some additional structure to the function space Θ, namely, making it a *reproducing kernel Hilbert space* (RKHS).

Definition 1.2.7 (Hilbert space). *A Hilbert space is an inner product vector space, which is complete under the induced metric.*

We will be dealing almost exclusively with Hilbert spaces of real-valued functions. The completeness condition is of little practical interest for us, whereas the existence of an inner product $\langle \cdot, \cdot \rangle$ is crucial, as it induces a norm $\|\theta\| = \sqrt{\langle \theta, \theta \rangle}$. This makes Hilbert spaces a specialization of normed spaces (Banach spaces). An important property of Hilbert spaces is that they are spanned by a countable basis.

Definition 1.2.8 (Reproducing kernel Hilbert space). *A RKHS is a Hilbert space of real-valued functions in which the evaluation functional is bounded and linear.*

The evaluation functional for some point $\mathbf{x} \in \mathcal{X}$ is a functional that, when applied to a function θ, returns its value at the point $\mathbf{x}$, i.e. $\theta(\mathbf{x})$. By the Riesz representation theorem the evaluation functional in a Hilbert space Θ is of the form $\langle k_{\mathbf{x}}, \cdot \rangle$, with $k_{\mathbf{x}}$ being a member of Θ. By definition, the evaluation functional satisfies for any $\mathbf{x} \in \mathcal{X}$, $\langle k_{\mathbf{x}}, \theta \rangle = \theta(\mathbf{x})$. In the case of RKHS the kernel k of the evaluation functional is a *positive-definite, symmetric kernel.* Namely, $k_{\mathbf{x}} = k(\mathbf{x}, \cdot) = k(\cdot, \mathbf{x}) \in \Theta$, and for every $\theta, \theta' \in \Theta$ it satisfies

$$\int_{\mathcal{X}^2} d\mathbf{x} d\mathbf{x}' \theta(\mathbf{x}) k(\mathbf{x}, \mathbf{x}') \theta(\mathbf{x}') \geq 0.$$

Moreover, the set of functions $k(\mathbf{x}, \cdot)$, for all $\mathbf{x} \in \mathcal{X}$, span Θ. A choice of a kernel function k uniquely defines a RKHS, and conversely (Cucker & Smale, 2001).

The reason we wanted to Θ to be a RKHS was because we needed a disciplined way to penalize the complexity of learned solutions. It turns out that by minimizing the empirical risk (Definition 1.2.5) while also paying a penalty depending on the RKHS norm of candidate hypotheses, the learning problem becomes well-posed, even in the nonparametric domain. An additional, quite useful result known as the *Representer theorem* tells us that, in the supervised setting (in which $\hat{y}_Z \equiv \hat{\theta}_Z$), we only need to search for solutions of a rather restricted form. In one of its formulations, this theorem reads as follows (see Kimeldorf & Wahba, 1971 for the basic result, with extensions and related results in Cox & O'Sullivan, 1990; Schölkopf & Smola, 2002; Csató & Opper, 2002).

Theorem 1.2.1 (Representer). *Denote by $\Omega : \mathbb{R}^+ \to \mathbb{R}$ a strictly monotonically increasing function. Then, for any instantaneous loss function ℓ_{ins} and a sample $Z = \{\mathbf{x}_i, y_i\}_{i=1}^t$, the solution of*

$$\min_{\hat{\theta}_Z \in \Theta} \left\{ \sum_{i=1}^t \ell_{ins}(\mathbf{x}_i, y_i, \hat{y}_Z(\mathbf{x}_i)) + \Omega\left(\|\hat{\theta}_Z\| \right) \right\},$$

is of the form

$$\hat{\theta}_Z(\mathbf{x}) = \sum_{i=1}^t \alpha_i k(\mathbf{x}_i, \mathbf{x}), \tag{1.2.2}$$

where $\alpha_i \in \mathbb{R}$ for all $i = 1, \ldots, t$.

Recall that $\hat{y}_Z(\mathbf{x})$ is the predicted observation provided by the hypothesis $\hat{\theta}_Z$ for the point $\mathbf{x}$. The Representer theorem implies that, although our search is conducted in an essentially infinite dimensional (function) space, we need only consider a finite t-dimensional subspace, which is spanned by the functions $\{k(\mathbf{x}_i, \cdot)\}_{i=1}^t$. Learning is thus reduced to finding the optimal vector $\boldsymbol{\alpha}_t = (\alpha_1, \ldots, \alpha_t)^\top$.

1.2.4 From Features to Kernels

Another key result with far reaching consequences concerning kernel algorithms is Mercer's theorem (Mercer, 1909). Mercer's theorem provides the theoretical backing for the commonly used "kernel trick" (see below), by implying that for any positive-definite kernel k there exists a mapping $\mathbf{x} \mapsto \boldsymbol{\phi}(\mathbf{x})$ such that $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{x}')$.

Theorem 1.2.2 (Mercer). *Let $k(\cdot, \cdot)$ be a positive-definite, symmetrical, continuous and bounded kernel function. Then, the (positive-definite) integral operator $\mathcal{T}_k : \Theta \to \Theta$ defined by*

$$\mathcal{T}_k \theta(\mathbf{x}) = \int_{\mathcal{X}} k(\mathbf{x}, \mathbf{x}') \theta(\mathbf{x}') \rho_{\mathbf{x}}(\mathbf{x}') d\mathbf{x}', \tag{1.2.3}$$

where $\rho_{\mathbf{x}}(\cdot)$ is the marginal distribution of $\mathbf{x}$, has a countable set of continuous eigenfunctions $\{\psi_i\}_{i=1}^\infty$ with their respective positive *eigenvalues $\{\lambda_i\}_{i=1}^\infty$, such that for almost all $\mathbf{x}, \mathbf{x}'$,*

$$k(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^\infty \lambda_i \psi_i(\mathbf{x}) \psi_i(\mathbf{x}')$$

If Θ is finite dimensional, the sum above should be replaced by $\sum_{i=1}^{|\Theta|}$. Mercer's theorem implies that Θ is spanned by the orthogonal set of eigenvectors[5] $\{\psi_i\}$. Also note that, due to the positivity of k, all of the eigenvalues $\{\lambda_i\}$ are positive. In the sequel we will assume that the eigenvalues are ordered so that $\lambda_1 \geq \lambda_2 \geq \ldots$ etc. The Spectral theorem (e.g., Cucker & Smale, 2001 Chapter 2, Theorem 2) further states, that the set $\{\lambda_i\}$ is either finite (in which case Θ is finite dimensional), or $\lambda_i \to 0$, as $i \to \infty$. An immediate consequence of Mercer's theorem is that, by defining $\phi_i = \sqrt{\lambda_i} \psi_i$, we obtain the result mentioned above, namely,

$$k(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^\infty \phi_i(\mathbf{x}) \phi_i(\mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{x}').$$

A useful corollary of this theorem, known as the *kernel trick* is the following:

Corollary 1.2.3 (Kernel Trick). *Any algorithm, which may be stated using only inner products between members of the input space, can be immediately replaced*

[5]If all eigenvalues are different, the eigenvectors of $\mathcal{T}_k$ serve as an orthogonal basis. This is due to the self-adjointness of the operator, which is due to the symmetry of its kernel. Degeneracy in the eigenvalues may be treated in the usual way, by explicitly constructing an orthogonal basis in the degenerate subspace.

with a new (kernel) algorithm, in which the inner products are replaced with kernel evaluations.

An algorithm transformed by application of the kernel trick into a nonlinear kernel-based version, may therefore be understood as a linear algorithm in the feature space. For instance, using Mercer's theorem, the kernel expression $\sum_{i=1}^{t} \alpha_i k(\mathbf{x}_i, \mathbf{x})$, resulting from the Representer theorem, may be written as $\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x})$, where

$$\mathbf{w} = \sum_{i=1}^{\top} \alpha_i \boldsymbol{\phi}(\mathbf{x}_i). \tag{1.2.4}$$

1.2.5 Sparsity in Kernel Methods

The representer theorem and the kernel trick provide us with the general forms of kernel-based solutions. In particular, we found that the number of learnable parameters in a kernel machine generally equals t, the size of the training data set. While this is most definitely an improvement over the typical dimensionality of the hypothesis space (i.e. infinity), for some applications it is nevertheless still too much, hence the need for *sparse* solutions. By seeking sparsity we mean that we would like a large fraction of the coefficients $\{\alpha_i\}_{i=1}^{t}$ to vanish. This has several beneficial effects. If a large number of vanishing coefficients may be detected early in the operation of the learning algorithm, their removal may allow for considerable computational savings. Even if vanishing coefficients are removed only once learning is complete, this still results in reduced computational requirements at query time, i.e. when the estimate $\hat{\theta}_Z(\mathbf{x})$ has to be computed. Finally, there are several theoretical results on compression bounds, indicating that sparsity may be used as a tool for controlling the capacity of a learning machine, much in the same way that the RKHS norm is used in Tikhonov-style regularization. Thus sparsity may also be beneficial for the generalization capability of a kernel machine (see Herbrich, 2002 and references therein, and Meir & Zhang, 2003).

One paradigmatic approach, used by Vapnik in his support vector regression (SVR) machine (Vapnik et al., 1997), is to encourage sparsity by employing an error tolerant loss function, together with a RKHS squared-norm regularization term. In algorithms for solving SVR, and in related regularization-networks algorithms (Evgeniou et al., 2000), sparsity is achieved by *elimination.* This means that, at the outset, these algorithms consider all training samples as potential contributing members of the kernel expansion, and once the optimization problem is solved they eliminate those samples, the coefficients of which vanish. An alternative approach is to obtain sparsity by *construction.* Here the algorithm starts with an empty representation, in which all coefficients vanish, and gradually adds samples according to some criterion. Constructive sparsification is normally used off-line (e.g., Vincent &

Bengio, 2002), in which case the algorithm is free to choose any one of the training samples at each step of the construction process. Due to the intractability of finding the best subset of samples (Natarajan, 1995), these algorithms usually resort to employing various *greedy* selection strategies, in which at each step the sample selected is the one that maximizes the amount of increase (or decrease) its addition induces in some empirical fitness (or error) criterion.

Another possible approach is to attain sparsity by projection. Roughly speaking, this is done by projecting the full solution onto a lower dimensional manifold, spanned by a subset of the kernel functions employed in the original solution. This reduction in dimensionality results in an approximation of the original solution, which may, or may not, be accompanied by some degradation in quality. It should be noted that this approach does not preclude the use of other approaches, as it may be employed as a post-processing stage performed on the solutions delivered by any kernel method.

Why should sparsification be at all possible? In order to answer this question we need to make some additional definitions.

Definition 1.2.9 (ϵ-cover). *Let $\mathcal{F}$ be a metric space (i.e. a vector space endowed with a metric $\|\cdot\|$, a Hilbert space being a special case). An ϵ-cover, $\mathcal{C}_\epsilon(\mathcal{F})$, is a set of points $\{f_i\}$, such that every member of $\mathcal{F}$ is in a ball of radius ϵ centered in one of the members of $\mathcal{C}_\epsilon(\mathcal{F})$. In other words, for all $f \in \mathcal{F}$ there exists some $f_i \in \mathcal{C}_\epsilon(\mathcal{F})$ such that $\|f - f_i\| \leq \epsilon$.*

Definition 1.2.10 (Covering numbers). *Let $\mathcal{F}$ be a metric space. The ϵ-covering number $\mathcal{N}^c_\epsilon(\mathcal{F})$ is the size of the smallest ϵ-cover for $\mathcal{F}$.*

Definition 1.2.11 (Packing numbers). *Let $\mathcal{F}$ be a metric space. The ϵ-packing number $\mathcal{N}^p_\epsilon(\mathcal{F})$ is the size of the largest subset of $\mathcal{F}$ the members of which are separated by a distance strictly larger than ε.*

It is a well known (and easy to prove) result that, when they exist (Anthony & Bartlett, 1999),

$$\mathcal{N}^p_{2\epsilon}(\mathcal{F}) \leq \mathcal{N}^c_\epsilon(\mathcal{F}) \leq \mathcal{N}^p_\epsilon(\mathcal{F}). \tag{1.2.5}$$

Let us return to our original question and formulate it in slightly more precise terms: How well can the complete (non-sparse) expression $f(\mathbf{x}) = \sum_{i=1}^t \alpha_i k(\mathbf{x}_i, \mathbf{x})$, be approximated by an expression $\hat{f}(\mathbf{x}) = \sum_{i=1}^t \hat{\alpha}_i k(\mathbf{x}_i, \mathbf{x})$, in which all coefficients $\{\hat{\alpha}_i\}$, except for a subset of size m, vanish? Unfortunately, it has been shown that this problem is NP-hard, due to the need to check all possible subsets (Natarajan, 1995). Nevertheless, computational considerations notwithstanding, let us proceed.

In the previous section we used Mercer's theorem to conclude that Θ is spanned by the basis functions $\{\phi_i\}$. Therefore, any hypothesis $\theta \in \Theta$, may be written as $\theta(\mathbf{x}) = \sum_i w_i \phi_i(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x})$. Moreover, the Representer theorem showed us that $\mathbf{w}$ is of the form $\mathbf{w} = \sum_{i=1}^t \alpha_i \boldsymbol{\phi}(\mathbf{x}_i)$. Measuring the approximation quality using the squared RKHS norm induced by k, it is easy to show that

$$\|\hat{f} - f\|_k^2 = \|\hat{\mathbf{w}} - \mathbf{w}\|^2 = (\hat{\mathbf{w}} - \mathbf{w})^\top (\hat{\mathbf{w}} - \mathbf{w}).$$

Thus the problem is transformed into that of finding a sparse approximation to $\mathbf{w}$. From Eq. 1.2.4 we see that $\mathbf{w}$ lies in $\text{span}\{\boldsymbol{\phi}(\mathbf{x}_i)\}_{i=1}^t$, which itself is a subspace of the *feature space*, defined by (slightly abusing notation)

$$\boldsymbol{\phi}(\mathcal{X}) \overset{\text{def}}{=} \text{span}\{\boldsymbol{\phi}(\mathbf{x}) : \ \mathbf{x} \in \mathcal{X}\}. \tag{1.2.6}$$

Now suppose that the ϵ-packing number for $\boldsymbol{\phi}(\mathcal{X})$ satisfies

$$\mathcal{N}_\epsilon^p(\boldsymbol{\phi}(\mathcal{X})) < t.$$

Then, from Eq. 1.2.5 it follows that there exists an ϵ-cover for $\boldsymbol{\phi}(\mathcal{X})$ the size of which is $\mathcal{N}_\epsilon^c(\boldsymbol{\phi}(\mathcal{X})) < t$. This, in turn, means that at least $t - \mathcal{N}_\epsilon^c(\boldsymbol{\phi}(\mathcal{X}))$ members of the set $\{\boldsymbol{\phi}(\mathbf{x}_i)\}_{i=1}^t$ are at a distance of ϵ or less to some other member of that set. Hence, at least such a number of coefficients, in Eq. 1.2.4, may be made to vanish, incurring an approximation error that is, at most, linear in ϵ.

In the worst case, the covering numbers $\mathcal{N}_\epsilon^c(\boldsymbol{\phi}(\mathcal{X}))$, display an exponential dependence on the dimensionality of $\boldsymbol{\phi}(\mathcal{X})$ (i.e. $O\left((1/\epsilon)^d\right)$). However, both the choice of kernel and the density $\rho_{\mathbf{x}}$ according to which data is sampled, often tend to result in a low volume feature space $\boldsymbol{\phi}(\mathcal{X})$. This is manifest in the decay rate of the eigenvalues of the operator $\mathcal{T}_k$, which are proportional to the side lengths of a box containing the data in $\boldsymbol{\phi}(\mathcal{X})$. The faster the eigenvalues decay, the slower is the rise of the covering numbers, and consequently the better prospects we have for quickly reaching a sample size t in which we will be able to obtain sparse solutions (see König, 1986 for details on eigenvalue decay rates).

1.2.6 Online Learning with Kernels

Kernel methods are not a natural choice for the practitioner seeking an online algorithm. In a nutshell, the major obstacles in applying kernel methods to online learning are:

1. Many kernel methods require random/multiple access to training samples,

2. their computational cost (both in time and space) is typically super-linear in

the size of the training set, and

3. their prediction (query) time often scales linearly with the training set size.

In the online learning scenario input samples are observed sequentially, one at a time. Such scenarios abound in diverse fields such as data mining, time series prediction, signal processing and reinforcement learning, to mention a few. In such cases there is a clear advantage to algorithms that do not need to relearn from scratch when new data arrive. In many of these applications there is an additional requirement for *real-time* operation, meaning that the algorithm's computational expenses per time-step should be bounded by a constant independent of the number of previously observed samples, for it is assumed that new samples arrive at a roughly constant rate. As may be inferred from the discussion above, kernel algorithms typically need to maintain a set of parameters the size of which equals the size of the training data set, which immediately precludes them from meeting this real-time criterion.

Let us suppose, however, that by using some sparsification mechanism, we are able to bound from above the number of parameters that are really required for solving the given problem. In this case, it should be possible to design kernel algorithms that meet the real-time criterion. The first part of the thesis (Chapters 2 and 3), is concerned with the development and application of such sparse, online algorithms in the supervised setting of function approximation, or regression. The kernel-RLS algorithm presented in Chapter 3 is based on a classical, well established frequentist learning paradigm. In the next section we consider an alternative, Bayesian learning paradigm – Gaussian processes – with which we will pursue, in the second part of the thesis, our ultimate goal: Solving reinforcement learning problems.

1.3 Gaussian Processes

Gaussian Processes (GPs) have been used extensively in recent years in supervised learning tasks such as classification and regression (e.g., Gibbs & MacKay, 1997; Williams, 1999). Based on probabilistic generative models, GP methods are theoretically attractive since they allow a Bayesian treatment of these problems, yielding full posterior distributions based both on one's prior beliefs and on the data observed, rather than the point-estimates usually provided by other methods. Since GPs may be defined directly in function space, they are not as restrictive as parametric models in terms of the hypothesis space in which learning takes place. Moreover, when both the prior distribution and the likelihood are Gaussian, the posterior distribution, conditioned on the observations, is also Gaussian and Bayes' rule yields closed-form expressions for the posterior moments, thus completely avoiding the difficulties associated with iterative optimization algorithms and their convergence behavior.

1.3.1 Definitions and Notation

A random process F is a (finite, countably, or uncountably infinite) set of random variables, each of which is assigned an index. F is said to be a Gaussian process if the variables belonging to any finite subset of F are jointly Gaussian. Here we will focus on GPs indexed by an *input, or state variable* $\mathbf{x}$. To keep the discussion as general as possible we only need to assume that $\mathbf{x}$ is a member of some set $\mathcal{X}$. We therefore allow $\mathcal{X}$ to be either finite, countably infinite, or even uncountably infinite, somewhat stretching the notion of index. F may be thought of as a random vector if $\mathcal{X}$ is finite, as a random series if $\mathcal{X}$ is countably infinite, and as a *random function* if $\mathcal{X}$ is uncountably infinite. In the latter case, each instantiation of F is a function $f : \mathcal{X} \to \mathbb{R}$. On the other hand, for a given $\mathbf{x}$, $F(\mathbf{x})$ is a random variable (RV), normally distributed jointly with the other components of F.

In order to perform Bayesian inference using GPs we need to define a *statistical generative model*, typically consisting of the following ingredients:

1. A *model-equation* relating the observed and unobserved random processes in our model, in which the latter is usually transformed and corrupted by some additive measurement noise to produce the former. The unobserved process is the subject of our Bayesian inference effort.

2. A distribution of the measurement noise terms. By noise, we refer to any additive random process in the model equation, the statistics of which is known (at least up to a few undetermined hyperparameters), and which is *not* the subject of our inference problem.

3. A prior distribution of the unobserved process. This is a necessary ingredient required for employing Bayes' rule.

Since F is a priori Gaussian, its prior distribution is fully specified by its mean and covariance,

$$\begin{aligned}\mathbf{E}_0\left[F(\mathbf{x})\right] &\overset{\text{def}}{=} f_0(\mathbf{x}),\\ \mathbf{Cov}_0\left[F(\mathbf{x}), F(\mathbf{x}')\right] &= \mathbf{E}_0\left[F(\mathbf{x})F(\mathbf{x}')\right] - f_0(\mathbf{x})f_0(\mathbf{x}') \overset{\text{def}}{=} k(\mathbf{x}, \mathbf{x}'),\end{aligned} \tag{1.3.7}$$

respectively, where $\mathbf{E}_0(\cdot)$ denotes the expectation with respect to the prior distribution. In order for $k(\cdot,\cdot)$ to be a legitimate covariance it is required to be symmetric and positive-definite. Symmetry means that $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$, for all $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$. For a finite $\mathcal{X}$, $k(\cdot,\cdot)$ is a matrix, and the second requirement translates to positive-definiteness of this matrix. When $\mathcal{X}$ is continuous, positive-definiteness is defined by the integral condition $\int_{\mathcal{X}^2} g(\mathbf{x})k(\mathbf{x}, \mathbf{x}')g(\mathbf{x}')d\mathbf{x}d\mathbf{x}' \geq 0$, $\forall g \in \ell_2$. Interestingly, these are exactly the requirements made of Mercer kernels, discussed above. Recall that, in kernel methods, $k(\cdot, \cdot)$ is usually referred to as the kernel function, and is viewed as an inner product in some high dimensional feature space. Under certain conditions, these two views are in fact equivalent, which is the reason we use the same notation for both functions (see Schölkopf & Smola, 2002 for details).

1.3.2 The Linear Statistical Model

Consider the following generative model:

$$Y = \mathbf{H}F + N, \tag{1.3.8}$$

where $\mathbf{H}$ is a general linear transformation. The case where both F and N are Gaussian and independent of each other, is known as the *linear statistical model* (Scharf, 1991). In the generative, forward view of this model, an observation Y is obtained by the following steps:

1. sample F from its prior distribution,
2. apply the transformation $\mathbf{H}$ to F,
3. sample N from the noise distribution,
4. compute Y using Eq. 1.3.8.

In the learning scenario, we are provided with a sequence of measurements $\{(\mathbf{x}_i, y_i)\}_{i=1}^t$, sampled using this generative model by observing it only at the indices $\mathbf{x}_1, \ldots, \mathbf{x}_t$. The resulting set of t equations, where t is the number of measurements performed, may be written as

$$Y_t = \mathbf{H}_t F + N_t, \tag{1.3.9}$$

where

$$Y_t = (Y(\mathbf{x}_1), \ldots, Y(\mathbf{x}_t))^\top,$$
$$N_t = (N(\mathbf{x}_1), \ldots, N(\mathbf{x}_t))^\top.$$

In the case where F is finite-dimensional, both $\mathbf{H}$ and $\mathbf{H}_t$ are matrices. In the continuous case, $\mathbf{H}$ is a linear integral operator, and $\mathbf{H}_t F$ is generally a set of t integrals, the i'th of which is of the form $\int_{\mathcal{X}} d\mathbf{x}' h_t(\mathbf{x}_i, \mathbf{x}')F(\mathbf{x}')$. *Sampling operators* are a special family of operators for which the i'th row of $\mathbf{H}_t F$ satisfies $(\mathbf{H}_t F)_i = \sum_{j=1}^t h_{i,j} F(\mathbf{x}_j)$. This means that, slightly abusing notation, we may write

$$Y_t = \mathbf{H}_t F_t + N_t, \tag{1.3.10}$$

where $\mathbf{H}_t$ is now a $t \times t$ matrix, $F_t = (F(\mathbf{x}_1), \ldots, F(\mathbf{x}_t))^\top$, and $N_t \sim \mathcal{N}\{\mathbf{0}, \mathbf{\Sigma}_t\}$. We will restrict our attention to generative models in which $\mathbf{H}$ is a sampling operator, and which therefore result in equations of the form (1.3.10).

Suppose now that we obtain a sequence of measurements $\{(\mathbf{x}_i, y_i)\}_{i=1}^t$, and we wish to apply Bayes' rule to compute the *posterior distribution* of F, conditioned on $Y(\mathbf{x}_i) = y_i$ for $i = 1, \ldots, t$. Fortunately, Gauss and later Markov have provided us with the answer (Scharf, 1991).

Theorem 1.3.1 (Gauss-Markov). *Let $\mathcal{N}$ denote the normal distribution, and let X and Y be random vectors distributed as*

$$\begin{pmatrix} X \\ Y \end{pmatrix} \sim \mathcal{N}\left\{ \begin{pmatrix} \mathbf{m}_x \\ \mathbf{m}_y \end{pmatrix}, \begin{bmatrix} \mathbf{K}_{xx} & \mathbf{K}_{xy} \\ \mathbf{K}_{yx} & \mathbf{K}_{yy} \end{bmatrix} \right\}$$

Then $X|Y \sim \mathcal{N}\left\{\hat{X}, \mathbf{P}\right\}$, where

$$\begin{aligned} \hat{X} &= \mathbf{m}_x + \mathbf{K}_{xy}\mathbf{K}_{yy}^{-1}(Y - \mathbf{m}_y) \\ \mathbf{P} &= \mathbf{K}_{xx} - \mathbf{K}_{xy}\mathbf{K}_{yy}^{-1}\mathbf{K}_{yx}. \end{aligned}$$

$\mathbf{P}$ is also known as the *Schur complement* of $\mathbf{K}_{xx}$ in the partitioned matrix $\begin{bmatrix} \mathbf{K}_{xx} & \mathbf{K}_{xy} \\ \mathbf{K}_{yx} & \mathbf{K}_{yy} \end{bmatrix}$ (Scharf, 1991).

Let us define the vector $\mathbf{f}_0 = (f_0(\mathbf{x}_1), \ldots, f_0(\mathbf{x}_t))^\top$, and the matrix $[\mathbf{K}_t]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$, for $i, j \in \{1, \ldots, t\}$. In our linear statistical model, we then have $F_t \sim \mathcal{N}\{\mathbf{f}_0, \mathbf{K}_t\}$, and from Eq. 1.3.10 we deduce that (recall that $\mathbf{\Sigma}_t = \mathbf{Cov}[N_t]$)

$$Y_t \sim \mathcal{N}\left\{\mathbf{H}_t\mathbf{f}_0,\, \mathbf{H}_t\mathbf{K}_t\mathbf{H}_t^\top + \mathbf{\Sigma}_t\right\}.$$

Consider a query point $\mathbf{x}$; we have

$$\begin{pmatrix} F(\mathbf{x}) \\ F_t \\ N_t \end{pmatrix} \sim \mathcal{N}\left\{ \begin{pmatrix} f_0(\mathbf{x}) \\ \mathbf{f}_0 \\ \mathbf{0} \end{pmatrix}, \begin{bmatrix} k(\mathbf{x},\mathbf{x}) & \mathbf{k}_t(\mathbf{x})^\top & \mathbf{0} \\ \mathbf{k}_t(\mathbf{x}) & \mathbf{K}_t & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{\Sigma}_t \end{bmatrix} \right\}, \tag{1.3.11}$$

where $\mathbf{k}_t(\mathbf{x}) = (k(\mathbf{x}_1, \mathbf{x}), \ldots, k(\mathbf{x}_t, \mathbf{x}))^\top$. Using Eq. 1.3.10 we make the transformation

$$\begin{pmatrix} F(\mathbf{x}) \\ F_t \\ Y_t \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \mathbf{I} & 0 \\ 0 & \mathbf{H}_t & \mathbf{I} \end{bmatrix} \begin{pmatrix} F(\mathbf{x}) \\ F_t \\ N_t \end{pmatrix} \tag{1.3.12}$$

Resulting in

$$\begin{pmatrix} F(\mathbf{x}) \\ F_t \\ Y_t \end{pmatrix} \sim \mathcal{N}\left\{ \begin{pmatrix} f_0(\mathbf{x}) \\ \mathbf{f}_0 \\ \mathbf{H}_t\mathbf{f}_0 \end{pmatrix}, \begin{bmatrix} k(\mathbf{x},\mathbf{x}) & \mathbf{k}_t(\mathbf{x})^\top & \mathbf{k}_t(\mathbf{x})^\top\mathbf{H}_t^\top \\ \mathbf{k}_t(\mathbf{x}) & \mathbf{K}_t & \mathbf{K}_t\mathbf{H}_t^\top \\ \mathbf{H}_t\mathbf{k}_t(\mathbf{x}) & \mathbf{H}_t\mathbf{K}_t & \mathbf{H}_t\mathbf{K}_t\mathbf{H}_t^\top + \mathbf{\Sigma}_t \end{bmatrix} \right\}, \tag{1.3.13}$$

The Gauss-Markov Theorem (1.3.1) may now be used to obtain the expressions for the posterior mean and covariance of $F(\mathbf{x})$, conditioned on Y_t:

$$(F(\mathbf{x})|Y_t) \sim \mathcal{N}\left\{\hat{F}_t(\mathbf{x}), P_t(\mathbf{x},\mathbf{x})\right\}, \quad \text{where}$$

$$\hat{F}_t(\mathbf{x}) = f_0(\mathbf{x}) + \mathbf{k}_t(\mathbf{x})^\top\mathbf{H}_t^\top\left(\mathbf{H}_t\mathbf{K}_t\mathbf{H}_t^\top + \mathbf{\Sigma}_t\right)^{-1}(Y_t - \mathbf{H}_t\mathbf{f}_0), \tag{1.3.14}$$

$$P_t(\mathbf{x},\mathbf{x}') = k(\mathbf{x},\mathbf{x}') - \mathbf{k}_t(\mathbf{x})^\top\mathbf{H}_t^\top\left(\mathbf{H}_t\mathbf{K}_t\mathbf{H}_t^\top + \mathbf{\Sigma}_t\right)^{-1}\mathbf{H}_t\mathbf{k}_t(\mathbf{x}'). \tag{1.3.15}$$

Note that the posterior mean, $\hat{F}_t(\mathbf{x})$, is linear in the measurements Y_t, while the posterior covariance, $P_t(\mathbf{x},\mathbf{x}')$, is altogether independent of Y_t. Also note that it is possible to decompose these expressions into input dependent terms (which depend on $\mathbf{x}$ and $\mathbf{x}'$), and terms that only depend on the training sample. Specifically,

$$\hat{F}_t(\mathbf{x}) = f_0(\mathbf{x}) + \mathbf{k}_t(\mathbf{x})^\top\boldsymbol{\alpha}_t, \tag{1.3.16}$$

$$P_t(\mathbf{x},\mathbf{x}') = k(\mathbf{x},\mathbf{x}') - \mathbf{k}_t(\mathbf{x})^\top\mathbf{C}_t\mathbf{k}_t(\mathbf{x}'). \tag{1.3.17}$$

where

$$\boldsymbol{\alpha}_t = \mathbf{H}_t^\top\left(\mathbf{H}_t\mathbf{K}_t\mathbf{H}_t^\top + \mathbf{\Sigma}_t\right)^{-1}(Y_t - \mathbf{H}_t\mathbf{f}_0),$$

$$\mathbf{C}_t = \mathbf{H}_t^\top\left(\mathbf{H}_t\mathbf{K}_t\mathbf{H}_t^\top + \mathbf{\Sigma}_t\right)^{-1}\mathbf{H}_t. \tag{1.3.18}$$

Eq. 1.3.16 and 1.3.17 lead us to conclude that $\boldsymbol{\alpha}_t$ and $\mathbf{C}_t$ are *sufficient statistics* for the posterior moments. Note, that this result may be thought of as the counterpart of the Representer theorem (Theorem 1.2.1) for Gaussian processes[6].

The posterior mean (Eq. 1.3.16) is our Bayesian estimator for F, while the posterior covariance provides an estimate of the accuracy of the predictions provided by the posterior mean. Assuming all our model assumptions are correct, the posterior mean possesses several attractive properties (Scharf, 1991):

1. It minimizes the posterior risk (Definition 1.2.6) for the squared loss.
2. It equals the *maximum a posteriori* (MAP) estimate, under the same assumptions.
3. It is the (unique) *minimum variance unbiased* (MVUB) estimator for F.

[6] A more general result is provided by the Parameterization lemma of Csató & Opper, 2002.

4. Even if the Gaussianity assumptions are dropped (but maintaining moment assumptions), the resulting estimator, although no longer interpretable as a posterior mean, is the *linear minimum mean squared error* (LMMSE) estimator.

These properties apply to the family of linear statistical models, as well as to a generalization thereof, known as the Kalman filter (Kalman, 1960; Scharf, 1991).

1.3.3 Gaussian Process Regression

As an illustrative example let us review the use of GPs for regression with white Gaussian noise. In this setup, we are provided with a sample of t training examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^t$. The model-equation for some $\mathbf{x} \in \mathcal{X}$ is

$$Y(\mathbf{x}) = F(\mathbf{x}) + N(\mathbf{x}), \tag{1.3.19}$$

where F is the GP corresponding to the unknown function from which the data are generated, N is a white noise GP, and Y is the observable process, modeled here as a noisy version of F. In other words, the operator $\mathbf{H}$ is here the identity operator, which qualifies it as a sampling operator. Eq. 1.3.19, evaluated for the training samples, may be written concisely as

$$Y_t = F_t + N_t, \tag{1.3.20}$$

In our example, we assume that the noise terms corrupting each sample are independently and identically distributed (IID), we therefore have

$$N_t \sim \mathcal{N}\left(\mathbf{0}, \sigma^2\mathbf{I}\right),$$

where σ^2 is the variance of each noise term. The last remaining item on our list to be specified is the prior over F, which we assume to be Gaussian with zero mean and covariance given by Eq. 1.3.7. The kernel function $k(\cdot,\cdot)$ encodes our prior knowledge concerning the correlations between the components of F at different points. Equivalently, k may be thought of as inducing a measure of proximity between the members of $\mathcal{X}$. It can also be shown that k defines the function space within which the search for the solution takes place (see Schölkopf & Smola, 2002 for more details). While the choice of prior mean is relatively innocuous, in the sense that the asymptotic solution is independent of it, the last point indicates that the choice of covariance kernel has far reaching consequences, which cannot be completely undone, even by an infinite amount of data[7]. Whichever way one wishes

[7]This is in contrast to priors over finite dimensional random processes, the influence of which vanishes asymptotically.

to view it, the choice of kernel codes one's prior knowledge and beliefs concerning the problem at hand, and the expected form of its solution.

Fig. 1.1 illustrates the GP regression setting as a graphical model, in which arrows (and lack thereof) mark the conditional independency relations between the nodes corresponding to the latent $F(\mathbf{x}_i)$ and the observed $Y(\mathbf{x}_i)$ variables.

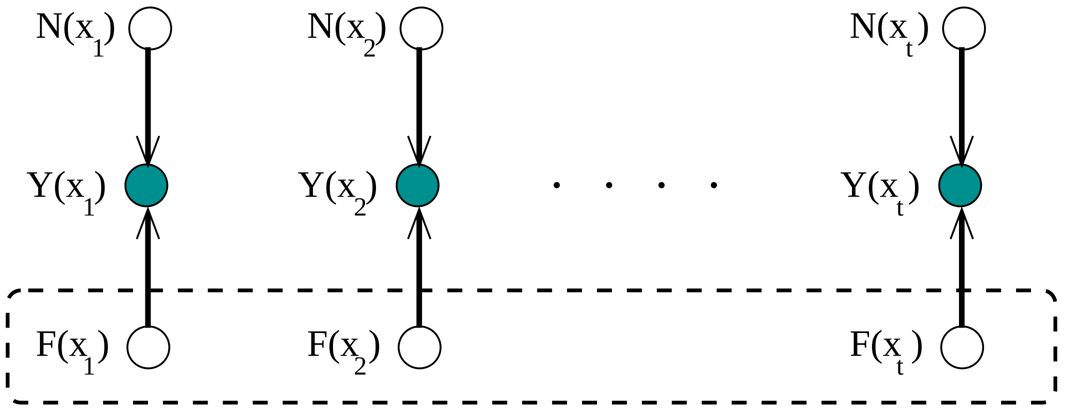


Figure 1.1: A directed graph illustrating the conditional independencies between the latent $F(\mathbf{x}_i)$ variables (bottom row), the noise variables $N(\mathbf{x}_i)$ (top row), and the observable $Y(\mathbf{x}_i)$ variables (middle row), in GP regression. All of the $F(\mathbf{x}_i)$ variables should be interconnected by arrows (forming a clique), due to the dependencies introduced by the prior. To avoid cluttering the diagram, this was marked by the dashed frame surrounding them.

Given the observed set of t training examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^t$, the posterior distribution of F, conditioned on this sample is given by Eq. 1.3.15, with $\mathbf{H}_t = \mathbf{I}$, $\mathbf{\Sigma}_t = \sigma^2\mathbf{I}$, and $\mathbf{f}_0 = \mathbf{0}$:

$$\mathbf{E}\left(F(\mathbf{x})|Y_t = \mathbf{y}_t\right) = f_0(\mathbf{x}) + \mathbf{k}_t(\mathbf{x})^\top(\mathbf{K}_t + \sigma^2\mathbf{I})^{-1}\mathbf{y}_t, \tag{1.3.21}$$

$$\mathbf{Cov}\left(F(\mathbf{x}), F(\mathbf{x}')|Y_t = \mathbf{y}_t\right) = k(\mathbf{x}, \mathbf{x}') - \mathbf{k}_t(\mathbf{x})^\top(\mathbf{K}_t + \sigma^2\mathbf{I})^{-1}\mathbf{k}_t(\mathbf{x}'), \tag{1.3.22}$$

where $\mathbf{y}_t = (y_1, \ldots, y_t)^\top$.

Figure 1.2 depicts the result of performing GP regression on a set of 10 samples, sampled from the *sinc* function, with added noise. The solid (black) line is the target function, the diamonds mark the training samples, and the (blue) dashed line is the posterior mean – the Bayesian estimator for F. The two (red) dotted lines mark one posterior standard deviation above and below the posterior mean. The posterior variance allows us to assign varying levels of confidence to predictions at different parts of the input space. In this example, almost everywhere, the target function is within the one standard deviation confidence interval from its estimate.

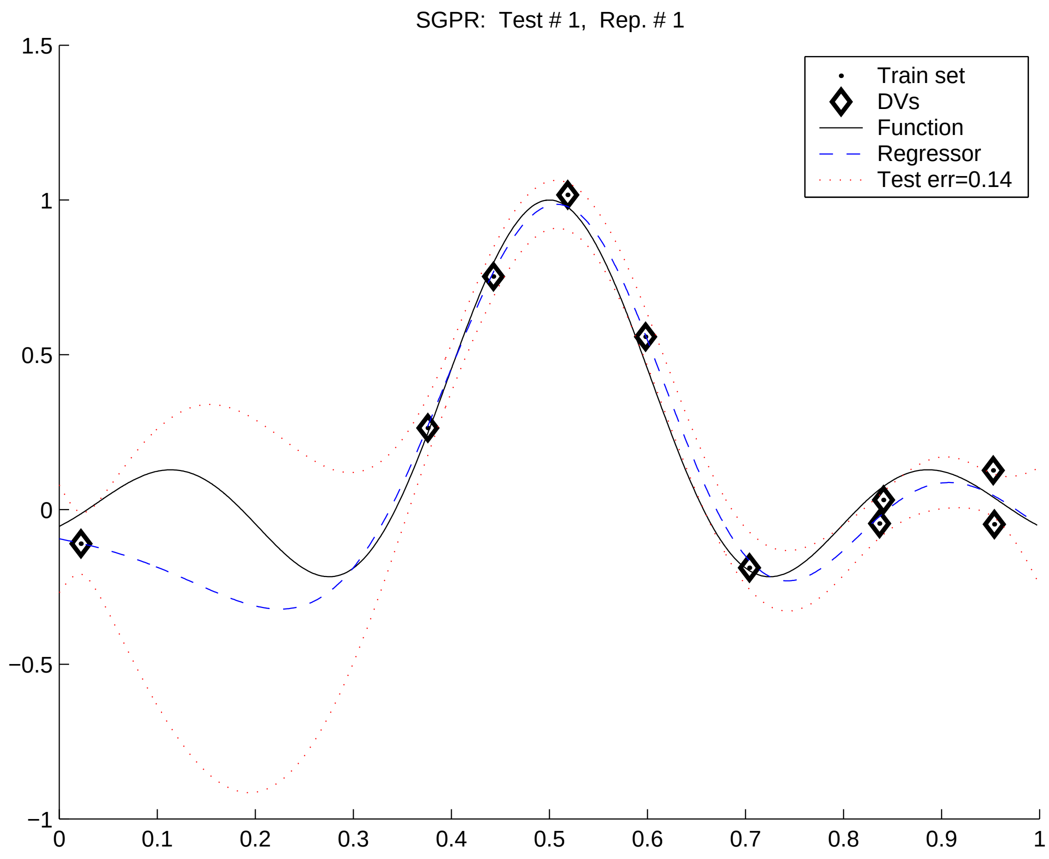


Figure 1.2: GP regression applied to the *sinc* function corrupted with IID Gaussian noise

1.3.4 Parametric Gaussian Processes

Although the nonparametric, kernel-based approach for GP regression, described above, offers much flexibility, it may sometimes be preferable to employ a parametric representation, under very similar assumptions. In the parametric setting, the GP F is assumed to consist of a linear combination of a finite number, n, of basis functions. The vector of n scalars returned by these basis functions, when they are evaluated at a given $\mathbf{x}$, is referred to as the *feature vector* of $\mathbf{x}$. Specifically, if ϕ_i is the i'th basis function, then

$$F(\mathbf{x}) = \sum_{i=1}^{n} \phi_i(\mathbf{x}) W_i = \boldsymbol{\phi}(\mathbf{x})^\top W,$$

where $\boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_n(\mathbf{x}))^\top$ is the feature vector of $\mathbf{x}$, and $W = (W_1, \ldots, W_n)^\top$. The randomness in F is now due to W being a random vector. According to our custom, the prior over W is assumed to be Gaussian, and with little loss of generality we postulate it to be distributed as[8]

$$W \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Our model-equation (Eq. 1.3.10) now becomes

$$Y_t = \mathbf{H}_t \mathbf{\Phi}_t^\top W + N_t, \tag{1.3.23}$$

where $\mathbf{\Phi}_t$ is the $n \times t$ matrix

$$\mathbf{\Phi}_t = [\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_t)] .$$

The Gauss-Markov Theorem may be applied here again to yield the mean and covariance of the posterior (Gaussian) distribution of W, conditioned on the observed data:

$$\mathbf{E}(W|Y_t) = \mathbf{\Phi}_t \mathbf{H}_t^\top \left(\mathbf{H}_t \mathbf{\Phi}_t^\top \mathbf{\Phi}_t \mathbf{H}_t^\top + \mathbf{\Sigma}_t \right)^{-1} Y_t \tag{1.3.24}$$

$$\mathbf{Cov}(W|Y_t) = \mathbf{I} - \mathbf{\Phi}_t \mathbf{H}_t^\top \left(\mathbf{H}_t \mathbf{\Phi}_t^\top \mathbf{\Phi}_t \mathbf{H}_t^\top + \mathbf{\Sigma}_t \right)^{-1} \mathbf{H}_t \mathbf{\Phi}_t^\top . \tag{1.3.25}$$

Since $F(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top W$, the posterior mean and covariance of F can now be easily computed:

$$\begin{aligned} \mathbf{E}(F(\mathbf{x})|Y_t) &= \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{\Phi}_t \mathbf{H}_t^\top \left(\mathbf{H}_t \mathbf{\Phi}_t^\top \mathbf{\Phi}_t \mathbf{H}_t^\top + \mathbf{\Sigma}_t \right)^{-1} Y_t \\ \mathbf{Cov}\left(F(\mathbf{x}), F(\mathbf{x}')|Y_t\right) &= \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{x}') - \\ &\quad \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{\Phi}_t \mathbf{H}_t^\top \left(\mathbf{H}_t \mathbf{\Phi}_t^\top \mathbf{\Phi}_t \mathbf{H}_t^\top + \mathbf{\Sigma}_t \right)^{-1} \mathbf{H}_t \mathbf{\Phi}_t^\top \boldsymbol{\phi}(\mathbf{x}'). \end{aligned} \tag{1.3.26}$$

It is instructive to note that, if our chosen feature vectors satisfy the inner product condition

$$\boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{x}') = k(\mathbf{x}, \mathbf{x}') \quad \text{for all } \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \tag{1.3.27}$$

than the two representations – parametric and nonparametric – become equivalent. This may be easily seen by noting that, in this case, $\mathbf{\Phi}_t^\top \boldsymbol{\phi}(\mathbf{x}) = \mathbf{k}_t(\mathbf{x})$, $\mathbf{\Phi}_t^\top \mathbf{\Phi}_t = \mathbf{K}_t$, and comparing the parametric posterior moments (1.3.26) with the nonparametric ones (1.3.15, assume $\mathbf{f}_0 = \mathbf{0}$). In fact, Mercer's theorem (Theorem 1.2.2) guarantees us that a set of features, for which the condition (1.3.27) is satisfied, *always exists*. However, this set may be infinite. In this sense, the nonparametric, kernel based

[8]The more general prior $\mathcal{N}(\mathbf{0}, \mathbf{S})$, where $\mathbf{S}$ is a general covariance matrix, may be reduced to this prior by linearly transforming the feature vector $\boldsymbol{\phi}(\mathbf{x})$ into a new set of features, $\boldsymbol{\psi}(\mathbf{x}) = \mathbf{S}^{1/2}\boldsymbol{\phi}(\mathbf{x})$, resulting in the exact same prior over F.

approach may be thought of as a generalization of the parametric approach, allowing us to incorporate an infinite number of features into our representation.

The following simple Lemma allows us to derive alternative expressions for the posterior mean and covariance.

Lemma 1.3.2. *If* $\mathbf{A}$ *is an* $n \times m$ *matrix,* $\mathbf{B}$ *an* $m \times n$ *matrix, and* $\mathbf{BA} + \mathbf{I}$ *is nonsingular, then* $\mathbf{AB} + \mathbf{I}$ *is also nonsingular, and* [9]

$$\mathbf{A}\left(\mathbf{BA} + \mathbf{I}\right)^{-1} = \left(\mathbf{AB} + \mathbf{I}\right)^{-1}\mathbf{A}.$$

Proof See Appendix C.

Assuming $\mathbf{\Sigma}_t^{-1}$ exists (this would usually follow from the positivity of $\mathbf{\Sigma}_t$), a simple application of Lemma 1.3.2 to Eq. 1.3.24, 1.3.25 results in,

$$\mathbf{E}\left(W|Y_t\right) = \left(\mathbf{\Phi}_t\mathbf{H}_t^\top\mathbf{\Sigma}_t^{-1}\mathbf{H}_t\mathbf{\Phi}_t^\top + \mathbf{I}\right)^{-1}\mathbf{\Phi}_t\mathbf{H}_t^\top\mathbf{\Sigma}_t^{-1}Y_t \tag{1.3.28}$$

$$\begin{aligned}\mathbf{Cov}\left(W|Y_t\right) &= \mathbf{I} - \left(\mathbf{\Phi}_t\mathbf{H}_t^\top\mathbf{\Sigma}_t^{-1}\mathbf{H}_t\mathbf{\Phi}_t^\top + \mathbf{I}\right)^{-1}\mathbf{\Phi}_t\mathbf{H}_t^\top\mathbf{\Sigma}_t^{-1}\mathbf{H}_t\mathbf{\Phi}_t^\top \\ &= \left(\mathbf{\Phi}_t\mathbf{H}_t^\top\mathbf{\Sigma}_t^{-1}\mathbf{H}_t\mathbf{\Phi}_t^\top + \mathbf{I}\right)^{-1}\end{aligned} \tag{1.3.29}$$

In the case of regression with IID Gaussian noise (with variance σ^2), we set $\mathbf{H}_t = \mathbf{I}$ and $\mathbf{\Sigma}_t = \sigma^2\mathbf{I}$, resulting in

$$\mathbf{E}\left(W|Y_t = \mathbf{y}_t\right) = \left(\mathbf{\Phi}_t\mathbf{\Phi}_t^\top + \sigma^2\mathbf{I}\right)^{-1}\mathbf{\Phi}_tY_t \tag{1.3.30}$$

$$\mathbf{Cov}\left(W|Y_t = \mathbf{y}_t\right) = \sigma^2\left(\mathbf{\Phi}_t\mathbf{\Phi}_t^\top + \sigma^2\mathbf{I}\right)^{-1} \tag{1.3.31}$$

These alternative expressions allow for considerable computational savings when $t \gg n$, since, instead of inverting the $t \times t$ matrix $\left(\mathbf{H}_t\mathbf{\Phi}_t^\top\mathbf{\Phi}_t\mathbf{H}_t^\top + \mathbf{\Sigma}_t\right)$, we are now only required to invert the $n \times n$ matrix $\left(\mathbf{\Phi}_t\mathbf{H}_t^\top\mathbf{\Sigma}_t^{-1}\mathbf{H}_t\mathbf{\Phi}_t^\top + \mathbf{I}\right)$. It may be readily verified that the posterior mean (1.3.28) is the solution to a regularized least-squares problem. Namely,

$$\hat{\mathbf{w}}_t = \mathbf{E}\left(W|Y_t = \mathbf{y}_t\right) = \underset{\mathbf{w}}{\operatorname{argmin}}\left\{\left\|\mathbf{y}_t - \mathbf{H}_t\mathbf{\Phi}_t^\top\mathbf{w}\right\|^2_{\mathbf{\Sigma}_t^{-1}} + \|\mathbf{w}\|^2\right\}, \tag{1.3.32}$$

where $\|\mathbf{u}\|^2_{\mathbf{\Sigma}_t^{-1}} \overset{\text{def}}{=} \mathbf{u}^\top\mathbf{\Sigma}_t^{-1}\mathbf{u}$. The problem (1.3.32) is easily recognized as the maximization of the log-posterior density of W. In other words, the posterior mean $\hat{\mathbf{w}}_t$

[9]Note that $\mathbf{I}$ on the l.h.s. is the $m \times m$ identity matrix, while on the r.h.s. it is the $n \times n$ identity matrix.

is also the *maximum a posteriori* (MAP) estimator of W, under our model assumptions. Taking (carefully) the limit $\mathbf{\Sigma}_t \rightarrow \mathbf{0}$, reduces (1.3.32) to a simple least-squares problem:

$$\min_{\mathbf{w}} \left\| \mathbf{y}_t - \mathbf{H}_t \mathbf{\Phi}_t^\top \mathbf{w} \right\|^2_{\mathbf{\Sigma}_t^{-1}}, \tag{1.3.33}$$

The solution of this problem is the *maximum likelihood* (ML) estimator of W, within our model. To summarize,

$$\hat{\mathbf{w}}_t^{MAP} = \left(\mathbf{\Phi}_t \mathbf{H}_t^\top \mathbf{\Sigma}_t^{-1} \mathbf{H}_t \mathbf{\Phi}_t^\top + \mathbf{I} \right)^{-1} \mathbf{\Phi}_t \mathbf{H}_t^\top \mathbf{\Sigma}_t^{-1} \mathbf{y}_t, \tag{1.3.34}$$

$$\hat{\mathbf{w}}_t^{ML} = \left(\mathbf{\Phi}_t \mathbf{H}_t^\top \mathbf{\Sigma}_t^{-1} \mathbf{H}_t \mathbf{\Phi}_t^\top \right)^{-1} \mathbf{\Phi}_t \mathbf{H}_t^\top \mathbf{\Sigma}_t^{-1} \mathbf{y}_t. \tag{1.3.35}$$

The same kind of analysis applies to the nonparametric case. Here again, we may invoke Lemma 1.3.2 to derive alternative expressions for $\boldsymbol{\alpha}_t$ and $\mathbf{C}_t$:

$$\boldsymbol{\alpha}_t = \left(\mathbf{H}_t^\top \mathbf{\Sigma}_t^{-1} \mathbf{H}_t \mathbf{K}_t + \mathbf{I} \right)^{-1} \mathbf{H}_t^\top \mathbf{\Sigma}_t^{-1} \mathbf{r}_{t-1}, \tag{1.3.36}$$

$$\mathbf{C}_t = \left(\mathbf{H}_t^\top \mathbf{\Sigma}_t^{-1} \mathbf{H}_t \mathbf{K}_t + \mathbf{I} \right)^{-1} \mathbf{H}_t^\top \mathbf{\Sigma}_t^{-1} \mathbf{H}_t. \tag{1.3.37}$$

Contrary to the parametric case, these alternative expressions do not offer us any computational advantage over the original ones. However, this alternative representation of the Bayesian solution serves as a link to the nonparametric MAP and ML solutions. In the nonparametric case we know from Eq. 1.3.16 that the solution is of the form $\boldsymbol{\alpha}^\top \mathbf{k}_t(\mathbf{x})$ (we assume $f_0 \equiv 0$). The MAP problem then becomes

$$\min_{\boldsymbol{\alpha}} \left\{ \| \mathbf{y}_t - \mathbf{H}_t \mathbf{K}_t \boldsymbol{\alpha} \|^2_{\mathbf{\Sigma}_t^{-1}} + \boldsymbol{\alpha}^\top \mathbf{K}_t \boldsymbol{\alpha} \right\}. \tag{1.3.38}$$

In the classical, frequentist view, the term $\boldsymbol{\alpha}^\top \mathbf{K}_t \boldsymbol{\alpha}$ is viewed as a regularization term. In fact, it can be easily shown that $\boldsymbol{\alpha}^\top \mathbf{K}_t \boldsymbol{\alpha} = \|f\|^2$, which is the squared RKHS norm of f, making this an instance of Tikhonov-style regularization. By imposing the stationarity conditions (equating the gradient w.r.t. $\boldsymbol{\alpha}$ to $\mathbf{0}$) the MAP solution may be shown to be

$$\boldsymbol{\alpha}_t^{MAP} = \left(\mathbf{H}_t^\top \mathbf{\Sigma}_t^{-1} \mathbf{H}_t \mathbf{K}_t + \mathbf{I} \right)^{-1} \mathbf{H}_t^\top \mathbf{\Sigma}_t^{-1} \mathbf{y}_t, \tag{1.3.39}$$

which is the same as the posterior mean parameter $\boldsymbol{\alpha}_t$ from Eq. 1.3.36. Again, in the limit of $\mathbf{\Sigma}_t \rightarrow 0$, the influence of the regularizing term $\boldsymbol{\alpha}^\top \mathbf{K}_t \boldsymbol{\alpha}$ vanishes, and we are left with the ML solution

$$\boldsymbol{\alpha}_t^{ML} = \left(\mathbf{H}_t^\top \mathbf{\Sigma}_t^{-1} \mathbf{H}_t \mathbf{K}_t \right)^{-1} \mathbf{H}_t^\top \mathbf{\Sigma}_t^{-1} \mathbf{y}_t, \tag{1.3.40}$$

As mentioned before, in the nonparametric case the ML solution is not very useful, as it is bound to overfit due to the lack of any capacity control mechanism. The high capacity of the ML solution may be demonstrated by observing that it doesn't make any errors on the training set, regardless of its size:

$$\begin{aligned}
\Big(\hat{y}(\mathbf{x}_1), \quad \ldots \quad , \hat{y}(\mathbf{x}_t)\Big)^\top_{ML} &= \mathbf{H}_t\mathbf{K}_t\boldsymbol{\alpha}_t^{ML} \\
&= \mathbf{H}_t\mathbf{K}_t\left(\mathbf{H}_t^\top\boldsymbol{\Sigma}_t^{-1}\mathbf{H}_t\mathbf{K}_t\right)^{-1}\mathbf{H}_t^\top\boldsymbol{\Sigma}_t^{-1}\mathbf{y}_t \\
&= \left(\mathbf{H}_t\mathbf{K}_t\mathbf{H}_t^\top\boldsymbol{\Sigma}_t^{-1}\right)^{-1}\mathbf{H}_t\mathbf{K}_t\mathbf{H}_t^\top\boldsymbol{\Sigma}_t^{-1}\mathbf{y}_t \\
&= \mathbf{y}_t \\
&= \Big(y(\mathbf{x}_1), \quad \ldots \quad , y(\mathbf{x}_t)\Big)^\top .
\end{aligned}$$

The third equality above is due to Lemma 1.3.2.

Finally, it is worth mentioning that the parametric linear statistical model (1.3.23) is a special case of the celebrated Kalman filter (Kalman, 1960), for the case where there are no state dynamics, i.e. where $W_{t+1} = W_t \equiv W$ is constant.

1.3.5 Summary and Remarks

We have seen that GPs, in the framework of the linear statistical model, can be used to perform Bayesian inference using both parametric and nonparametric representations. In either case, GPs deliver an estimator that takes into account both our prior knowledge and the observed data, in the form of the posterior mean. We stated without proof several optimality properties of this estimator, the proofs of which may be found elsewhere (e.g., Scharf, 1991). In addition to the posterior mean, GPs provide us with the posterior covariance, which may be used to provide confidence intervals for the predictions supplied by the mean. We also showed the connection between the GP estimator and the MAP, ML and least-squares solutions to the same problem.

The regression model (1.3.19) in which $\mathbf{H}$ is the identity operator is probably the simplest example of such Linear-Gaussian statistical models. However, $\mathbf{H}$ may represent any linear operation, such as arbitrary linear combinations of F at any number of different points, differentiation and integration of F, etc. Moreover, in several of these instances, the posterior moments may still be computed efficiently. It is quite remarkable that, until recently, this fact was largely ignored in the Machine Learning community (but see Solak et al., 2003; Graepel, 2003). In Chapter 4 of this thesis we will apply such linear statistical models to the solution of RL problems.

1.4 Markov Decision Processes

As mentioned above, Markov Decision Processes (MDPs), or a generalization thereof, known as partially observable MDPs (POMDPs, Kaelbling et al., 1998), provide the formal basis underlying RL methodology. A discrete time MDP is a tuple $(\mathcal{X}, \mathcal{U}, R, p)$, where

- $\mathcal{X}$ is the state space
- $\mathcal{U}$ is the action space
- $R : \mathcal{X} \to \mathbb{R}$ is the immediate reward[10], which may be a random process (see below for a definition), in which case $R(\mathbf{x})$ is distributed as $R(\mathbf{x}) \sim q(\cdot|\mathbf{x})$.
- $p : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to [0, 1]$ is the transition probability distribution, which we assume to be stationary.

In the context of RL it is useful to define several additional objects. A *stationary policy* $\mu : \mathcal{X} \times \mathcal{U} \to [0, 1]$ is a time-independent mapping from states to action selection probabilities. A given policy induces a *policy-dependent* state-transition probability distribution, defined as[11]

$$p^\mu(\mathbf{x}'|\mathbf{x}) = \int_{\mathcal{U}} d\mathbf{u} \mu(\mathbf{u}|\mathbf{x}) p(\mathbf{x}'|\mathbf{u}, \mathbf{x}). \tag{1.4.41}$$

Hence, for a fixed policy and a fixed initial state $\mathbf{x}_0$, the probability (density) of observing the sequence of states $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_t$ is $\prod_{i=1}^t p^\mu(\mathbf{x}_i|\mathbf{x}_{i-1})$.

Another useful quantity is the *discounted return*. The discounted return is a random process, defined as

$$D(\mathbf{x}) = \sum_{i=0}^{\infty} \gamma^i R(\mathbf{x}_i) | \mathbf{x}_0 = \mathbf{x}, \quad \text{where } \mathbf{x}_{i+1} \sim p^\mu(\cdot|\mathbf{x}_i) \text{ for all } i \geq 0, \tag{1.4.42}$$

and $\gamma \in [0, 1]$ is the discount factor[12]. The randomness in $D(\mathbf{x}_0)$, for any given state $\mathbf{x}_0$, is due both to the stochasticity of the sequence of states that follow $\mathbf{x}_0$, and to the randomness, or noise, in the rewards $R(\mathbf{x}_0), R(\mathbf{x}_1), \ldots$ etc., both of which jointly constitute the *intrinsic* randomness of the MDP.

Eq. 1.4.42 together with the stationarity of the MDP yield

$$D(\mathbf{x}) = R(\mathbf{x}) + \gamma D(\mathbf{x}'), \quad \text{where } \mathbf{x}' \sim p^\mu(\cdot|\mathbf{x}). \tag{1.4.43}$$

[10]The general case is $R : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to \mathbb{R}$; to simplify the exposition we assume that the reward associated with a transition from state $\mathbf{x}$ to state $\mathbf{x}'$ depends only on $\mathbf{x}$. However, the subsequent analysis can be easily generalized to accommodate less restrictive reward models.

[11]Here and in the sequel, whenever integration is performed over a finite or discrete space, the integral should be understood as a summation.

[12]When $\gamma = 1$ the policy must be proper, see Bertsekas and Tsitsiklis (1996)

The equality here should be understood as an equality in the distributions of the two sides of the equation. Let us define the expectation operator $\mathbf{E}_\mu$ as the expectation over all possible trajectories and all possible rewards collected therein. This allows us to define the *value function* $V(\mathbf{x})$ as the result of applying this expectation operator to the discounted return $D(\mathbf{x})$, i.e.

$$V(\mathbf{x}) = \mathbf{E}_\mu D(\mathbf{x}) \tag{1.4.44}$$

Thus, applying $\mathbf{E}_\mu$ to both sides of Eq. 1.4.43, and using the conditional expectation formula (Scharf, 1991), we get

$$\begin{aligned} V(\mathbf{x}) \overset{\text{def}}{=} \mathbf{E}_\mu D(\mathbf{x}) &= \mathbf{E}_\mu \left[R(\mathbf{x}) + \gamma D(\mathbf{x}') \right] \\ &= \bar{r}(\mathbf{x}) + \gamma \mathbf{E}_{\mathbf{x}'} \mathbf{E}_\mu \left[D(\mathbf{x}') | \mathbf{x}' \right] \\ &= \bar{r}(\mathbf{x}) + \gamma \mathbf{E}_{\mathbf{x}'} V(\mathbf{x}'), \end{aligned}$$

where

$$\begin{aligned} \mathbf{E}_{\mathbf{x}'} V(\mathbf{x}') &= \int_{\mathcal{X}} d\mathbf{x}' p^\mu(\mathbf{x}'|\mathbf{x}) V(\mathbf{x}'), \quad \text{and} \\ \bar{r}(\mathbf{x}) &= \int_{\mathbb{R}} dr q(r|\mathbf{x}) r \quad \text{is the expected reward at the state } \mathbf{x}. \end{aligned}$$

The equality we have just proved, namely that

$$V(\mathbf{x}) = \bar{r}(\mathbf{x}) + \gamma \mathbf{E}_{\mathbf{x}'} V(\mathbf{x}') \quad \forall \mathbf{x} \in \mathcal{X}, \tag{1.4.45}$$

is recognizable as the fixed-policy version of the Bellman equation[13] (Bellman, 1957). The policy that maximizes the expected discounted return from each state is called an optimal policy, and is denoted by μ^*. In the case of stationary MDPs, there exists a *deterministic* optimal policy[14]. With some abuse of notation, we denote the action selected by a deterministic policy μ, at a state $\mathbf{x}$, by $\mu(\mathbf{x})$.

The value function corresponding to an optimal policy is called the optimal value, and is denoted by V^*. While there may exist more than one optimal policies, the optimal value is unique (Bertsekas, 1995), and may be computed by solving the Bellman optimality equation

$$V^*(\mathbf{x}) = \bar{r}(\mathbf{x}) + \gamma \max_{\mathbf{u}} \int_{\mathcal{X}} d\mathbf{x}' p(\mathbf{x}'|\mathbf{u}, \mathbf{x}) V^*(\mathbf{x}') \quad \forall \mathbf{x} \in \mathcal{X}. \tag{1.4.46}$$

[13]A similar equation, satisfied by the variance of the discounted return, may be derived in an analogous manner, this time using the conditional variance formula, see Sobel, 1982 for details.

[14]This is no longer the case for POMDPs and Markov Games, see Kaelbling et al. (1998); Littman (1994).

Even if $\mathcal{X}$ is finite, and assuming for now, that both p and $\bar{r}$ are known, solving the set of equations (1.4.46) remains a non-trivial endeavor if $|\mathcal{X}|$ is large; since we are faced with a set of $|\mathcal{X}|$ nonlinear equations. Ignoring this obstacle for the moment, let us assume that we have already solved Eq. 1.4.46 for V^*. Can we now compute an optimal policy μ^* from V^*? The answer is affirmative. Specifically, for any $\mathbf{x} \in \mathcal{X}$, a deterministic optimal policy is given by

$$\mu^*(\mathbf{x}) = \arg\max_{\mathbf{u}} \int_{\mathcal{X}} d\mathbf{x}' p(\mathbf{x}'|\mathbf{u}, \mathbf{x}) V^*(\mathbf{x}') \tag{1.4.47}$$

1.4.1 Dynamic Programming

In the preceding section we noted the difficulty of solving Bellman's optimality equation. Luckily, two efficient and provably convergent dynamic programming (DP) algorithms (over which many variations exist) are available at our disposal. Value Iteration (VI) works by initializing a value estimate, for all $\mathbf{x} \in \mathcal{X}$, by $\hat{V}_0(\mathbf{x}) = \bar{r}(\mathbf{x})$, and using Eq. 1.4.46 as an update rule:

$$\hat{V}_i(\mathbf{x}) = \bar{r}(\mathbf{x}) + \gamma \max_{\mathbf{u}} \int_{\mathcal{X}} d\mathbf{x}' p(\mathbf{x}'|\mathbf{u}, \mathbf{x}) \hat{V}_{i-1}(\mathbf{x}'). \tag{1.4.48}$$

Algorithm 1 describes VI in pseudocode.

Algorithm 1 The Value Iteration Algorithm

```
For all $\mathbf{x} \in \mathcal{X}$, set  $\hat{V}_0(\mathbf{x}) := \bar{r}(\mathbf{x})$
$i := 0$
set $done := false$
while not done
  For all $\mathbf{x} \in \mathcal{X}$, set $\hat{V}_{i+1}(\mathbf{x}) := \bar{r}(\mathbf{x}) + \gamma \max_{\mathbf{u}} \int_{\mathcal{X}} d\mathbf{x}' p(\mathbf{x}'|\mathbf{u}, \mathbf{x}) \hat{V}_i(\mathbf{x}')$
  if $\|\hat{V}_{i+1} - \hat{V}_i\| \le \varepsilon$, set $done := true$
  $i := i + 1$
end while
For all $\mathbf{x} \in \mathcal{X}$, set $\hat{\mu}^*(\mathbf{x}) := \arg\max_{\mathbf{u}} \int_{\mathcal{X}} d\mathbf{x}' p(\mathbf{x}'|\mathbf{u}, \mathbf{x}) \hat{V}_i(\mathbf{x}')$
return $\hat{\mu}^*$
```

The termination condition used above (the **if** inside the loop) involves the computation of some norm of the difference between two subsequent value estimates. Several theoretical results exist, which provide bounds for the performance of $\hat{\mu}^*$ as a function of ε for different norms, including the maximum norm $\|\cdot\|_\infty$ (Williams & Baird, 1993), and weighted ℓ_2 norms (Munos, 2003). In practice, however, a less computationally cumbersome criterion is often used.

Policy Iteration (PI) works by the process of *policy improvement*. Specifically, it starts with some initial random deterministic policy. At each successive iteration, it evaluates the value function for the current policy, and then performs a policy improvement step, in which a new policy is generated by selecting the greedy action at each state, with respect to the values of the current policy. Iterating the policy evaluation – policy improvement process is known to produce a strictly monotonically improving sequence of policies. If the improved policy is the same as the policy improved upon, then we are assured that the optimal policy has been found. The policy evaluation step can be implemented by any algorithm that solves the set of equations (1.4.45). The pseudocode for PI is given in Algorithm 2.

Algorithm 2 The Policy Iteration Algorithm

```
For all $\mathbf{x} \in \mathcal{X}$, set  $\hat{\mu}_0(\mathbf{x})$ = some random action
$i := 0$
set $done := false$
while not done
  call subroutine $\hat{V}_i$ := PolicyEvaluation($\hat{\mu}_i$)
  For all $\mathbf{x} \in \mathcal{X}$, set $\hat{\mu}_{i+1}(\mathbf{x}) := \arg\max_{\mathbf{u}} \int_{\mathcal{X}} d\mathbf{x}' p(\mathbf{x}'|\mathbf{u},\mathbf{x}) \hat{V}_i(\mathbf{x}')$
  if $\hat{\mu}_{i+1} = \hat{\mu}_i$, set $done := true$
  $i := i + 1$
end while
return $\mu^* := \hat{\mu}_i$
```

For instance, if the number of states is finite (N), states can be enumerated using integer indices $i = 1, 2, \dots, N$. If N is sufficiently small, we can store the estimated state values in a lookup table, or a vector, $\hat{\mathbf{v}}$ of which the i'th component is $\hat{V}(i)$. Similarly, the components of the policy-dependent transition probability can be stored in a matrix $[\mathbf{P}^{\mu}]_{i,j} = p^{\mu}(j|i)$, and the mean rewards in a vector $\bar{\mathbf{r}}$. Then, the Bellman equation (1.4.45) may be written concisely as

$$\hat{\mathbf{v}} = \bar{\mathbf{r}} + \gamma \mathbf{P}^{\mu} \hat{\mathbf{v}}, \tag{1.4.49}$$

and solved for $\hat{\mathbf{v}}$ by

$$\hat{\mathbf{v}} = (\mathbf{I} - \gamma \mathbf{P}^{\mu})^{-1} \bar{\mathbf{r}}.$$

Note, that if the discount factor $\gamma = 0$, this reduces to $\hat{\mathbf{v}} = \bar{\mathbf{r}}$.

If the number of states renders the inversion of $\mathbf{I} - \gamma \mathbf{P}^{\mu}$ too expensive, we may resort to solving Eq. 1.4.49 by fixed-point iterations. That is, we iterate

$$\hat{\mathbf{v}}_{t+1} = \bar{\mathbf{r}} + \gamma \mathbf{P}^{\mu} \hat{\mathbf{v}}_t. \tag{1.4.50}$$

The pseudocode for this fixed-point policy evaluation (FPPE) algorithm is given in Algorithm 3 (read integrals as sums, as necessary). Note the similarity between this algorithm and the VI algorithm (Algorithm 1). In fact, when there is only one possible action at each state, the two algorithms coincide.

Algorithm 3 The Fixed-Point Policy Evaluation Algorithm

```
For all $\mathbf{x} \in \mathcal{X}$, set $\hat{V}_0(\mathbf{x}) := \bar{r}(\mathbf{x})$
$i := 0$
set $done := false$
while not done
  For all $\mathbf{x} \in \mathcal{X}$, set $\hat{V}_{i+1}(\mathbf{x}) := \bar{r}(\mathbf{x}) + \gamma \int_{\mathcal{X}} d\mathbf{x}' p^{\mu}(\mathbf{x}'|\mathbf{x}) \hat{V}_i(\mathbf{x}')$
  if $\|\hat{V}_{i+1} - \hat{V}_i\| \leq \varepsilon$, set $done := true$
  $i := i + 1$
end while
return $\hat{V}_i$
```

In many cases, such as the one just discussed, the policy evaluation routine only provides an approximation of the true value function. In such cases, the resulting algorithm is referred to as Approximate Policy Iteration (API). If the approximation error is smaller than half the smallest difference between the values of the best and second best actions, then API can still be shown to converge to the optimal policy. When compared with the VI iterations, each iteration of PI or API is rather expensive computationally. However, PI algorithms typically require surprisingly few iterations to converge (see Puterman, 1994; Bertsekas, 1995 for further details).

1.4.2 Reinforcement Learning

Most successful Reinforcement Learning algorithms are descended from one of the two DP algorithms described above, VI and PI. However, there are two major features distinguishing the RL setting from the traditional decision theoretic setting. First, while in decision theory it is assumed that the environment model is fully known, in RL no such assumption is made. Second, in RL, the learning process is usually assumed to take place *online*, namely, concurrently with the accumulation of actual or simulated data acquired by the learning agent as it explores its environment. These two features make RL a significantly more difficult challenge, and place serious constraints on any potential RL algorithm. Probably the two best known RL algorithms, TD(λ) (Sutton, 1988) and Q-learning (Watkins, 1989), serve well to demonstrate how RL methods handle these constraints. For simplicity we assume that the state and action spaces are finite, and that state values, or state-action values are stored explicitly in a lookup table.

TD(λ) is aimed at evaluating the value function for a fixed policy μ. The input to the algorithm is a sequence of state-reward couples, generated by the MDP controlled by the policy μ. The idea in TD(λ) is to gradually improve value estimates by moving them towards the weighted average of multi-step lookahead estimates, which take into account the observed rewards. In the simplest case, of $\lambda = 0$, this amounts to moving the value estimate of the current state, $\hat{V}(x_t)$, toward the 1-

step lookahead estimate $r_t + \gamma \hat{V}(\mathbf{x}_{t+1})$. Since the next state, $\mathbf{x}_{t+1}$, is sampled from the policy-dependent transition probability $p^\mu(\cdot|\mathbf{x}_t)$, the expected move of $\hat{V}(x_t)$ is in the direction of $r_t + \gamma \sum_{\mathbf{x}} p^\mu(\mathbf{x}|\mathbf{x}_t)\hat{V}(\mathbf{x})$. In effect, what was done here was to replace the computation of the mean value-estimate over all possible states that can follow $\mathbf{x}_t$, by the single term $\hat{V}(\mathbf{x}_{t+1})$, which is an unbiased estimator for this mean[15]. Similarly, the sampled reward, r_t, is an unbiased estimator for $\bar{r}(\mathbf{x}_t)$. In the long run, under some technical conditions, this has the effect of driving $\hat{V}$ to the solution of the Bellman equation (1.4.45) (Sutton, 1988; Watkins, 1989; Dayan, 1992; Dayan & Sejnowski, 1994). Thus, TD methods avoid making direct use of the transition model $\{p^\mu, q\}$ by sampling from it. TD(0) may also be thought of as an asynchronous, stochastic version of the FPPE algorithm described above. The pseudocode for TD(0) is given in Algorithm 4. The update term $r_{t-1} + \gamma\hat{V}(\mathbf{x}_t) - \hat{V}(\mathbf{x}_{t-1})$ is referred to as the *temporal difference*[16] at time-step t.

Algorithm 4 The Tabular TD(0) Algorithm

```
For all $\mathbf{x} \in \mathcal{X}$, **set** $\hat{V}(\mathbf{x}) := 0$
**for** $t = 1, 2, \ldots$
  **observe** $\mathbf{x}_{t-1}$, $r_{t-1}$, $\mathbf{x}_t$
  $\hat{V}(\mathbf{x}_{t-1}) := \hat{V}(\mathbf{x}_{t-1}) + \eta \left(r_{t-1} + \gamma\hat{V}(\mathbf{x}_t) - \hat{V}(\mathbf{x}_{t-1})\right)$
**end for**
**return** $\hat{V}$
```

In many cases, RL tasks are naturally divided into *learning episodes.* In such episodic learning tasks, the agent is placed at some (typically randomly chosen) initial state, and is then allowed to follow its policy until it reaches a *terminal absorbing state.* At this point the episode terminates and a new one may begin. A terminal state is modeled as a state with zero reward and with only self transitions, for any action. If the state at time t, $\mathbf{x}_t$, is terminal, then in the TD(λ) update, as well as in algorithms presented in the sequel, we *define* $\hat{V}(\mathbf{x}_t) = 0$.

Recall that a value estimation algorithm is only a subcomponent of a complete PI-type algorithm, for finding a near-optimal policy. The policy improvement step requires that we compute the improved policy by

$$\hat{\mu}_{i+1}(\mathbf{x}) := \arg\max_{\mathbf{u}} \int_{\mathcal{X}} d\mathbf{x}' p(\mathbf{x}'|\mathbf{u}, \mathbf{x})\hat{V}_i(\mathbf{x}')$$

This, of course, requires knowledge of the transition model $p(\mathbf{x}'|\mathbf{u}, \mathbf{x})$. As such knowledge is typically unavailable[17], we need to find a way to perform this maxi-

[15]Note, however, that it is not an unbiased estimator for $\sum_{\mathbf{x}} p^\mu(\mathbf{x}|\mathbf{x}_t)V(\mathbf{x})$!

[16]This is a slight departure from conventional notation, meant to ensure that the update at time-step t uses only information available up to that time. Hence, the temporal difference at time t is not defined as $r_t + \gamma\hat{V}_t(\mathbf{x}_{t+1}) - \hat{V}_t(\mathbf{x}_t)$, as this definition requires knowledge the state at time $t+1$.

[17]Even if the MDP model were available, computing the expectation may be too costly to perform.

mization efficiently, without requiring us to invoke the transition model. One way this may be done is by learning state-action values, also known as Q-values, rather than just state values. For a given policy μ, the Q-value for the state-action pair $(\mathbf{x}, \mathbf{u})$ is the expected discounted return over all trajectories starting from $\mathbf{x}$, for which the first action is $\mathbf{u}$, and with all subsequent actions chosen according to μ. The Q-values associated with μ may be shown to satisfy the following form of the Bellman equation (compare to Eq. 1.4.45):

$$\begin{aligned} Q(\mathbf{x}, \mathbf{u}) &= \bar{r}(\mathbf{x}) + \gamma \mathbf{E}_{\mathbf{x}'|\mathbf{u}} V(\mathbf{x}') \\ &= \bar{r}(\mathbf{x}) + \gamma \int_{\mathcal{X}} d\mathbf{x}' p(\mathbf{x}'|\mathbf{u}, \mathbf{x}) V(\mathbf{x}'), \end{aligned} \tag{1.4.51}$$

where $V(\mathbf{x}') = \int_{\mathcal{U}} d\mathbf{u}' \mu(\mathbf{u}'|\mathbf{x}') Q(\mathbf{x}', \mathbf{u}')$. The policy improvement step can now be performed by a single maximization operation per state: $\hat{\mu}_{i+1}(\mathbf{x}) := \arg\max_{\mathbf{u}} \hat{Q}_i(\mathbf{x}, \mathbf{u})$. The TD($\lambda$) algorithm may be used, essentially unchanged, to learn state-action values. This algorithm is known as SARSA (State-Action-Reward-State-Action) (Sutton & Barto, 1998). Algorithm 5 provides the pseudocode of SARSA(0).

Algorithm 5 The Tabular SARSA(0) Algorithm

```
For all $\mathbf{x} \in \mathcal{X}$ and $\mathbf{u} \in \mathcal{U}$ **set** $\hat{Q}^*(\mathbf{x}, \mathbf{u}) := 0$
**for** $t = 1, 2, \ldots$
  **observe** $\mathbf{x}_{t-1}$, $\mathbf{u}_{t-1}$, $r_{t-1}$, $\mathbf{x}_t$, $\mathbf{u}_t$
  $\hat{Q}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) := \hat{Q}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) + \eta \left( r_{t-1} + \gamma \hat{Q}(\mathbf{x}_t, \mathbf{u}_t) - \hat{Q}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \right)$
**end for**
**return** $\hat{Q}$
```

The Q-learning algorithm is to Value Iteration what SARSA(0) is to the FPPE algorithm. Namely, Q-learning is an asynchronous, stochastic version of VI, which learns Q-values. In this case, state-action values allow the maximization operation, inherent to all VI methods, to be carried out efficiently, and without requiring knowledge of the transition model. As usual, we denote the Q-function associated with an optimal policy μ^* by Q^*. Evidently, the optimal value function V^* is related to Q^* by

$$V^*(\mathbf{x}) = \max_{\mathbf{u}} Q^*(\mathbf{x}, \mathbf{u}),$$

as the maximization makes sure the first action is optimal, and all subsequent actions must also be optimal, by the definition of Q^*. Hence, the Q-values associated with the optimal policy may be shown to satisfy an analogue of Eq. 1.4.46

$$Q^*(\mathbf{x}, \mathbf{u}) = \bar{r}(\mathbf{x}) + \gamma \mathbf{E}_{\mathbf{x}'|\mathbf{u}} \max_{\mathbf{u}'} Q^*(\mathbf{x}', \mathbf{u}') \tag{1.4.52}$$

In the spirit of TD and SARSA, in which expectations were replaced with actual samples, we can, by analogy, derive the Q-learning algorithm. Algorithm 6 gives the pseudocode.

Algorithm 6 The Tabular Q-Learning Algorithm

```
For all $\mathbf{x} \in \mathcal{X}$ and $\mathbf{u} \in \mathcal{U}$ set  $\hat{Q}^*(\mathbf{x},\mathbf{u}) := 0$
for $t = 1, 2, \ldots$
  observe $\mathbf{x}_{t-1}$, $\mathbf{u}_{t-1}$, $r_{t-1}$, $\mathbf{x}_t$
  $\hat{Q}^*(\mathbf{x}_{t-1},\mathbf{u}_{t-1}) := \hat{Q}^*(\mathbf{x}_{t-1},\mathbf{u}_{t-1}) + \eta\left(r_{t-1} + \gamma \max_{\mathbf{u}'} \hat{Q}^*(\mathbf{x}_t,\mathbf{u}') - \hat{Q}^*(\mathbf{x}_{t-1},\mathbf{u}_{t-1})\right)$
end for
return $\hat{Q}^*$
```

Assuming $\hat{Q}^* = Q^*$ (i.e. Q-learning has converged to the optimum), then a (deterministic) optimal action for each state can be easily computed by a single maximization operation,

$$\mu^*(\mathbf{x}) = \arg\max_{\mathbf{u}} \hat{Q}^*(\mathbf{x},\mathbf{u}),$$

with ties broken arbitrarily. One of the attractive features of Q-learning, is that, regardless of the exploration policy μ, as long as μ avoids neglecting parts of the state and action spaces, the estimates $\hat{Q}^*$ will converge to the optimal Q-values Q^*. This is referred to as *off-policy learning*, generally meaning that the values of one policy are learned while another is being followed.

The TD, SARSA and Q-learning algorithms, as well as many other algorithms relying on a lookup-table representation, are useful in providing a proof-of-concept. However, real-world problems can rarely be solved using such representations, due to the large, and sometimes infinite, state and action spaces, which characterize such problems. Since a tabular representation is unfeasible, it is necessary, in such problems, to use some form of function approximation (FA) to represent the value function and possibly also the policy. This, however, gives rise to another difficulty, as many otherwise popular function approximation schemes (e.g., sigmoidal multilayer neural nets), when used to represent the value function, interfere with the contraction properties of the DP operators (Gordon, 1996). In reality, practitioners either ignore this problem and make do without any convergence guarantees, with mixed success (e.g., Tesauro, 1995; Schraudolph et al., 1994; Crites & Barto, 1996), or resort to using special forms of FA, that are well behaved (Singh et al., 1995; Boyan & Moore, 1995; Tsitsiklis & Van Roy, 1996; Gordon, 1996; Munos, 2000).

One particular form of FA, in which the approximation is linear in its parameters has emerged as particularly useful. There is a significant body of work providing convergence guarantees for the TD algorithm (Sutton, 1988) and several related algorithms, when used in conjunction with such linear approximation architectures

(Tsitsiklis & Van Roy, 1996; Konda & Tsitsiklis, 2000; Nedic & Bertsekas, 2003; Munos, 2003). However, linear parametric FA architectures are inherently limited in their expressive powers, since one must choose *a priori* a finite set of basis functions, the span of which constitutes the hypothesis space to which the value estimate belongs. If the true value function does not belong to this hypothesis space, the approximation to which these algorithms converge may be quite bad, depending on a measure of the distance between the hypothesis space and the true value function. Nonetheless, for the RL practitioner seeking a provably convergent, online, model-free algorithm for value estimation, the choice is limited to TD(λ) and some of its variants, such as SARSA(λ) (Sutton & Barto, 1998) and LSTD(λ), used in conjunction with a linear function approximation architecture (Bradtke & Barto, 1996; Boyan, 1999a). Let us briefly overview some of these methods.

1.4.3 TD Methods with Function Approximation

Temporal difference learning has been implemented with a wide range of function approximation schemes, including neural networks, tile codes (a.k.a. Cerebellar model articulation controllers, or CMACs), radial basis functions (RBFs), discretization-based methods, nearest-neighbors, to mention a few (see Sutton & Barto, 1998 for an overview). While in many of these instances, TD learning was successful, there are also quite a few examples in which TD methods have been shown to fail (e.g., Baird, 1995; Tsitsiklis & Van Roy, 1996). As mentioned above, for the class of linear FA architectures, i.e. in which the approximation is linear in the learned parameters, TD methods have been shown to possess desirable convergence properties - a rare commodity among RL algorithms (Tsitsiklis & Van Roy, 1996; Konda & Tsitsiklis, 2000; Nedic & Bertsekas, 2003). However, before focusing on a specific FA architecture, let us consider the general parametric setting.

In the parametric setting we postulate that the true value function is parameterized by a vector of unknown parameters $\boldsymbol{\theta}$, and our goal is to use observable data to estimate $\boldsymbol{\theta}$. In this context it is useful to define the *Bellman residual.*

Definition 1.4.1 (Bellman residual). *The Bellman residual of an estimator* $V_{\hat{\boldsymbol{\theta}}}$ *of the value function, at the state* $\mathbf{x}$*, is*

$$B(\hat{\boldsymbol{\theta}}, \mathbf{x}) = V_{\hat{\boldsymbol{\theta}}}(\mathbf{x}) - \bar{r}(\mathbf{x}) - \gamma \int_{\mathcal{X}} V_{\hat{\boldsymbol{\theta}}}(\mathbf{x}') p^{\mu}(\mathbf{x}'|\mathbf{x}) d\mathbf{x}'.$$

The Bellman residual is simply the difference between the two sides of the Bellman equation (1.4.45), when our value estimator is substituted for the value function. Since the true value function $V_{\boldsymbol{\theta}}$ uniquely solves the Bellman equation, $B(\hat{\boldsymbol{\theta}}, \mathbf{x}) \equiv 0$ only for $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}$.

A standard stochastic iterative algorithm, which may be applied for minimizing the mean squared Bellman residual (i.e. the squared Bellman residual averaged over states) is based on computing the gradient of the squared Bellman residual $\left(B(\hat{\boldsymbol{\theta}}, \mathbf{x}_i)\right)^2$ with respect to $\hat{\boldsymbol{\theta}}$, and making small adjustments to $\hat{\boldsymbol{\theta}}$ in the direction of the negative gradient. Provided that the Bellman residual is a sufficiently smooth function of $\hat{\boldsymbol{\theta}}$, and that the step-size decreases appropriately with time, asymptotic convergence to a local minimum is assured (Bertsekas & Tsitsiklis, 1996, Chapter 4). In our case, the gradient at $\mathbf{x}_{t-1}$ is

$$\nabla_{\hat{\boldsymbol{\theta}}} \left(B(\hat{\boldsymbol{\theta}}, \mathbf{x}_{t-1})\right)^2 = B(\hat{\boldsymbol{\theta}}, \mathbf{x}_{t-1}) \left(\nabla_{\hat{\boldsymbol{\theta}}} V_{\hat{\boldsymbol{\theta}}}(\mathbf{x}_{t-1}) - \gamma \nabla_{\hat{\boldsymbol{\theta}}} \int_{\mathcal{X}} V_{\hat{\boldsymbol{\theta}}}(\mathbf{x}') p^{\mu}(\mathbf{x}'|\mathbf{x}_{t-1}) d\mathbf{x}'\right). \tag{1.4.53}$$

Since the distributions p^{μ} and q are unknown, explicitly computing the r.h.s. of Eq. 1.4.53 is impossible[18]. However, assuming that we are nevertheless capable of generating samples from p^{μ} and q, we may use a stochastic approximation algorithm known as the Robbins-Monro algorithm (Bertsekas & Tsitsiklis, 1996, Chapter 4) to solve our problem. In this algorithm, each expectation is replaced by a single random sample generated from the corresponding distribution, yielding the stochastic update,

$$\hat{\boldsymbol{\theta}} := \hat{\boldsymbol{\theta}} - \eta_t \left(V_{\hat{\boldsymbol{\theta}}}(\mathbf{x}_{t-1}) - r_{t-1} - \gamma V_{\hat{\boldsymbol{\theta}}}(\mathbf{x}')\right) \left(\nabla_{\hat{\boldsymbol{\theta}}} V_{\hat{\boldsymbol{\theta}}}(\mathbf{x}_{t-1}) - \gamma \nabla_{\hat{\boldsymbol{\theta}}} V_{\hat{\boldsymbol{\theta}}}(\mathbf{x}'')\right), \tag{1.4.54}$$

where η_t is the time-dependent step-size, r_{t-1} is a reward sampled from $q(\cdot|\mathbf{x}_{t-1})$, and $\mathbf{x}'$ and $\mathbf{x}''$ are *independent random samples* from $p^{\mu}(\cdot|\mathbf{x}_{t-1})$. In the online setting we are only provided with a single such sample, namely $\mathbf{x}_t$. In the RL literature this seems to be perceived as a fundamental barrier prohibiting the online minimization of the mean squared Bellman residual (Werbos, 1990; Baird, 1995; Lagoudakis et al., 2002; Bertsekas & Tsitsiklis, 1996 Chapter 6.10.1). Note that this double-sample requirement may be safely ignored in, and only in, two degenerate special cases:

1. The state transitions of the MDP under the policy evaluated are deterministic - in which case both samples would yield the same result, and
2. the discount factor $\gamma = 0$, in which case the identity of successor states is irrelevant, since value function estimation degenerates to the estimation of the mean immediate reward (we assume that rewards depend only on the originating state and the action, but not on the successor state).

Otherwise, using $\mathbf{x}_t$ in both expectation terms results in (sometimes severely) biased value estimates (see Werbos, 1990, for a more detailed discussion).

[18]Even if p^{μ} is known, computing the integral may be computationally prohibitive.

TD(λ)

The version of TD(λ) employing function approximation is closely related to the Robbins-Monro type algorithm discussed above. Curiously, The parametric TD(0) algorithm is the result of using the update (1.4.54) with $\mathbf{x}' = \mathbf{x}_t$ and altogether ignoring the term $\nabla_{\hat{\boldsymbol{\theta}}} V_{\hat{\boldsymbol{\theta}}}(\mathbf{x}'')$, i.e.

$$\hat{\boldsymbol{\theta}} := \hat{\boldsymbol{\theta}} + \eta_t \left(r_{t-1} + \gamma V_{\hat{\boldsymbol{\theta}}}(\mathbf{x}_t) - V_{\hat{\boldsymbol{\theta}}}(\mathbf{x}_{t-1})\right) \nabla_{\hat{\boldsymbol{\theta}}} V_{\hat{\boldsymbol{\theta}}}(\mathbf{x}_{t-1}). \tag{1.4.55}$$

This is usually explained as a stochastic gradient update in which the estimator $V_{\hat{\boldsymbol{\theta}}}(\mathbf{x}_{t-1})$ is pushed toward its target value $r_{t-1} + \gamma V_{\hat{\boldsymbol{\theta}}}(\mathbf{x}_t)$, whereby $V_{\hat{\boldsymbol{\theta}}}(\mathbf{x}_t)$ is treated as if were fixed (i.e. independent of $\hat{\boldsymbol{\theta}}$). The use of target values that themselves depend on the estimated parameter, referred to as "bootstrapping", is at the same time the hallmark of TD methods and the source of their rather restricted convergence properties. Note, that for $\gamma = 0$, the update rule above is a simple gradient descent rule for regression, with the targets being the observed rewards.

Suppose now that $V_{\hat{\boldsymbol{\theta}}}$ is linear in $\hat{\boldsymbol{\theta}}$. In such linear FA schemes one has to define a number of *features* $\{\phi_i(\mathbf{x}) : \mathcal{X} \to \mathbb{R}, i \in \{1, \ldots, n\}\}$ that map the raw state vector $\mathbf{x}$ into a more refined form of representation, ideally reflecting the available prior knowledge regarding the domain at hand. These features should allow us to approximate, as closely as required, the value functions we are likely to encounter on our search for the optimal policy. In the linear-parametric case, the span of $\{\phi_i\}_{i=1}^n$ is our hypothesis space (denoted by Θ), as it specifies the complete repertoire of hypothetical value functions we are willing to consider. Having designed n such features, these may be aggregated into a *feature vector* $\boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_n(\mathbf{x}))^\top$, which is then used to represent the state $\mathbf{x}$. Therefore, using a linear FA, the estimate $\hat{V}$ for the value function is of the form

$$V_{\hat{\mathbf{w}}}(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \hat{\mathbf{w}}, \quad \text{and } \nabla_{\hat{\mathbf{w}}} V_{\hat{\mathbf{w}}}(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x}) \tag{1.4.56}$$

where $\hat{\mathbf{w}}$ is the vector of estimated parameters $[\hat{w}_1, \ldots, \hat{w}_n]^\top$. In order to streamline notation, from now on, we will denote the value estimate simply as $\hat{V}$, with its (linear) dependence on the estimated parameters remaining implicit. The subscript will be used to maintain a time index for the estimates. The TD(0) update (1.4.55) now becomes

$$\hat{\mathbf{w}}_t = \hat{\mathbf{w}}_{t-1} + \eta_t \boldsymbol{\phi}(\mathbf{x}_{t-1}) \left(r_{t-1} + \gamma \hat{V}_{t-1}(\mathbf{x}_t) - \hat{V}_{t-1}(\mathbf{x}_{t-1})\right), \tag{1.4.57}$$

where $\hat{V}_{t-1}(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \hat{\mathbf{w}}_{t-1}$ and η_t is a time dependent learning-rate parameter. Note that the parenthesized expression in this update is the temporal difference at time-step t.

TD(λ), as its name suggests, is a generalization of TD(0). In TD(0) the target toward which $\hat{V}(\mathbf{x}_t)$ is pushed is the one-step lookahead, bootstrapped estimate $r_t + \gamma\hat{V}(\mathbf{x}_{t+1})$. However, just as easily, one could define a τ-step update by defining the target as

$$\sum_{i=0}^{\tau-1} \gamma^i r_{t+i} + \gamma^\tau \hat{V}(\mathbf{x}_{t+\tau})$$

and using the τ-step temporal difference in the TD update rule. For $\gamma < 1$, any bootstrapping errors, due to the fact that $\hat{V}(\mathbf{x}_{t+\tau}) \neq V(\mathbf{x}_{t+\tau})$, will be diminished by a factor of γ^τ. However, the τ-step targets will tend to be noisier than their 1-step counterparts, since the first term, which is a sum of τ random variables collected along a random trajectory, becomes more dominant. This provides the rationale for averaging all the different τ-step targets, for $\tau = 1, 2, \ldots, \infty$. TD($\lambda$) averages them exponentially, λ being the exponential decay rate (i.e. the τ-step target is weighted by the factor $(1-\lambda)\lambda^{\tau-1}$). Setting $\lambda = 1$ is equivalent to taking the limit $\tau = \infty$. The resulting TD(1) algorithm may be shown to be an incremental, gradient-based method for solving the least-squares regression problem, in which the target values are Monte-Carlo samples of the discounted return (i.e. no bootstrapping is used, see Bertsekas & Tsitsiklis, 1996, Chapter 6.3.1).

Exponential averaging was chosen for TD(λ) for reasons of computational convenience[19], as it turns out that the resulting updates may be performed efficiently online. This is achieved by making use of an additional vector of *eligibility traces*. In the online setting each temporal difference is used only once, when it is observed, and is then discarded. The online updates for TD(λ) are (see Sutton & Barto, 1998; Bertsekas & Tsitsiklis, 1996 for details):

$$\begin{aligned} \hat{\mathbf{w}}_t &= \hat{\mathbf{w}}_{t-1} + \eta_t \mathbf{z}_{t-1}\left(r_{t-1} + \gamma\hat{V}_{t-1}(\mathbf{x}_t) - \hat{V}_{t-1}(\mathbf{x}_{t-1})\right), \quad \text{and} \qquad (1.4.58) \\ \mathbf{z}_t &= \gamma\lambda\mathbf{z}_{t-1} + \boldsymbol{\phi}(\mathbf{x}_t), \quad \text{with } \mathbf{z}_0 = \boldsymbol{\phi}(\mathbf{x}_0), \end{aligned}$$

where $\mathbf{z}_t$ is the eligibility vector at time-step t. Note that, for $\lambda = 0$, the TD(λ) update reduces to the update of Eq. 1.4.57. The pseudocode for TD(λ) with linear FA is given in Algorithm 7.

Various alternative update schemes for updating the eligibility traces have been proposed, some of which have been shown in experiments to outperform the basic

[19]Or, as Sutton and Barto put it (Sutton & Barto, 1998): "*The λ-return and TD(λ) methods use the λ parameter to shift from 1-step TD methods to Monte Carlo methods. The specific way this shift is done is interesting, but not obviously better or worse than the way it is done with simple n-step methods by varying n. Ultimately, the most compelling motivation for the way of mixing n-step backups is simply that there is a simple algorithm –TD(λ) – for achieving it. This is a mechanism issue rather than a theoretical one.*"

Algorithm 7 The TD(λ) Algorithm with Linear Function Approximation

```
Initialize $\hat{\mathbf{w}}_0 := \mathbf{0}$, $\mathbf{z}_0 := \boldsymbol{\phi}(\mathbf{x}_0)$
for $t = 1, 2, \ldots$
  observe $\mathbf{x}_{t-1}$, $r_{t-1}$, $\mathbf{x}_t$
  $\hat{\mathbf{w}}_t := \hat{\mathbf{w}}_{t-1} + \eta \mathbf{z}_{t-1} \left( r_{t-1} - \left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma \boldsymbol{\phi}(\mathbf{x}_t)\right)^\top \hat{\mathbf{w}}_{t-1} \right)$
  $\mathbf{z}_t := \gamma \lambda \mathbf{z}_{t-1} + \boldsymbol{\phi}(\mathbf{x}_t)$
end for
return $\hat{\mathbf{w}}_t$
```

scheme, described above (Singh & Sutton, 1996, see also Bertsekas & Tsitsiklis, 1996 Chapter 5.3.3). Heuristic protocols for varying λ as learning progresses have also been suggested (Sutton & Barto, 1998, Bertsekas & Tsitsiklis, 1996 Chapter 5.3.2). For lookup-table representations it was proposed to start learning with a high value of λ and decrease λ to 0 as learning progresses and bootstrap estimates improve. In fact, it has been demonstrated that, in general, the limiting solutions obtained by TD(λ) depend on λ, and that their quality can get worse as λ becomes smaller than 1 (Bertsekas & Tsitsiklis, 1996, Chapter 6.3.2). TD(1), on the other hand, is the only member of the TD(λ) family that is guaranteed to converge to the point in Θ that is closest (under some metric) to the parameters determining the true value function[20].

LSTD(λ)

Under some technical conditions, in the limit $t \to \infty$, the TD(λ) algorithm may be shown to converge to the solution of the following set of linear equations (Bertsekas & Tsitsiklis, 1996):

$$\mathbf{B}_t \hat{\mathbf{w}}_t = \mathbf{b}_t, \quad \text{where } \mathbf{B}_t = \sum_{i=0}^{t-1} \mathbf{z}_i \left(\boldsymbol{\phi}(\mathbf{x}_i) - \gamma \boldsymbol{\phi}(\mathbf{x}_{i+1})\right)^\top \quad \text{and } \mathbf{b}_t = \sum_{i=0}^{t-1} \mathbf{z}_i r_i. \tag{1.4.59}$$

For $t \geq n$, $\mathbf{B}_t$ is a square positive-definite matrix and is therefore invertible (Bertsekas & Tsitsiklis, 1996 Lemma 6.6). The LSTD(λ) algorithm (Bradtke & Barto, 1996; Boyan, 1999a) was proposed as a more data-efficient method than TD(λ), which, instead of making small updates, solves Eq. 1.4.59 directly. This may be done simply by maintaining and updating the matrix $\mathbf{B}_t$ and the vector $\mathbf{b}_t$, and solving for $\hat{\mathbf{w}}_t$ using $\hat{\mathbf{w}}_t = \mathbf{B}_t^{-1}\mathbf{b}_t$ whenever value estimates are required. This batch form of LSTD(λ) is described in Algorithm 8. It can be shown (e.g., Boyan, 1999b) that for a learning episode ending with a terminal state $\mathbf{x}_{t+1}$ (i.e. $\mathbf{x}_t$ is the last

[20]It is interesting to note that the convergence theory regarding TD(λ) with linear approximation architectures suggests a diametric protocol to the one proposed for the tabular case, namely, an increasing schedule for λ. This is due to the fact that the bound on the quality of the asymptotic solution improves as λ approaches 1.

Algorithm 8 The Batch LSTD(λ) Algorithm

```
Parameters: $\lambda$
Initialize $\mathbf{B}_0 = \mathbf{0}$, $\mathbf{b}_0 = \mathbf{0}$, $\mathbf{z}_0 := \boldsymbol{\phi}(\mathbf{x}_0)$,
for $t = 1, 2, \ldots$
  observe $\mathbf{x}_{t-1}$, $r_{t-1}$, $\mathbf{x}_t$
  $\mathbf{B}_t = \mathbf{B}_{t-1} + \mathbf{z}_{t-1}\left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)\right)^\top$
  $\mathbf{b}_t = \mathbf{b}_{t-1} + \mathbf{z}_{t-1}r_{t-1}$
  $\mathbf{z}_t = \gamma\lambda\mathbf{z}_{t-1} + \boldsymbol{\phi}(\mathbf{x}_t)$
end for
return $\hat{\mathbf{w}}_t = \mathbf{B}_t^{-1}\mathbf{b}_t$
```

non-terminal state in the episode), for $\lambda = 1$, $\mathbf{B}_{t+1}$ and $\mathbf{b}_{t+1}$ satisfy

$$\mathbf{B}_{t+1} = \sum_{i=0}^{t} \boldsymbol{\phi}(\mathbf{x}_i)\boldsymbol{\phi}(\mathbf{x}_i)^\top, \quad \text{and } \mathbf{b}_{t+1} = \sum_{i=0}^{t} \boldsymbol{\phi}(\mathbf{x}_i) \sum_{j=i}^{t-1} \gamma^{j-i} r_j.$$

This makes

$$\hat{\mathbf{w}}_{t+1} = \mathbf{B}_{t+1}^{-1}\mathbf{b}_{t+1} = \left(\boldsymbol{\Phi}_t\boldsymbol{\Phi}_t^\top\right)^{-1} \boldsymbol{\Phi}_t\mathbf{y}_t, \tag{1.4.60}$$

where the i'th component of $\mathbf{y}_t$, $y_i = \sum_{j=i}^{t} \gamma^{j-i} r_j$, the least squares estimate for $\mathbf{w}$, when regressed on the components of $\mathbf{y}_t$. More specifically, $\hat{\mathbf{w}}_{t+1}$ is the solution of the least squares problem

$$\min_{\mathbf{w}} \left\|\mathbf{y}_t - \boldsymbol{\Phi}_t^\top \mathbf{w}\right\|^2,$$

where now

$$\boldsymbol{\Phi}_t = \left[\boldsymbol{\phi}(\mathbf{x}_0), \ldots, \boldsymbol{\phi}(\mathbf{x}_t)\right].$$

As an alternative to the batch Algorithm 8, Eq. 1.4.59 may be solved recursively by application of the matrix inversion lemma (Scharf, 1991), resulting in the online updates:

$$\begin{aligned} \hat{\mathbf{w}}_t &= \hat{\mathbf{w}}_{t-1} - \mathbf{q}_t\left(r_{t-1} + \gamma\hat{V}_{t-1}(\mathbf{x}_t) - \hat{V}_{t-1}(\mathbf{x}_{t-1})\right), \quad \text{and} \\ \mathbf{M}_t &= \mathbf{M}_{t-1} - \mathbf{q}_t\left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)\right)^\top \mathbf{M}_{t-1}, \\ \text{where } \mathbf{q}_t &= \frac{\mathbf{M}_{t-1}\mathbf{z}_t}{1 + \left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)\right)^\top \mathbf{M}_{t-1}\mathbf{z}_t}. \end{aligned} \tag{1.4.61}$$

For initialization, set $\hat{\mathbf{w}}_0 = \mathbf{0}$ and $\mathbf{M}_0 \propto \mathbf{I}$. The pseudocode is provided in Algorithm 9. Note that, the term $r_{t-1} - \left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)\right)^\top \hat{\mathbf{w}} = r_{t-1} + \gamma\hat{V}_{t-1}(\mathbf{x}_t) - \hat{V}_{t-1}(\mathbf{x}_{t-1})$ is the temporal difference at time t.

Algorithm 9 The Recursive LSTD(λ) Algorithm

```
Parameters: $\lambda$, $\sigma$
Initialize $\hat{\mathbf{w}}_0 = \mathbf{0}$, $\mathbf{z}_0 = \boldsymbol{\phi}(\mathbf{x}_0)$, $\mathbf{M}_0 = \sigma^2\mathbf{I}$
for $t = 1, 2, \ldots$
  observe $\mathbf{x}_{t-1}$, $r_{t-1}$, $\mathbf{x}_t$
  $\mathbf{q}_t = \left(1 + \left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)\right)^\top \mathbf{M}_{t-1}\mathbf{z}_{t-1}\right)^{-1} \mathbf{M}_{t-1}\mathbf{z}_{t-1}$
  $\hat{\mathbf{w}}_t = \hat{\mathbf{w}}_{t-1} + \mathbf{q}_t \left(r_{t-1} - \left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)\right)^\top \hat{\mathbf{w}}_{t-1}\right)$
  $\mathbf{M}_t = \mathbf{M}_{t-1} - \mathbf{q}_t \left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)\right)^\top \mathbf{M}_{t-1}$
  $\mathbf{z}_t = \gamma\lambda\mathbf{z}_{t-1} + \boldsymbol{\phi}(\mathbf{x}_t)$
end for
return $\hat{\mathbf{w}}_t$
```

The convergence results of Tsitsiklis and Van Roy (1996), pertaining to both TD(λ) and LSTD(λ), deal only with the behavior of the approximation $\hat{V}_t$ in the asymptotic regime, and do not allow for an interpretation of $\hat{V}_t$ for finite t. Moreover, for $\lambda < 1$, the limit of convergence is in general different from the projection of the true value function on the hypothesis space Θ. This means that both TD(λ) and LSTD(λ) with $\lambda < 1$ may produce asymptotically inconsistent value estimates (see Bertsekas & Tsitsiklis, 1996 Chapter 6.3.2 for an example).

Chapter 2

On-Line Kernel Sparsification

Summary: This chapter lays some of the groundwork for subsequent chapters by introducing a simple kernel sparsification method. This method may be used to construct online a dictionary of representative states, in terms of which all kernel expansions may be expressed to a given accuracy. The proposed method leverages the fact that, although the dimensionality of the feature space induced by the kernel transformation may be infinite, the image of the input space under this transformation has a finite volume. The sparsification method is first described as resulting from an *approximate linear dependence* condition in feature space. Then, theoretical results pertaining to the size of the dictionary and the quality of the resulting approximation are presented. Next, a relation between our method and kernel principal component analysis is revealed. Finally, we discuss our method in the context of existing sparsification techniques. An earlier account of this method was published in Engel et al. (2002).

2.1 Introduction

Sparse solutions for kernel algorithms are desirable for two main reasons. First, instead of storing information pertaining to the entire history of training instances, sparsity allows the solution to be stored in memory in a compact form and to be easily used later. The sparser is the solution of a kernel algorithm, the less time and memory are consumed in both the learning and the operation (query) phases of the kernel machine. Second, sparsity is related to generalization ability, and is considered a desirable property in learning algorithms (see, e.g., Schölkopf & Smola, 2002; Herbrich, 2002) as well as in signal processing (e.g., Mallat, 1998). The ability of a kernel machine to correctly generalize from its learned experience to new data can be shown to improve as the number of its free variables decreases (as long as the training error does not increase), which means that sparsification may be used as a regularization instrument.

In the classic SVM framework sparsity is achieved by making use of error-tolerant cost functions in conjunction with an additional regularization term encouraging "flat" solutions by penalizing the squared norm of the weight vector (Schölkopf & Smola, 2002). For SV classification, it has been shown in Vapnik (1995) that the expected number of of SVs is bounded below by $(t-1)E(p_{err})$ where t is the number of training samples and $E(p_{err})$ is the expectation of the error probability on a test sample. In spite of claims to the contrary (Syed et al., 1999), it has been shown, both theoretically and empirically (Burges, 1996; Downs et al., 2001), that the solutions provided by SVMs are often not maximally sparse. It also stands to reason that once a sufficiently large training set has been learned, any additional training samples would not contain much new information and therefore should not cause a linear increase in the size of the solution.

We take a somewhat different approach toward sparsification, which is based on the following observation: Although the dimension of the feature space $\boldsymbol{\phi}(\mathcal{X})$ is usually very high, or even infinite; the *effective* dimensionality of the manifold spanned by the training feature vectors may be significantly lower. Consequently, the solution to any optimization problem conforming to the conditions required by the Representer Theorem (1.2.1) may be expressed, to arbitrary accuracy, by a set of linearly independent feature vectors that *approximately span* this manifold.

The output of a learning algorithm is a hypothesis, or predictor, which is a function of the training set. In the online setting the samples in the training set are temporally ordered and are indexed by the (discrete) time index in which they were observed. By the time t the training sample is therefore $((\mathbf{x}_i, y_i))_{i=1}^t$. In the case of kernel methods, the predictor at time t is typically of the form specified by the Representer Theorem, given by Eq. 1.2.2. Mercer's Theorem (1.2.2) shows that a predictor of this form is in fact a linear predictor in the feature space $\boldsymbol{\phi}(\mathcal{X})$ induced by the kernel:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{t} \alpha_i \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}) \rangle = \sum_{i=1}^{t} \alpha_i k(\mathbf{x}_i, \mathbf{x}) = \boldsymbol{\alpha}_t^\top \mathbf{k}_t(\mathbf{x}), \tag{2.1.1}$$

where we used the definitions

$$\mathbf{k}_t = (k(\mathbf{x}_1, \mathbf{x}), \ldots, k(\mathbf{x}_t, \mathbf{x}))^\top, \quad \text{and } \boldsymbol{\alpha}_t = (\alpha_1, \ldots, \alpha_t)^\top. \tag{2.1.2}$$

Suppose that the point $\mathbf{x}_t$ satisfies $\boldsymbol{\phi}(\mathbf{x}_t) = \sum_{i=1}^{t-1} a_i \boldsymbol{\phi}(\mathbf{x}_i)$, then in the predictor $\hat{f}$ at time t (and any subsequent time) there is no need to have a non-zero coefficient for $\boldsymbol{\phi}(\mathbf{x}_t)$ as it can be absorbed in the other terms. This idea would work when $\boldsymbol{\phi}(\mathcal{X})$ is low dimensional, or if the data happens to belong to a low dimensional subspace of the feature space. However, for many kernels $\boldsymbol{\phi}(\mathcal{X})$ is high dimensional, or even infinite dimensional. For example, if k is a Gaussian kernel then $\dim(\boldsymbol{\phi}(\mathcal{X})) = \infty$ (e.g., Schölkopf & Smola, 2002). In this case, unless $\mathbf{x}_t = \mathbf{x}_i$ for some $i < t$, the feature vector $\boldsymbol{\phi}(\mathbf{x}_t)$ will be linearly independent of $\{\boldsymbol{\phi}(\mathbf{x}_i)\}_{i=1}^{t-1}$. The solution we propose is to relax the requirement that $\boldsymbol{\phi}(\mathbf{x}_t)$ can be exactly written as a sum of $\{\boldsymbol{\phi}(\mathbf{x}_i)\}_{i=1}^{t-1}$ and to consider instead *approximate* linear dependency. Given a new sample $\mathbf{x}_t$, we will distinguish between two cases. In the first case, the sample is approximately dependent on past samples. Such a sample will be considered only through its effect on the predictor $\hat{f}$, but will not entail adding a term to the kernel expansion (2.1.1). A sample, the feature vector of which is not approximately dependent on past samples, will add an extra term to this expansion, and will therefore be admitted into a "dictionary" of representative samples. This essentially amounts an online projection of the feature vectors encountered during training to a low dimensional subspace spanned by a subset of the training samples, namely, those found in the dictionary.

In the next section we describe the online sparsification algorithm in detail. Section 2.3 states, proves and discusses some desirable theoretical properties of the algorithm, while Section 2.4 illuminates its connection to kernel PCA. In Section 2.5 we overview current sparsification methods and compare our method to them.

2.2 The Sparsification Procedure

The online prediction setup assumes we sequentially sample a stream of input/output pairs $((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots)$, $\mathbf{x}_i \in \mathcal{X}$, $y_i \in \mathbb{R}$. Assume that at time step t, after having observed $t-1$ training samples $\{\mathbf{x}_i\}_{i=1}^{t-1}$, we have collected a dictionary consisting of a subset of the training samples $\mathcal{D}_{t-1} = \{\tilde{\mathbf{x}}_j\}_{j=1}^{m_{t-1}}$, where by construction $\{\boldsymbol{\phi}(\tilde{\mathbf{x}}_j)\}_{j=1}^{m_{t-1}}$ are linearly independent feature vectors, and $m_t = |\mathcal{D}_t|$. Now we are presented with a new sample $\mathbf{x}_t$. We test whether $\boldsymbol{\phi}(\mathbf{x}_t)$ is approximately linearly dependent on the dictionary vectors. If not, we add it to the dictionary. Consequently, all training samples up to and including time t can be approximated as linear combinations of the vectors in $\mathcal{D}_t$.

To avoid adding the training sample $\mathbf{x}_t$ to the dictionary, we need to find coefficients $\boldsymbol{a} = (a_1, \ldots, a_{m_{t-1}})^\top$ satisfying the approximate linear dependence (ALD) condition

$$\left\| \sum_{j=1}^{m_{t-1}} a_j \boldsymbol{\phi}(\tilde{\mathbf{x}}_j) - \boldsymbol{\phi}(\mathbf{x}_t) \right\|^2 \le \nu,$$

where ν is a positive threshold parameter determining the level of accuracy of the approximation (indirectly, ν also controls the level of sparsity intended to be achieved). Finding the optimal coefficient vector $\boldsymbol{a}_t$ may be done by solving the least-squares problem,

$$\delta_t \overset{\text{def}}{=} \min_{\boldsymbol{a}} \left\| \sum_{j=1}^{m_{t-1}} a_j \boldsymbol{\phi}(\tilde{\mathbf{x}}_j) - \boldsymbol{\phi}(\mathbf{x}_t) \right\|^2 . \tag{2.2.3}$$

If the ALD condition in (2.2.3) holds, $\boldsymbol{\phi}(\mathbf{x}_t)$ can be approximated within a squared error ν by some linear combination of current dictionary members. Performing the minimization in (2.2.3) we can simultaneously check whether this condition may be satisfied and obtain the optimal coefficient vector $\boldsymbol{a}_t$ satisfying it with minimal squared error. Expanding (2.2.3) we note that it may be written entirely in terms of inner products (in $\boldsymbol{\phi}(\mathcal{X})$) between feature vectors $\boldsymbol{\phi}(\cdot)$,

$$\delta_t = \min_{\boldsymbol{a}} \Big\{ \sum_{i,j=1}^{m_{t-1}} a_i a_j \big\langle \boldsymbol{\phi}(\tilde{\mathbf{x}}_i), \boldsymbol{\phi}(\tilde{\mathbf{x}}_j) \big\rangle - 2 \sum_{j=1}^{m_{t-1}} a_j \big\langle \boldsymbol{\phi}(\tilde{\mathbf{x}}_j), \boldsymbol{\phi}(\mathbf{x}_t) \big\rangle + \big\langle \boldsymbol{\phi}(\mathbf{x}_t), \boldsymbol{\phi}(\mathbf{x}_t) \big\rangle \Big\}.$$

We employ the kernel trick (1.2.3) by replacing inner products between feature space vectors with the kernel function. We therefore make the substitution $\big\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}') \big\rangle = k(\mathbf{x}, \mathbf{x}')$, obtaining

$$\begin{aligned} \delta_t &= \min_{\boldsymbol{a}} \Big\{ \sum_{i,j=1}^{m_{t-1}} a_i a_j k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) - 2 \sum_{j=1}^{m_{t-1}} a_j k(\tilde{\mathbf{x}}_j, \mathbf{x}_t) + k(\mathbf{x}_t, \mathbf{x}_t) \Big\} \\ &= \min_{\boldsymbol{a}} \Big\{ \boldsymbol{a}^\top \tilde{\mathbf{K}}_{t-1} \boldsymbol{a} - 2 \boldsymbol{a}^\top \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) + k_{tt} \Big\} , \end{aligned} \tag{2.2.4}$$

where

$$\Big[\tilde{\mathbf{K}}_{t-1} \Big]_{i,j} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j), \quad \Big(\tilde{\mathbf{k}}_{t-1}(\mathbf{x}) \Big)_i = k(\tilde{\mathbf{x}}_i, \mathbf{x}), \quad k_{tt} = k(\mathbf{x}_t, \mathbf{x}_t), \tag{2.2.5}$$

with $i, j = 1, \ldots, m_{t-1}$. The optimization problem in (2.2.4) is a simple quadratic problem, which may be solved analytically, yielding the optimal vector of approximation coefficients $\boldsymbol{a}_t$, and allowing the evaluation of the ALD condition:

$$\boldsymbol{a}_t = \tilde{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) , \quad \delta_t = k_{tt} - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top \boldsymbol{a}_t \le \nu. \tag{2.2.6}$$

If $\delta_t > \nu$ then we must expand the current dictionary by augmenting it with $\mathbf{x}_t$: $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$. Using the expanded dictionary, $\boldsymbol{\phi}(\mathbf{x}_t)$ may be exactly represented (by itself) and δ_t is therefore set to zero.

Consequently, for every time-step i up to t we have

$$\boldsymbol{\phi}(\mathbf{x}_i) = \sum_{j=1}^{m_i} a_{i,j} \boldsymbol{\phi}(\tilde{\mathbf{x}}_j) + \boldsymbol{\phi}_i^{res}, \quad (\|\boldsymbol{\phi}_i^{res}\|^2 \le \nu), \tag{2.2.7}$$

with $\boldsymbol{\phi}_i^{res}$ denoting the i'th approximation residual vector. Let us define the matrix of approximation coefficients $\mathbf{A}_t$ by

$$[\mathbf{A}_t]_{i,j} = a_{i,j}, \quad \text{where } i \in \{1,\ldots,t\} \text{ and } j \in \{1,\ldots,m_i\}.$$

Due to the sequential nature of the algorithm, for $j > m_i$, $[\mathbf{A}_t]_{i,j} = 0$, as $\boldsymbol{\phi}(\mathbf{x}_i)$ is approximated using only dictionary terms that were admitted up to time-step i.

Choosing ν to be sufficiently small we can make the approximation error in $\boldsymbol{\phi}(\mathbf{x}_i) \approx \sum_{j=1}^{m_i} a_{i,j}\boldsymbol{\phi}(\tilde{\mathbf{x}}_j)$ arbitrarily small, and by setting $\nu = 0$ this, and subsequent approximations become exact. The full kernel (or Gram) matrix, $\mathbf{K}_t$, is defined by

$$[\mathbf{K}_t]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j) \text{ with } i,j = 1,\ldots,t.$$

In order to establish a sparse (or rather, low rank) approximation of the kernel matrix $\mathbf{K}_t$, let us define the matrix

$$\boldsymbol{\Phi}_t = \left[\boldsymbol{\phi}(\mathbf{x}_1), \quad \ldots \quad, \boldsymbol{\phi}(\mathbf{x}_t)\right].$$

Then, since $\boldsymbol{\phi}(\mathbf{x}_i) \approx \sum_{j=1}^{m_t} a_{i,j}\boldsymbol{\phi}(\tilde{\mathbf{x}}_j)$, we have

$$\mathbf{K}_t = \boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t \approx \mathbf{A}_t \tilde{\mathbf{K}}_t \mathbf{A}_t^\top. \tag{2.2.8}$$

Similarly, since $\mathbf{k}_t(\mathbf{x}) = \boldsymbol{\Phi}^\top \boldsymbol{\phi}(\mathbf{x})$ (see Eq. 2.1.1 and 2.1.2), we obtain

$$\mathbf{k}_t(\mathbf{x}) \approx \mathbf{A}_t \tilde{\mathbf{k}}_t(\mathbf{x}). \tag{2.2.9}$$

In practice, we will freely make the substitutions (2.2.8) and (2.2.9), with the understanding that the resulting expressions are approximate whenever $\nu > 0$.

Using the well known partitioned matrix inversion formula it is possible to derive a recursive formula for $\tilde{\mathbf{K}}_t^{-1}$ (see Appendix D.4):

$$\tilde{\mathbf{K}}_t = \begin{bmatrix} \tilde{\mathbf{K}}_{t-1} & \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) \\ \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top & k_{tt} \end{bmatrix} \quad \Rightarrow \quad \tilde{\mathbf{K}}_t^{-1} = \frac{1}{\delta_t} \begin{bmatrix} \delta_t \tilde{\mathbf{K}}_{t-1}^{-1} + \boldsymbol{a}_t \boldsymbol{a}_t^\top & -\boldsymbol{a}_t \\ -\boldsymbol{a}_t^\top & 1 \end{bmatrix}, \tag{2.2.10}$$

where $\boldsymbol{a}_t = \tilde{\mathbf{K}}_{t-1}^{-1}\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$, and δ_t given by Eq. 2.2.4. After performing this update $\boldsymbol{a}_t$ is set to $(0,0,\ldots,1)^t$, as the new dictionary $\mathcal{D}_t$ now contains $\mathbf{x}_t$ as its most recent member, and $\boldsymbol{\phi}(\mathbf{x}_t)$ may therefore be perfectly reproduced. Table 10 provides a pseudocode sketch of our sparsification algorithm. The output of the algorithm, as it is described here, is the dictionary $\mathcal{D}_t$. At little extra cost, this algorithm may also output any of the following: The dictionary kernel matrix $\tilde{\mathbf{K}}_t$, its inverse $\tilde{\mathbf{K}}_t^{-1}$ or the matrix of approximation coefficients $\mathbf{A}_t = [\boldsymbol{a}_1,\ldots,\boldsymbol{a}_t]^\top$. In the sequel, when this algorithm is used, its updates will usually be interleaved with the update equations of a master algorithm. It is therefore instructive to view it first in isolation, as shown in Table 10. Note that the computational cost for the t'th sample is bounded by

Algorithm 10 Online Kernel Sparsification Algorithm

```
                                                            Cost
Parameter: $\nu$
Initialize: $\mathcal{D}_1 = \{\mathbf{x}_1\}$, $m = 1$, $\tilde{\mathbf{K}}_1^{-1} = [1/k_{11}]$        $O(1)$
for $t = 2, 3 \ldots$
  Get new sample: $\mathbf{x}_t$
  Compute $\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$  (2.1.2)                     $O(m)$
  ALD test:
    $\boldsymbol{a}_t = \tilde{\mathbf{K}}_{t-1}^{-1}\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$                          $O(m^2)$
    $\delta_t = k_{tt} - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top \boldsymbol{a}_t$                    $O(m)$
    if $\delta_t > \nu$   % add $\mathbf{x}_t$ to dictionary                 $O(1)$
      $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$                              $O(1)$
      Compute $\tilde{\mathbf{K}}_t^{-1}$ (2.2.10)                           $O(m^2)$
      $\boldsymbol{a}_t = (\mathbf{0}, \ldots, 1)^\top$                                  $O(m)$
      $m := m + 1$                                           $O(1)$
    else   % dictionary unchanged
      $\mathcal{D}_t = \mathcal{D}_{t-1}$
    end if
  end for
Output: $\mathcal{D}_t$
```

$O(m_t^2)$. Assuming we are able to bound the size of the final dictionary, this property allows the algorithm described here to meet the on-line and real-time requirements.

It is illuminating to consider a Gaussian process view of this sparsification method. Assume that we have a random process, F, indexed by $\mathbf{x} \in \mathcal{X}$, sampled at a sequence of points $\mathbf{x}_1, \ldots, \mathbf{x}_t$. Define $F_t = (F(\mathbf{x}_1), \ldots, F(\mathbf{x}_t))^\top$, and let F be distributed according to a Gaussian prior distribution with a zero-mean and a covariance function $k(\cdot, \cdot)$, making it a Gaussian process. Hence the prior over F_t is $\mathcal{N}(\mathbf{0}, \mathbf{K}_t)$. Suppose we knew the values of the variables in F_{t-1}. What would then be the conditional mean and variance of $F(\mathbf{x}_t)$, given F_{t-1}? A priori we have

$$\begin{pmatrix} F_{t-1} \\ F(\mathbf{x}_t) \end{pmatrix} \sim \mathcal{N}\left\{ \begin{pmatrix} \mathbf{0} \\ 0 \end{pmatrix}, \begin{bmatrix} \mathbf{K}_{t-1} & \mathbf{k}_{t-1}(\mathbf{x}_t) \\ \mathbf{k}_{t-1}(\mathbf{x}_t)^\top & k(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix} \right\}.$$

The Gauss-Markov Theorem (1.3.1) may now be invoked to obtain the moments of the conditional (Gaussian) distribution:

$$\begin{aligned} \mathbf{E}\left[F(\mathbf{x}_t)|F_{t-1}\right] &= \mathbf{k}_{t-1}(\mathbf{x}_t)^\top \mathbf{K}_{t-1}^{-1} F_{t-1}, \\ \mathbf{Var}\left[F(\mathbf{x}_t)|F_{t-1}\right] &= k_{tt} - \mathbf{k}_{t-1}(\mathbf{x}_t)^\top \mathbf{K}_{t-1}^{-1} \mathbf{k}_{t-1}(\mathbf{x}_t). \end{aligned}$$

These expressions provide us with an alternative probabilistic view of our sparsification algorithm. The δ_t criterion, to which we previously assigned a geometrical interpretation (Eq. 2.2.6), is now the posterior variance of $F(\mathbf{x}_t)$, conditioned on the values of F at the dictionary points. If *knowing* $F(\tilde{\mathbf{x}}_1), \ldots, F(\tilde{\mathbf{x}}_{|\mathcal{D}|})$ confers enough information on the value of $F(\mathbf{x}_t)$, this conditional variance, δ_t, will be low (lower than ν), and we will not add $\mathbf{x}_t$ to the dictionary, and vice versa. The vector of approximation coefficient $\boldsymbol{a}_t = \tilde{\mathbf{K}}_{t-1}^{-1}\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$, which was derived above (2.2.6) as the solution of a least-squares problem, is now the vector of coefficients multiplying the components of $F(\tilde{\mathbf{x}}_1), \ldots, F(\tilde{\mathbf{x}}_{|\mathcal{D}|})$ for computing the conditional mean at $\mathbf{x}_t$.

2.3 Properties of the Sparsification Method

Let us now study some of the properties of the sparsification method. We first show that under mild conditions on the data and the kernel function, the dictionary is finite. We then discuss the question of how good the approximation really is, by showing that the sensitivity parameter ν controls how well the true kernel matrix is approximated.

2.3.1 A Bound on the Dictionary Size

Recall that the data points $\mathbf{x}_1, \mathbf{x}_2, \ldots$ are assumed to belong to some input set $\mathcal{X}$. Let $\boldsymbol{\phi}(\mathcal{X}) = \{\boldsymbol{\phi}(\mathbf{x}) : \mathbf{x} \in \mathcal{X}\}$. The following theorem holds regardless of the dimensionality of $\boldsymbol{\phi}(\mathcal{X})$, and essentially says that as long as $\mathcal{X}$ is compact, the set of dictionary vectors is finite.

Theorem 2.3.1. *Assume that $\boldsymbol{\phi}(\mathcal{X})$ is a compact set. Then for any training sequence $\{\mathbf{x}_i\}_{i=1}^{\infty}$ with $\mathbf{x}_i \in \mathcal{X}$ for all i, and for any $\nu > 0$, the size of the dictionary $|\mathcal{D}_t|$ is finite, and is bounded from above by the $\sqrt{\nu}$-packing number of $\boldsymbol{\phi}(\mathcal{X})$, for the ℓ_2 norm, $\mathcal{N}^p_{\sqrt{\nu}}(\boldsymbol{\phi}(\mathcal{X}))$.*

Proof The compactness of $\boldsymbol{\phi}(\mathcal{X})$ implies that for any $\varepsilon > 0$ a finite ε-cover exists. Recall that the covering number is finite if, and only if, the packing number is finite (e.g., Anthony & Bartlett, 1999), implying that the maximal number of ε-separated points in $\boldsymbol{\phi}(\mathcal{X})$ is finite, and given by $\mathcal{N}^p_{\varepsilon}(\boldsymbol{\phi}(\mathcal{X}))$. Next, observe that by construction of the algorithm, any two points $\boldsymbol{\phi}(\tilde{\mathbf{x}}_i)$ and $\boldsymbol{\phi}(\tilde{\mathbf{x}}_j)$ in the dictionary $\mathcal{D}_t$ obey $\|\boldsymbol{\phi}(\tilde{\mathbf{x}}_i) - \boldsymbol{\phi}(\tilde{\mathbf{x}}_j)\|_2 > \sqrt{\nu}$, i.e. are $\sqrt{\nu}$-separated. Therefore, there can be at most $\mathcal{N}^p_{\sqrt{\nu}}(\boldsymbol{\phi}(\mathcal{X}))$ members in the dictionary. □

Theorem 2.3.1 implies that after an initial period, the computational cost per time-step of the algorithm becomes independent of time and depends only on the dictionary size. It is this property that makes our framework practical for online real-time learning. Precise values for the size of the cover, and thus for the maximal size of the dictionary, can be obtained by making further assumptions on the domain $\mathcal{X}$ and the kernel k. In particular, well known bounds on the covering numbers, which depend on the behavior of the eigenvalues of the integral operator (1.2.3) may be applied (König, 1986).

A particular case, which is often of practical interest, is when $\mathcal{X}$ is a compact subset of a normed vector space (i.e. a Banach space; for instance, a compact subset of $\mathbb{R}^n$ with the ℓ_2 norm). In this case we may use the following proposition to conclude that $\boldsymbol{\phi}(\mathcal{X})$ is compact and therefore, according to Theorem 2.3.1, that the size of the dictionary is finite.

Proposition 2.3.2. *Assume that (i) k is a continuous Mercer kernel and (ii) $\mathcal{X}$ is a compact subset of a Banach space. Then $\boldsymbol{\phi}(\mathcal{X})$ is compact.*

Proof First, we claim that $\boldsymbol{\phi}$ is continuous. Given a sequence $\mathbf{z}_1, \mathbf{z}_2, \ldots$ of points in $\mathcal{X}$ such that $\mathbf{z}_i \to \mathbf{z}^*$ we have that $\|\boldsymbol{\phi}(\mathbf{z}_i) - \boldsymbol{\phi}(\mathbf{z}^*)\|_2^2 = \langle \boldsymbol{\phi}(\mathbf{z}_i), \boldsymbol{\phi}(\mathbf{z}_i)\rangle + \langle \boldsymbol{\phi}(\mathbf{z}^*), \boldsymbol{\phi}(\mathbf{z}^*)\rangle - \langle \boldsymbol{\phi}(\mathbf{z}_i), \boldsymbol{\phi}(\mathbf{z}^*)\rangle - \langle \boldsymbol{\phi}(\mathbf{z}^*), \boldsymbol{\phi}(\mathbf{z}_i)\rangle$. Writing this in terms of kernels we have $\|\boldsymbol{\phi}(\mathbf{z}_i) - \boldsymbol{\phi}(\mathbf{z}^*)\|_2^2 = k(\mathbf{z}_i, \mathbf{z}_i) + k(\mathbf{z}^*, \mathbf{z}^*) - k(\mathbf{z}_i, \mathbf{z}^*) - k(\mathbf{z}^*, \mathbf{z}_i)$. Since k itself is continuous we have that $\|\boldsymbol{\phi}(\mathbf{z}_i) - \boldsymbol{\phi}(\mathbf{z}^*)\|_2^2 \to 0$, so $\boldsymbol{\phi}$ is continuous.
Since $\mathcal{X}$ is compact and $\boldsymbol{\phi}$ is continuous, we conclude that $\boldsymbol{\phi}(\mathcal{X})$ is compact. □

2.3.2 Quality of Approximation

After establishing that the dictionary is finite we turn to study the effect of the approximation level ν on the approximation of the kernel matrix $\mathbf{K}$. Since the vector of predictions at the training data points may be written as

$$\begin{pmatrix} \hat{f}(\mathbf{x}_1) \\ \vdots \\ \hat{f}(\mathbf{x}_t) \end{pmatrix} = \mathbf{K}_t \boldsymbol{\alpha}_t,$$

a bound on the error in approximating $\mathbf{K}_t$ may be used to derive a bound on the error in $\hat{f}$ at those points. It could be argued that a more direct strategy for sparsification, aimed at directly minimizing the error in $\hat{f}$, would be a better choice. Such strategies were indeed explored elsewhere, and we discuss some of them in detail in section 2.5. Our choice of an *unsupervised* sparsification criterion, which does not take into account the measured observations $\{y_i\}_{i=1}^t$, is due to the non-stationary nature of the learning problems that we intend to apply our methods to, namely, reinforcement learning problems. In such problems, as policies change, so do the distributions of the measured variables (in RL these are the rewards). Since we want to maintain a single dictionary, rather than a constructing a different one for each policy, we opt for an unsupervised criterion.

In order to simplify the analysis we first consider an *off-line* version of the algorithm on a finite data set of size t. In this case, the dictionary is first constructed in the usual manner, and then the optimal expansion coefficient matrix $\mathbf{A}$ is computed for the entire data set at once. We use essentially the same notation as before except that now every quantity depending on an incomplete dictionary is redefined to depend on the entire dictionary. In order to remind us of this change we omit the time index in each such quantity. For instance, $\mathcal{D}$, m, and $\mathbf{K}$ denote the dictionary, its size, and the full kernel matrix, respectively.

Defining the matrices $\boldsymbol{\Phi} = [\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_t)]$, $\tilde{\boldsymbol{\Phi}} = [\boldsymbol{\phi}(\tilde{\mathbf{x}}_1), \ldots, \boldsymbol{\phi}(\tilde{\mathbf{x}}_m)]$, and $\boldsymbol{\Phi}^{res} = [\boldsymbol{\phi}_1^{res}, \ldots, \boldsymbol{\phi}_t^{res}]$, we may write (2.2.7) for all samples concisely as

$$\boldsymbol{\Phi} = \tilde{\boldsymbol{\Phi}} \mathbf{A}^\top + \boldsymbol{\Phi}^{res}. \tag{2.3.11}$$

As before, the optimal expansion coefficients for the sample $\mathbf{x}_i$ are found by $\boldsymbol{a}_i = \tilde{\mathbf{K}}^{-1}\tilde{\mathbf{k}}(\mathbf{x}_i)$, and $[\mathbf{A}]_{i,j} = a_{i,j}$. Premultiplying (2.3.11) by its transpose we get

$$\mathbf{K} = \mathbf{A}\tilde{\mathbf{K}}\mathbf{A}^\top + \boldsymbol{\Phi}^{res\top}\boldsymbol{\Phi}^{res}. \tag{2.3.12}$$

The cross term $\mathbf{A}\tilde{\boldsymbol{\Phi}}^\top \boldsymbol{\Phi}^{res}$ and its transpose vanish due to the ℓ_2-orthogonality of the residuals to the subspace spanned by the dictionary vectors. Defining the residual kernel matrix by $\mathbf{R} = \mathbf{K} - \mathbf{A}\tilde{\mathbf{K}}\mathbf{A}^\top$ it can be easily seen that, in both the online and the off-line cases, $(\operatorname{diag}\mathbf{R})_i = \delta_i$. Further, in the off-line case $\mathbf{R} = \boldsymbol{\Phi}^{res\top}\boldsymbol{\Phi}^{res}$ and is therefore positive semi-definite. As a consequence, in the off-line case we may bound the norm of $\mathbf{R}$, thus justifying our approximation. Recall that for a matrix $\mathbf{R}$ the matrix 2-norm is defined as $\|\mathbf{R}\|_2 = \max_{\mathbf{u}:\|\mathbf{u}\|_2=1}\|\mathbf{R}\mathbf{u}\|_2$.

Proposition 2.3.3. *In the off-line case $\|\mathbf{R}\|_2 \leq t\nu$.*

Proof Let $\lambda_i^{\mathbf{R}}$ be the i-th eigenvalue of $\mathbf{R}$. We recall from linear algebra that $\|\mathbf{R}\|_2 = \max_i |\lambda_i^{\mathbf{R}}|$. Since $(\operatorname{diag}\mathbf{R})_i = \delta_i$, we have that $\sum_{i=1}^t \lambda_i^{\mathbf{R}} = \operatorname{tr}\mathbf{R} = \sum_{i=1}^t \delta_i$. Moreover, since $\mathbf{R}$ is positive semi-definite $\lambda_i^{\mathbf{R}} \geq 0$ for all i. Therefore $\|\mathbf{R}\|_2 = \max_i \lambda_i^{\mathbf{R}} \leq \sum_{i=1}^t \delta_i \leq t\nu$. □

As a corollary, a similar bound can be placed on the covariance of the residuals $\|\mathbf{C}^{res}\| = \frac{1}{t}\|\boldsymbol{\Phi}^{res}\boldsymbol{\Phi}^{res\top}\| = \max_i \lambda_i^{\mathbf{R}}/t \leq \sum_{i=1}^t \delta_i/t \leq \nu$.

Considering the online case, it is clear that once the dictionary stops growing, for the remaining samples the algorithm behaves exactly like its off-line counterpart. Using Theorem 2.3.1 we know that if $\boldsymbol{\phi}(\mathcal{X})$ is compact, for any $\nu > 0$ the dictionary is finite, and therefore a bound similar to that of Proposition 2.3.3 holds beyond some finite time. More specifically, in the online case we have $\|\mathbf{R}\|_2 \leq t\nu + B$, where B is a random positive constant accounting for the size of the residuals up to the point when the dictionary reaches its final size. The corresponding bound for the residual covariance is therefore $\|\mathbf{C}^{res}\| \leq \nu + \frac{B}{t}$.

2.4 Online Sparsification as Approximate PCA

The online sparsification method described above has close connections to kernel principal components analysis (PCA) (Schölkopf et al., 1998). PCA is the optimal unsupervised dimensionality reduction method, minimizing the mean squared error. PCA is widely used in signal processing, machine learning, psychology, data analysis and visualization, and other fields involving high dimensional data. PCA is often used to remove noise from data. In such applications the noisy data is projected onto the first principal directions (i.e. on the subspace spanned by the first eigenvectors of the data's covariance matrix, where the eigenvectors are ordered by non-increasing eigenvalues). This is done due to the, sometimes implicit, assumption that the variance in the remaining directions is due to uninformative noise. In Schölkopf et al. (1998) it was shown that solving the eigenvalue problem of the kernel (Gram) matrix $\mathbf{K}_t = \mathbf{\Phi}_t^\top \mathbf{\Phi}_t$, is essentially equivalent to performing PCA on the data in the feature space $\boldsymbol{\phi}(\mathcal{X})$, referred to as *kernel-PCA*. In this section we show that our sparsification method is essentially an approximate form of kernel-PCA.

Let us first describe the optimal dimensionality reduction procedure. Let the covariance matrix at time t, be $\mathbf{C}_t \overset{\text{def}}{=} \frac{1}{t}\mathbf{\Phi}_t\mathbf{\Phi}_t^\top$. $\mathbf{C}_t$ has (at most) t positive eigenvalues $\{\lambda_1, \ldots, \lambda_t\}$ and a corresponding orthonormal set of eigenvectors $\{\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_t\}$, forming an orthogonal basis for the subspace of $\boldsymbol{\phi}(\mathcal{X})$ that contains $\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_t)$. Let us define a projection operator $P_\mathcal{S}$ onto the span of a subset of these eigenvectors. For an arbitrary subset $\Psi_\mathcal{S} = \{\boldsymbol{\psi}_i : i \in \mathcal{S} \subseteq \{1, \ldots, t\}\}$, it is well known that projecting the training data onto the subspace spanned by the eigenvectors in $\Psi_\mathcal{S}$, entails a mean squared error of $\frac{1}{t}\sum_{i=1}^t \|(I - P_\mathcal{S})\boldsymbol{\phi}(x_i)\|^2 = \sum_{i \notin \mathcal{S}} \lambda_i$. An immediate consequence of this is that an optimal m dimensional projection, with respect to the mean squared error criterion, is one in which $\Psi_\mathcal{S}$ consists of the first m eigenvectors (i.e. $\mathcal{S} = \{1, \ldots, m\}$).

Similarly, we can also define a projection operator onto the span of the dictionary vectors found by our sparsification method. Let

$$P_\mathcal{D} = \tilde{\mathbf{\Phi}} \left(\tilde{\mathbf{\Phi}}^\top \tilde{\mathbf{\Phi}}\right)^{-1} \tilde{\mathbf{\Phi}}^\top \tag{2.4.13}$$

denote this projection operator. Using the ALD condition (2.2.6) we can place a bound on the mean squared error due to the projection $P_{\mathcal{D}}$:

$$\begin{aligned}\frac{1}{t}\sum_{i=1}^{t}\|\boldsymbol{\phi}(\mathbf{x}_i)-P_{\mathcal{D}}(\boldsymbol{\phi}(\mathbf{x}_i))\|^2 &= \frac{1}{t}\sum_{i=1}^{t}\|(I-P_{\mathcal{D}})\boldsymbol{\phi}(\mathbf{x}_i)\|^2\\ &= \frac{1}{t}\sum_{i=1}^{t}\delta_i\\ &\le ((t-m_t)/t)\nu\\ &\le \nu,\end{aligned}$$

where the first inequality is due to the fact that for the m_t dictionary vectors the error is zero. However, since the size of the dictionary is not known a priori, the second inequality provides a more useful bound.

In the preceding paragraph we showed that the mean squared error of the projection performed by the online sparsification procedure is bounded by ν. We now show that, for a sufficiently small ν, the projection performed by the sparsification method, essentially preserves all the "important" eigenvectors, which are used in the optimal PCA projection.

Theorem 2.4.1. *Let $P_{\mathcal{D}}$ be the projection operator defined in Eq. 2.4.13, for a dictionary $\mathcal{D}$ constructed using a threshold ν. The i'th normalized eigenvector $\boldsymbol{\psi}_i$ of the empirical covariance matrix $\mathbf{C} = \frac{1}{t}\boldsymbol{\Phi}\boldsymbol{\Phi}^\top$, with eigenvalue $\lambda_i > 0$, satisfies $\|P_{\mathcal{D}}\boldsymbol{\psi}_i\|^2 \ge 1-\nu/\lambda_i$.*

Proof Any $\boldsymbol{\psi}_i$ for which $\lambda_i > 0$ may be expanded in terms of the feature vectors corresponding to the data points. It is well known (e.g., Schölkopf et al., 1998) that every respective expansion coefficient vector is itself an eigenvector of the kernel matrix $\mathbf{K}$ with eigenvalue $t\lambda_i$. We may therefore write $\boldsymbol{\psi}_i = \frac{1}{\sqrt{t\lambda_i}}\boldsymbol{\Phi}\mathbf{u}_i$ where $\mathbf{K}\mathbf{u}_i = t\lambda_i\mathbf{u}_i$ and $\|\mathbf{u}_i\| = 1$. Substituting into $\|P_{\mathcal{D}}\boldsymbol{\psi}_i\|^2$ we have $\|P_{\mathcal{D}}\boldsymbol{\psi}_i\|^2 = \boldsymbol{\psi}_i^\top P_{\mathcal{D}}\boldsymbol{\psi}_i = \frac{1}{t\lambda_i}\mathbf{u}_i^\top\boldsymbol{\Phi}^\top P_{\mathcal{D}}\boldsymbol{\Phi}\mathbf{u}_i$. Recalling that $P_{\mathcal{D}}$ is the projection operator onto the span of the dictionary we have $P_{\mathcal{D}}\boldsymbol{\phi}(\mathbf{x}_i) = \tilde{\boldsymbol{\Phi}}\boldsymbol{a}_i$ and $P_{\mathcal{D}}\boldsymbol{\Phi} = \tilde{\boldsymbol{\Phi}}\mathbf{A}^\top$. Therefore, $\boldsymbol{\Phi}^\top P_{\mathcal{D}}\boldsymbol{\Phi} = \mathbf{A}\tilde{\mathbf{K}}\mathbf{A}^\top$ and

$$\begin{aligned}\|P_{\mathcal{D}}\boldsymbol{\psi}_i\|^2 &= \frac{1}{t\lambda_i}\mathbf{u}_i^\top\mathbf{A}\tilde{\mathbf{K}}\mathbf{A}^\top\mathbf{u}_i\\ &= \frac{1}{t\lambda_i}\mathbf{u}_i^\top(\mathbf{K}-\mathbf{R})\mathbf{u}_i\\ &= 1-\frac{1}{t\lambda_i}\mathbf{u}_i^\top\mathbf{R}\mathbf{u}_i\\ &\ge 1-\frac{\|\mathbf{R}\|_2}{t\lambda_i}\\ &\ge 1-\frac{\nu}{\lambda_i},\end{aligned}$$

where the last inequality is due to Proposition 2.3.3. □

Theorem 2.4.1 establishes the connection between our sparsification algorithm and kernel PCA, since it implies that eigenvectors whose eigenvalues are significantly larger than ν are projected almost in their entirety onto the span of the dictionary vectors, and the quality of the approximation improves linearly with the ratio λ_i/ν. In comparison, kernel PCA projects the data solely onto the span of the first few eigenvectors of $\mathbf{C}$. We are therefore led to regard our sparsification method as an approximate form of kernel PCA, with the caveat that our method does not diagonalize $\mathbf{C}$, and so cannot extract individual orthogonal features, as kernel PCA does. Computationally however, our method is significantly cheaper ($O(m^2)$ memory and $O(tm^2)$ time) than exact kernel PCA ($O(t^2)$ memory and $O(t^2p)$ time where p is the number of extracted components).

2.5 Comparison to Other Sparsification Schemes

Several approaches to sparsification of kernel-based solutions have been proposed in the literature. As already mentioned above, SVMs for classification and regression attempt to achieve sparsity by utilizing error insensitive cost functions. The solution produced by an SVM consists of a linear combination of kernel evaluations, one per training sample, where in some cases a (sometimes large) fraction of the combination coefficients vanish.

In SVR, and more generally in regularization networks (Evgeniou et al., 2000), sparsity is achieved by *elimination*. This means that, at the outset, these algorithms consider all training samples as potential contributing members of the expansion (3.1.1); and upon solving the optimization problem they eliminate those samples whose coefficients vanish. This approach has several major disadvantages. First, with a training set of t samples, during learning the SVM algorithm must maintain a matrix of size $t \times t$ and update a full set of t coefficients. This means that, even if the end result turns out to be very sparse, the training algorithm will not be able to take full advantage of this sparsity in terms of efficiency. As a consequence, even the current state-of-the art SVM algorithm scales super-linearly in t (Collobert & Bengio, 2001). Second, in SVMs the solution's sparsity depends on the level of noise in the training data; this effect is especially pronounced in the case of regression. Finally, SVM solutions are known to be non-maximally sparse. This is due to the special form of the SVM quadratic optimization problem, in which the constraints limit the level of sparsity attainable (Osuna & Girosi, 1998).

Shortly after the introduction of SVMs to the machine learning community it was realized by Burges (1996) that the solutions provided by SVMs, both for classification and regression, may often be made significantly sparser, without altering the resulting predictor/classifier. It was also shown in this paper, as well as in Burges and Schölkopf (1997), that additional sparsity may be attained by allowing small changes to be made in the SVM solution, with little or no degradation in generalization ability. Burges' idea is based on using a "reduced-set" of feature space vectors to approximate the original solution. In Burges' method the reduced-set of feature vectors, apart from its size, is virtually unconstrained and therefore the algorithmic complexity of finding reduced-set solutions is rather high, posing a major obstacle to the widespread use of this method. In Schölkopf et al. (1999) and Downs et al. (2001) it was suggested that restricting the reduced set to be a subset of the training samples would help alleviate the computational cost associated with the original reduced-set method. This was backed by empirical results on several problems including handwritten digit recognition. All of these reduced-set methods achieve sparsity by elimination, meaning that they are designed to be used as a post-processing stage, after a kernel solution is obtained from some main algorithm (e.g., SVM) whose level of sparsity is deemed unsatisfactory by the user. Ultimately, reduced-set methods are based on the identification of (approximate) linear dependencies between feature space vectors and their subsequent elimination. For a more thorough account of reduced-set methods see Chapter 18 of Schölkopf and Smola (2002).

An alternative approach is to obtain sparsity by *construction*. Here the algorithm starts with an empty representation, in which all coefficients vanish, and gradually adds samples. Constructive sparsification is normally used off-line (e.g., Vincent & Bengio, 2002), in which case the algorithm is free to choose any one of the training samples at each step of the construction process. The sample selected at each step is typically the one that maximizes the amount of increase (or decrease) its addition induces in some fitness (or error) criterion. Such greedy strategies are resorted to because of a general hardness result for the problem of finding the best subset of samples in a sparse approximation framework (Natarajan, 1995), along with some positive results concerning the convergence rates for sparse greedy algorithms (Natarajan, 1995; Zhang, 2003). Examples of such methods are Smola and Schölkopf (2000); Smola and Bartlett (2001); Zhang (2003); see also Chapter 10 of Schölkopf and Smola (2002). These methods are also closely related to the kernel Matching Pursuit algorithm (Vincent & Bengio, 2002).

Greedy methods represent the opposite extreme to reduced-set methods along the elimination-construction axis of sparsification methods. Another (orthogonal) dimension along which sparsification methods may be measured is the one differentiating between supervised and unsupervised sparsification. Supervised sparsification is geared toward optimizing a supervised error criterion (e.g., the mean-squared error in regression tasks), while unsupervised sparsification attempts to faithfully reproduce the the images of input samples in feature space. Examples for supervised sparsification are Burges (1996); Tipping (2001a); Smola and Bartlett (2001); Vincent and Bengio (2002), of which Tipping (2001a) is unique in that it aims at achieving sparsity by taking a Bayesian approach in which a prior favoring sparse solutions is employed [1]. In Smola and Bartlett (2001) a greedy sparsification method is suggested that is specific to Gaussian Process regression and is similar to kernel Matching Pursuit (Vincent & Bengio, 2002).

Examples of unsupervised sparsification are Smola and Schölkopf (2000); Williams and Seeger (2001); Fine and Scheinberg (2001); Tipping (2001b). In Smola and Schölkopf (2000) a randomized-greedy selection strategy is used to reduce the rank of the kernel matrix $\mathbf{K}$ while Williams and Seeger (2001) uses a purely random strategy based on the Nyström method to achieve the same goal. In Fine and Scheinberg (2001) the incomplete Cholesky factorization algorithm is used to yield yet another reduced-rank approximation to $\mathbf{K}$. Employing low-rank approximations to $\mathbf{K}$ is essentially equivalent to using low-dimensional approximations of the feature vectors corresponding to the training samples. As principal component analysis (PCA) is known to deliver the optimal unsupervised dimensionality reduction for the mean-squared reconstruction error criterion, it is therefore natural to turn to kernel PCA (Schölkopf et al., 1998) as a sparsification device. Indeed, many of the unsupervised methods mentioned above are closely related to kernel PCA. In Tipping (2001b) a sparse variant of kernel PCA is proposed, based on a Gaussian generative model. The general idea in both cases is to project the entire feature space onto the low dimensional manifold spanned by the first eigenvectors of the sample covariance in feature space, corresponding to the leading/non-zero eigenvalues.

While many of the sparsification methods discussed above are constructive in nature, progressively building increasingly richer representations with time, they are not applicable to the online setting. In this setting input samples are not randomly accessible, instead they are given as a temporal stream of data in which only one sample is observed at any given time. This imposes an additional constraint on any sparsification method that attempts to represent the entire training history using some representative sample. Namely, at any point in time the algorithm must decide whether to add the current sample to its representation, or discard it. Clearly, none of the methods described above is suitable for on-line applications. This problem of online sparsification is not as well studied in the kernel community, and is the one we address with our sparsification algorithm.

Our method is most closely related to a sparsification method used By Csató and Opper (2001); Csató and Opper (2002) in the context of learning with Gaussian

[1] It should be noted that support vector machines may also be cast within a probabilistic Bayesian framework, see Sollich (2002).

Processes (Gibbs & MacKay, 1997; Williams, 1999). Csató and Opper's method also incrementally constructs a dictionary of input samples on which all other data are projected (with the projection performed in the feature space $\phi(\mathcal{X})$). However, while in our method the criterion used to decide whether a sample should be added to the dictionary is based only on the distance (in $\phi(\mathcal{X})$) between the new sample and the span of previously stored dictionary samples, their method also takes into account the estimate of the regressor (or classifier) on the new point and its difference from the target value. Consequently, the dictionary constructed by their method depends on the function being estimated and on the sample noise, while our method disregards these completely. We defer further discussion of the differences between these two methods to Section 3.4.4 in the next chapter.

Chapter 3

Kernel Recursive Least Squares

Summary: Before approaching the daunting challenge of applying kernel methods to reinforcement learning problems, an intermediate goal is set – to derive kernel-based algorithms for regression, which may be used in the online setting. In this chapter we describe such an online kernel algorithm for nonlinear regression, which is based on the recursive least squares algorithm. We refer to this algorithm as *kernel-RLS*. The online sparsification method of the previous chapter plays a crucial role in this algorithm. After a short introduction we present our algorithm. We then prove a generalization bound, followed by an extensive suit of experiments, testing the algorithm on several estimation problems from machine learning and signal processing. We compare kernel-RLS with a related method for sparse Gaussian Process regression, and conclude with a discussion. This work was first published in Engel et al. (2004).

3.1 Introduction

The celebrated recursive least-squares (RLS) algorithm (e.g. Kailath et al., 2000; Haykin, 1996; Scharf, 1991) is a popular and practical algorithm used extensively in signal processing, communications and control. The algorithm is an efficient online method for finding linear predictors minimizing the mean squared error over the training data. We consider the classic system identification setup (e.g. Ljung, 1999), where we assume access to a recorded sequence of input and output samples

$$Z^t = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_t, y_t))$$

arising from some unknown source. In the classic regression (or function approximation) framework the input-output pairs $(\mathbf{x}_i, y_i)$ are assumed to be independently and identically distributed (IID) samples from some distribution $p(Y, \mathbf{X})$. In signal processing applications the inputs more typically consist of lagged values of the outputs y_i, as would be the case for autoregressive (AR) sources, and/or samples of some other signal (ARMA and MA models, respectively). In the prediction problem, one attempts to find the best predictor $\hat{y}_t = \hat{f}(\mathbf{x}_t)$ for y_t given $Z^{t-1} \cup \{\mathbf{x}_t\}$. In this context, one is often interested in online applications, where the predictor is updated following the arrival of each new sample. Online algorithms are useful in learning scenarios where input samples are observed sequentially, one at a time (e.g., data mining, time series prediction, reinforcement learning). In such cases there is a clear advantage to algorithms that do not need to relearn from scratch when new data arrive. In many of these applications there is an additional requirement for *real-time* operation, meaning that the algorithm's computational cost per time-step should be bounded by a constant independent of time, for it is assumed that new samples arrive at a roughly constant rate.

Standard approaches to the prediction problem usually assume a simple parametric form, e.g., $\hat{f}(\mathbf{x}) = \big\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \big\rangle$, where $\mathbf{w}$ is a vector of parameters and $\boldsymbol{\phi}$ is a *fixed, finite dimensional mapping.* In the classic least-squares approach, one then attempts to find the value of $\mathbf{w}$ that minimizes the squared error $\sum_{i=1}^{t} \big(y_i - \big\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_i) \big\rangle\big)^2$. The RLS algorithm is used to recursively solve this least-squares problem for $\mathbf{w}$. Given a new sample $(\mathbf{x}_t, y_t)$, the number of computations performed by RLS to compute a new minimum least-squares estimate of $\mathbf{w}$ is independent of t, making it suitable for real-time applications.

Kernel methods present an alternative to the parametric approach. Solutions attained by these methods are non-parametric in nature and are typically of the form

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{t} \alpha_i k(\mathbf{x}_i, \mathbf{x}) \ , \qquad (3.1.1)$$

where $\{\mathbf{x}_i\}_{i=1}^{t}$ are the training data points. The Representer Theorem (1.2.1) assures us that in most practical cases, we need not look any further than an expression of the form (3.1.1).

Since the number of of tunable parameters in kernel solutions equals the size of the training data-set, one must introduce some form of regularization, in order to control the capacity of the resulting predictors. For instance, in SVR regularization is attained by using the so-called "ε-insensitive" error tolerant cost function, in conjunction with an additional regularization (penalty) term encouraging "flat" solutions. The end effect of this form of regularization is that SVR solutions are typically sparse – meaning that many of the α_i variables vanish in the SVR solution (3.1.1). However, as discussed in Section 2.5 methods that achieve sparsity by elimination, like SVR, are not amenable to online implementations. In a nutshell, the major obstacles in applying kernel methods to online prediction problems are: (i) Many kernel methods require random/multiple access to training samples, (ii) their computational cost (both in time and space) is typically super-linear in the size of the training set, and (iii) their prediction (query) time often scales linearly with the training set size.

In Chapter 2 (and earlier, in Engel et al., 2002) we proposed a solution to this problem by an online constructive sparsification method based on sequentially admitting into the kernel representation only samples that cannot be approximately represented using linear combinations (in feature space) of previously admitted samples. In Engel et al. (2002) our sparsification method was used to construct an online SVR-like algorithm, whereas here we will use it to derive a nonparametric kernel-based version of the RLS algorithm. The kernel-RLS (KRLS) algorithm proposed here is capable of efficiently and recursively solving nonlinear least-squares prediction problems, and is therefore particularly useful in applications requiring online or real-time operation.

In the next section we derive the KRLS algorithm. We then derive a data-dependent generalization bound for KRLS. We proceed to test KRLS on several supervised learning and signal processing tasks. Next, we compare KRLS with a related method known as sparse Gaussian process regression. We close this chapter with a discussion.

3.2 The Kernel RLS Algorithm

The RLS algorithm is used to incrementally train a linear regression model, parameterized by a weight vector $\mathbf{w}$, of the form $\hat{f}(\mathbf{x}) = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle$, where, as before, $\boldsymbol{\phi}(\mathbf{x})$ is the feature vector associated with the input $\mathbf{x}$. We assume that a standard preprocessing step has been performed in order to absorb the bias term into the weight vector $\mathbf{w}$ (i.e. by redefining $\mathbf{w}$ as $(\mathbf{w}^\top, b)^\top$ and $\boldsymbol{\phi}$ as $(\boldsymbol{\phi}^\top, 1)^\top$); see Engel et al. (2002) for details. In the simplest form of the RLS algorithm we minimize at each time step t the sum of the squared errors:

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{t} \left(\hat{f}(\mathbf{x}_i) - y_i \right)^2 = \|\boldsymbol{\Phi}_t^\top \mathbf{w} - \mathbf{y}_t\|^2, \tag{3.2.2}$$

where we have defined the vector $\mathbf{y}_t = (y_1, \ldots, y_t)^\top$. Ordinarily, we would minimize (3.2.2) with respect to $\mathbf{w}$ and obtain $\mathbf{w}_t = \operatorname{argmin}_{\mathbf{w}} \|\boldsymbol{\Phi}_t^\top \mathbf{w} - \mathbf{y}_t\|^2 = \left(\boldsymbol{\Phi}_t^\top\right)^\dagger \mathbf{y}_t$, where $\left(\boldsymbol{\Phi}_t^\top\right)^\dagger$ is the pseudo-inverse of $\boldsymbol{\Phi}_t^\top$. The classic RLS algorithm (Ljung, 1999), based on the *matrix inversion lemma* allows one to minimize the loss $\mathcal{L}(\mathbf{w})$ recursively and online without recomputing the matrix $\left(\boldsymbol{\Phi}_t^\top\right)^\dagger$ at each step.

As mentioned above, the feature space may be of very high dimensionality, rendering the handling and manipulation of matrices such as $\boldsymbol{\Phi}_t$ and $\boldsymbol{\Phi}_t \boldsymbol{\Phi}_t^\top$ prohibitive.

Algorithm 11 The Kernel RLS Algorithm. On the right we bound the number of operations per time-step for each line of the pseudo-code. The overall per time-step computational cost is bounded by $O(m^2)$ (we assume that kernel evaluations require $O(1)$ time)

```
                                                                  Cost
Parameter: $\nu$
Initialize: $\tilde{\mathbf{K}}_1 = [k_{11}]$, $\tilde{\mathbf{K}}_1^{-1} = [1/k_{11}]$,
        $\boldsymbol{\alpha}_1 = (y_1/k_{11})$, $\mathbf{P}_1 = [1]$, $m = 1$
for $t = 2, 3 \ldots$
  1. Get new sample: $(\mathbf{x}_t, y_t)$
  2. Compute $\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$                          $O(m)$
  3. ALD test:
    $\boldsymbol{a}_t = \tilde{\mathbf{K}}_{t-1}^{-1}\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$                        $O(m^2)$
    $\delta_t = k_{tt} - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top \boldsymbol{a}_t$                    $O(m)$
    if $\delta_t > \nu$   % add $\mathbf{x}_t$ to dictionary                $O(1)$
      $\mathcal{D}_t = \{\mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}\}$                       $O(1)$
      Compute $\tilde{\mathbf{K}}_t^{-1}$ (2.2.10) with $\tilde{\boldsymbol{a}}_t = \boldsymbol{a}_t$         $O(m^2)$
      $\boldsymbol{a}_t = (\mathbf{0}, \ldots, 1)^\top$                           $O(m)$
      Compute $\mathbf{P}_t$ (3.2.9)                               $O(m)$
      Compute $\boldsymbol{\alpha}_t$ (3.2.10)                              $O(m)$
      $m := m + 1$                                          $O(1)$
    else   % dictionary unchanged
      $\mathcal{D}_t = \mathcal{D}_{t-1}$
      $\mathbf{q}_t = \frac{\mathbf{P}_{t-1}\boldsymbol{a}_t}{1+\boldsymbol{a}_t^\top \mathbf{P}_{t-1}\boldsymbol{a}_t}$                          $O(m^2)$
      Compute $\mathbf{P}_t$ (3.2.7)                               $O(m^2)$
      Compute $\boldsymbol{\alpha}_t$ (3.2.8)                               $O(m^2)$
Output: $\mathcal{D}_t$, $\boldsymbol{\alpha}_t$
```

Fortunately, as can be easily verified[1], we may express the optimal weight vector as

$$\mathbf{w}_t = \sum_{i=1}^{t} \alpha_i \boldsymbol{\phi}(\mathbf{x}_i) = \boldsymbol{\Phi}_t \boldsymbol{\alpha} \ , \tag{3.2.3}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_t)^\top$. Substituting into (3.2.2), slightly abusing notation, we have

$$\mathcal{L}(\boldsymbol{\alpha}) = \|\mathbf{K}_t \boldsymbol{\alpha} - \mathbf{y}_t\|^2 \ . \tag{3.2.4}$$

Theoretically, the minimizer of (3.2.4) is given by $\boldsymbol{\alpha}_t = \mathbf{K}_t^\dagger \mathbf{y}_t$, which can be computed recursively using the classic RLS algorithm. The problem with this approach is threefold. First, for large datasets simply maintaining $\mathbf{K}$ in memory, estimating the coefficient vector $\boldsymbol{\alpha}$ and evaluating new points could prove prohibitive both in terms of space and time. Second, the order of the model produced (i.e. the size of the vector $\boldsymbol{\alpha}$, which will be dense in general), would be equal to the number of training samples, causing severe overfitting. Third, in many cases the eigenvalues of the matrix $\mathbf{K}_t$ decay rapidly to 0, which means that inverting it would be numerically unstable.

By making use of the sparsification method of Chapter 2 we can overcome these shortcomings. The basic idea is to use the smaller (sometimes much smaller) set of dictionary samples $\mathcal{D}_t$ in the expansion of the weight vector $\mathbf{w}_t$, instead of the entire training set.

Using the approximation (see Eq. 2.3.11) $\boldsymbol{\Phi}_t \approx \tilde{\boldsymbol{\Phi}}_t \mathbf{A}^\top$ in (3.2.3), we obtain

$$\mathbf{w}_t = \boldsymbol{\Phi}_t \boldsymbol{\alpha}_t \approx \tilde{\boldsymbol{\Phi}}_t \mathbf{A}_t^\top \boldsymbol{\alpha}_t = \tilde{\boldsymbol{\Phi}}_t \tilde{\boldsymbol{\alpha}}_t = \sum_{j=1}^{|\mathcal{D}_t|} \tilde{\alpha}_j \boldsymbol{\phi}(\tilde{\mathbf{x}}_j), \tag{3.2.5}$$

where $\tilde{\boldsymbol{\alpha}}_t \overset{\text{def}}{=} \mathbf{A}_t^\top \boldsymbol{\alpha}_t$ is a vector of m reduced coefficients. The loss becomes

$$\mathcal{L}(\tilde{\boldsymbol{\alpha}}) = \|\boldsymbol{\Phi}_t^\top \tilde{\boldsymbol{\Phi}}_t \tilde{\boldsymbol{\alpha}} - \mathbf{y}_t\|^2 = \|\mathbf{A}_t \tilde{\mathbf{K}}_t \tilde{\boldsymbol{\alpha}} - \mathbf{y}_t\|^2 \ , \tag{3.2.6}$$

and its minimizer is $\tilde{\boldsymbol{\alpha}}_t = (\mathbf{A}_t \tilde{\mathbf{K}}_t)^\dagger \mathbf{y}_t = \tilde{\mathbf{K}}_t^{-1} \mathbf{A}_t^\dagger \mathbf{y}_t$.

In the online scenario, at each time step t we are faced with either one of the following two cases:

1. $\boldsymbol{\phi}(\mathbf{x}_t)$ is ALD on $\mathcal{D}_{t-1}$, i.e. $\delta_t \le \nu$ and $\boldsymbol{a}_t$ are given by (2.2.4). In this case $\mathcal{D}_t = \mathcal{D}_{t-1}$, and consequently $\tilde{\mathbf{K}}_t = \tilde{\mathbf{K}}_{t-1}$.

2. $\delta_t > \nu$, therefore $\boldsymbol{\phi}(\mathbf{x}_t)$ is not ALD on $\mathcal{D}_{t-1}$. $\mathbf{x}_t$ is added to the dictionary, i.e.

[1]Simply add to $\mathbf{w}_t$ some vector $\bar{\mathbf{w}}$ orthogonal to $\{\boldsymbol{\phi}(\mathbf{x}_i)\}_{i=1}^t$ and substitute into (3.2.2).

$\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$, and the dimension of $\tilde{\mathbf{K}}_t$ increases by 1.

We now derive the KRLS update equations for each of these cases.

Case 1 Here, only $\mathbf{A}$ changes between time steps: $\mathbf{A}_t = [\mathbf{A}_{t-1}^\top, \boldsymbol{a}_t]^\top$. Therefore $\mathbf{A}_t^\top \mathbf{A}_t = \mathbf{A}_{t-1}^\top \mathbf{A}_{t-1} + \boldsymbol{a}_t \boldsymbol{a}_t^\top$, and $\mathbf{A}_t^\top \mathbf{y}_t = \mathbf{A}_{t-1}^\top \mathbf{y}_{t-1} + \boldsymbol{a}_t y_t$. Note that $\tilde{\mathbf{K}}_t$ is unchanged. Defining $\mathbf{P}_t = (\mathbf{A}_t^\top \mathbf{A}_t)^{-1}$ we apply the matrix inversion Lemma (e.g., Scharf, 1991) to obtain a recursive formula for $\mathbf{P}_t$:

$$\mathbf{P}_t = \mathbf{P}_{t-1} - \frac{\mathbf{P}_{t-1}\boldsymbol{a}_t\boldsymbol{a}_t^\top \mathbf{P}_{t-1}}{1 + \boldsymbol{a}_t^\top \mathbf{P}_{t-1}\boldsymbol{a}_t} \,. \tag{3.2.7}$$

Defining $\mathbf{q}_t = \frac{\mathbf{P}_{t-1}\boldsymbol{a}_t}{1+\boldsymbol{a}_t^\top \mathbf{P}_{t-1}\boldsymbol{a}_t}$ we derive the KRLS update rule for $\tilde{\boldsymbol{\alpha}}$:

$$\begin{aligned}
\tilde{\boldsymbol{\alpha}}_t &= \tilde{\mathbf{K}}_t^{-1}\mathbf{P}_t\mathbf{A}_t^\top \mathbf{y}_t \\
&= \tilde{\mathbf{K}}_t^{-1}\left(\mathbf{P}_{t-1} - \mathbf{q}_t\boldsymbol{a}_t^\top \mathbf{P}_{t-1}\right)\left(\mathbf{A}_{t-1}^\top \mathbf{y}_{t-1} + \boldsymbol{a}_t y_t\right) \\
&= \tilde{\boldsymbol{\alpha}}_{t-1} + \tilde{\mathbf{K}}_t^{-1}\left(\mathbf{P}_t\boldsymbol{a}_t y_t - \mathbf{q}_t\boldsymbol{a}_t^\top \tilde{\mathbf{K}}_t\tilde{\boldsymbol{\alpha}}_{t-1}\right) \\
&= \tilde{\boldsymbol{\alpha}}_{t-1} + \tilde{\mathbf{K}}_t^{-1}\mathbf{q}_t\left(y_t - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top \tilde{\boldsymbol{\alpha}}_{t-1}\right),
\end{aligned} \tag{3.2.8}$$

where the last equality is based on $\mathbf{q}_t = \mathbf{P}_t\boldsymbol{a}_t$, and $\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) = \tilde{\mathbf{K}}_t\boldsymbol{a}_t$.

Case 2 Here, $\mathbf{K}_t \neq \mathbf{K}_{t-1}$, but the partitioned matrix inverse formula is used to obtain (see Eq. 2.2.10 and Appendix D.4)

$$\tilde{\mathbf{K}}_t = \begin{bmatrix} \tilde{\mathbf{K}}_{t-1} & \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) \\ \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top & k_{tt} \end{bmatrix} \quad \Rightarrow \quad \tilde{\mathbf{K}}_t^{-1} = \frac{1}{\delta_t}\begin{bmatrix} \delta_t\tilde{\mathbf{K}}_{t-1}^{-1} + \boldsymbol{a}_t\boldsymbol{a}_t^\top & -\boldsymbol{a}_t \\ -\boldsymbol{a}_t^\top & 1 \end{bmatrix},$$

where $\tilde{\boldsymbol{a}}_t = \tilde{\mathbf{K}}_{t-1}^{-1}\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$. Note that $\tilde{\boldsymbol{a}}_t$ equals the $\boldsymbol{a}_t$ computed for the ALD test (2.2.4), so there is no need to recompute $\tilde{\mathbf{K}}_{t-1}^{-1}\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$. Furthermore, $\boldsymbol{a}_t = (0, \ldots, 1)^\top$ since $\boldsymbol{\phi}(\mathbf{x}_t)$ is exactly representable by itself. Therefore

$$\begin{aligned}
\mathbf{A}_t &= \begin{bmatrix} \mathbf{A}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix}, \; \mathbf{A}_t^\top \mathbf{A}_t = \begin{bmatrix} \mathbf{A}_{t-1}^\top \mathbf{A}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix}, \\
\mathbf{P}_t &= \begin{bmatrix} \mathbf{P}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix},
\end{aligned} \tag{3.2.9}$$

where $\mathbf{0}$ is a vector of zeros of appropriate length. The KRLS update rule for $\tilde{\boldsymbol{\alpha}}$ is:

$$\begin{aligned}
\tilde{\boldsymbol{\alpha}}_t &= \tilde{\mathbf{K}}_t^{-1}(\mathbf{A}_t^\top \mathbf{A}_t)^{-1}\mathbf{A}_t^\top \mathbf{y}_t \\
&= \tilde{\mathbf{K}}_t^{-1}\begin{bmatrix} (\mathbf{A}_{t-1}^\top \mathbf{A}_{t-1})^{-1}\mathbf{A}_{t-1}^\top \mathbf{y}_{t-1} \\ y_t \end{bmatrix} \\
&= \begin{bmatrix} \tilde{\boldsymbol{\alpha}}_{t-1} - \frac{\tilde{\boldsymbol{a}}_t}{\delta_t}(y_t - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top \tilde{\boldsymbol{\alpha}}_{t-1}) \\ \frac{1}{\delta_t}(y_t - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top \tilde{\boldsymbol{\alpha}}_{t-1}) \end{bmatrix},
\end{aligned} \tag{3.2.10}$$

where for the final equality we used $\tilde{\boldsymbol{a}}_t^\top \tilde{\mathbf{K}}_{t-1} = \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top$. The algorithm in pseudo-code is described in Algorithm 11. On the right-hand side of the table we bound the number of operations per time-step for each line of the pseudo-code. The overall per time-step computational cost is bounded by $O(m^2)$ (we assume that kernel evaluations require $O(1)$ time). [2]

3.3 A Generalization Bound

In this section we present a data-dependent generalization bound whose generality makes it applicable to the KRLS algorithm. Most of the currently available bounds for classification and regression assume a bounded loss function. While this assumption is often acceptable for classification, this is clearly not the case for regression. Recently a generalization error bound for unbounded loss functions was established in Meir and Zhang (2003), where the boundedness assumption is replaced by a moment condition. Theorem 3.3.1 below is a slightly revised version of Theorem 8 in Meir and Zhang (2003).

We quote the general theorem, and then apply it to the specific case of the squared loss studied in this work. For each function f, consider the loss function

$$\ell_{ins}(\mathbf{x}, y, f(\mathbf{x})) = \ell_{ins}(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$$

viewed as a function from $\mathcal{X} \times \mathcal{Y}$ to $\mathbb{R}$. In our context the function f is given by $f(\mathbf{x}) = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle$. Let $Z^t = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_t, y_t)\}$ be a set of t IID samples drawn from some distribution $\rho(X, Y)$. Set

$$\ell_{av}(f) = \mathbf{E}_{X,Y} \ell_{ins}(Y, f(X)) \quad \text{and } \hat{\ell}_{av}(f) = \frac{1}{t} \sum_{i=1}^{t} \ell_{ins}(y_i, f(\mathbf{x}_i)).$$

Furthermore, let

$$\mathcal{L}_{\mathcal{F}} = \{\ell_{ins}(y, f(\mathbf{x})) : \ f \in \mathcal{F}\}$$

be a class of functions (defined on $\mathcal{X} \times \mathcal{Y}$).

In order to establish useful generalization bounds we introduce a classic complexity measure for a class of functions $\mathcal{F}$. Let $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_t)$ be a sequence of independent and identically distributed $\{\pm 1\}$-valued random variables, such that $\text{Prob}(\xi_i = 1) = 1/2$. The *empirical* Rademacher complexity of $\mathcal{F}$ (e.g., van der

[2]The C source code for the KRLS algorithm may be downloaded from http://visl.technion.ac.il/~yaki/

Vaart & Wellner, 1996) is defined as

$$\hat{R}_t(\mathcal{F}) = \mathbf{E}_{\boldsymbol{\xi}} \sup_{f\in\mathcal{F}} \left\{ \frac{1}{t}\sum_{i=1}^{t} \xi_i f(\mathbf{x}_i) \right\}.$$

The Rademacher complexity $R_t(\mathcal{F})$ is given by $R_t(\mathcal{F}) = \mathbf{E}\hat{R}_t(\mathcal{F})$, where the expectation is taken with respect to the marginal product distribution ρ^t over $\{\mathbf{x}_1, \ldots, \mathbf{x}_t\}$.

Theorem 3.3.1 (Meir & Zhang, 2003). *Let $\mathcal{F}$ be a class of functions mapping from a domain $\mathcal{X}$ to $\mathbb{R}$, and let $\{(\mathbf{x}_i, y_i)\}_{i=1}^t$, $\mathbf{x}_i \in \mathcal{X}$, $y_i \in \mathbb{R}$, be independently selected according to a probability measure ρ. Assume there exists a positive real number M such that for all positive ζ*

$$\log \mathbf{E}_{X,Y} \sup_{f\in\mathcal{F}} \cosh(2\zeta \ell_{ins}(Y, f(X))) \leq \zeta^2 M^2/2. \tag{3.3.11}$$

Then with probability of at least $1-\delta$ over samples of length t, every $f \in \mathcal{F}$ satisfies

$$\ell_{av}(f) \leq \hat{\ell}_{av}(f) + 2R_t(\mathcal{L}_{\mathcal{F}}) + M\sqrt{\frac{2\log(1/\delta)}{t}}.$$

(References to δ here and in the rest of this section are unrelated to the δ_t used in the context of the sparsification). Note that the condition (3.3.11) replaces the standard uniform boundedness used in many approaches. In order for this result to be useful we need to present an upper bound on the Rademacher complexity $R_t(\mathcal{L}_{\mathcal{F}})$. Assume initially that $(1/2)\|\mathbf{w}\|^2 \leq A$ (where $\|\mathbf{w}\|^2 = \langle \mathbf{w}, \mathbf{w} \rangle$). We quote the following result from Meir and Zhang (2003) (see Eq. (13)).

Lemma 3.3.2. *Consider the class of functions $\mathcal{F}_A = \{f(\mathbf{x}) = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle : (1/2)\|\mathbf{w}\|^2 \leq A\}$. Then*

$$\hat{R}_t(\mathcal{F}_A) \leq \sqrt{\frac{2A}{t}\left(\frac{1}{t}\sum_{i=1}^{t} \|\boldsymbol{\phi}(\mathbf{x}_i)\|^2\right)}. \tag{3.3.12}$$

We assume for simplicity that $k(\mathbf{x}, \mathbf{x}') \leq B$ for all $\mathbf{x}$ and $\mathbf{x}'$. In the following sequence of inequalities we use the notation $\mathbf{w} \in \Omega_A$ to indicate that $(1/2)\|\mathbf{w}\|^2 \leq A$. Denoting the expectation with respect to the product distribution ρ^t over the sample

Z^t by $\mathbf{E}_Z$, and keeping in mind that $\mathbf{E}\xi_i = 0$ for any i we have that

$$\begin{aligned}
R_t(\mathcal{L}_{\mathcal{F}_A}) &= \mathbf{E}_Z \mathbf{E}_{\boldsymbol{\xi}} \sup_{\mathbf{w}\in\Omega_A} \frac{1}{t}\sum_{i=1}^{t} \xi_i \left(y_i - \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_i)\rangle\right)^2 \\
&= \mathbf{E}_Z \mathbf{E}_{\boldsymbol{\xi}} \sup_{\mathbf{w}\in\Omega_A} \frac{1}{t}\sum_{i=1}^{t} \xi_i \left(\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_i)\rangle^2 - 2\langle \mathbf{w}, y_i\boldsymbol{\phi}(\mathbf{x}_i)\rangle\right) \\
&\le \mathbf{E}_Z \mathbf{E}_{\boldsymbol{\xi}} \sup_{\mathbf{w}\in\Omega_A} \frac{1}{t}\sum_{i=1}^{t} \xi_i \left(\sqrt{2AB} + 2|y_i|\right) \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_i)\rangle \\
&\overset{(a)}{\le} \frac{\sqrt{2AB}}{t}\mathbf{E}_Z \sqrt{\sum_{i=1}^{t}\left(\sqrt{2AB} + 2|y_i|\right)^2} \\
&\overset{(b)}{\le} \frac{\sqrt{2AB}}{t}\mathbf{E}_Z \sqrt{\sum_{i=1}^{t}(4AB + 8y_i^2)} \\
&\overset{(c)}{\le} \frac{\sqrt{2AB}}{t}\sqrt{4tAB + 8t\mathbf{E}[Y^2]} \\
&\overset{(d)}{\le} \frac{2\sqrt{2}AB + 4\sqrt{AB\mathbf{E}[Y^2]}}{\sqrt{t}} .
\end{aligned}$$

where (a) used (3.3.12) with $\boldsymbol{\phi}(\mathbf{x}_i)$ replaced by $(\sqrt{2AB} + 2|y_i|)\boldsymbol{\phi}(\mathbf{x}_i)$ and the fact that $k(\mathbf{x},\mathbf{x}) = \|\boldsymbol{\phi}(\mathbf{x}_i)\|^2 \le B$, (b) used $(a+b)^2 \le 2a^2 + 2b^2$, (c) made use of Jensen's inequality, and (d) is based on $\sqrt{x+y} \le \sqrt{x} + \sqrt{y}$.

The derived bound on $R_t(\mathcal{L}_{\mathcal{F}_A})$ is formulated in terms of the parameter A. In order to remove this dependence we use a (by now) standard trick based on the union bound. Let $\{A_i\}_{i=1}^{\infty}$ and $\{p_i\}_{i=1}^{\infty}$ be sets of positive numbers such that $\limsup A_i = \infty$ and $\sum_i p_i = 1$ (for example, $p_i = 1/i(i+1)$). We apply Theorem 3.3.1 for each value of A_i replacing δ by $p_i\delta$. A simple utilization of the union bound, and some algebraic manipulations (described in the proof of Theorem 10 in Meir and Zhang (2003)) allow one to establish the following bound, where all reference to the parameter A has been eliminated.

Theorem 3.3.3. *Let $\mathcal{F}$ be a class of functions of the form $f(\mathbf{x}) = \langle \mathbf{w}, \phi(\mathbf{x})\rangle$, and let $\{(\mathbf{x}_i, y_i)\}_{i=1}^{t}$, $\mathbf{x}_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$, be independently selected according to a probability measure ρ. Assume further that (3.3.11) holds. Fix any number g_0 and set $\tilde{g}(\mathbf{w}) = 2\max((1/2)\|\mathbf{w}\|^2, g_0)$ Then with probability at least $1-\delta$ over samples of length t, every $f \in \mathcal{F}$ satisfies*

$$\begin{aligned}
\mathbf{E}\left(Y - f(X)\right)^2 \le \frac{1}{t}\sum_{i=1}^{t}(y_i - f(\mathbf{x}_i))^2 + \frac{4\sqrt{2}B\tilde{g}(\mathbf{w}) + 8\sqrt{B\tilde{g}(\mathbf{w})\mathbf{E}[Y^2]}}{\sqrt{t}} \\
+ M\sqrt{\frac{4\log\log_2(2\tilde{g}(\mathbf{w})/g_0) + 2\log(1/\delta)}{t}} .
\end{aligned}$$

The essential feature of the bound in Theorem 3.3.3 is that it holds uniformly for any $\mathbf{w}$, and thus in particular to the weight vector $\mathbf{w}$ obtained using the KRLS algorithm. Observe that the price paid for eliminating the parameter A is an extra term of order $\log\log \tilde{g}(\mathbf{w})$. Note also that using the sparse dual representation (3.2.5) $\mathbf{w} = \sum_{j=1}^{|\mathcal{D}_t|} \tilde{\alpha}_j \boldsymbol{\phi}(\tilde{\mathbf{x}}_j)$ the bound can be phrased in terms of the coefficients $\tilde{\alpha}_j$ using $\|\mathbf{w}\|^2 = \tilde{\boldsymbol{\alpha}}_t^\top \tilde{\mathbf{K}}_t \tilde{\boldsymbol{\alpha}}_t$.

3.4 Experiments

In this section we experimentally demonstrate the potential utility and practical efficacy of the KRLS algorithm in a range of machine learning and signal processing applications. We begin by exploring the scaling behavior of KRLS on a simple, static nonlinear regression problem. We then move on to test KRLS on several well-known benchmark regression problems, both synthetic and real. As a point of reference we use the highly efficient SVM package SVMTorch (Collobert & Bengio, 2001), with which we compare our algorithm. Next, we move to the domain of time series prediction (TSP). The most common approach to the TSP problem is based on identifying a generally nonlinear auto-regressive model of the series. This approach essentially reduces the TSP problem to a regression problem with the caveat that samples can no longer be assumed to be IID. Numerous learning architectures and algorithms have been thrown at this problem with mixed results (see e.g., Weigend & Gershenfeld, 1994). One of the more successful general purpose algorithms tested on TSP is again the SVM (Müller et al., 1997); however SVMs are inherently limited by their off-line mode of training, and their poor scaling properties. We argue that KRLS is a more appropriate tool in this domain, and to support this claim we test KRLS on two well known and difficult time series prediction problems. Finally, we apply KRLS to a nonlinear channel equalization problem, on which SVMs were reported to perform well (Sebald & Bucklew, 2000). All tests were run on a 256Mb, 667MHz Pentium 3 Linux workstation.

3.4.1 Nonlinear Regression

We report the results of experiments comparing the KRLS algorithm (coded in C) to the state-of-the-art SVR implementation SVMTorch (Collobert & Bengio, 2001). Throughout, the best parameter values[3] (for C, ε, ν) were found by using a 10-fold cross-validation procedure in which we looked for the minimum average root-mean-squared error (RMSE) over a range of parameter values, spaced logarithmically. The

[3] ε is the parameter defining the width of the error-insensitive zone used in the SVR loss function, while C determines the trade-off between the empirical loss term and the regularization term. See Schölkopf and Smola (2002), Chapter 9 for precise definitions.

best values for each parameter were found for a single learning scenario in which the training set consisted of 1000 training samples corrupted by Gaussian noise with a standard deviation of 0.1. This set of parameters was then used throughout the experiments. No attempt was made to optimize parameters for each tested scenario separately, as this would be prohibitively time-consuming. The kernel we used was Gaussian,

$$k(\mathbf{x},\mathbf{x}') = \exp\left(-\|\mathbf{x}-\mathbf{x}'\|^2/(2\sigma^2)\right) \ , \tag{3.4.13}$$

and a similar procedure was used to find the value of the kernel width parameter σ for which each algorithm performed best. All the results reported below are averaged over 50 independent randomly generated training sets.

We first used KRLS for learning the 2-dimensional *Sinc-Linear* function $\sin(x_1)/x_1 + x_2/10$ defined on the domain $[-10, 10]^2$. The kernel is Gaussian with a standard deviation $\sigma = 4.4$ for both algorithms. The other SVR parameters are $C = 50,000$ and $\varepsilon = 0.05$, while the KRLS parameter is $\nu = 0.001$. Learning was performed on a random set of samples corrupted by additive IID zero-mean Gaussian noise. Testing was performed on an independent random sample of 1000 noise-free points.

Figure 3.1 depicts the results of two tests. In the first we fixed the noise level (noise STD 0.1) and varied the number of training samples from 5 to 50,000, with each training set drawn independently. We then plotted the test-set error (top left), the number of support/dictionary vectors as a percentage of the training set (top-center), and the CPU time (top-right) for each algorithm. As can be seen, the solution produced by KRLS significantly improves upon the SVR solution, both in terms of generalization performance and sparsity (with a maximum of 72 dictionary samples) for training set sizes of 200 samples or more. In terms of speed, KRLS outperforms SVMTorch over the entire range of training set sizes, by one to three orders of magnitude. In fact, looking at the asymptotic slopes of the two CPU-time graphs we observe that, while SVMTorch exhibits a super-linear dependence on the sample size (slope 1.8), KRLS scales linearly (slope 1.0) as required of a real-time algorithm.

In the second test we fixed the training sample size at 1000 and varied the level of noise in the range 10^{-6} to 10. We note that SVMTorch suffers from an incorrect estimation of the noise level by its ε parameter, in all respects. Most notably, in the presence of high noise, the sparsity of its solution deteriorates drastically. In contrast, due to its unsupervised method of sparsification, KRLS produces a sparse solution with complete disregard of the level of noise. Moreover, in terms of generalization, the KRLS solution is at least as good as the SVR solution.

We tested our algorithm on three additional synthetic data-sets, *Friedman 1,2*

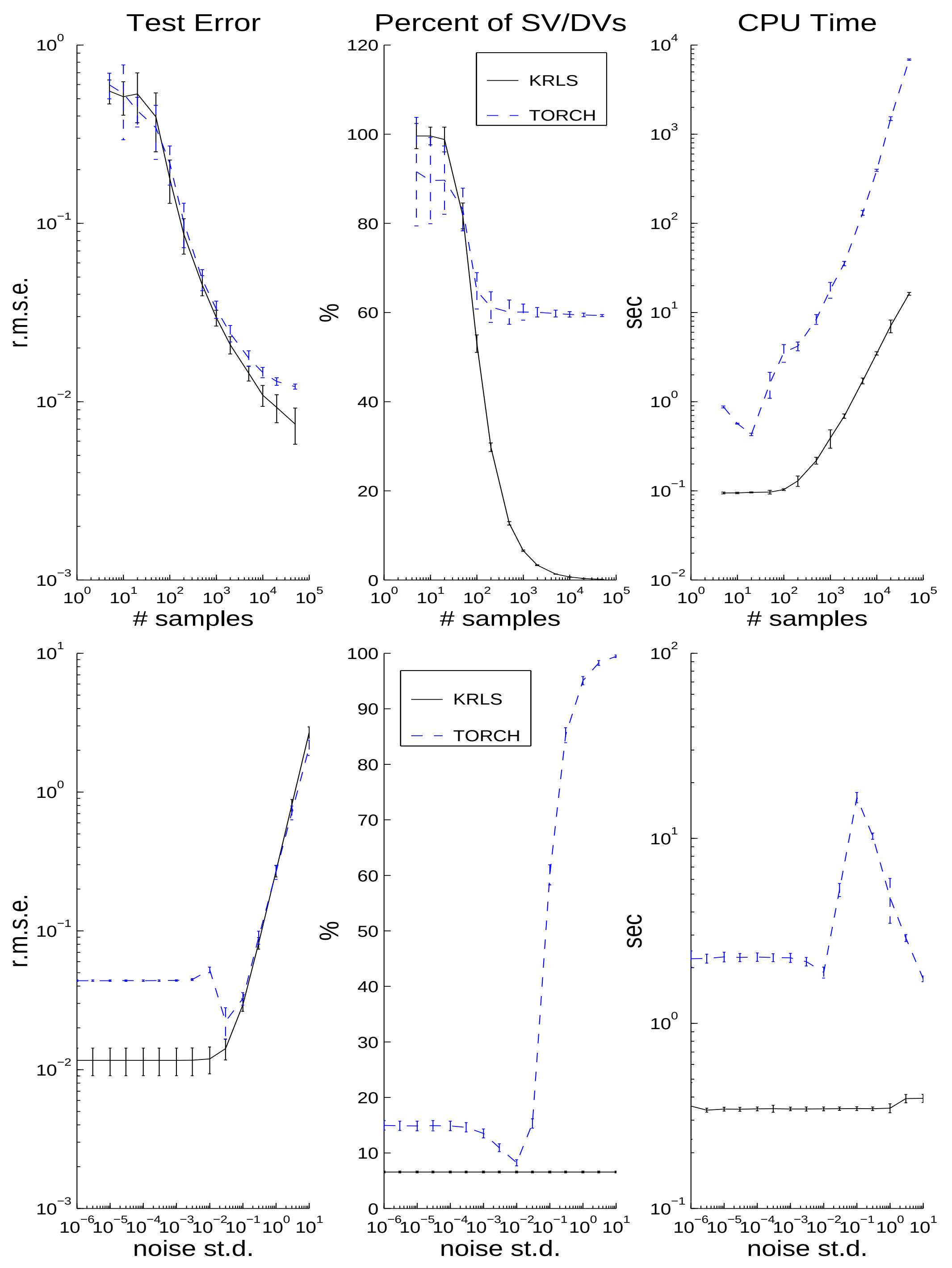


Figure 3.1: Scaling properties of KRLS and SVMTorch with respect to sample size (top) and noise magnitude (bottom), on the *Sinc-Linear* function. Error bars mark one standard deviation above and below the mean.

and *3*, due to Friedman (1991). Both training and test sets were 1000 samples long, and introduced noise was zero-mean Gaussian with a standard deviation of 0.1. For these data-sets a simple preprocessing step was performed which consisted of scaling

Parameters: $t = 1000$, $d = 10$, $C = 10^4$, $\varepsilon = 0.01$, $\sigma = 0.8$, $\nu = 10^{-3}$	**Friedman-1**	RMSE	STD	% SV	CPU
	SVMTorch	0.61	0.03	91.9	24.63
	KRLS	0.55	0.02	100.0	26.97
Parameters: $t = 1000$, $d = 4$, $C = 10^6$, $\varepsilon = 10^{-4}$, $\sigma = 0.5$, $\nu = 10^{-4}$	**Friedman-2**	RMSE	STD	% SV	CPU
	SVMTorch	0.20	0.01	97.3	725.22
	KRLS	0.18	0.02	36.1	10.24
Parameters: $t = 1000$, $d = 4$, $C = 10^5$, $\varepsilon = 0.1$, $\sigma = 0.5$, $\nu = 10^{-3}$	**Friedman-3**	RMSE	STD	% SV	CPU
	SVMTorch	0.09	0.01	37.6	2.32
	KRLS	0.09	0.01	23.3	4.74

Table 3.1: Results on the synthetic *Friedman* data-sets. The columns, from left to right, list the average R.M.S. test-set error, its standard deviation, average percentage of support/dictionary vectors, and the average CPU time in seconds used by the respective algorithm.

the input variables to the unit hyper-cube, based on their minimum and maximum values. The results are summarized in Table 3.1.

We also tested KRLS on two real-world data-sets - *Boston housing* and *Comp-activ*, both from Delve[4]. The task in the *Boston* data-set (506 samples, 13 dimensions) is to predict median value of owner-occupied homes in $1000's for various Boston neighborhoods, based on census data. In the *Comp-activ* (cpuSmall) data-set (8192 samples, 12 dimensions) the task is to predict the users' CPU utilization percentage in a multi-processor, multi-user computer system, based on measures of system activity. A preprocessing step, like the one used on the *Friedman* data-sets, was performed here as well. Generalization performance was checked using 106 (*Boston*) and 2192 (*Comp-activ*) left-out test samples. The results are summarized in Table 3.2.

Parameters: $t = 6000$, $d = 12$, $C = 10^6$, $\varepsilon = 0.5$, $\sigma = 0.9$, $\nu = 0.001$	**Comp-activ**	RMSE	STD	% SV	CPU
	SVMTorch	3.13	0.07	81.9	130.4
	KRLS	3.09	0.08	3.4	19.52
Parameters: $t = 400$, $d = 13$, $C = 10^6$, $\varepsilon = 1$, $\sigma = 1.3$, $\nu = 0.001$	**Boston**	RMSE	STD	% SV	CPU
	SVMTorch	3.16	0.49	58.0	5.57
	KRLS	3.52	0.48	38.6	0.53

Table 3.2: Results on the real-world *Comp-activ* and *Boston* data-sets.

3.4.2 Time Series Prediction

Digital signal processing is a rich application domain in which the classic RLS algorithm has been applied extensively. Online and real-time constraints are often

[4]http://www.cs.toronto.edu/~delve/data/datasets.html

imposed upon signal processing algorithms, making many of the recently developed kernel-based machine learning algorithms irrelevant for such tasks. The development of KRLS may help to fill this gap, indeed, it is in such tasks that the recursive, online operation of KRLS becomes particularly useful, if not essential.

An important and well studied problem in both the machine learning and the signal processing communities is the prediction of time series. We tested KRLS on both synthetically generated and real-world time series – The well studied Mackey–Glass time series, and the Laser (A) time series from the Santa Fe time series prediction competition (Weigend & Gershenfeld, 1994). The Mackey–Glass series is synthetically generated by numerical integration of a time-delay differential equation. The Laser time series is taken from real measurements of the intensity of a far-infrared NH3 laser. Both of these series exhibit chaotic behavior, making the task of multi-step prediction exponentially difficult as a function of the prediction horizon. Nevertheless, short-term multi-step prediction is feasible, depending on the intrinsic predictability of the dynamics. In the Mackey–Glass series it is possible to make accurate predictions 100's of time steps into the future, while for the Laser series the useful limit seems to be about 100. Throughout the time-series experiments we used the Gaussian kernel (3.4.13), the width of which is specified by the parameter σ.

Mackey–Glass Time Series

Our first experiment is with the Mackey–Glass chaotic time series. This time series may be generated by numerical integration of a time-delay differential equation that was proposed as a model of white blood cell production (Mackey & Glass, 1977):

$$\frac{dy}{dt} = \frac{ay(t-\tau)}{1+y(t-\tau)^{10}} - by(t), \tag{3.4.14}$$

where $a = 0.2$, $b = 0.1$. For $\tau > 16.8$ the dynamics become chaotic; we therefore conducted our tests using two value for τ, corresponding to weakly chaotic behavior at $\tau = 17$ and a more difficult case at $\tau = 30$. Eq. 3.4.14 was numerically integrated using the Euler method. Initial conditions were generated by drawing $\mathbf{x}_0$ from a uniform distribution over the interval $[0.1, 2]$, with $\mathbf{x}_t = 0$ for $t < 0$.

Training sets 1000 samples long were generated by sampling the series at 1 time-unit intervals, with the respective test sets consisting of the subsequent 200 samples. The embedding dimension was fixed at 10 with an embedding delay of 4 samples, i.e. $\mathbf{x}_t = (y(t-4), y(t-8), \ldots, y(t-40))$. The parameters for each algorithm were selected by searching for the minimum 200-step RMS iterated prediction error, averaged over 10 independent training and validation sets. The parameters found for the Mackey–Glass(17) series are, for SVMTorch $\sigma = 0.7$, $C = 10^3$, $\varepsilon = 10^{-5}$, while for

KRLS they are $\sigma = 0.5$, $\nu = 10^{-4}$. For the Mackey–Glass(30) series the SVMTorch parameters were unchanged, while for KRLS ν remained the same and $\sigma = 0.6$. The test results for SVMTorch and KRLS on both series, averaged over 50 independent trials, are given in Table 3.3.

MG(17)	RMSE(1)	STD(1)	RMSE(200)	STD(200)	% SV	CPU
SVMTorch	0.0023	0.0003	0.0187	0.0080	30.48	4.18
KRLS	0.0004	0.0002	0.0027	0.0016	27.05	6.85
MG(30)	RMSE(1)	STD(1)	RMSE(200)	STD(200)	% SV	CPU
SVMTorch	0.006	0.003	0.034	0.036	50.2	7.23
KRLS	0.006	0.004	0.033	0.028	51.9	12.00

Table 3.3: 1-step and 200-step iterated prediction results on the Mackey–Glass time series with $\tau = 17$ (MG(17)) and $\tau = 30$ (MG(30))

On the Mackey–Glass(30) series KRLS and SVM perform comparably, both in terms of prediction accuracy and in terms of sparsity. However, on the Mackey–Glass(17) series KRLS significantly outperforms SVM in its prediction accuracy. Fig. 3.2 shows examples of two test sets, one for each series, and the iterated predictions produced by the two algorithms.

Santa Fe Laser Time Series

Our next experiment is with the chaotic laser time series (data set A) from the Santa Fe time series competition (Weigend & Gershenfeld, 1994) [5] This is a particularly difficult time series to predict, due both to its chaotic dynamics and to the fact that only three "intensity collapse" events occur in the training set. The accurate prediction of these events is crucial to achieving a low prediction error on the test set. The training data consists of 1000 samples, with the test data being the subsequent 100 samples (see Fig. 3.3). The task is to predict the test series with minimum mean squared error. Looking at the competition results the difficulty of this task becomes apparent: Only two of the 14 contestants achieved prediction accuracies that are significantly better than simply predicting the mean of the training series. The winning entry achieved a normalized mean squared error (NMSE - the mean squared error divided by the series' variance) of 0.028, by utilizing a complex and highly specialized neural network architecture adapted by a temporal version of the backpropagation algorithm. Moreover, the final 25 steps of the predicted sequence were hand-picked by adjoining to the initial 75–step prediction a similar sequence taken from the training set. The second place entry used an approach based on local linear models and achieved a NMSE of 0.080 on the test set.

[5] http://www-psych.stanford.edu/ãndreas/Time-Series/SantaFe.html

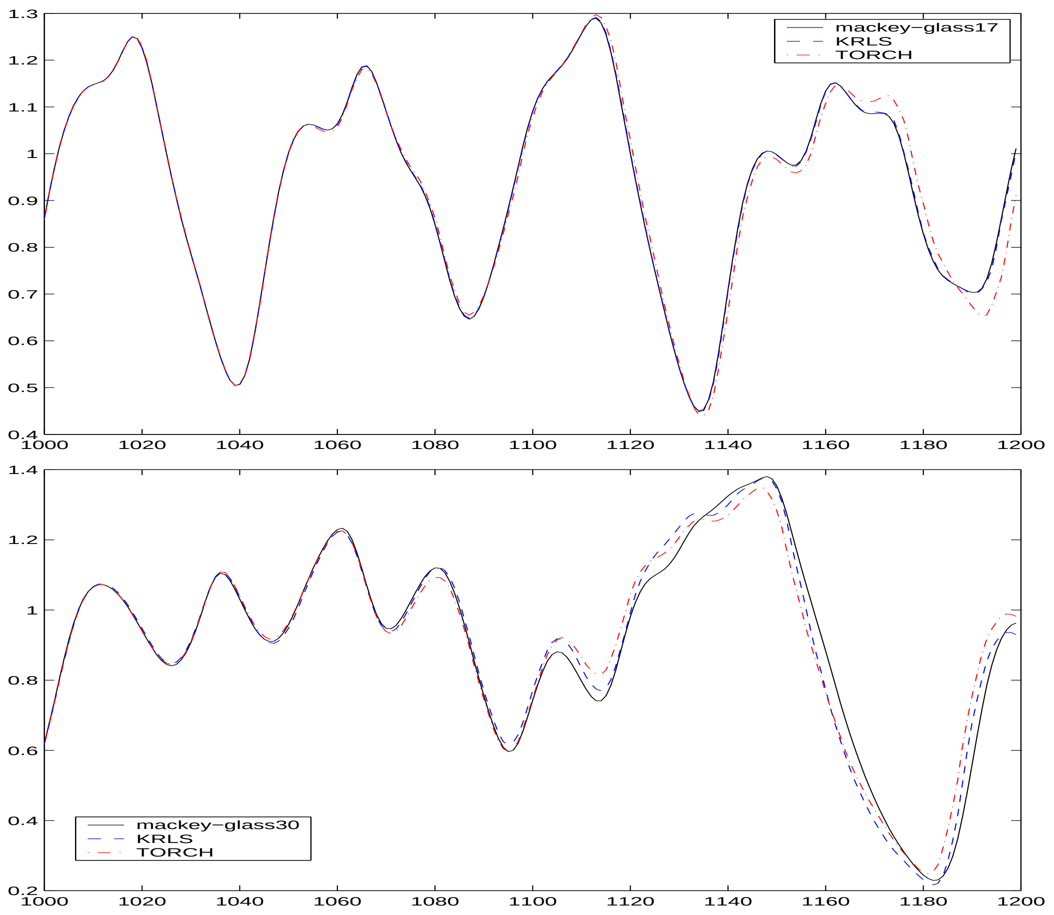


Figure 3.2: Multi-step iterated predictions for the Mackey–Glass time series with $\tau = 17$ (top) and $\tau = 30$ (bottom).

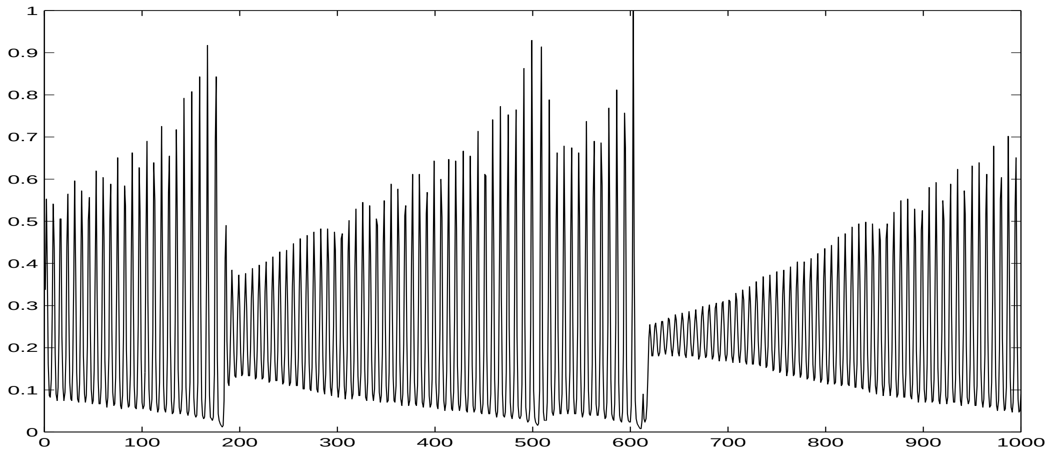


Figure 3.3: The Santa Fe competition laser training series (data set A)

We attempted to learn this series with the KRLS algorithm. The naive approach, as used above for the Mackey–Glass series, would be to minimize the 1-

step prediction error by defining the training samples as $\{(\mathbf{x}_i, y_i)\}_{i=1}^t$ with $\mathbf{x}_i = (y_{i-1}, y_{i-2}, \ldots, y_{i-d})$, where d is the model order and t the number of training samples [6]. Predicting the test series is performed by *iterated prediction* in which successive 1-step predictions are fed back into the predictor as inputs for the prediction of subsequent series values. Unfortunately this naive approach works well for this series only if we are really interested in performing well on 1-step predictions. Since our goal is to provide multi-step predictions, a more sophisticated method is called for. The method we used is as follows: Run KRLS to minimize the 1-step prediction MSE, as in the naive approach. After this pass over the training set is complete, we compute the optimal 1-step estimates $(\hat{y}_1^1, \ldots, \hat{y}_t^1)$. Next, we continue running KRLS on a new data set, which is of the same length as the original data, and in which $\mathbf{x}_t = (\hat{y}_{t-1}^1, y_{t-2}, \ldots, y_{t-d})$. We now have 2-step estimates $(\hat{y}_1^2, \ldots, \hat{y}_t^2)$, based on the real data and the 1-step estimates obtained in the previous step. On the i'th step of this iterative process the appended training set consists of samples of the form $\mathbf{x}_t = (\hat{y}_{t-1}^{i-1}, \hat{y}_{t-2}^{i-2}, \hat{y}_{t-3}^{i-3}, \ldots, \hat{y}_{t-i+1}^1, y_{t-i}, \ldots, y_{t-d})$ if $i \leq d$; or $\mathbf{x}_t = (\hat{y}_{t-1}^{i-1}, \hat{y}_{t-2}^{i-2}, \hat{y}_{t-3}^{i-3}, \ldots, \hat{y}_{t-d}^{i-d})$ if $i > d$.

We may iterate this process until some prespecified prediction horizon n is reached. Assuming that the dictionary ceases to grow at an early stage, The end result is a good approximation to the minimizer of the mean squared error over the entire $n \times t$ - long dataset, in which equal weight is given to 1-step through n-step prediction accuracy. The idea behind this somewhat complex scheme is to improve the stability of the iterative multi-step predictor with respect to small errors in its predicted values, since these are fed back into the predictor as inputs for predictions at subsequent time steps. Note that this scheme relies heavily on the recursiveness of KRLS; the implementation of such a scheme using a batch algorithm such as SVM would be considerably more difficult and costly.

The free parameters of the algorithm were tuned as follows. First we normalized the series, so that its values lie in the range $[0, 1]$. We then performed hold-out tests to determine the values of σ, ν, d and n, where ν is the ALD threshold parameter, d is the model order and n is the prediction horizon for the iterations described above. The training sets we used were a. samples 1-700, b. samples 1-800, and c. samples 1-900. Their respective held-out sets were a. 701-800, b. 801-900 and c. 901-1000. The parameters were optimized with respect to the mean squared error of the multi-step iterated prediction on the held out sets. When insignificant differences in prediction accuracy were observed, the parameter value incurring a lower computational cost was preferred (i.e. preference to high values of ν and σ and low values of d and n). The values found are: $\sigma = 0.9$, $\nu = 0.01$, $d = 40$ and $n = 6$. The NMSE prediction error on the competition test set (samples 1001-1100)

[6] $y_0, \ldots, y_{1-d}$ are assumed to be equal to zero.

achieved by KRLS is 0.026, which is slightly better than the winning entry in the competition. The KRLS prediction and the true continuation are shown in Fig. 3.4. [7]

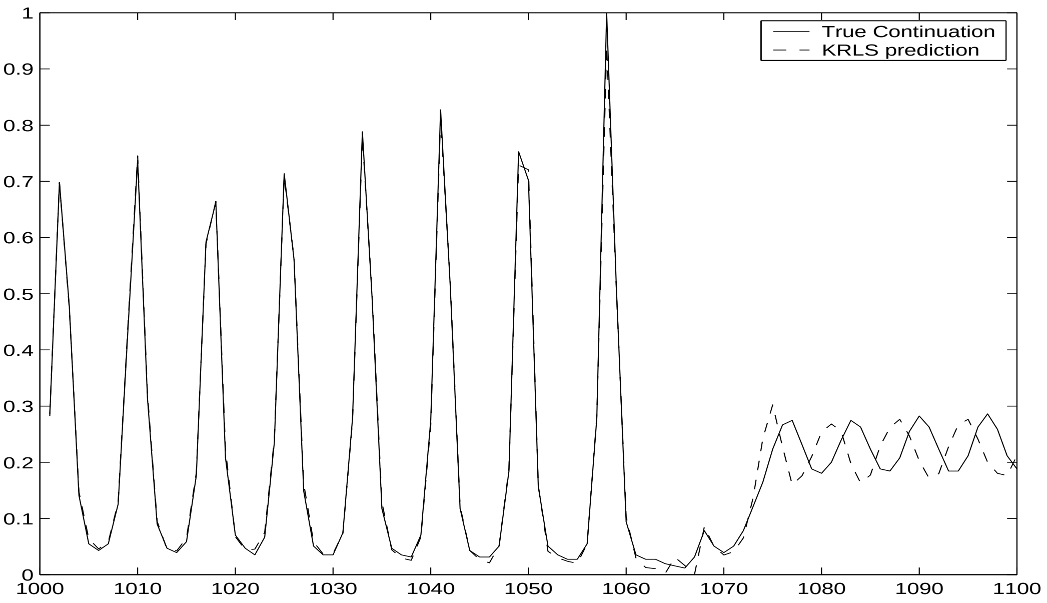


Figure 3.4: KRLS predicting 100 steps into the future (dashed line) on the laser time series. The true continuation is shown as a solid line. Note that on the first 60 steps the prediction error is hardly noticeable.

3.4.3 Channel Equalization

In Sebald and Bucklew (2000) SVMs were applied to nonlinear channel equalization problems with considerable success. One of the reservations made by the authors concerning the use of SVMs in this application domain was due to the inability of SVMs to be trained online. We suggest KRLS as a viable alternative that performs similarly in terms of error rates, but may be trained online, and often produces solutions that are much sparser than SVM solutions in less time, especially for large amounts of data.

Let us briefly state the channel equalization problem. A binary signal is fed into a generally nonlinear channel. At the receiver end of the channel the signal is further corrupted by additive IID (usually Gaussian) noise, and is then observed as $(y_1, \ldots, y_t)$. The aim of channel equalization is to construct an "inverse" filter that reproduces (possibly with some delay) the original signal with as low an error rate as possible. In order to attain this goal a known random binary signal $(u_1, \ldots, u_t)$ is sent through the channel and the corresponding noisy observa-

[7] In this experiment we used a Matlab version of KRLS, for which the entire training session took less than 5 minutes. The C implementation typically runs 5–10 times faster.

Table 3.4: Results of the channel equalization experiment

	SVM	KRLS
D=0 BER	0.174 ± 0.008	0.173 ± 0.010
Sparsity (%)	41.9 ± 4.3	9.7 ± 0.6
CPU Time (sec)	5.44 ± 1.06	0.15 ± 0.03
D=1 BER	0.067 ± 0.020	0.065 ± 0.006
Sparsity (%)	14.4 ± 2.5	9.7 ± 0.6
CPU Time (sec)	0.65 ± 0.19	0.15 ± 0.03
D=2 BER	0.048 ± 0.015	0.046 ± 0.004
Sparsity (%)	10.1 ± 2.1	9.6 ± 0.7
CPU Time (sec)	0.74 ± 0.24	0.15 ± 0.03

tions $(y_1, \ldots, y_t)$ are used to adapt the equalizer. Just as the time-series prediction problem can be cast as a regression problem, it is easy to see that channel equalization may be reduced to a classification problem, with samples $\{(\mathbf{x}_i, u_i)\}_{i=1}^t$, $\mathbf{x}_i = (y_{i+D}, y_{i+D-1}, \ldots, y_{i+D-d+1})$, where d is the model order and D is the equalization time lag. As KRLS minimizes the MSE, its choice as a learning algorithm for equalization seems questionable, since equalization aims at reducing the bit error rate (BER), which is the number of misclassifications. Nevertheless we show that the performance of KRLS solutions, as measured by the BER criterion, is at least as good as SVM classification solutions.

In this experiment we replicate the setup used in the first simulation in Sebald and Bucklew (2000). The nonlinear channel model is defined by $x_t = u_t + 0.5u_{t-1}$, $y_t = x_t - 0.9x_t^3 + n_\sigma$, where n_σ is white Gaussian noise with a variance of 0.2. As in Sebald and Bucklew (2000) a SVM with a third degree polynomial kernel $k(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^\top \mathbf{x}' + 1\right)^3$ was used, with the other parameters being $t = 500$ and $C/t = 5$. KRLS was found to perform best with an eighth degree polynomial kernel[8] and $\nu = 0.1$. Testing was performed on a 5000 sample random test sequence. The results are presented in Table 3.4; each entry is the result of averaging over 100 repeated independent tests.

Our results for SVMTorch are in agreement with the results reported in Sebald and Bucklew (2000) (i.e. within the error bars). The results in Table 3.4 show that, in this task, SVM and KRLS perform comparably in terms of the generalization bit error rate. Sparsity-wise, KRLS delivers uniformly sparser solutions than SVM. The advantage in sparsity is quite marked for $D = 0$ and diminishes for higher values of D. Note the robustness of the KRLS solutions in terms of sparsity and computation time with respect to variations in the equalizer delay D.

[8]We did not try any other type of kernel other than the polynomial one.

3.4.4 Comparison with Sparse Gaussian Processes

A method bearing close similarities to ours is that of Csató and Opper (2002), who suggested an online algorithm for sparse Gaussian process regression (SGPR) in a Bayesian setting. Our least squares approach may be interpreted as a form of maximum likelihood estimation based on similar assumptions. Since in Gaussian process regression the *posterior* moments are evaluated, SGPR must assume some probabilistic model for the measurement process. Conventionally it is assumed that measurements are corrupted by additive IID Gaussian noise, and in consequence SGPR requires an additional parameter estimating the measurement noise variance that is not required when using KRLS.

The distinction between SGPR and our method becomes apparent if this parameter is given a lower value than the true noise variance. In this case, given some fixed dictionary, SGPR would favor fitting the dictionary points at the price of committing larger errors on other points. In the limit, SGPR would fit only the dictionary points and would ignore completely all other points. This means that to compensate for an erroneous noise model, SGPR would have to resort to increasing the size of its dictionary, sacrificing the sparseness of its solution. In comparison, our sparsification method weighs all points equally and maintains the same level of sparsity irrespective of the intensity of the measurement noise. This last property is due to the unsupervised form of sparsification employed by KRLS. While the sparsification algorithm used in SGPR considers the error performed in estimating each new sample's target value (y_t) if that sample is not added to the dictionary, our method considers the error incurred in representing the sample points ($\boldsymbol{\phi}(\mathbf{x}_t)$) themselves. In consequence, no property of the target values, the level of measurement noise included, has any effect on the KRLS dictionary. While in some cases this may seem to be a drawback, unsupervised sparsification helps ensure the robustness of KRLS in the face of an unknown noise model. Not withstanding, it is possible (and straightforward) to incorporate any sparsification procedure into KRLS, supervised or not, as long as it may be implemented online.

In Fig. 3.5 we demonstrate these differences using the *sinc* function in a simple one-dimensional regression problem. The sample consists of 10 points corrupted by IID Gaussian noise with a standard deviation of 0.1. We let the SGPR noise model underestimate the noise by a factor of 5 (i.e. it assumes a noise st.d. of 0.02). At the top of Fig. 3.5 we show the predictions of KRLS and SGPR based on this data. Due to its misestimate of the noise SGPR severely overfits the data while requiring more dictionary vectors (DVs) (nine) than KRLS does (seven). The bottom of Fig. 3.5 shows what happens if, for the same data, we employ in SGPR the same unsupervised sparsification criterion used by KRLS (note the change of scale). Indeed, SGPR no longer overfits, unfortunately, as mentioned above, in this

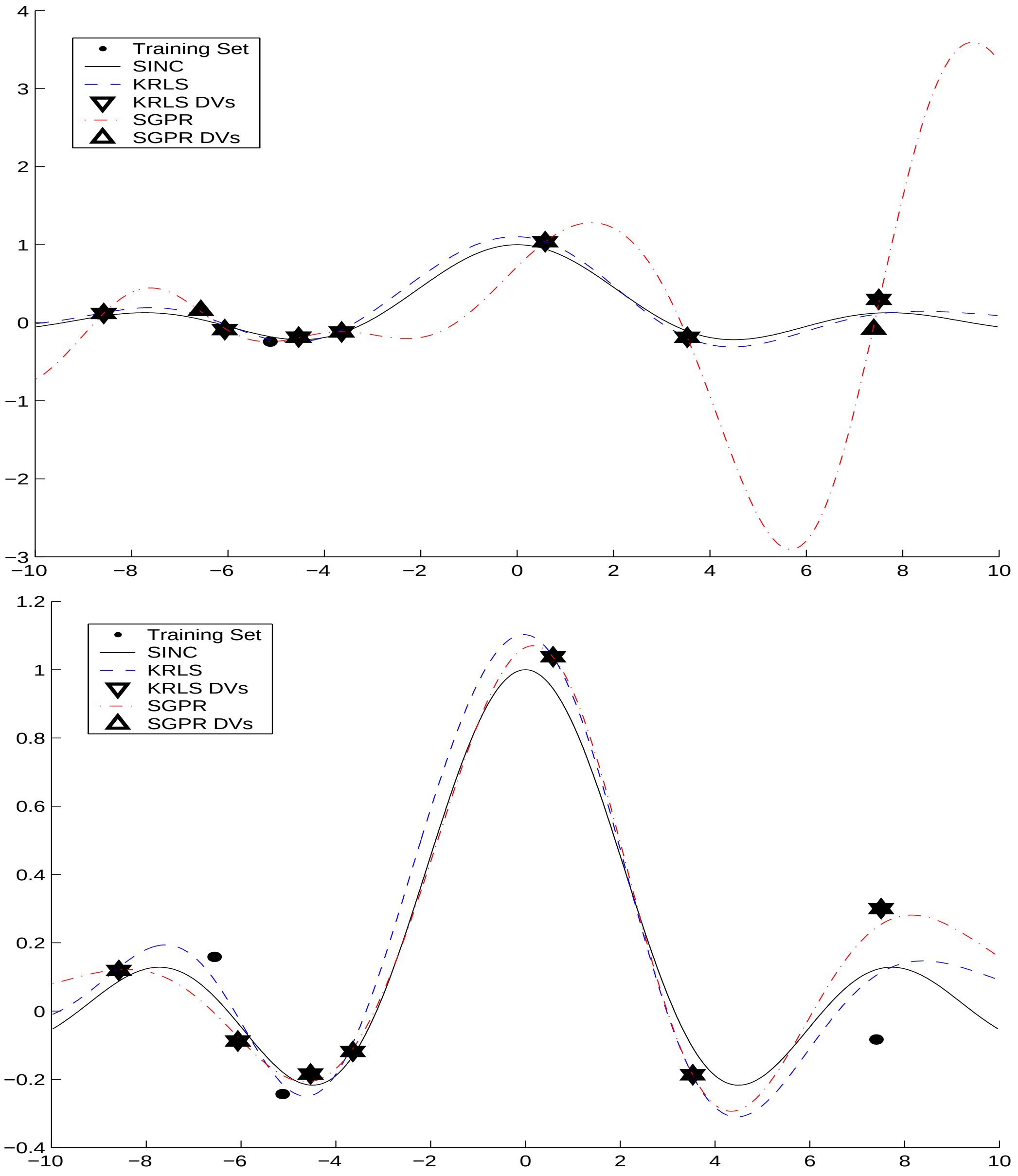


Figure 3.5: KRLS compared to SGPR with underestimated measurement noise on the *sinc* function. At the top SGPR uses its original supervised sparsification criterion, while at the bottom it uses the same unsupervised criterion used by KRLS.

case its estimate is heavily biased toward fitting the dictionary points at the expense of non-dictionary points. This effect is most pronounced in the vicinity of the two rightmost points. While the KRLS estimate lies roughly midway between the two, SGPR's estimate is much closer to the upper point, which belongs to its dictionary, almost ignoring the lower, non-dictionary point. This dictionary-bias effect may also be arrived at analytically. However, such a derivation is beyond the scope of this chapter.

3.5 Discussion and Conclusion

We have presented a nonlinear kernel-based version of the popular RLS algorithm. The algorithm requires two key ingredients: expressing all RLS related operations in terms of inner products in the feature space (which can be calculated using the kernel function in the input space), and sparsifying the data so that the dimension of the samples in the feature space remains bounded.

Our choice of unsupervised sparsification was motivated by the kind of online non-stationary learning problems that we aimed at solving at the outset (i.e. reinforcement learning problems). Choosing a supervised criterion invariably requires the user to make some extra assumptions on the measurement process that are not needed when using an unsupervised criterion. When the relevant domain knowledge is available it may be possible to capitalize on it. However, if these prior assumptions are incorrect or simply cease to apply at some stage, the results may be difficult to predict, as we demonstrated both for support vector regression in Section 3.4.1 and for sparse Gaussian process regression in Section 3.4.4.

Let us summarize the main contributions of this chapter. We presented a computationally efficient online algorithm possessing performance guarantees similar to those of the RLS algorithm (e.g. Scharf, 1991). Essentially, this means that the information content of each sample is fully extracted before that sample is discarded, since a second iteration over a previously learned training set would cause no change, whereas online gradient-based algorithms usually benefit from data recycling. Due to the unsupervised nature of our sparsification mechanism the sparsity of the solution is immune both to increase in noise and in training set size. We have also been able to present data-dependent generalization bounds that may be expressed in terms of the dictionary obtained. Finally, experiments indicate that the algorithm compares favorably with the state-of-the-art SVR algorithm SVMTorch in both generalization performance and computation time. In some cases the KRLS algorithm produces much sparser solutions with higher robustness to noise. The usefulness of KRLS was also demonstrated in the RLS algorithm's traditional application domain – signal processing. Here too, our algorithm compares favorably with current state-of-the-art algorithms.

An important future research direction would be to employ our online sparsification method, in conjunction with the kernel trick, to "kernelize" other algorithms that are recursive in nature, such as the Kalman filter (e.g., Haykin, 1996). A nonlinear kernelized version of the Kalman filter may be able to circumvent the inherent problem of handling nonlinearities, which was only partially resolved by the extended Kalman filter.

As far as the KRLS algorithm is concerned, many directions for modification and improvement are open. Further exploration into the connection of KRLS with maximum likelihood estimation is in order. Other directions include the utilization of specialized kernels tailored to specific problems, such as time series prediction; optimization of the kernel function by tuning its hyper-parameters; noise estimation (in the spirit of adaptive RLS); an exponentially weighted version of KRLS; and adaptive model order identification, as in Sayed and Kailath (1994).

Chapter 4

Reinforcement Learning with Gaussian Processes

Summary: This chapter represents the major contribution of this thesis. We start by tackling the policy evaluation problem of RL, using a GP based approach. The approach we suggest, termed Gaussian Process Temporal Difference (GPTD) learning, employs Bayesian methodology to infer a posterior distribution over value functions, conditioned on the state-reward trajectory observed while running a MDP. The GPTD framework may be used with both parametric and nonparametric representations of the value function, and applied to general MDPs with infinite state and action spaces. Various flavors of the basic model yield several different online algorithms, including a SARSA-like algorithm designed for learning state-action values, providing the basis for model-free policy improvement. Connections between the GPTD framework and existing algorithms are explored, providing a new view on familiar methods such as TD(λ) and LSTD(λ). Parts of the work presented in this chapter were published in Engel et al. (2003); Engel et al. (2005).

4.1 Introduction

The main goal of this thesis is to solve RL problems, and in this chapter this challenge is finally met. The first hurdle we clear is the basic RL problem of policy evaluation, or value estimation. Algorithms for solving this problem, such as the celebrated TD(λ), serve as a necessary algorithmic component of RL algorithms that are based on the policy iteration method (see Section 1.4.1).

Although the KRLS algorithm presented in Chapter 3 may be modified to produce a kernel-based policy evaluation algorithm, a different path is taken here. In this chapter we suggest a Gaussian process (GP) based framework, which uses linear statistical models to connect, in a probabilistic way, the underlying hidden value function with the observed rewards, for a MDP controlled by some fixed policy (see Section 1.3.2). We generically refer to this framework (as well as to some of the resulting algorithms) as *Gaussian Process Temporal Difference* (GPTD) learning.

The Bayesian paradigm, upon which GP inference is based, allows us to "invert" such statistical models by employing Bayes' rule, resulting in a *posterior distribution over value functions*, conditioned on the observations performed. Since the Normal distribution is self-conjugate, the application of Bayes' rule results in a closed form expression for the posterior (normal) distribution, given explicitly by the Gauss-Markov theorem (1.3.1). Moreover, the computation of the posterior moments is typically amenable to a sequential implementation, in which simple updates are performed for each new sample observed. We take advantage of this fact to obtain online algorithms, which maintain an exact, or almost exact, representation of the posterior distribution, at all times.

GPs are by default nonparametric (although they may also be defined parametrically, as shown in Section 1.3). This means that the search for the value function is conducted directly in a generally infinite dimensional function space. This naturally alleviates some of the limitations associated with linear architectures (discussed in Section 1.4.3), albeit at the cost of introducing some computational complications. Furthermore, by using a Mercer kernel as our basic representational object, rather than a set of basis functions, we are better equipped to solve RL problems in domains for which constructing such a basis is problematic (e.g., Kaelbling et al., 1998; M. Littman & Singh, 2002; Dzeroski et al., 1998).

On the other hand, by employing parametrically defined GPs within the GPTD framework, we are able to derive parametric variants of the same algorithms. Since these parametric algorithms conduct their search within a predefined finite-dimensional function space (spanned by a finite set of basis functions), their per-sample computational cost can be bounded in advance, resulting in efficient implementations that are comparable to the recursive implementation of the LSTD(λ) algorithm (see Section 1.4.3).

In RL, what we are really after is an optimal, or at least a good suboptimal action selection *policy*. Value estimates of states do not contain sufficient information for choosing actions without resorting to the MDP's model. A possible alternative is to learn state-action values, also known as Q-values. Once the Q-values of the current policy are known, model-free policy improvement may be performed (see Section 1.4.2). We therefore present an additional GPTD algorithm, called GPSARSA, which may be used to compute, online, posterior distributions over state-action value functions.

The rest of this chapter is organized as follows. In the following section we present the two forms of representations – parametric and nonparametric – that will be used in the sequel. Next, we describe a GPTD statistical model, as well as several corresponding algorithms, for handling MDPs with deterministic dynamics. Section 4.4 handles the general case of MDPs with stochastic transitions. Section

4.5 explores the relationship between our GPTD models and conventional temporal difference methods. Section 4.6 extends the GPTD model to the estimation of state action values, resulting in the GPSARSA algorithm. We conclude with a summary and discussion.

4.2 Parametric and Nonparametric Representations

As mentioned above, we model our subjective, extrinsic uncertainty[1] concerning the value function by modeling it as a Gaussian process V. As we have seen in Section 1.3, V may be represented using two distinct forms of representation – parametric and nonparametric. The former is based on parameterizing the value function by a finite set of random variables $\{w_1, \ldots, w_n\}$, arranged in a vector W, in such a way as to preserve the linearity of the generative model. This inevitably leads to the following parameterization

$$V(\mathbf{x}) = \sum_{j=1}^{n} \phi_j(\mathbf{x}) w_j = \boldsymbol{\phi}(\mathbf{x})^\top W, \tag{4.2.1}$$

where $\boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_n(\mathbf{x}))^\top$ is a vector of n features, which do not depend on any of the parameters w_j. The prior over W is assumed to be Gaussian, and with little loss of generality we postulate it to be distributed as

$$W \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

This induces a prior distribution over the value random process, which is also Gaussian, with easily computed moments:

$$\begin{aligned}\mathbf{E}[V(\mathbf{x})] &= \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{E}(W) = 0 \\ \mathbf{Cov}[V(\mathbf{x}), V(\mathbf{x}')] &= \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{E}[WW^\top] \boldsymbol{\phi}(\mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{x}').\end{aligned}$$

The parametric approach is most suitable in cases where domain knowledge may be used to construct a set of basis functions whose span includes good approximations of the true value function. Note that $\mathbf{E}$ denotes the expectation with respect to the extrinsic Bayesian uncertainty discussed above, and should not be confused with $\mathbf{E}_\mu$.

The alternative to the parametric approach is to define the value prior directly in function space, as explained in Section 1.3. In this case we impose a Gaussian prior over value functions, $V \sim \mathcal{N}(0, k(\cdot, \cdot))$, meaning that V is a GP for which, a

[1]This is in contrast to the *intrinsic* randomness of the MDP, which is due to the stochasticity of the state transitions and the rewards.

priori,

$$\mathbf{E}\left[V(\mathbf{x})\right] = 0, \quad \text{and } \mathbf{Cov}\left[V(\mathbf{x}), V(\mathbf{x}')\right] = k(\mathbf{x}, \mathbf{x}'), \quad \text{for all } \mathbf{x}, \mathbf{x}' \in \mathcal{X},$$

where k is a positive-definite kernel function. In the nonparametric framework, domain knowledge is expressible in the choice of the kernel function k. k should be chosen to code one's prior beliefs concerning the correlations between the values of states in the domain at hand, or, equivalently, the smoothness properties of the function space to which instances of V belong. This nonparametric approach has the advantage that, in general, it is no longer restricted to value approximations that lie in the span of any finite number of basis functions. In a sense, the basis functions are determined by the data, rather than being fixed a priori, as in the parametric approach. This benefit, however, comes at a cost: The number of parameters that need to be estimated is no longer fixed, but rather increases as the the number of data samples grows. This would seem to preclude the application of the nonparametric approach to any but the smallest of problems. However, as we shall see, this difficulty may be resolved by employing the kernel sparsification procedure, introduced in Chapter 2.

4.3 Deterministic Transitions

The general framework of Markov Decision Processes can be specialized in several ways. For instance, if the state space is restricted to a single state, Reinforcement Learning degenerates into a class of problems known as *Bandit problems*, studied thoroughly in the literature (e.g., Berry & Fristedt, 1985). Another, somewhat more complex special case, is the one in which, although the rewards may be noisy, the transitions between states are deterministic. Here, the RL problem reduces to finding optimal paths in weighted directed graphs with stochastic weights, the statistics of which are unknown (the weights correspond to the rewards, see Puterman (1994)). In this section we present a GP-based statistical generative model for value estimation in such deterministic MDPs, as well as recursive online algorithms for updating the posterior value moments.

4.3.1 A Statistical Generative Model

In the deterministic case, the Bellman equation (1.4.45), which relates values and rewards degenerates into

$$\bar{R}(\mathbf{x}) = V(\mathbf{x}) - \gamma V(\mathbf{x}') \tag{4.3.2}$$

where $\mathbf{x}'$ is the (deterministic) state succeeding $\mathbf{x}$, under the policy μ. We also assume that the noise in the rewards is independently distributed (ID) Gaussian with, possibly state dependent, variance $\sigma^2(\mathbf{x})$. Formally, this means that the reward $R(\mathbf{x})$, at some state $\mathbf{x}$, satisfies $R(\mathbf{x}) = \bar{R}(\mathbf{x}) + N(\mathbf{x})$ where $\bar{R}(\mathbf{x})$ is the expected reward for that state. The assumption on the Gaussianity of the noise process is required to preserve the analytic tractability of the resulting model. The independence assumption is due to the determinicity of the state transitions. When we treat the general case, of stochastic transitions, this assumption will have to be abandoned.

Assume we are given a sequence of rewards sampled along some trajectory $\mathbf{x}_0, \ldots, \mathbf{x}_t$, generated by the MDP from $p_0(\mathbf{x}_0)\Pi_{i=1}^t p^\mu(\mathbf{x}_i|\mathbf{x}_{i-1})$ (p_0 is an arbitrary probability distribution for the first state). Then, at the i'th time step we have $R(\mathbf{x}_i) = \bar{R}(\mathbf{x}_i) + N(\mathbf{x}_i)$. Let us aggregate the sequence of rewards corresponding to the sample path, and their respective noise terms into the vectors

$$R_t = \left(R(\mathbf{x}_0), \ldots, R(\mathbf{x}_t)\right)^\top, \quad \text{and } N_t = \left(N(\mathbf{x}_0), \ldots, N(\mathbf{x}_t)\right)^\top, \tag{4.3.3}$$

respectively. Due to our ID noise assumption, the noise vector N_{t-1} is normally distributed as

$$N_{t-1} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{\Sigma}_t\right), \quad \text{with } \mathbf{\Sigma}_t = \begin{bmatrix} \sigma_0^2 & 0 & \ldots & 0 \\ 0 & \sigma_1^2 & \ldots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma_{t-1}^2 \end{bmatrix}, \tag{4.3.4}$$

where we used the abbreviation $\sigma_i = \sigma(\mathbf{x}_i)$. Writing the Bellman equations (4.3.2) for the points belonging to the sample path, and substituting $R(\mathbf{x}_i) = \bar{R}(\mathbf{x}_i) + N(\mathbf{x}_i)$, we obtain the following set of t equations

$$R(\mathbf{x}_i) = V(\mathbf{x}_i) - \gamma V(\mathbf{x}_{i+1}) + N(\mathbf{x}_i), \quad i = 0, 1, \ldots, t-1$$

This set of linear equations may be concisely written as

$$R_{t-1} = \mathbf{H}_t V_t + N_{t-1} \tag{4.3.5}$$

where the value random vector V_t is defined by

$$V_t = \left(V(\mathbf{x}_0), \ldots, V(\mathbf{x}_t)\right)^\top, \tag{4.3.6}$$

and the $t \times (t+1)$ matrix $\mathbf{H}_t$ defined by

$$\mathbf{H}_t = \begin{bmatrix} 1 & -\gamma & 0 & \dots & 0 \\ 0 & 1 & -\gamma & \dots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \dots & 1 & -\gamma \end{bmatrix}. \tag{4.3.7}$$

Eq. 4.3.5, along with a prior distribution for V (defined either parametrically or non-parametrically, in Section 4.2), and a measurement noise distribution, completely specify a *statistical generative model* connecting the value and reward random processes. In order to infer value estimates from a sequence of actually observed rewards, the same generative model may be "inverted", so to speak, using Bayes' rule, to derive a *posterior value distribution* conditioned on the observed rewards.

Since both V_t and N_t are normally distributed, the generative model (4.3.5) belongs to the family of *linear statistical models*, discussed in Section 1.3.2. Consequently, our treatment of this model in Sections 1.3.2, and of its parametric form in Section 1.3.4, may be applied in full to our generative model, with $\mathbf{H}_t$ given by Eq. 4.3.7. Before proceeding with the derivation of the posterior distribution and algorithms for computing it, let us first consider what happens to our model equations (4.3.5) in episodic learning scenarios.

4.3.2 Episodic Tasks

Many RL tasks are episodic in nature. Whereas in continual learning tasks, in which the RL agent is placed in some (possibly random) starting state, and is then allowed to wander off indefinitely, in episodic tasks the state space is assumed to contain an absorbing, *terminal state*, into which the agent is assured to transition after a finite, but possibly random, duration of time. A terminal state may be thought of as a state from which there are only zero-reward self-transitions, for all actions. In episodic RL tasks, when such a state is reached, the episode terminates and the agent is placed in a new, usually random state to begin a new episode. For instance, in shortest-path type of problems, in which the task is to find a least-cost (maximum expected return) path to a member of a goal set, a terminal state is defined as a formal fictitious state to which all goal states transition deterministically[2].

As far as value estimation is concerned, the key property of terminal states is that, once a terminal state is reached, all subsequent rewards are zero. Therefore, both the discounted return and the value of a terminal state vanish, implying that the discounted return and the value of a goal state (or more generally, the state

[2]In shortest-path problems the discount factor γ is usually assumed to equal 1. With some possible abuse of terminology, we allow $\gamma \leq 1$.

immediately preceding the terminal state) are equal to that state's reward and expected reward, respectively. Specifically, if at time-step t a goal state is reached, the final equation in the system of equations (4.3.5) becomes

$$R(\mathbf{x}_t) = V(\mathbf{x}_t) + N(\mathbf{x}_t). \tag{4.3.8}$$

Hence, $\mathbf{H}_{t+1}$ becomes the following $(t+1) \times (t+1)$ square *invertible* matrix (its determinant equals 1),

$$\mathbf{H}_{t+1} = \begin{bmatrix} 1 & -\gamma & 0 & \dots & 0 \\ 0 & 1 & -\gamma & \dots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \dots & 1 & -\gamma \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix}. \tag{4.3.9}$$

Correspondingly, Eq. (4.3.5) becomes

$$R_t = \mathbf{H}_{t+1} V_t + N_t. \tag{4.3.10}$$

After a sequence of such learning episodes, each ending in a terminal state, $\mathbf{H}_{t+1}$ is a square block-diagonal matrix with each block, corresponding to one of the episodes, being of the form of Eq. (4.3.9).

4.3.3 Parametric GP Temporal Difference Learning

We are now in a position to write the closed form expression for the posterior moments, conditioned on an observed sequence of rewards $\mathbf{r}_{t-1} = (r_0, \dots, r_{t-1})^\top$.

In the parametric case, the posterior moments are given by Eq. 1.3.24 and 1.3.25, with Y_t replaced by R_{t-1}, $\mathbf{\Sigma}_t$ given by Eq. 4.3.4, and $\mathbf{H}_t$ given by Eq. 4.3.7:

$$\hat{\mathbf{w}}_t \stackrel{\text{def}}{=} \mathbf{E}\left[W | R_{t-1} = \mathbf{r}_{t-1}\right] = \mathbf{\Phi}_t \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{r}_{t-1}, \tag{4.3.11}$$

$$\mathbf{P}_t \stackrel{\text{def}}{=} \mathbf{Cov}\left[W | R_{t-1} = \mathbf{r}_{t-1}\right] = \mathbf{I} - \mathbf{\Phi}_t \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{H}_t \mathbf{\Phi}_t^\top, \tag{4.3.12}$$

where $\mathbf{Q}_t = \left(\mathbf{H}_t \mathbf{\Phi}_t^\top \mathbf{\Phi}_t \mathbf{H}_t^\top + \mathbf{\Sigma}_t\right)^{-1}$.

Here, $\mathbf{\Phi}_t$ is the $n \times (t+1)$ matrix

$$\mathbf{\Phi}_t = \left[\boldsymbol{\phi}(\mathbf{x}_0), \dots, \boldsymbol{\phi}(\mathbf{x}_t)\right]. \tag{4.3.13}$$

For convenience, let us define the $d \times 1$ vector $\mathbf{\Delta}\boldsymbol{\phi}_t$ and the $d \times t$ matrix $\mathbf{\Delta\Phi}_t$, by

$$\mathbf{\Delta}\boldsymbol{\phi}_t \stackrel{\text{def}}{=} \boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t), \quad \text{and } \mathbf{\Delta\Phi}_t \stackrel{\text{def}}{=} \mathbf{\Phi}_t \mathbf{H}_t^\top = \left[\mathbf{\Delta}\boldsymbol{\phi}_1, \quad \dots \quad, \mathbf{\Delta}\boldsymbol{\phi}_t\right], \tag{4.3.14}$$

respectively[3]. As we have seen in Section 1.3.4, alternative, more computationally efficient expressions (if $t > n$, where n is the number of basis functions) may be obtained:

$$\mathbf{P}_t = \left(\mathbf{\Delta\Phi}_t \mathbf{\Sigma}_t^{-1} \mathbf{\Delta\Phi}_t^\top + \mathbf{I}\right)^{-1}, \tag{4.3.15}$$

$$\hat{\mathbf{w}}_t = \mathbf{P}_t \mathbf{\Delta\Phi}_t \mathbf{\Sigma}_t^{-1} \mathbf{r}_{t-1}. \tag{4.3.16}$$

Algorithm 12 A parametric Batch-GPTD algorithm for deterministic MDPs

```
Initialize $\mathbf{B}_0 = \mathbf{0}$, $\mathbf{b}_0 = \mathbf{0}$
for $t = 1, 2, \ldots$
  observe $\mathbf{x}_{t-1}$, $r_{t-1}$, $\mathbf{x}_t$
  $\mathbf{\Delta}\boldsymbol{\phi}_t = \boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)$
  $\mathbf{B}_t = \mathbf{B}_{t-1} + \frac{1}{\sigma_{t-1}^2}\mathbf{\Delta}\boldsymbol{\phi}_t\mathbf{\Delta}\boldsymbol{\phi}_t^\top$
  $\mathbf{b}_t = \mathbf{b}_{t-1} + \frac{1}{\sigma_{t-1}^2}\mathbf{\Delta}\boldsymbol{\phi}_t r_{t-1}$
end for
return $\mathbf{P}_t = (\mathbf{B}_t + \mathbf{I})^{-1}$, $\hat{\mathbf{w}}_t = \mathbf{P}_t\hat{\mathbf{b}}_t$
```

Both the matrix $\mathbf{B}_t \overset{\text{def}}{=} \mathbf{\Delta\Phi}_t \mathbf{\Sigma}_t^{-1} \mathbf{\Delta\Phi}_t^\top$ and the vector $\mathbf{b}_t \overset{\text{def}}{=} \mathbf{\Delta\Phi}_t \mathbf{\Sigma}_t^{-1} \mathbf{r}_{t-1}$ may be computed sequentially:

$$\mathbf{B}_t = \mathbf{\Delta\Phi}_t \mathbf{\Sigma}_t^{-1} \mathbf{\Delta\Phi}_t^\top = \sum_{i=1}^{t} \frac{1}{\sigma_{i-1}^2} \mathbf{\Delta}\boldsymbol{\phi}_i \mathbf{\Delta}\boldsymbol{\phi}_i^\top = \mathbf{B}_{t-1} + \frac{1}{\sigma_{t-1}^2} \mathbf{\Delta}\boldsymbol{\phi}_t \mathbf{\Delta}\boldsymbol{\phi}_t^\top,$$

$$\mathbf{b}_t = \mathbf{\Delta\Phi}_t \mathbf{\Sigma}_t^{-1} \mathbf{r}_{t-1} = \sum_{i=1}^{t} \frac{1}{\sigma_{i-1}^2} \mathbf{\Delta}\boldsymbol{\phi}_i r_{i-1} = \mathbf{b}_{t-1} + \frac{1}{\sigma_{t-1}^2} \mathbf{\Delta}\boldsymbol{\phi}_t r_{t-1}$$

This results in our first (batch) GPTD algorithm, described in Algorithm 12.

Note that, by its construction, the posterior covariance matrix $\mathbf{P}_t$ is positive definite. Further, if the columns of $\mathbf{\Delta\Phi}_t$ span $\mathbb{R}^n$, $\mathbf{P}_t$ will tend to vanish as $t \to \infty$. Similarly to LSTD(λ), it is possible to obtain a recursive implementation of this algorithm, using the Matrix Inversion lemma (see Appendix D.4). A pseudocode description is given in Algorithm 13. The derivation of this algorithm may be found in Appendix A.1.1. As in the recursive implementation of LSTD(λ) (Algorithm 9), the term

$$\begin{aligned} d_t &= r_{t-1} - \mathbf{\Delta}\boldsymbol{\phi}_t^\top \hat{\mathbf{w}}_{t-1} \\ &= r_{t-1} - \left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)\right)^\top \hat{\mathbf{w}}_{t-1} \\ &= r_{t-1} + \gamma\hat{V}_{t-1}(\mathbf{x}_t) - \hat{V}_{t-1}(\mathbf{x}_{t-1}) \end{aligned}$$

[3] If $\mathbf{x}_t$ is the last state of an episode, then $\mathbf{\Delta}\boldsymbol{\phi}_t = \boldsymbol{\phi}(\mathbf{x}_t)$.

is the temporal difference at time t.

Algorithm 13 A recursive parametric GPTD algorithm for deterministic MDPs

```
Initialize $\hat{\mathbf{w}}_0 = \mathbf{0}$, $\mathbf{P}_0 = \mathbf{I}$
for $t = 1, 2, \ldots$
  observe $\mathbf{x}_{t-1}$, $r_{t-1}$, $\mathbf{x}_t$
  $\boldsymbol{\Delta\phi}_t = \boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)$
  $\mathbf{q}_t = \left(\sigma_{t-1}^2 + \boldsymbol{\Delta\phi}_t^\top \mathbf{P}_{t-1}\boldsymbol{\Delta\phi}_t\right)^{-1} \mathbf{P}_{t-1}\boldsymbol{\Delta\phi}_t$
  $\hat{\mathbf{w}}_t = \hat{\mathbf{w}}_{t-1} + \mathbf{q}_t \left(r_{t-1} - \boldsymbol{\Delta\phi}_t^\top \hat{\mathbf{w}}_{t-1}\right)$
  $\mathbf{P}_t = \mathbf{P}_{t-1} - \mathbf{q}_t \boldsymbol{\Delta\phi}_t^\top \mathbf{P}_{t-1}$
end for
return $\hat{\mathbf{w}}_t$, $\mathbf{P}_t$
```

A thorough comparison with the LSTD(λ) algorithm is obviously called for. Let us, however, postpone this comparison until we gain more insight on GPTD methods.

It is also possible to derive a recursive algorithm from the original, symmetric expressions for the posterior moments (4.3.11, 4.3.12). The resulting updates are:

$$\hat{\mathbf{w}}_t = \hat{\mathbf{w}}_{t-1} + \frac{1}{s_t}\mathbf{p}_t d_t, \quad \text{and } \mathbf{P}_t = \mathbf{P}_{t-1} - \frac{1}{s_t}\mathbf{p}_t\mathbf{p}_t^\top,$$

where we defined $\mathbf{p}_t = \mathbf{P}_{t-1}\boldsymbol{\Delta\phi}_t$, $s_t = \sigma_{t-1}^2 + \boldsymbol{\Delta\phi}_t^\top \mathbf{P}_{t-1}\boldsymbol{\Delta\phi}_t$, and $d_t = r_{t-1} - \boldsymbol{\Delta\phi}_t^\top \hat{\mathbf{w}}_{t-1}$. The derivation may be found in Appendix A.1.2. This algorithm is essentially the same as Algorithm 13, with $\mathbf{p}_t/s_t$ substituting for $\mathbf{q}_t$, and $\mathbf{p}_t$ substituting for $\mathbf{P}_{t-1}\boldsymbol{\Delta\phi}_t$. Its pseudocode is given in Algorithm 14.

Algorithm 14 Another recursive parametric GPTD algorithm for deterministic MDPs

```
Initialize $\hat{\mathbf{w}}_0 = \mathbf{0}$, $\mathbf{P}_0 = \mathbf{I}$
for $t = 1, 2, \ldots$
  observe $\mathbf{x}_{t-1}$, $r_{t-1}$, $\mathbf{x}_t$
  $\boldsymbol{\Delta\phi}_t = \boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)$
  $\mathbf{p}_t = \mathbf{P}_{t-1}\boldsymbol{\Delta\phi}_t$
  $s_t = \sigma_{t-1}^2 + \boldsymbol{\Delta\phi}_t^\top \mathbf{p}_t$
  $\hat{\mathbf{w}}_t = \hat{\mathbf{w}}_{t-1} + \frac{\mathbf{p}_t}{s_t}\left(r_{t-1} - \boldsymbol{\Delta\phi}_t^\top \hat{\mathbf{w}}_{t-1}\right)$
  $\mathbf{P}_t = \mathbf{P}_{t-1} - \frac{1}{s_t}\mathbf{p}_t\mathbf{p}_t^\top$
end for
return $\hat{\mathbf{w}}_t$, $\mathbf{P}_t$
```

4.3.4 Nonparametric GP Temporal Difference Learning

In the nonparametric case we define a prior distribution directly in the space of value functions. Our nonparametric generative model is therefore (see Eq. 4.3.5):

$$R_{t-1} = \mathbf{H}_t V_t + N_{t-1},$$

with a priori value and noise distributions (see Eq. 1.3.11)

$$\begin{pmatrix} V(\mathbf{x}) \\ V_t \\ N_{t-1} \end{pmatrix} \sim \mathcal{N}\left\{ \begin{pmatrix} 0 \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}, \begin{bmatrix} k(\mathbf{x},\mathbf{x}) & \mathbf{k}_t(\mathbf{x})^\top & \mathbf{0}^\top \\ \mathbf{k}_t(\mathbf{x}) & \mathbf{K}_t & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{\Sigma}_t \end{bmatrix} \right\},$$

Following the prescription given in Section 1.3.2, we make the transformation to the variables $V(\mathbf{x})$ and R_{t-1}, resulting in

$$\begin{pmatrix} V(\mathbf{x}) \\ R_{t-1} \end{pmatrix} \sim \mathcal{N}\left\{ \begin{pmatrix} 0 \\ \mathbf{0} \end{pmatrix}, \begin{bmatrix} k(\mathbf{x},\mathbf{x}) & \mathbf{k}_t(\mathbf{x})^\top \mathbf{H}_t^\top \\ \mathbf{H}_t \mathbf{k}_t(\mathbf{x}) & \mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \mathbf{\Sigma}_t \end{bmatrix} \right\}.$$

Application of the Gauss-Markov theorem yields the (marginal) posterior distribution of the value at a query point $\mathbf{x}$, conditioned on the observed sequence of rewards $\mathbf{r}_{t-1} = (r_0, \ldots, r_{t-1})^\top$:

$$(V(\mathbf{x}) | R_{t-1} = \mathbf{r}_{t-1}) \sim \mathcal{N}\left\{ \hat{V}_t(\mathbf{x}), P_t(\mathbf{x},\mathbf{x}) \right\} \ , \tag{4.3.17}$$

where

$$\begin{aligned} \hat{V}_t(\mathbf{x}) &= \mathbf{k}_t(\mathbf{x})^\top \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{r}_{t-1}, \\ P_t(\mathbf{x},\mathbf{x}') &= k(\mathbf{x},\mathbf{x}') - \mathbf{k}_t(\mathbf{x})^\top \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{H}_t \mathbf{k}_t(\mathbf{x}'), \\ \text{with} \quad \mathbf{Q}_t &= (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \mathbf{\Sigma}_t)^{-1}. \end{aligned} \tag{4.3.18}$$

The expressions above can be cast in a somewhat more concise form, by separating the input dependent terms from the learned terms:

$$\hat{V}_t(\mathbf{x}) = \boldsymbol{\alpha}_t^\top \mathbf{k}_t(\mathbf{x}), \quad P_t(\mathbf{x},\mathbf{x}') = k(\mathbf{x},\mathbf{x}') - \mathbf{k}_t(\mathbf{x})^\top \mathbf{C}_t \mathbf{k}_t(\mathbf{x}'), \tag{4.3.19}$$

where $\boldsymbol{\alpha}_t = \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{r}_{t-1}$ and $\mathbf{C}_t = \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{H}_t$ are independent of $\mathbf{x}$ and $\mathbf{x}'$.

Whereas in the parametric model the sufficient statistics required to compute the posterior moments of the value are the $n \times 1$ vector $\hat{\mathbf{w}}_t$ and the $n \times n$ matrix $\mathbf{P}_t$; here we are required to maintain a $(t+1) \times 1$ vector $\boldsymbol{\alpha}_t$ and a $(t+1) \times (t+1)$ matrix $\mathbf{C}_t$. However, before tackling the computational issues caused by the nonparametric nature of the representation, let us consider the problem of *exactly* updating the posterior parameters $\boldsymbol{\alpha}_t$ and $\mathbf{C}_t$ sequentially. The derivation of these updates may be found in Appendix A.1.3. Here we quote the results. Let us define the $(t+1)\times 1$ vector $\mathbf{h}_t$, the $t \times 1$ vector $\boldsymbol{\Delta}\mathbf{k}_t$ and the scalar Δk_{tt} by

$$\begin{aligned}
\mathbf{h}_t &= (0,\dots,1,-\gamma)^\top, \\
\boldsymbol{\Delta}\mathbf{k}_t &= \mathbf{k}_{t-1}(\mathbf{x}_{t-1}) - \gamma \mathbf{k}_{t-1}(\mathbf{x}_t), \\
\Delta k_{tt} &= k(\mathbf{x}_{t-1},\mathbf{x}_{t-1}) - 2\gamma k(\mathbf{x}_{t-1},\mathbf{x}_t) + \gamma^2 k(\mathbf{x}_t,\mathbf{x}_t),
\end{aligned}$$

respectively. Then,

$$\boldsymbol{\alpha}_t = \begin{pmatrix} \boldsymbol{\alpha}_{t-1} \\ 0 \end{pmatrix} + \frac{\mathbf{c}_t}{s_t} d_t\,, \quad \mathbf{C}_t = \begin{bmatrix} \mathbf{C}_{t-1} & ,\mathbf{0} \\ \mathbf{0}^\top & ,0 \end{bmatrix} + \frac{1}{s_t}\mathbf{c}_t\mathbf{c}_t^\top. \tag{4.3.20}$$

where

$$\begin{aligned}
\mathbf{c}_t &= \mathbf{h}_t - \begin{pmatrix} \mathbf{C}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t \\ 0 \end{pmatrix}, \\
d_t &= r_{t-1} - \boldsymbol{\Delta}\mathbf{k}_t^\top \boldsymbol{\alpha}_{t-1}, \\
s_t &= \sigma_{t-1}^2 + \Delta k_{tt} - \boldsymbol{\Delta}\mathbf{k}_t^\top \mathbf{C}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t.
\end{aligned} \tag{4.3.21}$$

Pseudocode for this algorithm is provided in Algorithm 15.

Algorithm 15 A recursive nonparametric GPTD algorithm for deterministic MDPs

```
Initialize $\boldsymbol{\alpha}_0 = 0$, $\mathbf{C}_0 = 0$
for $t = 1, 2, \dots$
  observe $\mathbf{x}_{t-1}$, $r_{t-1}$, $\mathbf{x}_t$
  $\mathbf{h}_t = (0,\dots,1,-\gamma)^\top$
  $\boldsymbol{\Delta}\mathbf{k}_t = \mathbf{k}_{t-1}(\mathbf{x}_{t-1}) - \gamma \mathbf{k}_{t-1}(\mathbf{x}_t)$
  $\Delta k_{tt} = k(\mathbf{x}_{t-1},\mathbf{x}_{t-1}) - 2\gamma k(\mathbf{x}_{t-1},\mathbf{x}_t) + \gamma^2 k(\mathbf{x}_t,\mathbf{x}_t)$
  $\mathbf{c}_t = \mathbf{h}_t - \begin{pmatrix} \mathbf{C}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t \\ 0 \end{pmatrix}$
  $d_t = r_{t-1} - \boldsymbol{\Delta}\mathbf{k}_t^\top \boldsymbol{\alpha}_{t-1}$
  $s_t = \sigma_{t-1}^2 + \Delta k_{tt} - \boldsymbol{\Delta}\mathbf{k}_t^\top \mathbf{C}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t$
  $\boldsymbol{\alpha}_t = \begin{pmatrix} \boldsymbol{\alpha}_{t-1} \\ 0 \end{pmatrix} + \frac{\mathbf{c}_t}{s_t} d_t$
  $\mathbf{C}_t = \begin{bmatrix} \mathbf{C}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t}\mathbf{c}_t\mathbf{c}_t^\top$
end for
return $\boldsymbol{\alpha}_t$, $\mathbf{C}_t$
```

4.3.5 Interpretation

The parametric value model described in Section 4.3.3 is clearly a special case of the celebrated Kalman filter (KF) (Kalman, 1960). The KF model-equations in our case are:

$$W_t = W_{t-1}, \quad \text{with } W_0 \sim \mathcal{N}\{\mathbf{0}, \mathbf{I}\}, \tag{4.3.22}$$

$$R(\mathbf{x}_{t-1}) = \mathbf{h}_t^\top \mathbf{\Phi}_t^\top W_t + N(\mathbf{x}_{t-1}), \quad \text{with } N(\mathbf{x}_{t-1}) \sim \mathcal{N}\{0, \sigma_{t-1}^2\}. \tag{4.3.23}$$

In KF terminology, W_t is the (random) *state variable*, the time evolution of which is controlled by Eq. 4.3.22, while Eq. 4.3.23 defines the dependence of the measured variables (the rewards) on the state. The temporal differences $d_t = r_{t-1} + \hat{V}_{t-1}(\mathbf{x}_t) - \hat{V}_{t-1}(\mathbf{x}_{t-1})$, appearing in both the parametric and nonparametric updates, are known as *measurement innovations*. The vector $\mathbf{q}_t = \frac{\mathbf{p}_t}{s_t}$ is referred to as the *Kalman gain*, and its nonparametric counterpart $\frac{\mathbf{c}_t}{s_t}$ plays a similar role in the nonparametric updates. Our model is a degenerate KF model, since the "state variable"[4] W has no dynamics. This suggests several ways in which our model may be extended, the discussion of which is deferred for the time being. The nonparametric case is a generalization of this KF model to an infinite dimensional state variable, namely, the value random process. We are not aware of any work extending the KF to the nonparametric case in this way; in fact, such an extension would probably be rather difficult to deal with, unless the state dynamics are of a special form, as they are in our case.

The update rules derived in Sections 4.3.3 and 4.3.4 have several interesting properties. In both the parametric and the nonparametric case, the matrix $\mathbf{Q}_t^{-1}$ is the self-covariance matrix of R_{t-1}. In the nonparametric case

$$\begin{aligned}
\mathbf{Cov}[R_{t-1}] &= \mathbf{E}[(\mathbf{H}_t V_t + N_t)(\mathbf{H}_t V_t + N_t)^\top] \\
&= \mathbf{E}[(\mathbf{H}_t V_t)(\mathbf{H}_t V_t)^\top] + \mathbf{E}[N_t N_t^\top] \\
&= \mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \mathbf{\Sigma}_t \\
&= \mathbf{Q}_t^{-1},
\end{aligned}$$

while in the parametric case $\mathbf{E}[V_t V_t^\top] = \mathbf{\Phi}_t^\top \mathbf{\Phi}_t$ and $\mathbf{Cov}[R_{t-1}] = \mathbf{H}_t \mathbf{\Phi}_t^\top \mathbf{\Phi}_t \mathbf{H}_t^\top + \mathbf{\Sigma}_t = \mathbf{Q}_t^{-1}$. Therefore, in either case, s_t is the Schur complement of the last element on the diagonal of the matrix $\mathbf{Q}_t^{-1}$ (see Appendix D). Invoking the Gauss-Markov theorem (Theorem 1.3.1) provides us with an alternative interpretation for s_t: s_t is the *posterior variance* of $R(\mathbf{x}_{t-1})$, conditioned on the previously observed values of

[4]Not to be confused with the MDP's state variable x.

$R(\mathbf{x}_0), \ldots, R(\mathbf{x}_{t-2})$, namely,

$$s_t = \mathbf{Var}\left[R(\mathbf{x}_{t-1})|R_{t-2}\right].$$

This means that s_t is positive; in fact, it is easy to show that $s_t \geq \sigma^2_{t-1}$. Finally, in the parametric case we have

$$\begin{aligned}
\mathbf{p}_t &= \mathbf{P}_{t-1}\boldsymbol{\Delta\phi}_t \\
&= \mathbf{P}_{t-1}\left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)\right) \\
&= \mathbf{Cov}\left[W, W^\top\left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)\right)|R_{t-2}\right] \\
&= \mathbf{Cov}\left[W, V(\mathbf{x}_{t-1}) - \gamma V(\mathbf{x}_t)|R_{t-2}\right].
\end{aligned}$$

Similarly, in the nonparametric case, for any query point $\mathbf{x} \in \mathcal{X}$,

$$\begin{aligned}
\mathbf{c}_t^\top\mathbf{k}_t(\mathbf{x}) &= \mathbf{h}_t^\top\mathbf{k}_t(\mathbf{x}) - \boldsymbol{\Delta}\mathbf{k}_t^\top\mathbf{C}_{t-1}\mathbf{k}_{t-1}(\mathbf{x}) \\
&= k(\mathbf{x}_{t-1}, \mathbf{x}) - \gamma k(\mathbf{x}_t, \mathbf{x}) - \mathbf{k}_{t-1}(\mathbf{x}_{t-1})^\top\mathbf{C}_{t-1}\mathbf{k}_{t-1}(\mathbf{x}) + \gamma\mathbf{k}_{t-1}(\mathbf{x}_t)^\top\mathbf{C}_{t-1}\mathbf{k}_{t-1}(\mathbf{x}) \\
&= P_{t-1}\left(\mathbf{x}, \mathbf{x}_{t-1}\right) - \gamma P_{t-1}\left(\mathbf{x}, \mathbf{x}_t\right) \\
&= \mathbf{Cov}\left[V(\mathbf{x}), V(\mathbf{x}_{t-1}) - \gamma V(\mathbf{x}_t)|R_{t-2}\right].
\end{aligned}$$

Moreover, we can show that in the parametric case (see Appendix C.2)

$$\begin{aligned}
\frac{\sigma^2_{t-1}}{s_t}\mathbf{p}_t &= \sigma^2_{t-1}\frac{\mathbf{Cov}\left[W, V(\mathbf{x}_{t-1}) - \gamma V(\mathbf{x}_t)|R_{t-2}\right]}{\mathbf{Var}\left[R(\mathbf{x}_{t-1})|R_{t-2}\right]} \\
&= \mathbf{Cov}\left[W, V(\mathbf{x}_{t-1}) - \gamma V(\mathbf{x}_t)|R_{t-1}\right] \\
&= \mathbf{Cov}\left[W, V(\mathbf{x}_{t-1}) - \gamma V(\mathbf{x}_t) - R(\mathbf{x}_{t-1})|R_{t-1}\right]. \qquad (4.3.24)
\end{aligned}$$

Whereas in the nonparametric case, for any $\mathbf{x} \in \mathcal{X}$ (see Appendix C.3),

$$\begin{aligned}
\frac{\sigma^2_{t-1}}{s_t}\mathbf{c}_t^\top\mathbf{k}_t(\mathbf{x}) &= \sigma^2_{t-1}\frac{\mathbf{Cov}\left[V(\mathbf{x}), V(\mathbf{x}_{t-1}) - \gamma V(\mathbf{x}_t)|R_{t-2}\right]}{\mathbf{Var}\left[R(\mathbf{x}_{t-1})|R_{t-2}\right]} \\
&= \mathbf{Cov}\left[V(\mathbf{x}), V(\mathbf{x}_{t-1}) - \gamma V(\mathbf{x}_t)|R_{t-1}\right] \\
&= \mathbf{Cov}\left[V(\mathbf{x}), V(\mathbf{x}_{t-1}) - \gamma V(\mathbf{x}_t) - R(\mathbf{x}_{t-1})|R_{t-1}\right]. \qquad (4.3.25)
\end{aligned}$$

Considered together, the above observations provide a rather intuitive picture of the updates of our value estimator (the posterior mean): The change in every component of either $\hat{\mathbf{w}}_t$ or $\boldsymbol{\alpha}_t$ is proportional to an error signal, given in both cases by the current temporal difference d_t. For each component, this change is modulated by a measure of the correlation between that component and the error signal, given by $\mathbf{p}_t/s_t$ or $\mathbf{c}_t/s_t$, respectively[5].

[5]The $R(\mathbf{x}_{t-1})$ terms in Eq. 4.3.24 and 4.3.25, have no effect on the covariance, as $R(\mathbf{x}_{t-1})$ is

Finally, σ_{t-1}^2/s_t may be thought of as an adaptive step-size, which is proportional to the reliability[6] attributed to the current reward measurement $R(\mathbf{x}_{t-1})$, given all previously observed rewards. Thus, all other things being equal, observations deemed unreliable will cause smaller changes to the value estimate than those considered to be more reliable.

As we progress through this chapter, we will encounter increasingly more complex algorithms (conceptually, rather than computationally). However, the temporal difference term d_t, the correlation vectors $\mathbf{p}_t$ and $\mathbf{c}_t$, and the reward posterior-variance term s_t, will continue to reappear under different guises in all of these algorithms, playing the same roles they assume in the recursive algorithms described above.

We have seen in Section 4.3.3 that, in the parametric case, the computation of the exact posterior may be performed efficiently online, in $O(n^2)$ time per sample and $O(n^2)$ memory. In the nonparametric case we have, in effect, a new feature vector component for each new sample observed, making the cost of adding the t'th sample $O(t^2)$ in both time and memory. This would seem to make the nonparametric form of GPTD computationally infeasible in all but the smallest and simplest domains. In Chapter 2 we described an efficient online kernel sparsification method. We will now invoke this method again, in order to reduce the computational costs associated with the nonparametric GPTD updates (4.3.20), to a level that would allow us to compute good approximations of the GP posterior *online*.

4.3.6 GPTD with Sparse Nonparametric Representations

We are now ready to merge the sparsification method developed in Chapter 2 into both the representation and the recursive updates of the GPTD posterior moments, derived in Section 4.3.4. Recall the sparse approximations (2.2.8, 2.2.9), which we repeat here for clarity:

$$\mathbf{K}_t \approx \mathbf{A}_t \tilde{\mathbf{K}}_t \mathbf{A}_t^\top, \quad \mathbf{k}_t(\mathbf{x}) \approx \mathbf{A}_t \tilde{\mathbf{k}}_t(\mathbf{x}).$$

Substituting these approximations into the exact GP solution (4.3.19) we obtain

$$\hat{V}_t(\mathbf{x}) = \tilde{\mathbf{k}}_t(\mathbf{x})^\top \tilde{\boldsymbol{\alpha}}_t, \quad P_t(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}') - \tilde{\mathbf{k}}_t(\mathbf{x})^\top \tilde{\mathbf{C}}_t \tilde{\mathbf{k}}_t(\mathbf{x}'), \tag{4.3.26}$$

being conditioned upon.

[6]Reliability being defined here as the inverse variance.

where we define

$$\begin{aligned}
\tilde{\mathbf{H}}_t &= \mathbf{H}_t \mathbf{A}_t, & \tilde{\mathbf{Q}}_t &= \left(\tilde{\mathbf{H}}_t \tilde{\mathbf{K}}_t \tilde{\mathbf{H}}_t^\top + \mathbf{\Sigma}_t\right)^{-1}, \\
\tilde{\boldsymbol{\alpha}}_t &= \tilde{\mathbf{H}}_t^\top \tilde{\mathbf{Q}}_t \mathbf{r}_{t-1}, & \tilde{\mathbf{C}}_t &= \tilde{\mathbf{H}}_t^\top \tilde{\mathbf{Q}}_t \tilde{\mathbf{H}}_t.
\end{aligned} \tag{4.3.27}$$

Note that the parameters we are required to store and update in order to evaluate the posterior mean and covariance are now $\tilde{\boldsymbol{\alpha}}_t$ and $\tilde{\mathbf{C}}_t$, the dimensions of which are $|\mathcal{D}_t| \times 1$ and $|\mathcal{D}_t| \times |\mathcal{D}_t|$, respectively. Recall from Chapter 2 that $\mathcal{D}_t$ is the dictionary at time t, $\tilde{\mathbf{K}}_t$ is the $|\mathcal{D}_t| \times |\mathcal{D}_t|$ kernel matrix of the dictionary members, and $\boldsymbol{a}_t$ is a $|\mathcal{D}_t| \times 1$ vector of least-squares coefficients for approximating $\boldsymbol{\phi}(\mathbf{x}_t)$ using the dictionary vectors (see Algorithm 10).

Algorithm 16 A recursive sparse GPTD algorithm for deterministic MDPs

```
Parameters: $\nu$
Initialize $\mathcal{D}_0 = \{\mathbf{x}_0\}$, $\tilde{\mathbf{K}}_0^{-1} = 1/k(\mathbf{x}_0, \mathbf{x}_0)$, $\boldsymbol{a}_0 = (1)$, $\tilde{\boldsymbol{\alpha}}_0 = 0$, $\tilde{\mathbf{C}}_0 = 0$
for $t = 1, 2, \ldots$
  observe $\mathbf{x}_{t-1}$, $r_{t-1}$, $\mathbf{x}_t$
  $\boldsymbol{a}_t = \tilde{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$
  $\delta_t = k(\mathbf{x}_t, \mathbf{x}_t) - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top \boldsymbol{a}_t$
  $\mathbf{\Delta}\tilde{\mathbf{k}}_t = \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_{t-1}) - \gamma \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$
  $d_t = r_{t-1} - \tilde{\boldsymbol{\alpha}}_{t-1}^\top \mathbf{\Delta}\tilde{\mathbf{k}}_t$
  if $\delta_t > \nu$
    compute $\tilde{\mathbf{K}}_t^{-1}$ (2.2.10)
    $\boldsymbol{a}_t = (0, \ldots, 1)^\top$
    $\tilde{\mathbf{h}}_t = (\boldsymbol{a}_{t-1}, -\gamma)^\top$
    $\Delta k_{tt} = \boldsymbol{a}_{t-1}^\top \left(\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_{t-1}) - 2\gamma \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)\right) + \gamma^2 k_{tt}$
    $\tilde{\mathbf{c}}_t = \tilde{\mathbf{h}}_t - \begin{pmatrix} \tilde{\mathbf{C}}_{t-1} \mathbf{\Delta}\tilde{\mathbf{k}}_t \\ 0 \end{pmatrix}$
    $s_t = \sigma_{t-1}^2 + \Delta k_{tt} - \mathbf{\Delta}\tilde{\mathbf{k}}_t^\top \tilde{\mathbf{C}}_{t-1} \mathbf{\Delta}\tilde{\mathbf{k}}_t$
    $\tilde{\boldsymbol{\alpha}}_{t-1} = \begin{pmatrix} \tilde{\boldsymbol{\alpha}}_{t-1} \\ 0 \end{pmatrix}$
    $\tilde{\mathbf{C}}_{t-1} = \begin{bmatrix} \tilde{\mathbf{C}}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix}$
  else
    $\tilde{\mathbf{h}}_t = \boldsymbol{a}_{t-1} - \gamma \boldsymbol{a}_t$
    $\Delta k_{tt} = \tilde{\mathbf{h}}_t^\top \mathbf{\Delta}\tilde{\mathbf{k}}_t$
    $\tilde{\mathbf{c}}_t = \tilde{\mathbf{h}}_t - \tilde{\mathbf{C}}_{t-1} \mathbf{\Delta}\tilde{\mathbf{k}}_t$
    $s_t = \sigma_{t-1}^2 + \Delta k_{tt} - \mathbf{\Delta}\tilde{\mathbf{k}}_t^\top \tilde{\mathbf{C}}_{t-1} \mathbf{\Delta}\tilde{\mathbf{k}}_t$
  end if
  $\tilde{\boldsymbol{\alpha}}_t = \tilde{\boldsymbol{\alpha}}_{t-1} + \frac{\tilde{\mathbf{c}}_t}{s_t} d_t$
  $\tilde{\mathbf{C}}_t = \tilde{\mathbf{C}}_{t-1} + \frac{1}{s_t} \tilde{\mathbf{c}}_t \tilde{\mathbf{c}}_t^\top$
end for
return $\mathcal{D}_t$, $\tilde{\boldsymbol{\alpha}}_t$, $\tilde{\mathbf{C}}_t$
```

We will now state the recursive update formulas for $\tilde{\boldsymbol{\alpha}}_t$ and $\tilde{\mathbf{C}}_t$. The complete derivation may be found in Appendix A.1.4. At each time step t we may be faced with either one of the following two cases. Either $\mathcal{D}_t = \mathcal{D}_{t-1}$ or $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$. In both cases we use the definitions

$$\boldsymbol{\Delta}\tilde{\mathbf{k}}_t = \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_{t-1}) - \gamma\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t), \quad \text{and } d_t = r_{t-1} - \tilde{\boldsymbol{\alpha}}_{t-1}^\top \boldsymbol{\Delta}\tilde{\mathbf{k}}_t.$$

Case 1. $\mathcal{D}_t = \mathcal{D}_{t-1}$:

$$\tilde{\boldsymbol{\alpha}}_t = \tilde{\boldsymbol{\alpha}}_{t-1} + \frac{\tilde{\mathbf{c}}_t}{s_t} d_t\,, \quad \tilde{\mathbf{C}}_t = \tilde{\mathbf{C}}_{t-1} + \frac{1}{s_t}\tilde{\mathbf{c}}_t\tilde{\mathbf{c}}_t^\top, \tag{4.3.28}$$

where

$$\begin{aligned} \tilde{\mathbf{c}}_t &= \tilde{\mathbf{h}}_t - \tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t \\ s_t &= \sigma_{t-1}^2 + \Delta k_{tt} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top \tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t, \end{aligned} \tag{4.3.29}$$

with the definitions

$$\begin{aligned} \tilde{\mathbf{h}}_t &= \boldsymbol{a}_{t-1} - \gamma \boldsymbol{a}_t, \\ \Delta k_{tt} &= \tilde{\mathbf{h}}_t^\top \boldsymbol{\Delta}\tilde{\mathbf{k}}_t. \end{aligned}$$

Case 2. $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$:

$$\tilde{\boldsymbol{\alpha}}_t = \begin{pmatrix} \tilde{\boldsymbol{\alpha}}_{t-1} \\ 0 \end{pmatrix} + \frac{\tilde{\mathbf{c}}_t}{s_t} d_t\,, \quad \tilde{\mathbf{C}}_t = \begin{bmatrix} \tilde{\mathbf{C}}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t}\tilde{\mathbf{c}}_t\tilde{\mathbf{c}}_t^\top, \tag{4.3.30}$$

where

$$\begin{aligned} \tilde{\mathbf{c}}_t &= \tilde{\mathbf{h}}_t - \begin{pmatrix} \tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t \\ 0 \end{pmatrix}, \\ s_t &= \sigma_{t-1}^2 + \Delta k_{tt} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top \tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t, \end{aligned} \tag{4.3.31}$$

with the definitions

$$\begin{aligned} \tilde{\mathbf{h}}_t &= \begin{pmatrix} \boldsymbol{a}_{t-1} \\ 0 \end{pmatrix} - \gamma \boldsymbol{a}_t = \begin{pmatrix} \boldsymbol{a}_{t-1} \\ -\gamma \end{pmatrix}, \\ \Delta k_{tt} &= \boldsymbol{a}_{t-1}^\top \left(\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_{t-1}) - 2\gamma\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)\right) + \gamma^2 k_{tt}. \end{aligned}$$

Pseudocode for this algorithm is provided in Algorithm 16.

4.3.7 Experiments

In this section we present several experiments meant to demonstrate the strengths and weaknesses of the deterministic GPTD algorithm. We start with a simple maze in order to demonstrate GPTD's ability to provide an uncertainty measure for the value estimate, as well as its data efficiency, i.e. its ability to extract reasonable value maps from very little data. We then move beyond value estimation to the more challenging task of learning the optimal policy (or a good approximation thereof). We use a more difficult maze and experiment with both deterministic and stochastic state transitions.

Our experimental test-bed is a continuous 2-dimensional square world with unit-length sides. An agent roaming this world may be located at any point, but can perform only a finite number of actions. The actions are 0.1-long steps in one of the 8 major compass winds, with an added Gaussian noise with a standard deviation of 0.05. Time is also discrete $t = 0, 1, 2, \ldots$. In this world there may be one or more rectangular goal regions and possibly also obstacles - piecewise linear curves, which the agent cannot cross. As long as the agent is not in a goal region it receives a reward of -1 per time-step. Upon reaching a goal state the agent is given a zero reward and is then placed at some random state to begin a new episode.

We begin with a simple experiment. The maze, shown in Fig. 4.1, has a single goal region stretching the entire length of the south wall of the maze (from y=0 to y=0.1). We chose the non-homogeneous polynomial kernel $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}' + 1)^5$, which corresponds to features that are monomials of up to degree 5 in the coordinates (Schölkopf & Smola, 2002), and subtracted 0.5 from each coordinate to avoid any asymmetric bias. The exploration policy is a stochastic one in which a southward move is taken with probability 0.8, otherwise a random move is performed. In Fig. 4.1 we show the results after a *single trial* in which 12 states were visited including a final goal state. This example demonstrates the efficiency of the algorithm when the kernel function is chosen judiciously. As can be seen at the bottom of Fig. 4.1 a single policy iteration sweep (i.e. choosing the greedy action with respect to the value function estimate) extracts a near-optimal policy for a large section of the maze surrounding the states visited. Looking at the variance map, it can be seen that in the proximity of the visited states the uncertainty in the value prediction is significantly lower than in other regions.

As already mentioned, a value estimation algorithm is usually a component in a larger RL system whose aim is to learn the *optimal* policy, namely, the one maximizing the total payoff per trial, or in our case, the one minimizing the time it takes the agent to reach a goal region. One such RL algorithm that has worked

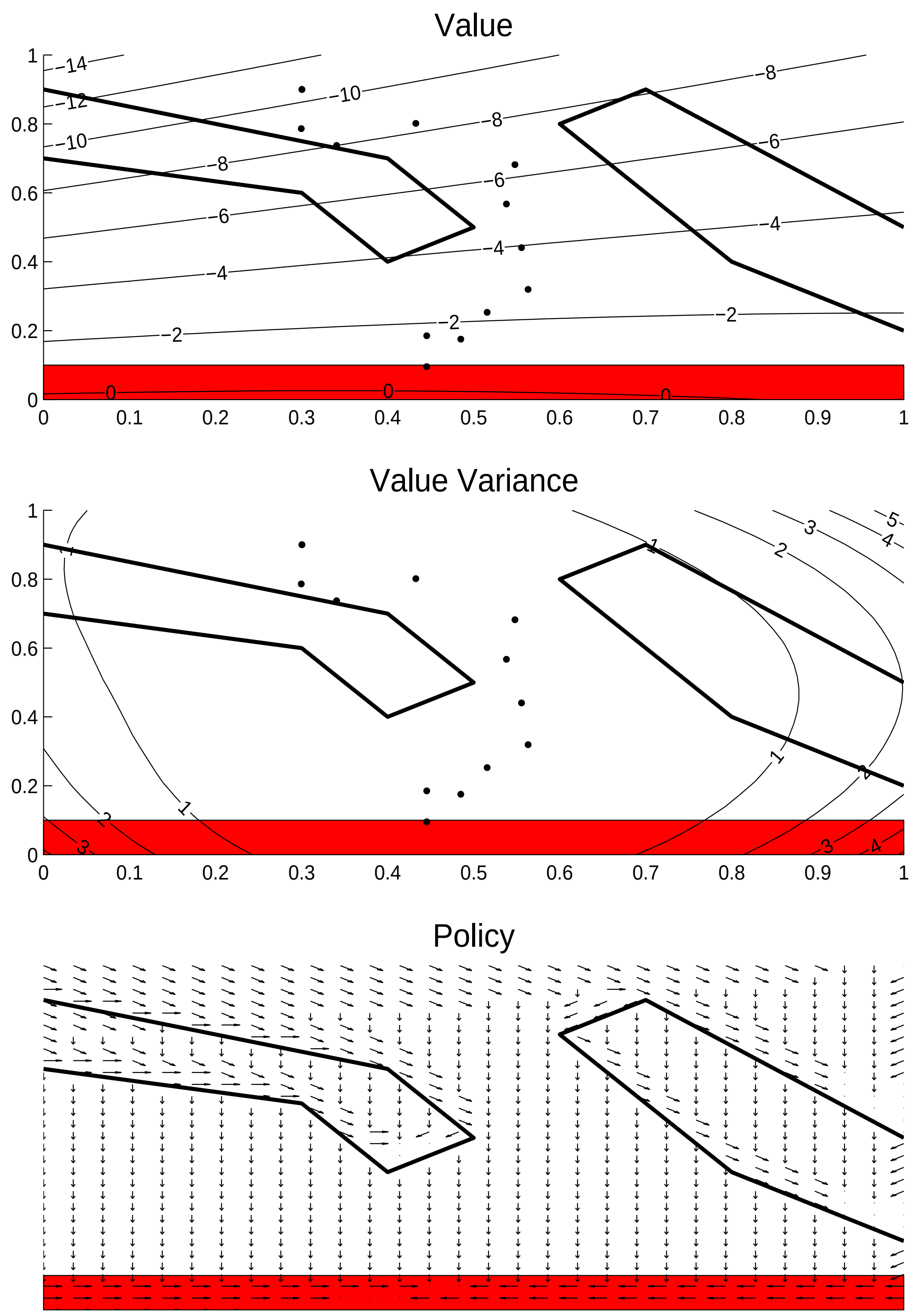


Figure 4.1: The results of a single 12-step trial on the simple maze shown in the figures, sampled on a 30 by 30 grid. From top to bottom: Top - the points visited during the trial and contour lines of the value function estimate. Center - The variance of the value estimates. Bottom - A greedy policy with respect to the value estimate.

surprisingly well in certain domains (e.g. Tesauro, 1995), although it possess no theoretical guarantees for convergence when used with function approximation, is Optimistic Policy Iteration (OPI) (Bertsekas & Tsitsiklis, 1996). In OPI the agent does not follow a fixed stationary policy; instead, at each time step it utilizes a model of its environment and its current value estimate to guess the expected payoff for each of the actions available to it. It then greedily chooses the highest ranking action, breaking ties randomly. We ran an agent utilizing OPI on the maze shown in Fig. 4.2 for 100 trials, once with deterministic transitions, and a second time with the noisy transitions described above. The kernel we used here was Gaussian $k(\mathbf{x}, \mathbf{x}') = c \exp\left(-\|\mathbf{x} - \mathbf{x}'\|^2/(2\sigma_k^2)\right)$, where $\sigma_k = 0.1$. The feature space induced by this choice is infinite-dimensional (Schölkopf & Smola, 2002). The value maps learned and their corresponding greedy policies, are shown in Fig. 4.2. Note that while the policies learned are both similar and quite close to optimal, the value estimates are different. More specifically, the value estimates in the stochastic case seem to be dampened. The variance maps are omitted, since over the entire maze the variance estimate is close to zero.

4.3.8 Remarks

This concludes our treatment of GPTD-learning in MDPs with deterministic transitions. As we saw in the previous section, when this model is used for learning the values of a stochastic MDP, the resulting value estimates are biased. Our next goal is therefore to perform GP-based value estimation in MDPs with stochastic transitions. As hinted above, we will do this by constructing an appropriate generative model that accounts for the randomness in both rewards and trajectories, thus enabling us to handle general MDPs.

4.4 Stochastic Transitions

In the first part of this chapter we derived a linear statistical model specifically geared for modeling the values of a MDP with deterministic trajectories. This model is strictly correct if the state transitions of the underlying MDP are deterministic, if the policy controlling the MDP is deterministic as well, and if the rewards are corrupted by white Gaussian noise[7]. While the latter assumption is relatively innocuous, the former two constitute a serious handicap to the applicability of the GPTD model described above to general RL problems. In the following sections we derive a (perhaps surprisingly) similar GPTD model, which allows for stochasticity in both transitions and rewards.

[7]Alternatively, if the discount factor is zero the first two assumptions are unnecessary, since the GPTD model degenerates into a simple GP regression model, with the caveat that samples are not drawn IID.

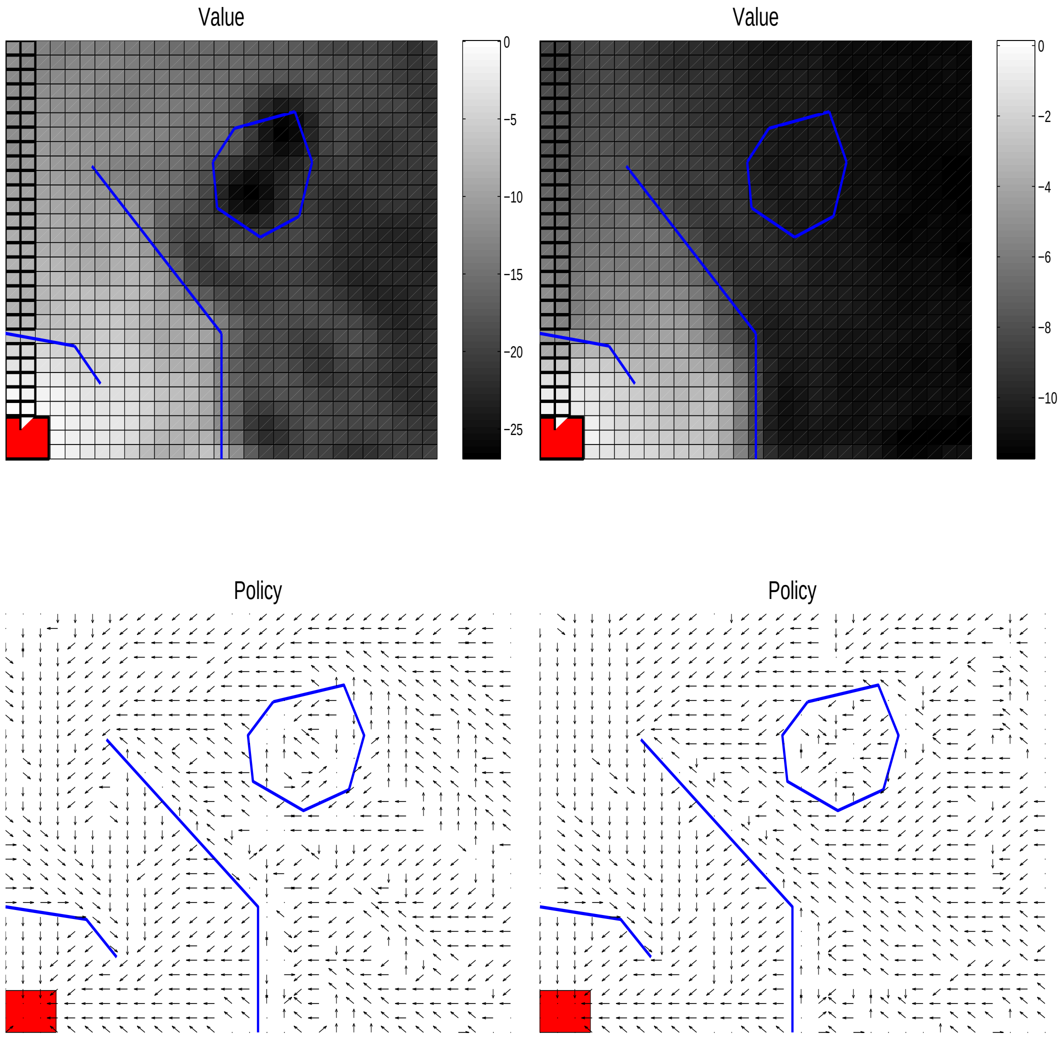


Figure 4.2: The results after 100 trials on the more difficult maze shown in the figures. GPTD with OPI is used to find a near-optimal policy. The two figures on the left show the results for deterministic state transitions while the two on the right depict the results for stochastic transitions. For each pair the final value function is shown at the top and its corresponding greedy policy at the bottom. The results shown are samples over a 30 by 30 grid.

4.4.1 A Statistical Generative Model

In Section 1.4 we showed that the value V is the result of taking the expectation of the discounted return D with respect to the randomness in the trajectory and in the rewards collected therein (Eq. 1.4.44). In the classic frequentist approach $V(\cdot)$ is no longer random, since it is the *true*, albeit unknown, value function induced by the policy μ. Adopting the Bayesian view, we may still view the value $V(\cdot)$ as a random entity by assigning it additional randomness that is due to our subjective uncertainty regarding the MDP's model $\{p, q\}$. We do not know what the true functions p and q are, which means that we are also uncertain about the true value function. We choose to model this additional *extrinsic* uncertainty by defining V as a random process indexed by the state variable $\mathbf{x}$. In this context it is useful to consider a decomposition of the discounted return into its mean (the value) and a zero-mean residual ΔV:

$$D(\mathbf{x}) = \mathbf{E}_\mu D(\mathbf{x}) + D(\mathbf{x}) - \mathbf{E}_\mu D(\mathbf{x}) = V(\mathbf{x}) + \Delta V(\mathbf{x})$$
$$\text{where } \Delta V(\mathbf{x}) \stackrel{\text{def}}{=} D(\mathbf{x}) - V(\mathbf{x}) \tag{4.4.32}$$

This decomposition is useful, since it separates the two sources of uncertainty inherent in the discounted return process D: For a known MDP model, V becomes deterministic and the randomness in D is fully attributed to the intrinsic randomness in the state-reward trajectory, modeled by ΔV. On the other hand, in a MDP in which both transitions and rewards are deterministic but otherwise unknown, ΔV becomes deterministic (i.e. identically zero), and the randomness in D is due solely to the extrinsic uncertainty, modeled by V.

Substituting Eq. 4.4.32 into Eq. 1.4.43 and rearranging we get

$$R(\mathbf{x}) = V(\mathbf{x}) - \gamma V(\mathbf{x}') + N(\mathbf{x},\mathbf{x}'), \quad \text{where } \mathbf{x}' \sim p^\mu(\cdot|\mathbf{x}), \text{ and}$$
$$N(\mathbf{x},\mathbf{x}') \stackrel{\text{def}}{=} \Delta V(\mathbf{x}) - \gamma \Delta V(\mathbf{x}'). \tag{4.4.33}$$

As before, we are provided with a sample trajectory $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_t$, and we may write the model equations (4.4.33) for these samples, resulting in the following set of t equations

$$R(\mathbf{x}_i) = V(\mathbf{x}_i) - \gamma V(\mathbf{x}_{i+1}) + N(\mathbf{x}_i, \mathbf{x}_{i+1}) \quad \text{for } i = 0, \ldots, t-1.$$

Using our standard definitions for R_t, V_t, $\mathbf{H}_t$ (Eq. 4.3.3, 4.3.6, 4.3.7), and with $N_t = (N(\mathbf{x}_0,\mathbf{x}_1), \ldots, N(\mathbf{x}_{t-1},\mathbf{x}_t))^\top$, we again have

$$R_{t-1} = \mathbf{H}_t V_t + N_t. \tag{4.4.34}$$

Eq. 4.4.34 is of the same structural form as Eq. 4.3.10 derived for the deterministic case. However, in order to fully define a complete probabilistic generative model, we need also to specify the distribution of the noise process N_t. It is here that the difference between the deterministic and the stochastic models is manifest.

4.4.2 A Correlated Noise Model

According to our custom, we model the residuals $\Delta V_t = (\Delta V(\mathbf{x}_0), \ldots, \Delta V(\mathbf{x}_t))^\top$ as a Gaussian process[8]. Particularly, this means that the distribution of the vector ΔV_t is completely specified by its mean and covariance. Another assumption we make is that each of the residuals $\Delta V(\mathbf{x}_i)$ is generated independently of all the others. We will discuss the implications of this assumption in Section 4.4.3. We are now ready to proceed with the derivation of the distribution of the noise process N_t.

By definition (Eq. 4.4.32), $\mathbf{E}_\mu[\Delta V(\mathbf{x})] = 0$ for all $\mathbf{x}$, so we have $\mathbf{E}_\mu[N(\mathbf{x}_i, \mathbf{x}_{i+1})] = 0$. Turning to the covariance, we have

$$\mathbf{E}_\mu[N(\mathbf{x}_i, \mathbf{x}_{i+1})N(\mathbf{x}_j, \mathbf{x}_{j+1})] = \mathbf{E}_\mu[(\Delta V(\mathbf{x}_i) - \gamma\Delta V(\mathbf{x}_{i+1}))(\Delta V(\mathbf{x}_j) - \gamma\Delta V(\mathbf{x}_{j+1}))].$$

According to our assumption regarding the independence of the residuals, for $i \neq j$, $\mathbf{E}_\mu\big[\Delta V(\mathbf{x}_i)\Delta V(\mathbf{x}_j)\big] = 0$. In contrast, $\mathbf{E}_\mu\big[\Delta V(\mathbf{x}_i)^2\big] = \mathbf{Var}_\mu\big[D(\mathbf{x}_i)\big]$ is generally larger than zero, unless both transitions and rewards are deterministic. Denoting $\sigma_i^2 = \mathbf{Var}\big[D(\mathbf{x}_i)\big]$, these observations may be summarized into the distribution of ΔV_t: $\Delta V_t \sim \mathcal{N}\big(\mathbf{0}, \operatorname{diag}(\boldsymbol{\sigma}_t)\big)$ where $\boldsymbol{\sigma}_t = \Big(\sigma_0^2,\ \sigma_1^2,\ \ldots\ ,\sigma_t^2\Big)^\top$, and $\operatorname{diag}(\cdot)$ denotes a diagonal matrix whose diagonal entries are the components of the argument vector. Since $N_{t-1} = \mathbf{H}_t\Delta V_t$, we have $N_{t-1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_t)$ with,

$$\begin{aligned}\boldsymbol{\Sigma}_t &= \mathbf{H}_t \operatorname{diag}(\boldsymbol{\sigma}_t)\mathbf{H}_t^\top \qquad (4.4.35)\\ &= \begin{bmatrix} \sigma_0^2 + \gamma^2\sigma_1^2 & -\gamma\sigma_1^2 & 0 & \ldots & 0 \\ -\gamma\sigma_1^2 & \sigma_1^2 + \gamma^2\sigma_2^2 & -\gamma\sigma_2^2 & \ldots & 0 \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & \ldots & & -\gamma\sigma_{t-1}^2 \\ 0 & 0 & \ldots & -\gamma\sigma_{t-1}^2 & \sigma_{t-1}^2 + \gamma^2\sigma_t^2 \end{bmatrix}.\end{aligned}$$

Let us briefly compare our new model with the deterministic GPTD model proposed in Section 4.3. Both models result in the same form of Eq. 4.4.34. However, in the deterministic GPTD model, the Gaussian noise term N_t had a diagonal covariance matrix (white noise), while in the stochastic model N_t is colored with a tri-diagonal covariance matrix. Note, also, that as the discount factor γ is reduced to zero, the two models tend to coincide. This is reasonable, since, the more strongly the future is discounted, the less it should matter whether the transitions are deterministic or stochastic. Fig. 4.3 illustrates the conditional independency

[8]This may not be a correct assumption in general; however, in the absence of any prior information concerning the distribution of the residuals, it is the *simplest* assumption we can make, since the Gaussian distribution possesses the highest entropy among all distributions with the same covariance. It is also possible to relax the Gaussianity requirement on both the prior and the noise. The resulting estimator may then be shown to be the *linear minimum mean-squared error* estimator for the value.

relations between the latent value variables $V(\mathbf{x}_i)$, the noise variables $\Delta V(\mathbf{x}_i)$, and the observable rewards $R(\mathbf{x}_i)$. Unlike GP regression, there are vertices connecting variables from different time steps, making the ordering of samples important. Also note that, for the last state in each episode ($\mathbf{x}_t$, in the figure), $R(\mathbf{x}_t)$ depends only on $V(\mathbf{x}_t)$ and $\Delta V(\mathbf{x}_t)$ (as in Eq. 4.3.8).

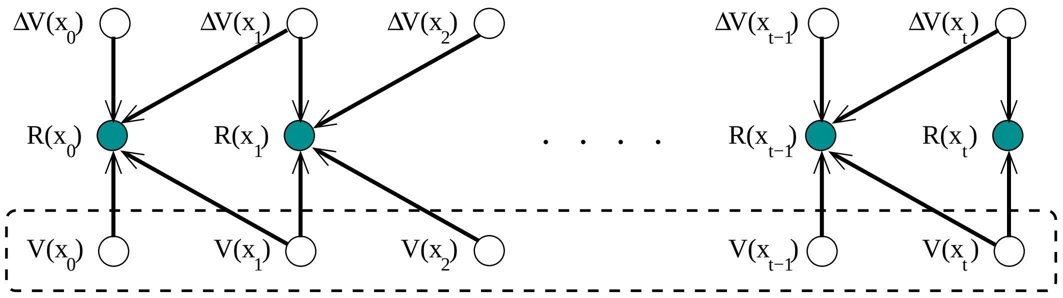


Figure 4.3: A graph illustrating the conditional independencies between the latent $V(\mathbf{x}_i)$ value variables (bottom row), the noise variables $\Delta V(\mathbf{x}_i)$ (top row), and the observable $R(\mathbf{x}_i)$ reward variables (middle row), in the GPTD model. As in the case of GP regression, all of the $V(\mathbf{x}_i)$ variables should be connected by arrows, due to the dependencies introduced by the prior. To avoid cluttering the diagram, this was marked by the dashed frame surrounding them.

4.4.3 Relation to Monte-Carlo Simulation

The assumption on the independence of the residuals made in Section 4.4.2 can be related to the well known *Monte-Carlo* method for value estimation (see Bertsekas & Tsitsiklis, 1996 Chapters 5, 6, and Sutton & Barto, 1998). Using Monte-Carlo policy evaluation reduces the problem of estimating the value function into a supervised regression problem, in which the target values for the regression are samples of the discounted return. Suppose that the last non-terminal state in the current episode is $\mathbf{x}_t$, then the Monte-Carlo training set is $(\mathbf{x}_i, y_i)_{i=0}^t$ with

$$y_i = \sum_{j=i}^{t} \gamma^{j-i} r_j, \tag{4.4.36}$$

where r_j is the reward observed at the j'th time step. The results of Section 4.3.2 concerning episodic learning tasks remain essentially unchanged in the stochastic case. The last equation in our generative model is $R(\mathbf{x}_t) = V(\mathbf{x}_t) + N(\mathbf{x}_t)$, since $V(\mathbf{x}_{t+1}) = 0$, and we therefore have

$$R_t = H_{t+1} V_t + N_t,$$

with $\mathbf{H}_{t+1}$ a square $(t+1) \times (t+1)$ matrix, as given by Eq. 4.3.9.

The validity of our model may be substantiated by performing a *whitening* transformation on Eq. (4.3.10). Since the noise covariance matrix $\mathbf{\Sigma}_t$ is positive definite, there exists a square matrix $\mathbf{\Sigma}_t^{-1/2}$ satisfying $\mathbf{\Sigma}_t^{-1/2^\top}\mathbf{\Sigma}_t^{-1/2} = \mathbf{\Sigma}_t^{-1}$. Multiplying Eq. (4.3.10) by $\mathbf{\Sigma}_t^{-1/2}$ we then get $\mathbf{\Sigma}_t^{-1/2}R_t = \mathbf{\Sigma}_t^{-1/2}\mathbf{H}_t V_t + \mathbf{\Sigma}_t^{-1/2}N_t$. The transformed noise term $\mathbf{\Sigma}_t^{-1/2}N_t$ has a covariance matrix given by $\mathbf{\Sigma}_t^{-1/2}\mathbf{\Sigma}_t\mathbf{\Sigma}_t^{-1/2^\top} = \mathbf{\Sigma}_t^{-1/2}(\mathbf{\Sigma}_t^{-1/2^\top}\mathbf{\Sigma}_t^{-1/2})^{-1}\mathbf{\Sigma}_t^{-1/2^\top} = \mathbf{I}$. Thus the transformation $\mathbf{\Sigma}_t^{-1/2}$ whitens the noise. In our case, a whitening matrix is given by

$$\mathbf{H}_{t+1}^{-1} = \begin{bmatrix} 1 & \gamma & \gamma^2 & \dots & \gamma^t \\ 0 & 1 & \gamma & \dots & \gamma^{t-1} \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \tag{4.4.37}$$

(showing that $\mathbf{H}_{t+1}^{-1}$ is indeed given by the matrix above is an easy exercise). The transformed model is $\mathbf{H}_{t+1}^{-1}R_t = V_t + N'_t$ with *white* Gaussian noise, since $N'_t = \mathbf{H}_{t+1}^{-1}N_t \sim \mathcal{N}\{\mathbf{0}, \text{diag}(\boldsymbol{\sigma}_t)\}$. Let us look at the i'th equation (i.e. row) of this transformed model:

$$R(\mathbf{x}_i) + \gamma R(\mathbf{x}_{i+1}) + \ldots + \gamma^{t-i}R(\mathbf{x}_t) = V(\mathbf{x}_i) + N'(\mathbf{x}_i),$$

where $N'(\mathbf{x}_i) \sim \mathcal{N}\{0, \sigma_i\}$. This is exactly the generative model we would use if we wanted to learn the value function by performing GP regression using Monte-Carlo samples of the discounted-return as our targets (see Section 1.3.3).

Let us denote $\mathbf{y}_t = (y_0, \ldots, y_t)^\top$, where $y_i = \sum_{j=i}^t \gamma^{j-i}r_j$. In the parametric case, assuming a constant noise variance σ^2, the posterior moments are given by (see Eq. 1.3.30, 1.3.31)

$$\begin{aligned} \hat{\mathbf{w}}_{t+1} &= \mathbf{E}\left(W | Y_t = \mathbf{y}_t\right) = \left(\mathbf{\Phi}_t\mathbf{\Phi}_t^\top + \sigma^2\mathbf{I}\right)^{-1}\mathbf{\Phi}_t\mathbf{y}_t \\ \mathbf{P}_{t+1} &= \mathbf{Cov}\left(W | Y_t = \mathbf{y}_t\right) = \sigma^2\left(\mathbf{\Phi}_t\mathbf{\Phi}_t^\top + \sigma^2\mathbf{I}\right)^{-1}. \end{aligned} \tag{4.4.38}$$

In the nonparametric case the parameters $\boldsymbol{\alpha}_{t+1}$ and $\mathbf{C}_{t+1}$ defining the posterior moments are given by (see Eq. 1.3.18)

$$\boldsymbol{\alpha}_{t+1} = \left(\mathbf{K}_t + \sigma^2\mathbf{I}\right)^{-1}\mathbf{y}_t, \quad \text{and } \mathbf{C}_{t+1} = \left(\mathbf{K}_t + \sigma^2\mathbf{I}\right)^{-1}.$$

This equivalence uncovers the implicit assumption underlying MC value estimation; namely, that the samples of the discounted return used for regression are statistically independent. In a typical online RL scenario, this assumption is clearly incorrect, as the samples of the discounted return are based on trajectories that partially overlap (e.g., for two consecutive states, $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$, the respective trajectories only differ by a single state – $\mathbf{x}_i$). This may help explain the frequently observed advantage of TD methods using $\lambda < 1$ over the corresponding Monte-Carlo (i.e. $\lambda = 1$) methods. The major benefit in using the GPTD formulation is that it immediately allows us to derive exact updates of the parameters of the posterior value mean and covariance online, rather than waiting until the end of the episode.

As we have done for the case of deterministic transitions, in the following sections we will derive a family of algorithms. The first is a batch algorithm for computing the parametric solution, while the second recursively computes the same solution. The third exactly solves the nonparametric case, and the fourth combines sparsification with the nonparametric solution. We refer to this family of algorithms as Monte-Carlo (MC) GPTD algorithms.

4.4.4 Parametric Monte-Carlo GPTD Learning

As already mentioned, in the parametric framework the value process is parameterized by $V(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top W$, and therefore

$$V_t = \boldsymbol{\Phi}_t^\top W, \quad \text{where } \boldsymbol{\Phi}_t = \Big[\boldsymbol{\phi}(\mathbf{x}_0), \quad \ldots \quad , \quad \boldsymbol{\phi}(\mathbf{x}_t)\Big]. \tag{4.4.39}$$

As in Section 4.3.3, the posterior moments are given by

$$\begin{aligned}\hat{\mathbf{w}}_t &\overset{\text{def}}{=} \mathbf{E}\left[W|R_{t-1} = \mathbf{r}_{t-1}\right] = \boldsymbol{\Delta\Phi}_t \left(\boldsymbol{\Delta\Phi}_t^\top \boldsymbol{\Delta\Phi}_t + \boldsymbol{\Sigma}_t\right)^{-1} \mathbf{r}_{t-1} \\ \mathbf{P}_t &\overset{\text{def}}{=} \mathbf{Cov}\left[W|R_{t-1} = \mathbf{r}_{t-1}\right] = \mathbf{I} - \boldsymbol{\Delta\Phi}_t \left(\boldsymbol{\Delta\Phi}_t^\top \boldsymbol{\Delta\Phi}_t + \boldsymbol{\Sigma}_t\right)^{-1} \boldsymbol{\Delta\Phi}_t^\top,\end{aligned} \tag{4.4.40}$$

where $\boldsymbol{\Delta\Phi}_t = \boldsymbol{\Phi}_t \mathbf{H}_t^\top$ (see Eq. 4.3.14). The alternative expressions for the posterior moments,

$$\hat{\mathbf{w}}_t = \left(\boldsymbol{\Delta\Phi}_t \boldsymbol{\Sigma}_t^{-1} \boldsymbol{\Delta\Phi}_t^\top + \mathbf{I}\right)^{-1} \boldsymbol{\Delta\Phi}_t \boldsymbol{\Sigma}_t^{-1} \mathbf{r}_{t-1}, \quad \text{and } \mathbf{P}_t = \left(\boldsymbol{\Delta\Phi}_t \boldsymbol{\Sigma}_t^{-1} \boldsymbol{\Delta\Phi}_t^\top + \mathbf{I}\right)^{-1}, \tag{4.4.41}$$

offer the advantage of a smaller matrix inversion problem.

In the preceding section we showed that, at the end of an episode, $\mathbf{x}_t$ being the last non-terminal state in the episode, $\mathbf{H}_{t+1}$ is a square invertible matrix. This means that

$$\boldsymbol{\Sigma}_{t+1}^{-1} = {\mathbf{H}_{t+1}^\top}^{-1} \operatorname{diag}(\boldsymbol{\sigma}_t)^{-1} {\mathbf{H}_{t+1}}^{-1}$$

Substituting this into Eq. 4.4.41, we get

$$\mathbf{P}_{t+1} = \left(\boldsymbol{\Phi}_t \operatorname{diag}(\boldsymbol{\sigma}_t)^{-1} \boldsymbol{\Phi}_t^\top + \mathbf{I}\right)^{-1}, \quad \hat{\mathbf{w}}_t = \mathbf{P}_{t+1} \boldsymbol{\Phi}_t \operatorname{diag}(\boldsymbol{\sigma}_t)^{-1} \mathbf{H}_{t+1}^{-1} \mathbf{r}_t.$$

for a constant noise variance σ^2 this becomes

$$\mathbf{P}_{t+1} = \sigma^2 \left(\mathbf{\Phi}_t \mathbf{\Phi}_t^\top + \sigma^2 \mathbf{I} \right)^{-1}, \quad \hat{\mathbf{w}}_{t+1} = \mathbf{P}_{t+1} \mathbf{\Phi}_t \mathbf{H}_{t+1}^{-1} \mathbf{r}_t.$$

Let us denote $\mathbf{B}_{t+1} = \mathbf{\Phi}_t \operatorname{diag}(\boldsymbol{\sigma}_t)^{-1} \mathbf{\Phi}_t^\top$ and $\mathbf{b}_{t+1} = \mathbf{\Phi}_t \operatorname{diag}(\boldsymbol{\sigma}_t)^{-1} \mathbf{H}_{t+1}^{-1} \mathbf{r}_t$. It is easy to verify that $\mathbf{B}_t$, and $\mathbf{b}_t$ may be obtained using the following recursions

$$\mathbf{B}_t = \mathbf{B}_{t-1} + \frac{1}{\sigma_{t-1}^2} \boldsymbol{\phi}(\mathbf{x}_{t-1}) \boldsymbol{\phi}(\mathbf{x}_{t-1})^\top, \quad \text{and } \mathbf{b}_t = \mathbf{b}_{t-1} + \frac{1}{\sigma_{t-1}^2} \mathbf{z}_{t-1} r_{t-1},$$

with $\mathbf{z}_t$ defined as in Eq. 1.4.58, for $\lambda = 1$, i.e.

$$\mathbf{z}_t = \gamma \mathbf{z}_{t-1} + \boldsymbol{\phi}(\mathbf{x}_t), \quad \text{with } \mathbf{z}_0 = \boldsymbol{\phi}(\mathbf{x}_0).$$

These sequential updates may be used to give rise to a batch algorithm, similar to the batch version of the LSTD(1) algorithm (Algorithm 8). The pseudocode is given in Algorithm 17.

Algorithm 17 A batch parametric Monte-Carlo GPTD algorithm

```
Initialize $\mathbf{B}_0 = \mathbf{0}$, $\mathbf{b}_0 = \mathbf{0}$, $\mathbf{z}_0 = \boldsymbol{\phi}(\mathbf{x}_0)$,
for $t = 1, 2, \ldots$
  observe $\mathbf{x}_{t-1}$, $r_{t-1}$, $\mathbf{x}_t$
  $\mathbf{B}_t = \mathbf{B}_{t-1} + \frac{1}{\sigma_{t-1}^2} \boldsymbol{\phi}(\mathbf{x}_{t-1}) \boldsymbol{\phi}(\mathbf{x}_{t-1})^\top$
  $\mathbf{b}_t = \mathbf{b}_{t-1} + \frac{1}{\sigma_{t-1}^2} \mathbf{z}_{t-1} r_{t-1}$
  $\mathbf{z}_t = \gamma \mathbf{z}_{t-1} + \boldsymbol{\phi}(\mathbf{x}_t)$
end for
return $\mathbf{P}_t = (\mathbf{B}_t + \mathbf{I})^{-1}$, $\hat{\mathbf{w}}_t = \mathbf{P}_t \mathbf{b}_t$
```

Although the noise covariance in the MC-GPTD model is no longer diagonal, it is nevertheless still possible to derive a recursive, online algorithm to compute the posterior moments. Rather than starting from the alternative expressions (4.3.15, 4.3.16), we base the derivation of the updates on the original, symmetric expressions of Eq. 4.4.40. The derivation is given in full in Appendix A.2.1. The resulting recursive updates are[9]:

$$\hat{\mathbf{w}}_t = \hat{\mathbf{w}}_{t-1} + \frac{1}{s_t} \mathbf{p}_t d_t, \quad \mathbf{P}_t = \mathbf{P}_{t-1} - \frac{1}{s_t} \mathbf{p}_t \mathbf{p}_t^\top, \tag{4.4.42}$$

[9]Since our model is essentially the Kalman Filter (KF) model, less the state dynamics, we could have used the measurement updates of the KF (Scharf, 1991, Chapters 7, 8) to derive the GPTD updates. This, however, is complicated by our correlated noise model, which would require us to introduce auxiliary state variables. Instead, we take the possibly longer, but more enlightening route, and derive the updates directly.

where

$$\begin{aligned}
\mathbf{p}_t &= \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\mathbf{p}_{t-1} + \mathbf{P}_{t-1}\left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)\right) \\
d_t &= \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}d_{t-1} + r_{t-1} - \left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)\right)^\top \hat{\mathbf{w}}_{t-1} \\
s_t &= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 - \frac{\gamma^2\sigma_{t-1}^4}{s_{t-1}} + \left(\mathbf{p}_t + \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\mathbf{p}_{t-1}\right)^\top \left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)\right)
\end{aligned} \tag{4.4.43}$$

It can be easily verified that these recursions should be initialized as follows:

$$\hat{\mathbf{w}}_0 = \mathbf{0}, \quad \mathbf{P}_0 = \mathbf{I}, \quad \mathbf{p}_0 = \mathbf{0}, \quad d_0 = 0, \quad 1/s_0 = 0.$$

Algorithm 18 provides the pseudocode for this algorithm.

Algorithm 18 A recursive parametric Monte-Carlo GPTD algorithm

```
**Initialize** $\hat{\mathbf{w}}_0 = \mathbf{0}$, $\mathbf{P}_0 = \mathbf{I}$, $\mathbf{p}_0 = \mathbf{0}$, $d_0 = 0$, $1/s_0 = 0$
**for** $t = 1, 2, \ldots$
  **observe** $\mathbf{x}_{t-1}$, $r_{t-1}$, $\mathbf{x}_t$
  $\boldsymbol{\Delta\phi}_t = \boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)$
  $\mathbf{p}_t = \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\mathbf{p}_{t-1} + \mathbf{P}_{t-1}\boldsymbol{\Delta\phi}_t$
  $d_t = \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}d_{t-1} + r_{t-1} - \boldsymbol{\Delta\phi}_t^\top\hat{\mathbf{w}}_{t-1}$
  $s_t = \sigma_{t-1}^2 + \gamma^2\sigma_t^2 - \frac{\gamma^2\sigma_{t-1}^4}{s_{t-1}} + \left(\mathbf{p}_t + \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\mathbf{p}_{t-1}\right)^\top \boldsymbol{\Delta\phi}_t$
  $\hat{\mathbf{w}}_t = \hat{\mathbf{w}}_{t-1} + \frac{1}{s_t}\mathbf{p}_t d_t$
  $\mathbf{P}_t = \mathbf{P}_{t-1} - \frac{1}{s_t}\mathbf{p}_t\mathbf{p}_t^\top$
**end for**
**return** $\hat{\mathbf{w}}_t$, $\mathbf{P}_t$
```

4.4.5 Nonparametric Monte-Carlo GPTD Learning

Recall that in the nonparametric framework we bypass the parameterization of the value process by placing a prior directly in the space of value functions. From Section 4.3.4, we know already that the posterior mean and covariance are given, respectively, by

$$\hat{V}_t(\mathbf{x}) = \boldsymbol{\alpha}_t^\top\mathbf{k}_t(\mathbf{x}), \quad P_t(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}') - \mathbf{k}_t(\mathbf{x})^\top\mathbf{C}_t\mathbf{k}_t(\mathbf{x}'),$$

where $\boldsymbol{\alpha}_t = \mathbf{H}_t^\top\mathbf{Q}_t\mathbf{r}_{t-1}$, $\mathbf{C}_t = \mathbf{H}_t^\top\mathbf{Q}_t\mathbf{H}_t$, and $\mathbf{Q}_t = (\mathbf{H}_t\mathbf{K}_t\mathbf{H}_t^\top + \boldsymbol{\Sigma}_t)^{-1}$. As in the parametric case, it is possible to derive recursive updates for $\boldsymbol{\alpha}_t$ and $\mathbf{C}_t$. The complete derivation may be found in Appendix A.2.2. The updates are:

$$\boldsymbol{\alpha}_t = \begin{pmatrix} \boldsymbol{\alpha}_{t-1} \\ 0 \end{pmatrix} + \frac{\mathbf{c}_t}{s_t} d_t, \quad \mathbf{C}_t = \begin{bmatrix} \mathbf{C}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t} \mathbf{c}_t \mathbf{c}_t^\top$$

with

$$\begin{aligned} d_t &= \frac{\gamma \sigma_{t-1}^2}{s_{t-1}} d_{t-1} + r_{t-1} - \Delta \mathbf{k}_t^\top \boldsymbol{\alpha}_{t-1}, \\ \mathbf{c}_t &= \frac{\gamma \sigma_{t-1}^2}{s_{t-1}} \begin{pmatrix} \mathbf{c}_{t-1} \\ 0 \end{pmatrix} + \mathbf{h}_t - \begin{pmatrix} \mathbf{C}_{t-1} \Delta \mathbf{k}_t \\ 0 \end{pmatrix}, \\ s_t &= \sigma_{t-1}^2 + \gamma^2 \sigma_t^2 - \frac{\gamma^2 \sigma_{t-1}^4}{s_{t-1}} + \Delta k_{tt} - \Delta \mathbf{k}_t^\top \mathbf{C}_{t-1} \Delta \mathbf{k}_t + \frac{2\gamma \sigma_{t-1}^2}{s_{t-1}} \mathbf{c}_{t-1}^\top \Delta \mathbf{k}_t. \end{aligned}$$

Above we made use of some definitions made in Section 4.3.4, which we repeat here for clarity:

$$\begin{aligned} \mathbf{h}_t &= (0, \ldots, 1, -\gamma)^\top, \\ \boldsymbol{\Delta} \mathbf{k}_t &= \mathbf{k}_{t-1}(\mathbf{x}_{t-1}) - \gamma \mathbf{k}_{t-1}(\mathbf{x}_t), \\ \Delta k_{tt} &= k(\mathbf{x}_{t-1}, \mathbf{x}_{t-1}) - 2\gamma k(\mathbf{x}_{t-1}, \mathbf{x}_t) + \gamma^2 k(\mathbf{x}_t, \mathbf{x}_t). \end{aligned}$$

It may be readily verified that these recursions should be initialized as follows:

$$\boldsymbol{\alpha}_0 = 0, \quad \mathbf{C}_0 = 0, \quad \mathbf{c}_0 = 0, \quad d_0 = 0, \quad 1/s_0 = 0.$$

The pseudocode for this algorithm is given in Algorithm 19.

4.4.6 Sparse Nonparametric Monte-Carlo GPTD Learning

The fourth member of the family of MC-GPTD algorithms combines the sparsification method of Chapter 2 with the nonparametric method of Section 4.4.5. Recall from Section 4.3.6 the expressions for the posterior moments (4.3.26):

$$\hat{V}_t(\mathbf{x}) = \tilde{\mathbf{k}}_t(\mathbf{x})^\top \tilde{\boldsymbol{\alpha}}_t, \quad \text{and } P_t(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}') - \tilde{\mathbf{k}}_t(\mathbf{x})^\top \tilde{\mathbf{C}}_t \tilde{\mathbf{k}}_t(\mathbf{x}'),$$

where

$$\begin{aligned} \tilde{\boldsymbol{\alpha}}_t &= \tilde{\mathbf{H}}_t^\top \tilde{\mathbf{Q}}_t \mathbf{r}_{t-1}, & \tilde{\mathbf{C}}_t &= \tilde{\mathbf{H}}_t^\top \tilde{\mathbf{Q}}_t \tilde{\mathbf{H}}_t, \\ \tilde{\mathbf{H}}_t &= \mathbf{H}_t \mathbf{A}_t, & \tilde{\mathbf{Q}}_t &= \left(\tilde{\mathbf{H}}_t \tilde{\mathbf{K}}_t \tilde{\mathbf{H}}_t^\top + \boldsymbol{\Sigma}_t \right)^{-1}. \end{aligned}$$

When compared with the discussion of Section 4.3.6, the only difference lies in the form of the noise covariance $\boldsymbol{\Sigma}_t$, which now is given by the tridiagonal matrix of

Algorithm 19 A recursive nonparametric Monte-Carlo GPTD algorithm

```
Initialize $\boldsymbol{\alpha}_0 = \mathbf{0}$, $\mathbf{C}_0 = 0$, $\mathbf{p}_0 = \mathbf{0}$, $d_0 = 0$, $1/s_0 = 0$
for $t = 1, 2, \ldots$
  observe $\mathbf{x}_{t-1}$, $r_{t-1}$, $\mathbf{x}_t$
  $\mathbf{h}_t = (0, \ldots, 1, -\gamma)^\top$
  $\boldsymbol{\Delta}\mathbf{k}_t = \mathbf{k}_{t-1}(\mathbf{x}_{t-1}) - \gamma \mathbf{k}_{t-1}(\mathbf{x}_t)$
  $\Delta k_{tt} = k(\mathbf{x}_{t-1}, \mathbf{x}_{t-1}) - 2\gamma k(\mathbf{x}_{t-1}, \mathbf{x}_t) + \gamma^2 k(\mathbf{x}_t, \mathbf{x}_t)$
  $d_t = \frac{\gamma \sigma_{t-1}^2}{s_{t-1}} d_{t-1} + r_{t-1} - \boldsymbol{\Delta}\mathbf{k}_t^\top \boldsymbol{\alpha}_{t-1}$
  $\mathbf{c}_t = \frac{\gamma \sigma_{t-1}^2}{s_{t-1}} \begin{pmatrix} \mathbf{c}_{t-1} \\ 0 \end{pmatrix} + \mathbf{h}_t - \begin{pmatrix} \mathbf{C}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t \\ 0 \end{pmatrix}$
  $s_t = \sigma_{t-1}^2 + \gamma^2 \sigma_t^2 - \frac{\gamma^2 \sigma_{t-1}^4}{s_{t-1}} + \Delta k_{tt} - \boldsymbol{\Delta}\mathbf{k}_t^\top \mathbf{C}_{t-1} \boldsymbol{\Delta}\mathbf{k}_t + \frac{2\gamma \sigma_{t-1}^2}{s_{t-1}} \mathbf{c}_{t-1}^\top \boldsymbol{\Delta}\mathbf{k}_t$
  $\boldsymbol{\alpha}_t = \begin{pmatrix} \boldsymbol{\alpha}_{t-1} \\ 0 \end{pmatrix} + \frac{\mathbf{c}_t}{s_t} d_t$
  $\mathbf{C}_t = \begin{bmatrix} \mathbf{C}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t} \mathbf{c}_t \mathbf{c}_t^\top$
end for
return $\boldsymbol{\alpha}_t$, $\mathbf{C}_t$
```

Eq. 4.4.35.

Recall from Chapter 2 that $\mathcal{D}_t$ is the dictionary at time t, $\tilde{\mathbf{K}}_t$ is the $|\mathcal{D}_t| \times |\mathcal{D}_t|$ kernel matrix of the dictionary members, and $\boldsymbol{a}_t$ is a $|\mathcal{D}_t| \times 1$ vector of least-squares coefficients for approximating $\boldsymbol{\phi}(\mathbf{x}_t)$ with the dictionary vectors (see Algorithm 10).

Let us state the recursive update formulas for $\tilde{\boldsymbol{\alpha}}_t$ and $\tilde{\mathbf{C}}_t$. The complete derivation may be found in Section A.2.3. As usual, at each time step t we may be faced with either one of the following two cases. Either $\mathcal{D}_t = \mathcal{D}_{t-1}$ or $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$. In either case we use the definition

$$\boldsymbol{\Delta}\tilde{\mathbf{k}}_t = \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_{t-1}) - \gamma \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t).$$

Case 1. $\mathcal{D}_t = \mathcal{D}_{t-1}$:

$$\tilde{\boldsymbol{\alpha}}_t = \tilde{\boldsymbol{\alpha}}_{t-1} + \frac{\tilde{\mathbf{c}}_t}{s_t} d_t\,, \quad \tilde{\mathbf{C}}_t = \tilde{\mathbf{C}}_{t-1} + \frac{1}{s_t} \tilde{\mathbf{c}}_t \tilde{\mathbf{c}}_t^\top, \tag{4.4.44}$$

where

$$\tilde{\mathbf{c}}_t = \frac{\gamma \sigma_{t-1}^2}{s_{t-1}} \tilde{\mathbf{c}}_{t-1} + \tilde{\mathbf{h}}_t - \tilde{\mathbf{C}}_{t-1} \boldsymbol{\Delta}\tilde{\mathbf{k}}_t$$

$$d_t = \frac{\gamma \sigma_{t-1}^2}{s_{t-1}} d_{t-1} + r_{t-1} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top \tilde{\boldsymbol{\alpha}}_{t-1}$$

$$s_t = \sigma_{t-1}^2 + \gamma^2 \sigma_t^2 + \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top \left(\tilde{\mathbf{c}}_t + \frac{\gamma \sigma^2}{s_{t-1}} \tilde{\mathbf{c}}_{t-1} \right) - \frac{\gamma^2 \sigma_{t-1}^4}{s_{t-1}}, \tag{4.4.45}$$

Algorithm 20 A sparse recursive nonparametric Monte-Carlo GPTD algorithm

```
Parameters: $\nu$
Initialize $\mathcal{D}_0 = \{\mathbf{x}_0\}$, $\tilde{\mathbf{K}}_0^{-1} = 1/k(\mathbf{x}_0,\mathbf{x}_0)$, $\boldsymbol{a}_0 = (1)$, $\tilde{\boldsymbol{\alpha}}_0 = 0$, $\tilde{\mathbf{C}}_0 = 0$, $\tilde{\mathbf{c}}_0 = 0$, $d_0 = 0$, $1/s_0 = 0$
for $t = 1, 2, \ldots$
  observe $\mathbf{x}_{t-1}$, $r_{t-1}$, $\mathbf{x}_t$
  $\boldsymbol{a}_t = \tilde{\mathbf{K}}_{t-1}^{-1}\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$
  $\delta_t = k(\mathbf{x}_t,\mathbf{x}_t) - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top \boldsymbol{a}_t$
  $\boldsymbol{\Delta}\tilde{\mathbf{k}}_t = \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_{t-1}) - \gamma\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$
  $d_t = \frac{\gamma\sigma_{t-1}^2}{s_{t-1}} d_{t-1} + r_{t-1} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top \tilde{\boldsymbol{\alpha}}_{t-1}$
  if $\delta_t > \nu$
    compute $\tilde{\mathbf{K}}_t^{-1}$ (2.2.10)
    $\boldsymbol{a}_t = (0, \ldots, 1)^\top$
    $\tilde{\mathbf{h}}_t = (\boldsymbol{a}_{t-1}, -\gamma)^\top$
    $\Delta k_{tt} = \boldsymbol{a}_{t-1}^\top \left(\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_{t-1}) - 2\gamma\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)\right) + \gamma^2 k_{tt}$
    $\tilde{\mathbf{c}}_t = \frac{\gamma\sigma_{t-1}^2}{s_{t-1}} \begin{pmatrix}\tilde{\mathbf{c}}_{t-1} \\ 0\end{pmatrix} + \tilde{\mathbf{h}}_t - \begin{pmatrix}\tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t \\ 0\end{pmatrix}$
    $s_t = \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \Delta k_{tt} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top \tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t + \frac{2\gamma\sigma_{t-1}^2}{s_{t-1}}\tilde{\mathbf{c}}_{t-1}^\top \boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \frac{\gamma^2\sigma_{t-1}^4}{s_{t-1}}$
    $\tilde{\boldsymbol{\alpha}}_{t-1} = \begin{pmatrix}\tilde{\boldsymbol{\alpha}}_{t-1} \\ 0\end{pmatrix}$
    $\tilde{\mathbf{C}}_{t-1} = \begin{bmatrix}\tilde{\mathbf{C}}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0\end{bmatrix}$
  else
    $\tilde{\mathbf{h}}_t = \boldsymbol{a}_{t-1} - \gamma\boldsymbol{a}_t$
    $\Delta k_{tt} = \tilde{\mathbf{h}}_t^\top \boldsymbol{\Delta}\tilde{\mathbf{k}}_t$
    $\tilde{\mathbf{c}}_t = \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\tilde{\mathbf{c}}_t + \tilde{\mathbf{h}}_t - \tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t$
    $s_t = \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top \left(\tilde{\mathbf{c}}_t + \frac{\gamma\sigma^2}{s_{t-1}}\tilde{\mathbf{c}}_{t-1}\right) - \frac{\gamma^2\sigma_{t-1}^4}{s_{t-1}}$
  end if
    $\tilde{\boldsymbol{\alpha}}_t = \tilde{\boldsymbol{\alpha}}_{t-1} + \frac{\tilde{\mathbf{c}}_t}{s_t} d_t$
    $\tilde{\mathbf{C}}_t = \tilde{\mathbf{C}}_{t-1} + \frac{1}{s_t}\tilde{\mathbf{c}}_t\tilde{\mathbf{c}}_t^\top$
end for
return $\mathcal{D}_t$, $\tilde{\boldsymbol{\alpha}}_t$, $\tilde{\mathbf{C}}_t$
```

with the definitions

$$\begin{aligned}\tilde{\mathbf{h}}_t &= \boldsymbol{a}_{t-1} - \gamma \boldsymbol{a}_t, \\ \Delta k_{tt} &= \tilde{\mathbf{h}}_t^\top \boldsymbol{\Delta}\tilde{\mathbf{k}}_t.\end{aligned}$$

Case 2. $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$:

$$\tilde{\boldsymbol{\alpha}}_t = \begin{pmatrix} \tilde{\boldsymbol{\alpha}}_{t-1} \\ 0 \end{pmatrix} + \frac{\tilde{\mathbf{c}}_t}{s_t} d_t \,, \quad \tilde{\mathbf{C}}_t = \begin{bmatrix} \tilde{\mathbf{C}}_{t-1} & , \mathbf{0} \\ \mathbf{0}^\top & , 0 \end{bmatrix} + \frac{1}{s_t} \tilde{\mathbf{c}}_t \tilde{\mathbf{c}}_t^\top \,, \tag{4.4.46}$$

where

$$\begin{aligned}\tilde{\mathbf{c}}_t &= \frac{\gamma \sigma_{t-1}^2}{s_{t-1}} \begin{pmatrix} \tilde{\mathbf{c}}_{t-1} \\ 0 \end{pmatrix} + \tilde{\mathbf{h}}_t - \begin{pmatrix} \tilde{\mathbf{C}}_{t-1} \boldsymbol{\Delta}\tilde{\mathbf{k}}_t \\ 0 \end{pmatrix}, \\ d_t &= \frac{\gamma \sigma_{t-1}^2}{s_{t-1}} d_{t-1} + r_{t-1} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top \tilde{\boldsymbol{\alpha}}_{t-1}, \\ s_t &= \sigma_{t-1}^2 + \gamma^2 \sigma_t^2 + \Delta k_{tt} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top \tilde{\mathbf{C}}_{t-1} \boldsymbol{\Delta}\tilde{\mathbf{k}}_t + \frac{2\gamma \sigma_{t-1}^2}{s_{t-1}} \tilde{\mathbf{c}}_{t-1}^\top \boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \frac{\gamma^2 \sigma_{t-1}^4}{s_{t-1}},\end{aligned} \tag{4.4.47}$$

with the definitions

$$\begin{aligned}\tilde{\mathbf{h}}_t &= \begin{pmatrix} \boldsymbol{a}_{t-1} \\ 0 \end{pmatrix} - \gamma \boldsymbol{a}_t = \begin{pmatrix} \boldsymbol{a}_{t-1} \\ -\gamma \end{pmatrix}, \\ \Delta k_{tt} &= \boldsymbol{a}_{t-1}^\top \left(\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_{t-1}) - 2\gamma \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) \right) + \gamma^2 k_{tt}.\end{aligned}$$

The pseudocode for this algorithm is provided in Algorithm 20.

4.5 Connections with Other TD Methods

In Section 4.3, we derived several GP-based algorithms for learning a posterior distribution over value functions, for MDPs with stochastic rewards, but deterministic transitions. This was done by using a generative model of the form of (4.3.5), in which the covariance of the noise term was diagonal: $\boldsymbol{\Sigma}_t = \text{diag}(\boldsymbol{\sigma}_{t-1})$. In Section 4.4 we derived a second GPTD model that overcomes the limitation of the first algorithm to deterministic transitions. We did this by invoking a useful decomposition of the discounted return random process into the value process, modeling our uncertainty concerning the MDP's model, and a zero-mean residual (4.4.32), modeling the MDP's intrinsic stochasticity; and by additionally assuming independence of the residuals. Surprisingly, all that this amounts to is the replacement of the diagonal noise covariance with a tridiagonal, correlated noise covariance:

$\mathbf{\Sigma}_t = \mathbf{H}_t \operatorname{diag}(\boldsymbol{\sigma}_t)\mathbf{H}_t^\top$. This change induces a model, which we have shown to be effectively equivalent to GP regression on Monte-Carlo samples of the discounted return.

We are therefore inclined to adopt a broader view of GPTD as a general GP-based framework for Bayesian modeling of value functions, encompassing all generative models of the form $R_{t-1} = \mathbf{H}_t V_t + N_{t-1}$, with $\mathbf{H}_t$ given by (4.3.7) (or (4.3.9) at the end of an episode), a Gaussian prior placed on V, and an arbitrary zero-mean Gaussian noise process N. No doubt, most such models will be meaningless from a value estimation point of view, while others will not admit efficient recursive algorithms for computing the posterior value moments. However, if the noise covariance $\mathbf{\Sigma}_t$ is suitably chosen, and if it is additionally *simple* in some way, then we may be able to derive such a recursive algorithm to compute complete posterior value distributions, on-line. In this section we show that by employing alternative forms of noise covariance, we are able to obtain GP-based variants of LSTD(λ) (Bradtke & Barto, 1996; Boyan, 1999a).

4.5.1 A Maximum Likelihood Variant

In certain cases, one may prefer to forgo specifying a prior over the weight vector W. In such cases, one can no longer perform Bayesian analysis, but it is nevertheless still possible to perform classical maximum-likelihood (ML) inference to find the value of W, $\hat{\mathbf{w}}^{ML}$, for which the observed data is most likely, according to the generative model of Eq. (4.4.34). Due to the Gaussianity assumption, ignoring terms independent of W, the log-likelihood of the sequence of measured rewards is

$$\log \Pr(R_{t-1} = \mathbf{r}_{t-1}|W) \propto -\left(\mathbf{r}_{t-1} - \mathbf{\Delta\Phi}_t^\top W\right)^\top \mathbf{\Sigma}_t^{-1} \left(\mathbf{r}_{t-1} - \mathbf{\Delta\Phi}_t^\top W\right). \quad (4.5.48)$$

The maximum-likelihood (ML) problem therefore reduces to the minimization of a simple quadratic form:

$$\min_{\mathbf{w}} \left\{ \left(\mathbf{r}_{t-1} - \mathbf{\Delta\Phi}_t^\top \mathbf{w}\right)^\top \mathbf{\Sigma}_t^{-1} \left(\mathbf{r}_{t-1} - \mathbf{\Delta\Phi}_t^\top \mathbf{w}\right) \right\}, \quad (4.5.49)$$

the solution of which is given by (recall that $\mathbf{\Delta\Phi}_t = \mathbf{\Phi}_t \mathbf{H}_t^\top$)

$$\hat{\mathbf{w}}_t^{ML} = \left(\mathbf{\Phi}_t \mathbf{H}_t^\top \mathbf{\Sigma}_t^{-1} \mathbf{H}_t \mathbf{\Phi}_t^\top\right)^{-1} \mathbf{\Phi}_t \mathbf{H}_t^\top \mathbf{\Sigma}_t^{-1} \mathbf{r}_{t-1}. \quad (4.5.50)$$

As we have seen in Section 4.4.3, at the end of an episode $\mathbf{H}_t$ is invertible and is given by Eq. 4.4.37. The inverse noise covariance, assuming for simplicity a constant noise variance σ^2, is given by

$$\mathbf{\Sigma}_t^{-1} = \frac{1}{\sigma^2} \mathbf{H}_{t+1}^{\top}{}^{-1} \mathbf{H}_{t+1}{}^{-1}.$$

Therefore, Eq. 4.5.50 becomes

$$\hat{\mathbf{w}}_{t+1}^{ML} = \left(\mathbf{\Phi}_t \mathbf{\Phi}_t^\top\right)^{-1} \mathbf{\Phi}_t \mathbf{H}_{t+1}^{-1} \mathbf{r}_t \tag{4.5.51}$$

From the formula for $\mathbf{H}_{t+1}^{-1}$ we infer that $\mathbf{H}_{t+1}^{-1}\mathbf{r}_t = \mathbf{y}_t$, where the i'th component of $\mathbf{y}_t$ is $y_i = \sum_{j=i}^{t} \gamma^{j-i} r_j$ (see Eq. 4.4.36). Comparing this with the LSTD(1) estimate of Eq. 1.4.60, we conclude that our parametric ML solution is precisely the LSTD(1) solution.

It is equally straightforward to show that TD(1) may be derived as a gradient ascent method for maximizing the log-likelihood of the parametric MC-GPTD model, with a fixed noise variance σ^2. The derivation may be found in Appendix B.

As with any other ML estimator, it should be cautioned that the ML variant will tend to overfit the data, until the number of samples considerably exceeds the dimensionality of the feature space $\boldsymbol{\phi}(\mathcal{X})$ (i.e. the number of independent adjustable parameters). GPTD solutions avoid overfitting by virtue of the regularizing influence of the prior. For this reason, for all practical purposes, we consider MC-GPTD, in its original form, to be preferable to its ML variant, namely, LSTD(1).

A natural question to ask at this point is whether LSTD(λ) for $\lambda < 1$ may also be derived as a ML solution arising from some GPTD generative model, with a different noise covariance. We address this issue next.

4.5.2 LSTD(λ) as a Maximum Likelihood Algorithm

LSTD(λ) with linear function approximation solves the equation set 1.4.59, repeated here for clarity:

$$\mathbf{B}_t \hat{\mathbf{w}}_t = \mathbf{b}_t, \quad \text{where } \mathbf{B}_t = \sum_{i=0}^{t-1} \mathbf{z}_i \left(\boldsymbol{\phi}(\mathbf{x}_i) - \gamma \boldsymbol{\phi}(\mathbf{x}_{i+1})\right)^\top \quad \text{and } \mathbf{b}_t = \sum_{i=0}^{t-1} \mathbf{z}_i r_i. \tag{4.5.52}$$

In both TD(λ) and LSTD(λ) a vector $\mathbf{z}_t$ of eligibilities is maintained, using the recursion

$$\mathbf{z}_t = \gamma\lambda \mathbf{z}_{t-1} + \boldsymbol{\phi}(\mathbf{x}_t), \quad \text{with } \mathbf{z}_0 = \boldsymbol{\phi}(\mathbf{x}_0).$$

These eligibility vectors may be arranged in an $n \times t$ *eligibility matrix* $\mathbf{Z}_t^{(\lambda)}$, defined by

$$\mathbf{Z}_t^{(\lambda)} = \left[\mathbf{z}_0, \ldots, \mathbf{z}_{t-1}\right].$$

Using $\mathbf{Z}_t^{(\lambda)}$ and the definition of $\mathbf{\Delta\Phi}_t$ (4.3.14), we may write $\mathbf{B}_t$ and $\mathbf{b}_t$ from Eq. 1.4.59, as

$$\mathbf{B}_t = \mathbf{Z}_t^{(\lambda)}\mathbf{\Delta\Phi}_t^\top, \quad \text{and } \mathbf{b}_t = \mathbf{Z}_t^{(\lambda)}\mathbf{r}_{t-1}.$$

$\mathbf{B}_t$ is an $n \times n$ square matrix, and for $t \geq n$, assuming no state is visited twice, it is of full rank (i.e. its null-space is $\{\mathbf{0}\}$ and $\mathbf{B}_t^{-1}$ exists). We may therefore premultiply Eq. 1.4.59 by $\mathbf{B}_t^\top \mathbf{G}_t$, where $\mathbf{G}_t$ is an arbitrary $d \times d$ symmetric positive-definite matrix, with no change in the solution whatsoever. This results in

$$\mathbf{\Delta\Phi}_t {\mathbf{Z}_t^{(\lambda)}}^\top \mathbf{G}_t \mathbf{Z}_t^{(\lambda)} \mathbf{\Delta\Phi}_t^\top \hat{\mathbf{w}}_t = \mathbf{\Delta\Phi}_t {\mathbf{Z}_t^{(\lambda)}}^\top \mathbf{G}_t \mathbf{Z}_t^{(\lambda)} \mathbf{r}_{t-1}. \tag{4.5.53}$$

Eq. 4.5.53 is the set of normal equations resulting from the least squares problem

$$\min_{\mathbf{w}} \left\{ \left(\mathbf{r}_{t-1} - \mathbf{\Delta\Phi}_t^\top \mathbf{w}\right)^\top {\mathbf{Z}_t^{(\lambda)}}^\top \mathbf{G}_t \mathbf{Z}_t^{(\lambda)} \left(\mathbf{r}_{t-1} - \mathbf{\Delta\Phi}_t^\top \mathbf{w}\right) \right\}, \tag{4.5.54}$$

which, in turn, may be interpreted as arising from a maximum likelihood problem over jointly Gaussian random variables:

$$\max_{\mathbf{w}} \{\log \Pr(\mathbf{r}_{t-1}|\mathbf{w})\}, \quad \text{where}$$
$$\Pr(\mathbf{r}_{t-1}|\mathbf{w}) \propto \exp\left(-\frac{1}{2}(\mathbf{r}_{t-1} - \mathbf{\Delta\Phi}_t^\top \mathbf{w})^\top {\mathbf{Z}_t^{(\lambda)}}^\top \mathbf{G}_t \mathbf{Z}_t^{(\lambda)} (\mathbf{r}_{t-1} - \mathbf{\Delta\Phi}_t^\top \mathbf{w})\right).$$

Note that, regardless of the choice of $\mathbf{G}_t$, the minimum in (4.5.54) is zero, since $\mathbf{Z}_t^{(\lambda)}\left(\mathbf{r}_{t-1} - \mathbf{\Delta\Phi}_t^\top \mathbf{w}\right) = 0$ is a set of n equations in n variables, and therefore $\mathbf{Z}_t^{(\lambda)}\left(\mathbf{r}_{t-1} - \mathbf{\Delta\Phi}_t^\top \mathbf{w}\right)$ can be made to vanish.

Comparing Eq. 4.5.54 with Eq. 4.5.49, we conclude that LSTD(λ) implicitly assumes that rewards and values are connected by a Gaussian process model of the by now familiar (parametric) GPTD form, namely:

$$R_{t-1} = \mathbf{H}_t V_t + N_{t-1} = \mathbf{H}_t \mathbf{\Phi}_t^\top W + N_{t-1},$$

in which the inverse covariance matrix of the the noise process N_{t-1} satisfies

$$\mathbf{\Sigma}_t^{-1} \propto {\mathbf{Z}_t^{(\lambda)}}^\top \mathbf{G}_t \mathbf{Z}_t^{(\lambda)}.$$

The noise covariance matrix $\mathbf{\Sigma}_t$ itself does not generally exist, since, for $t > n$, ${\mathbf{Z}_t^{(\lambda)}}^\top \mathbf{G}_t \mathbf{Z}_t^{(\lambda)}$ is rank deficient (with rank n). This means that there are $t - n$ orthogonal directions in $\mathbb{R}^t$ (defined by an orthogonal basis for the null space of ${\mathbf{Z}_t^{(\lambda)}}^\top \mathbf{G}_t \mathbf{Z}_t^{(\lambda)}$) in which the variance is infinite, which means that deviations in these directions have no effect on the likelihood. The nonexistence of $\mathbf{\Sigma}_t$ is less of a problem than it might seem, since the ML, MAP and Bayesian solutions may all be written entirely in terms of $\mathbf{\Sigma}_t^{-1}$ (see Eq. 4.5.49, 4.3.15, 4.3.16), and it is therefore only important that $\mathbf{\Sigma}_t^{-1}$ exists. In any event, the inverse of $\mathbf{\Sigma}_t^{-1} + \varepsilon \mathbf{I}$ for any $\varepsilon > 0$ exists and may be used as a substitute for $\mathbf{\Sigma}_t$.

It now becomes a simple exercise to derive a new set of GPTD algorithms, referred to as GPTD(λ), which are based on the LSTD(λ) noise model, i.e. using $\mathbf{\Sigma}_t^{-1} = \frac{1}{\sigma^2}\mathbf{Z}_t^{(\lambda)^\top}\mathbf{G}_t\mathbf{Z}_t^{(\lambda)}$. For the parametric GPTD(λ) model, the posterior moments, given by (Eq. 4.3.15, 4.3.16), are

$$\begin{aligned}\mathbf{P}_t^{(\lambda)} &= \sigma^2\left(\mathbf{\Delta\Phi}_t\mathbf{Z}_t^{(\lambda)^\top}\mathbf{G}_t\mathbf{Z}_t^{(\lambda)}\mathbf{\Delta\Phi}_t^\top + \sigma^2\mathbf{I}\right)^{-1}, \\ \hat{\mathbf{w}}_t^{(\lambda)} &= \frac{1}{\sigma^2}\mathbf{P}_t^{(\lambda)}\mathbf{\Delta\Phi}_t\mathbf{Z}_t^{(\lambda)^\top}\mathbf{G}_t\mathbf{Z}_t^{(\lambda)}\mathbf{r}_{t-1}.\end{aligned} \tag{4.5.55}$$

Whereas for the nonparametric GPTD(λ), we have (see Eq. 1.3.36, 1.3.37)

$$\begin{aligned}\boldsymbol{\alpha}_t^{(\lambda)} &= \left(\mathbf{H}_t^\top\mathbf{Z}_t^{(\lambda)^\top}\mathbf{G}_t\mathbf{Z}_t^{(\lambda)}\mathbf{H}_t\mathbf{K}_t + \sigma^2\mathbf{I}\right)^{-1}\mathbf{H}_t^\top\mathbf{Z}_t^{(\lambda)^\top}\mathbf{G}_t\mathbf{Z}_t^{(\lambda)}\mathbf{r}_{t-1}, \\ \mathbf{C}_t^{(\lambda)} &= \left(\mathbf{H}_t^\top\mathbf{Z}_t^{(\lambda)^\top}\mathbf{G}_t\mathbf{Z}_t^{(\lambda)}\mathbf{H}_t\mathbf{K}_t + \sigma^2\mathbf{I}\right)^{-1}\mathbf{H}_t^\top\mathbf{Z}_t^{(\lambda)^\top}\mathbf{G}_t\mathbf{Z}_t^{(\lambda)}\mathbf{H}_t.\end{aligned} \tag{4.5.56}$$

In the parametric case, for an arbitrary $\mathbf{G}_t$, taking the limit $\sigma \to 0$ brings us back to the familiar LSTD(λ). Moreover, for any $\sigma > 0$, and for $\lambda = 1$ there exists a particular choice of $\mathbf{G}_t$ for which we are returned to the MC-GPTD model of Sections 4.4.4 and 4.4.5, as we show next for the parametric model.

4.5.3 GPTD(λ)

In this section we consider a possible choice for the matrix $\mathbf{G}_t$ in Eq. 4.5.54. First, however, let us recall that the eligibility vectors satisfy $\mathbf{z}_t = \sum_{i=0}^t (\gamma\lambda)^{t-i}\boldsymbol{\phi}(\mathbf{x}_i)$. We may therefore write $\mathbf{Z}_{t+1}^{(\lambda)}$ as

$$\mathbf{Z}_{t+1}^{(\lambda)} = \mathbf{\Phi}_t\begin{bmatrix} 1 & \gamma\lambda & (\gamma\lambda)^2 & \dots & (\gamma\lambda)^t \\ 0 & 1 & \gamma\lambda & \dots & (\gamma\lambda)^{t-1} \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & \dots & 1\end{bmatrix}. \tag{4.5.57}$$

Suppose that $\mathbf{x}_t$ is the final state in an episode. Recalling $\mathbf{H}_{t+1}^{-1}$ from Eq. 4.4.37, for $\lambda = 1$ we can write

$$\mathbf{Z}_{t+1}^{(1)} = \mathbf{\Phi}_t\mathbf{H}_{t+1}^{-1}$$

$\mathbf{P}^{(1)}_{t+1}$ of Eq. 4.5.55 then becomes

$$\mathbf{P}^{(1)}_{t+1} = \sigma^2 \left(\mathbf{\Phi}_t \mathbf{\Phi}_t^\top \mathbf{G}_t \mathbf{\Phi}_t \mathbf{\Phi}_t^\top + \sigma^2 \mathbf{I} \right)^{-1},$$

while $\hat{\mathbf{w}}^{(1)}_{t+1}$ becomes

$$\hat{\mathbf{w}}^{(1)}_{t+1} = \left(\mathbf{\Phi}_t \mathbf{\Phi}_t^\top \mathbf{G}_t \mathbf{\Phi}_t \mathbf{\Phi}_t^\top + \sigma^2 \mathbf{I} \right)^{-1} \mathbf{\Phi}_t \mathbf{\Phi}_t^\top \mathbf{G}_t \mathbf{\Phi}_t \mathbf{H}^{-1}_{t+1} \mathbf{r}_t.$$

These expressions for the posterior moments suggest that a reasonable choice for $\mathbf{G}_t$ would be $\mathbf{G}_t = (\mathbf{\Phi}_t \mathbf{\Phi}_t^\top)^{-1}$, since then, for $\lambda = 1$, we are returned to the MC-GPTD solution,

$$\mathbf{P}^{(1)}_{t+1} = \sigma^2 \left(\mathbf{\Phi}_t \mathbf{\Phi}_t^\top + \sigma^2 \mathbf{I} \right)^{-1} \quad \text{and } \hat{\mathbf{w}}^{(1)}_{t+1} = \left(\mathbf{\Phi}_t \mathbf{\Phi}_t^\top + \sigma^2 \mathbf{I} \right)^{-1} \mathbf{\Phi}_t \mathbf{H}^{-1}_{t+1} \mathbf{r}_t.$$

These solutions are recognized as the parametric MC-GPTD posterior moments (see Eq. 4.4.38).

Our argument in favor of this particular choice of the inverse noise covariance (through the choice of $\mathbf{G}_t$) is based on the limiting behavior of the resulting solutions when λ approaches 1 (in which case it coincides with the MC-GPTD solution), and when σ approaches 0 (in which case it coincides with LSTD(λ)). Apart from these continuity arguments, the question of how a noise model such as the one suggested above may be theoretically justified, remains unanswered. In particular, this means that, even with complete knowledge of the underlying MDP, the optimal choice of the parameter λ is currently an issue that has to be resolved empirically. However, the same could be said about LSTD(λ). As far as we are aware, the only way LSTD(λ) with linear function approximation may be justified, or indeed derived (apart from the post hoc asymptotic bounds of Tsitsiklis & Van Roy, 1996), is either by analogy to the case of a lookup-table based representation, or by a continuity argument based on the fact that as λ approaches 1, the resulting algorithm approaches LSTD(1), which is known to perform least-squares regression on Monte-Carlo samples of the discounted return (Bertsekas & Tsitsiklis, 1996; Sutton & Barto, 1998; Boyan, 1999a).

In the following section we perform an empirical comparison of the LSTD(λ) and GPTD(λ) algorithm, on a simple MDP, in order to get some idea concerning the accuracy of their respective value estimates.

4.5.4 Experiments

In this section we perform an experimental comparison between the parametric form of GPTD(λ) and LSTD(λ). Our experimental testbed is a simple 10 state random-walk MDP. The 10 states are arranged linearly from state 1 on the left to state 10 on the right. The left-hand wall is a retaining barrier, meaning that if a left step is made from state 1, the state transitioned to is again state 1. State 10 is a zero reward absorbing state. The agent may either make a step to the left or to the right

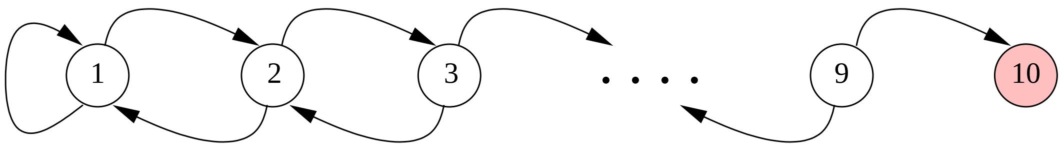


Figure 4.4: The 10 state random walk domain

with probabilities $\Pr(left)$ and $\Pr(right)$, respectively. ($\Pr(left) + \Pr(right) = 1$). The reward is -1 for states 1–9 and 0 for state 10. This domain is illustrated in Figure 4.4.

In order to perform the comparison on equal footing, we restrict our attention to LSTD(λ) and GPTD(λ) with a linear parametric function approximation architecture, using a set of 10 radial basis functions (RBFs), centered in each one of the states, each with a width of 1. I.e. for the state j, $\phi_i(j) = exp\left(-(j-i)^2/2\right)$, with $i, j \in \{1, \ldots, 10\}$. This way, both methods use the same parametric representation, and learn within the same hypothesis space.

In a simple MDP such as this one, it is quite easy to compute (even analytically, in certain cases) the state values. These may then be used to compute the error between the true values and the estimates provided by each algorithm. Here we compare LSTD(λ) with GPTD(λ) for 5 equally spaced value of λ: 1, 0.75, 0.5, 0.25, 0. Two experiments were made, one for $\Pr(right) = 0.8$ and the other for $\Pr(right) = 0.6$. The probability distribution of the number of steps until absorption, along with its mean and standard deviation, for each starting state, are shown in Fig. 4.5.

It can be seen from Fig. 4.5 that in such random walks the distribution of the discounted return is asymmetric, heavy-tailed and quite far from being Gaussian. This becomes increasingly so as $\Pr(right)$ is reduced. Consequently, very few of the assumptions upon which GPTD is based hold.

In the experiments, each algorithm was run on an identical series of 1000 episodes. At the end of each episode, the squared error between the estimated value and the true value for each state was computed. Each such experiment was repeated 10 times allowing us to average together the squared errors for each state, resulting in an estimate for the root mean squared (RMS) error, for each state and each algorithm.

In Fig. 4.6 we show the results of this comparison. The graphs show the RMS errors, for each algorithm, averaged over the 10 states, for several "snapshot" points along the series of 1000 learning episodes. At the top the results for $\Pr(right) = 0.8$ are shown, and at the bottom are the results for the more difficult problem (from the GPTD standpoint) of $\Pr(right) = 0.6$.

As can be seen in both instances, except after the first trial, the GPTD(λ) solution almost always either equals or dominates the LSTD(λ) solution.

4.6 Policy Improvement with GPSARSA

In Chapter 1, SARSA was described as a fairly straightforward extension of the TD algorithm (Sutton & Barto, 1998), in which state-action values are estimated. This allows policy improvement steps to be performed without requiring any additional knowledge on the MDP model. The idea is to use the stationary policy μ being followed in order to define a new, augmented process, the state space of which is $\mathcal{X}' = \mathcal{X} \times \mathcal{U}$, (i.e. the original state space augmented by the action space), while maintaining the same reward model. This augmented process is Markovian with transition probabilities $p'(\mathbf{x}', \mathbf{u}'|\mathbf{x}, \mathbf{u}) = p^{\mu}(\mathbf{x}'|\mathbf{x})\mu(\mathbf{u}'|\mathbf{x}')$. SARSA is simply the TD algorithm applied to this new process. The same reasoning may be applied to derive a GPSARSA algorithm from the GPTD algorithm.

In the nonparametric case, all we need is to define a covariance kernel function over state-action pairs, i.e. $k : (\mathcal{X} \times \mathcal{U}) \times (\mathcal{X} \times \mathcal{U}) \rightarrow \mathbb{R}$. Since states and actions are different entities it makes sense to decompose k into a state-kernel k_x and an action-kernel k_u:

$$k(\mathbf{x}, \mathbf{u}, \mathbf{x}', \mathbf{u}') = k_x(\mathbf{x}, \mathbf{x}')k_u(\mathbf{u}, \mathbf{u}').$$

If both k_x and k_u are kernels we know that k is also a legitimate kernel (Schölkopf & Smola, 2002), and just as the state-kernel codes our prior beliefs concerning correlations between the values of different states, so should the action-kernel code our prior beliefs on value correlations between different actions. In the parametric case we would similarly define a set of basis functions over $\mathcal{X} \times \mathcal{U}$. Here we will treat the more interesting nonparametric case.

All that remains now is to run the sparse nonparametric GPTD algorithm (20) on the augmented state-reward sequence, using the new state-action kernel function. Action selection may be performed by ε-greedily choosing the highest ranking action, and slowly decreasing ε toward zero. However, we may run into difficulties trying to find the highest ranking action from a large or even infinite number of possible actions. This may be solved by sampling the value estimates for a few randomly chosen actions and maximize only among these, or alternatively using a fast iterative maximization method, such as the quasi-Newton method or conjugate gradients. Ideally, we should design the action kernel in such a way as to provide a closed-form expression for the greedy action.

4.6.1 Experiments

As an example let us consider again the two-dimensional continuous world described in Section 4.3.7. Rather than restricting the action space to 8 actions, as we did before, now we allow the agent to take a 0.1-long step in *any direction*. For each time step until it reaches a goal state the agent is penalized with a negative reward of -1; if it hits an obstacle it is returned to its original position. Let us represent an action as the unit vector pointing in the direction of the corresponding move, thus making $\mathcal{U}$ the unit circle. We leave the space kernel k_x unspecified and focus on the action kernel. Let us define k_u as follows,

$$k_u(\mathbf{u}, \mathbf{u}') = 1 + \frac{(1-b)}{2}(\mathbf{u}^\top \mathbf{u}' - 1),$$

where b is a constant in $[0,1]$. Since $\mathbf{u}^\top \mathbf{u}'$ is the cosine of the angle between $\mathbf{u}$ and $\mathbf{u}'$, $k_u(\mathbf{u}, \mathbf{u}')$ attains its maximal value of 1 when the two actions are the same, and its minimal value of b when the actions are 180 degrees apart. Setting b to a positive value is reasonable, since even opposite actions from the same state are expected, a priori, to have positively correlated values. However, the most valuable feature of this kernel is its linearity, which makes it possible to maximize the value estimate over the actions analytically.

Assume that the agent runs GPSARSA, so that it maintains a dictionary of state-action pairs $D_t = \{(\tilde{\mathbf{x}}_i, \tilde{\mathbf{u}}_i)\}_{i=1}^m$. The agent's value estimate for its current state $\mathbf{x}$ and an arbitrary action $\mathbf{u}$ is

$$\begin{aligned}\hat{V}(\mathbf{x}, \mathbf{u}) &= \sum_{i=1}^m \tilde{\boldsymbol{\alpha}}_i k_x(\tilde{\mathbf{x}}_i, \mathbf{x}) k_u(\tilde{\mathbf{u}}_i, \mathbf{u}) \\ &= \sum_{i=1}^m \tilde{\boldsymbol{\alpha}}_i k_x(\tilde{\mathbf{x}}_i, \mathbf{x}) \left(1 + \frac{(1-b)}{2}(\tilde{\mathbf{u}}_i^\top \mathbf{u} - 1)\right).\end{aligned}$$

Maximizing this expression with respect to $\mathbf{u}$ amounts to maximizing $\sum_{i=1}^m \beta_i(\mathbf{x})\tilde{\mathbf{u}}_i^\top \mathbf{u}$ subject to the constraint $\|\mathbf{u}\| = 1$, where $\beta_i(\mathbf{x}) \stackrel{\text{def}}{=} \tilde{\boldsymbol{\alpha}}_i k_x(\tilde{\mathbf{x}}_i, \mathbf{x})$. Solving this problem using a single Lagrange multiplier results in the greedy action $\mathbf{u}^* = \frac{1}{\lambda}\sum_{i=1}^m \beta_i(\mathbf{x})\tilde{\mathbf{u}}_i$ where λ is a normalizing constant. It is also possible to maximize the variance estimate. This may be used to select non-greedy exploratory moves, by choosing the action the value of which the agent is least certain about. Performing this maximization amounts to solving a 2×2 Eigenvalue problem.

Our experimental test-bed is the continuous state-action maze described above. In order to introduce stochasticity into the transitions, beyond the randomness inherent in the ε-greedy policy, we corrupt the moves chosen by the agent with a zero-mean uniformly distributed angular noise in the range of ± 30 degrees. A GPSARSA-learning agent was put through 200 episodes, each of which consisted of placing it in a random position in the maze, and letting it roam the maze until it reaches a goal position (success) or until 100 time-steps elapse, whichever happens first. At each episode, ε was set to $10/(10+T)$, T being the number of successful episodes completed up to that point. The σ parameter of the intrinsic variance was fixed at 1 for all states. The state kernel was Gaussian $k(\mathbf{x},\mathbf{x}') = k(\mathbf{x},\mathbf{x}') = c\exp\left(-\|\mathbf{x}-\mathbf{x}'\|^2/(2\sigma_k^2)\right)$, with $\sigma_k = 0.2$ and a $c = 10$ (c is the prior variance of V, since $k(\mathbf{x},\mathbf{x}) = c$). The action kernel was the linear kernel described above. The parameter ν controlling the sparsity of the solution (see Chapter 2) was $\nu = 0.1$, resulting in a dictionary that saturates at 150-160 state-action pairs.

The results of such a run on four different mazes, the first of which is identical to the one used in Section 4.3.7, are shown in Fig. 4.7. Note that the "dampening" of the value estimates, which was observed in the experiments of Section 4.3.7 when the GPTD algorithm for deterministic transitions was used with stochastic transitions, is now absent as expected.

By choosing the Gaussian kernel we impose a smoothness property on candidate value functions. This smoothness assumption naturally breaks down near the obstacles, which is the reason for the appearance of some artifacts across or near obstacles, such as "whirlpools" and suboptimal paths. Notice however, that since the actual move performed in each step is corrupted by a significant angular noise, the learned policies correctly exhibit a preference for paths going through wide rather than narrow openings, when such a choice exists.

4.7 Summary and Discussion

In this chapter we presented a Bayesian formulation of the fundamental RL problem of policy evaluation. This was done by casting the probabilistic relation between the value function and the observed rewards as a linear statistical generative model over normally distributed random processes (Eq. 4.3.5). The Bayesian solution for the policy evaluation problem is embodied in the posterior value distribution conditioned on the observed state-reward trajectory. This posterior is computed by employing Bayes' rule to "invert" the set of equations provided by the generative model. Apart from the value estimates given by the posterior mean, the Bayesian solution also provides the variance of values around this mean, supplying the practitioner with a measure of the accuracy of value estimates.

In Rasmussen and Kuss (2004) an alternative approach to employing GPs in RL is proposed. The approach in that paper is fundamentally different from the generative approach of the GPTD framework. In Rasmussen and Kuss (2004) one GP is used to learn the MDP's transition model, while another is used to estimate the value. This leads to an inherently off-line algorithm, which is not capable of interacting with the controlled system directly and updating its estimates as additional data arrive. There are several other shortcomings that limit the usefulness of that framework. First, the state dynamics is assumed to be factored, in the sense that each state coordinate is assumed to evolve in time independently of all others. This is a rather strong assumption that is not likely to be satisfied in most real problems. Moreover, it is also assumed that the reward function is completely known in advance, and is of a very special form – either polynomial or Gaussian. Finally, the covariance kernels used are also restricted to be either polynomial or Gaussian or a mixture of the two, due to the need to integrate over products of GPs. This considerably diminishes the appeal of employing GPs, since one of the main reasons for using them, and kernel methods in general, is the richness of expression inherent in the ability to construct arbitrary kernels, reflecting domain and problem-specific knowledge, and defined over sets of diverse objects, such as text documents and DNA sequences (to name only two), and not just points in metric space. Finally, the value function is only modeled at a predefined set of *support states*, and is solved only for them. No method is proposed to ensure that this set of states is representative in any way.

Bayesian methods, such as ours, typically require the user to impart more domain specific knowledge to the learning system, than do classical methods. Such domain knowledge may be encoded in the prior and in the measurement model (e.g., in the measurement noise covariance). In many cases such domain knowledge is available, at least in a rough form. For instance, in the RL context, the user will usually know whether the MDP under investigation follows deterministic or stochastic dynamics. She will usually also have a good idea on the range of values of the rewards, and what makes states and actions similar or different from each other. All of this information may be incorporated into a GPTD generative model, and capitalized upon in the subsequent Bayesian posterior analysis. We made use of these inherent qualities of the Bayesian approach by defining different generative models, one which is suitable only for MDPs with deterministic transitions and a second one for MDPs with stochastic transitions. In the nonparametric case, the kernel function k is used to encode notions of similarity and dissimilarity between states, or more correctly, how the values of different states are correlated[10]. Having said that, there are well known methods in the Bayesian and GP literature for learning the hyperparameters of a Bayesian model (a.k.a. model selection). These

[10]For instance, using the Gaussian kernel $c\exp\left(-\|\mathbf{x}'-\mathbf{x}\|^2/(2\sigma^2)\right)$, apart from encoding the belief that nearby states are more similar than states that are far apart, also states that, a priori, we believe that the variance of the values at any point in $\mathcal{X}$ is c.

include maximizing the likelihood of the observations with respect to these hyperparameters, or alternatively, treating them as random variables (parameterized by hyper-hyperparameters) and performing Bayesian inference on them as well (this is known as a hierarchical Bayesian model). As these techniques are adequately covered elsewhere (e.g., Mackay, 1997; Gibbs & MacKay, 1997; Williams, 1999; Seeger, 2003) we chose not to elaborate on this subject here.

As discussed in Section 1.2.1, in classical frequentist approaches[11], the risk being minimized depends on the true but unknown hypothesis θ – in our case the true value function. This means that for a fixed θ, the estimator minimizing the risk, is optimal in the frequentist sense. That is, if we indefinitely repeat the same experiment, each time with a new sequence of state-action-reward triplets, the minimum risk estimator will incur the lowest total loss. However, in the RL setting, one typically observes only a single sampled trajectory[12] and would therefore prefer to have a decision procedure that behaves optimally for *that* trajectory. The Bayesian approach provides just that, by minimizing a risk that is conditioned on the specific observed history – the posterior Bayesian risk.

We showed in Sections 4.5.2 and 4.5.3 that the familiar family of LSTD(λ) algorithms are in fact classical parametric maximum-likelihood algorithms based on statistical generative models having the same structure as our GPTD model, and employing a specific form of λ-dependent noise covariance. As such, LSTD(λ) solutions are limit points of GPTD solutions. In this sense, our GPTD framework subsumes all other TD methods employing linear function approximation architectures. This, we hope, should alleviate concerns regarding the applicability of the assumptions underlying the GPTD model, as the same assumptions are also implicitly made by LSTD(λ) and TD(λ).

By employing a nonparametric, kernel-based extension of the conventional parametric linear statistical model we were able to derive kernel-based variants of our algorithms. These kernel algorithms offer a degree of representational flexibility that is impossible to attain using standard linear FA architectures, in which a set of basis functions must be defined in advance. By using the sparsification method described in Chapter 2 we were able to obtain, for the first time to the best of our knowledge, practical nonparametric online algorithms for solving the policy evaluation problem. The constructive process through which these algorithms sparsify the full nonparametric solution may be thought of as a process of basis construction, in which new basis functions are added if and when they are needed for maintaining a sufficiently accurate approximation of the complete non-sparse solution (the basis functions are the kernel functions evaluated at the dictionary states, namely $\{k(\tilde{\mathbf{x}}_j, \cdot)\}_{j=1}^{|\mathcal{D}|}$). As

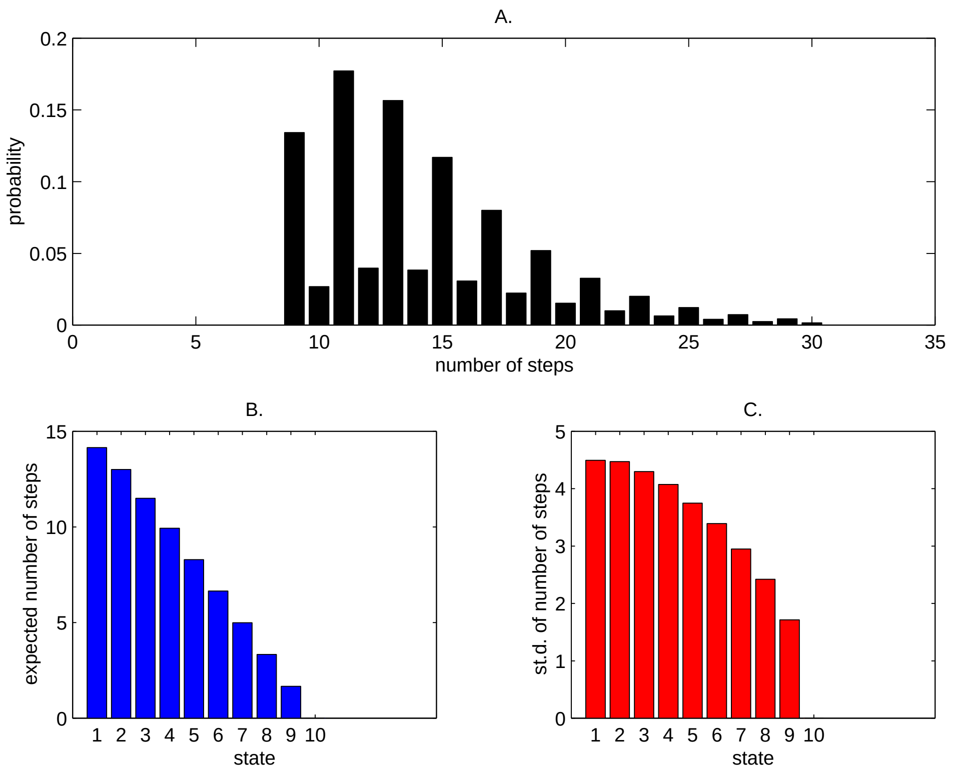
[11]LSTD(1) is one such approach, as it delivers a least-squares estimate.

[12]This is true regardless of whether that trajectory is divided into episodes or not.

such they enjoy both the representational flexibility of the complete nonparametric solution while also taking advantage of redundancies in the Hilbert space induced by the kernel. When the dictionary constructed by these algorithms ceases to grow (and we are assured by Theorem 2.3.1 that this will eventually happen), then from that point on the algorithm may be considered as a (linear) parametric algorithm.

Taking the frequentist perspective, the posterior mean provided by the Bayesian analysis may be viewed as the solution of a Tikhonov-regularized least-squares problem (see Section 1.3.4). Although in the frequentist view, such solutions are considered biased, they are known to be consistent (i.e. asymptotically unbiased). Specifically, this implies that in the parametric case, in the limit of an infinite trajectory, both GPTD(λ) (including MC-GPTD = GPTD(1)) and LSTD(λ) converge to the same solution (provided the same λ is used). This should remove any concerns regarding the convergence of GPTD(λ), as the asymptotic convergence properties of LSTD(λ) are well understood (Tsitsiklis & Van Roy, 1996). When a sparse nonparametric representation is used, we already know that, after a finite number of transitions are observed, our sparse algorithms are effectively computing parametric solutions, using a basis consisting of the kernel functions evaluated at the dictionary states. Therefore, the same conclusions regarding the convergence of our parametric methods also apply to our sparse nonparametric methods. More intricate convergence properties of GPs, such as generalization bounds and learning curves (but only in the context of supervised learning), are presently the subject of intense research (e.g., Opper & Vivarelli, 1999; Malzahn & Opper, 2001; Seeger, 2003; Sollich & Williams, 2005). However, the GPTD model is generally more complex than the conventional GP models used in the supervised setting. Deriving results analogous to those mentioned above, in the RL setting, is left as an open direction for future research.

Another contribution is the extension of GPTD to the estimation of state-action values, or Q-values, leading to the GPSARSA algorithm. Learning Q-values makes the task of policy improvement in the absence of a transition model tenable, even when the action space is continuous, as demonstrated by the example in Section 4.6.1. The availability of confidence intervals for Q-values significantly expands the repertoire of possible exploration strategies. In finite MDPs, strategies employing such confidence intervals have been experimentally shown to perform more efficiently then conventional ε-greedy or Boltzmann sampling strategies, e.g., Kaelbling (1993); Dearden et al. (1998); Even-Dar et al. (2003). GPSARSA allows such methods to be applied to infinite MDPs, and it remains to be seen whether significant improvements can be so achieved for realistic problems with continuous space and action spaces.



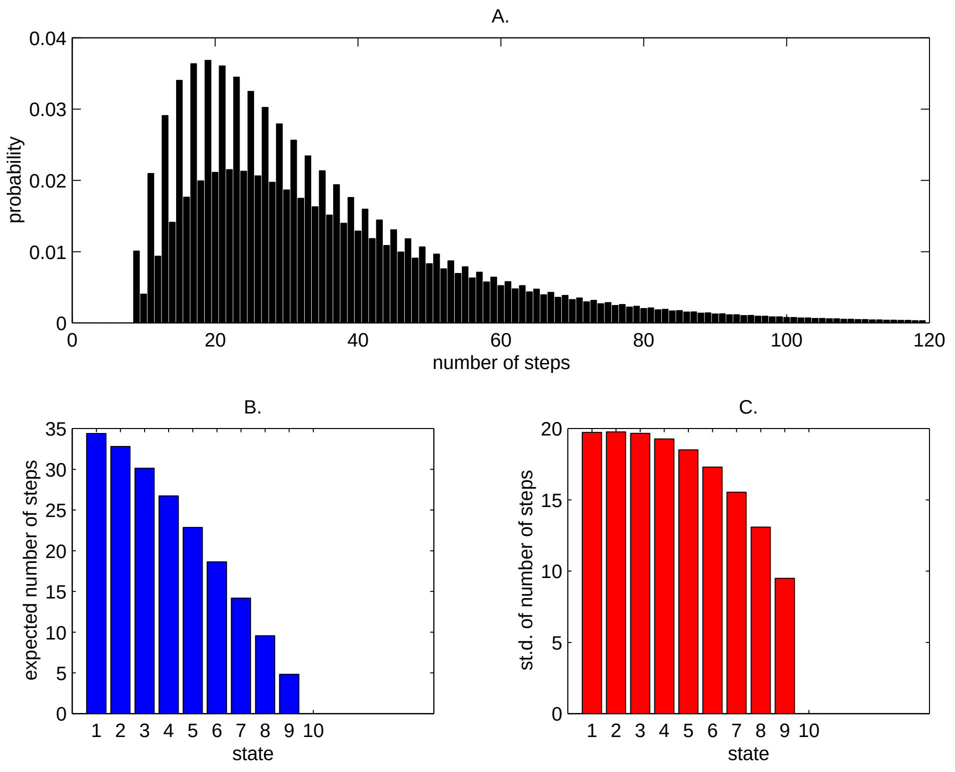
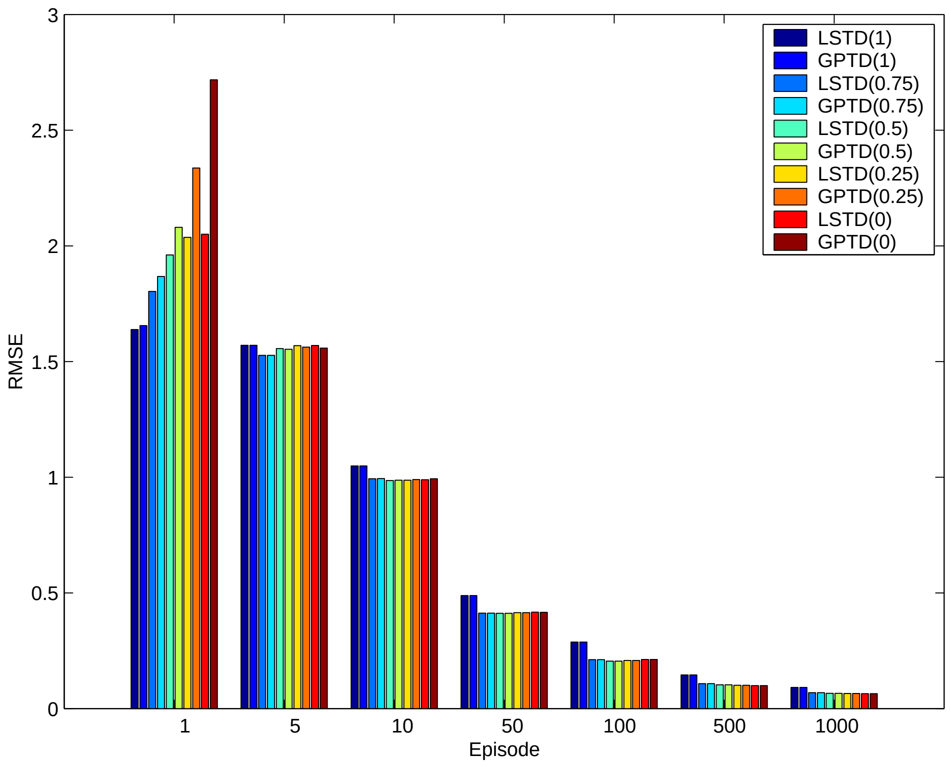


Figure 4.5: A 10 state one-dimensional random walk, with an absorbing state 10. The probability of making a step to the right is 0.8 (top) or 0.6 (bottom). A. The probability distribution of the number steps until absorption, starting from state 1. B. The expected number of time-steps until absorption, for each state. C. The standard deviation of the number of time-steps until absorption, for each state.



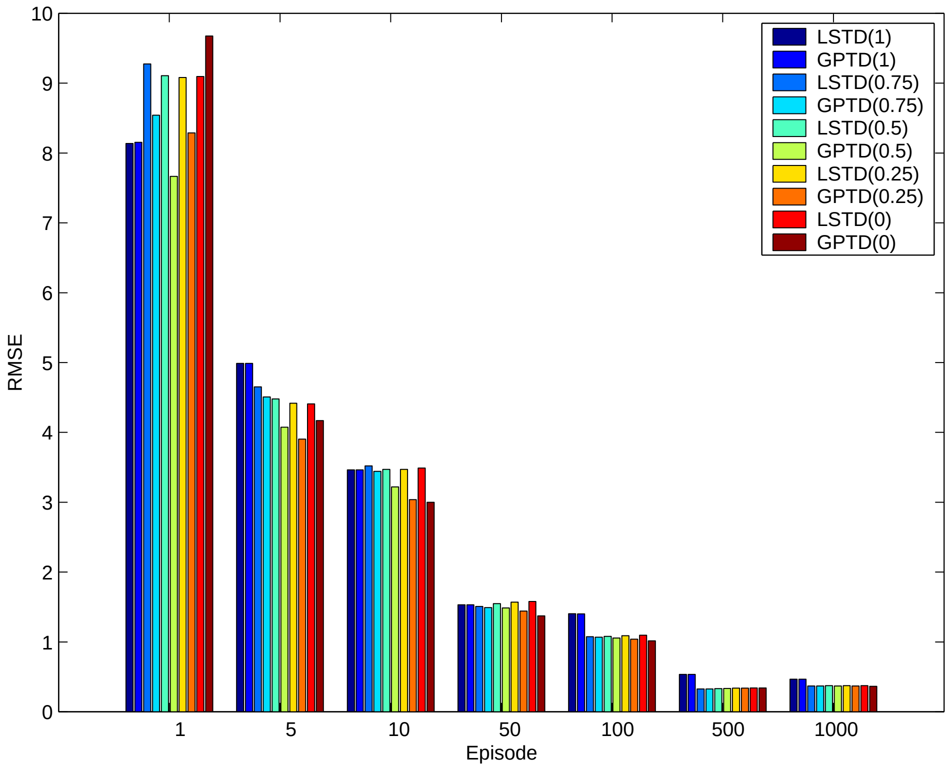


Figure 4.6: Comparison results for the $\Pr(right) = 0.8$ random walk (top) and the $\Pr(right) = 0.6$ random walk (bottom).

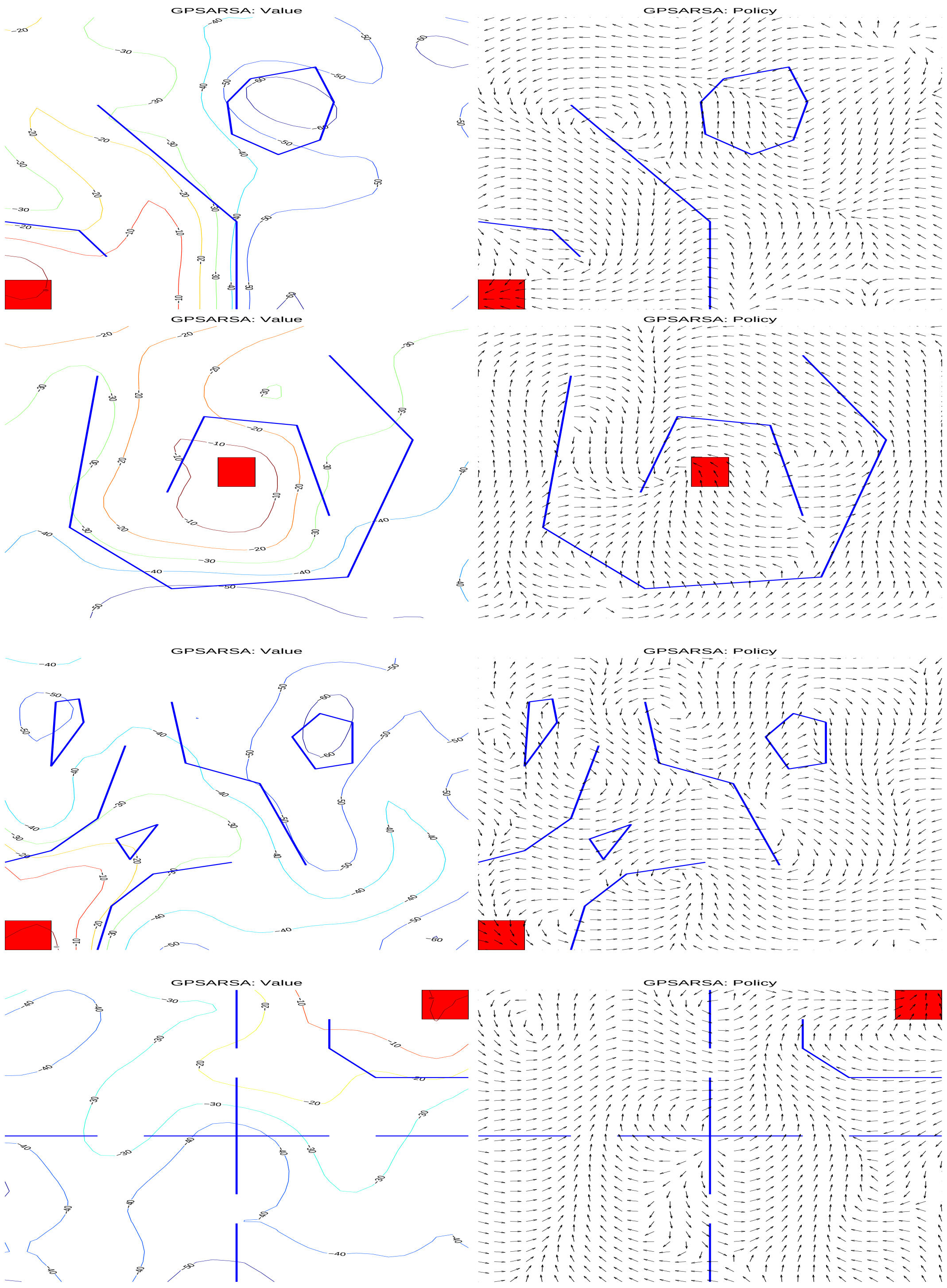


Figure 4.7: The posterior value mean (left) and the corresponding greedy policy (right) for four different mazes, after 200 learning episodes. In each maze, the goal region is the dark (red) rectangle.

Chapter 5

Conclusion

The most significant contribution of this thesis is in presenting a new Bayesian framework for reasoning about value functions in reinforcement learning problems. The strengths of this GP-based Bayesian approach lie in its ability to provide probabilistic predictions amounting to a complete posterior distribution over value functions, conditioned on the observed history of state-action-reward transitions. The ability of the GPTD framework to provide a confidence measure on value predictions in large MDPs is quite unique, and has many potential uses. By employing different forms of noise statistics we were able to obtain a generative model specifically tailored to MDPs with deterministic transitions, as well as a family of models capable of handling general MDPs.

The GPTD framework presented here may be used either parametrically or nonparametrically. In the former case, interesting connections between GPTD algorithms and classical parametric RL algorithms are revealed, providing insight into the implicit assumptions underlying the latter class of algorithms. A set of efficient recursive algorithms for computing the posterior moments were derived, corresponding to the two generative models mentioned above. When phrased nonparametrically, the GPTD framework results in a family of kernel algorithms that search for a solution to the value estimation problem in an generally infinite dimensional function space. The use of general Mercer kernels as the basic representational element is another unique feature of our method, which would allow practitioners to solve RL problems in domains for which the parametric, feature-vector based approach is unsuitable. A sequential sparsification method, presented in Chapter 2, is essential for overcoming the computational difficulties associated with this search. Using this method allowed us to derive recursive, efficient, online algorithms for computing the posterior value moments in the nonparametric case.

The kernel-RLS algorithm described in Chapter 3, which also employs our sparsification method, is another important contribution. Online kernel algorithms, such as KRLS should be considered as viable alternatives to traditional parametric algorithms in application domains requiring sequential, online mode of operation, such as signal processing and data mining. Kernel methods have been slow in penetrating these domains due to the off-line, non-sequential nature of most kernel algorithms, and their difficulty in handling large amounts of data. Algorithms such as KRLS should help rectify this situation.

For want of space, some of the other contributions accomplished during the thesis work had to be left out from this thesis. These include

- An online algorithm, which may be used to learn in an unsupervised manner, maps of Markov processes and MDPs embedded in Euclidean space. Distances in the resulting maps reflect the strength of interaction among the states of the MDP (Engel & Mannor, 2001).
- A sparse online SVR-like regression algorithm (Engel et al., 2002). As in the kernel-RLS algorithm, the sparsification method of Chapter 2 is employed to reduce the number of variables used in the maximization of the dual SVR Lagrangian. This results in a highly efficient regression algorithm, performing on a par with state-of-the-art kernel regression algorithms.
- An additional GPTD algorithm for collaborative/parallelized policy evaluation. This algorithm may be used to accurately merge posterior value GPs learned by multiple GPTD-learning agents into a single unified value GP (Engel & Mannor, 2005).

5.1 Future Work

The emphasis in this thesis has been on developing the theory of Gaussian processes for RL. Empirical work was therefore largely restricted to verifying that our technical results bear out in reality. Future work (underway) is aimed at solving large scale RL problems using the GPTD framework, taking leverage on the particular advantages specific to this framework, such as the nonparametric form of representation and the confidence intervals it provides.

Standing challenges for future work include balancing exploration and exploitation in RL using the value confidence intervals provided by GPTD methods; further exploring the space of GPTD models by considering additional noise covariance structures; application of the GPTD methodology to POMDPs; creating a GP-based Actor-Critic architecture; GPQ-Learning for off-policy learning of the optimal policy; parallelizing and distributing GPTD and GPSARSA among multiple agents; analyzing the convergence properties of GPTD; and searching for the optimal noise covariance.

Appendix A

Derivation of Recursive GPTD Algorithms

A.1 Deterministic Transitions

A.1.1 Parametric GPTD Updates

Recall the alternative expressions for the parametric posterior moments (4.3.15, 4.3.16):

$$\mathbf{P}_t = \left(\mathbf{\Delta\Phi}_t \mathbf{\Sigma}_t^{-1} \mathbf{\Delta\Phi}_t^\top + \mathbf{I}\right)^{-1}$$
$$\hat{\mathbf{w}}_t = \mathbf{P}_t \mathbf{\Sigma}_t^{-1} \mathbf{\Delta\Phi}_t \mathbf{r}_{t-1},$$

where

$$\mathbf{\Delta}\boldsymbol{\phi}_t \overset{\text{def}}{=} \boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma \boldsymbol{\phi}(\mathbf{x}_t), \quad \mathbf{\Delta\Phi}_t \overset{\text{def}}{=} \mathbf{\Phi}_t \mathbf{H}_t^\top = [\mathbf{\Delta}\boldsymbol{\phi}_1, \ldots, \mathbf{\Delta}\boldsymbol{\phi}_t],$$

and $\mathbf{\Sigma}_t = \text{diag}(\sigma_0^2, \ldots, \sigma_{t-1}^2)$. We therefore have

$$\mathbf{\Sigma}_t = \begin{bmatrix} \mathbf{\Sigma}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & \sigma_{t-1}^2 \end{bmatrix}.$$

Consequently,

$$\mathbf{P}_t^{-1} = \mathbf{P}_{t-1}^{-1} + \frac{1}{\sigma_{t-1}^2} \mathbf{\Delta}\boldsymbol{\phi}_t \mathbf{\Delta}\boldsymbol{\phi}_t^\top,$$

and by the Matrix Inversion lemma,

$$\mathbf{P}_t = \mathbf{P}_{t-1} - \frac{\mathbf{P}_{t-1} \mathbf{\Delta}\boldsymbol{\phi}_t \mathbf{\Delta}\boldsymbol{\phi}_t^\top \mathbf{P}_{t-1}}{\sigma_{t-1}^2 + \mathbf{\Delta}\boldsymbol{\phi}_t^\top \mathbf{P}_{t-1} \mathbf{\Delta}\boldsymbol{\phi}_t}.$$

Let us define

$$\hat{\mathbf{q}}_t = \frac{\mathbf{P}_{t-1}\boldsymbol{\Delta\phi}_t}{\sigma_{t-1}^2 + \boldsymbol{\Delta\phi}_t^\top \mathbf{P}_{t-1}\boldsymbol{\Delta\phi}_t},$$

then

$$\mathbf{P}_t = \mathbf{P}_{t-1} - \hat{\mathbf{q}}_t \boldsymbol{\Delta\phi}_t^\top \mathbf{P}_{t-1}.$$

We are now ready to derive the recursive update rule for the posterior mean $\hat{\mathbf{w}}_t$.

$$\begin{aligned}
\hat{\mathbf{w}}_t &= \mathbf{P}_t \boldsymbol{\Delta\Phi}_t \boldsymbol{\Sigma}_t^{-1}\mathbf{r}_{t-1} \\
&= \mathbf{P}_t \left[\boldsymbol{\Delta\Phi}_{t-1}, \quad \boldsymbol{\Delta\phi}_t\right] \begin{bmatrix} \boldsymbol{\Sigma}_{t-1}^{-1} & \mathbf{0} \\ \mathbf{0}^\top & \sigma_{t-1}^{-2} \end{bmatrix} \begin{pmatrix} \mathbf{r}_{t-2} \\ r_{t-1} \end{pmatrix} \\
&= \mathbf{P}_t \left(\boldsymbol{\Delta\Phi}_{t-1}\boldsymbol{\Sigma}_{t-1}^{-1}\mathbf{r}_{t-2} + \frac{1}{\sigma_{t-1}^2}\boldsymbol{\Delta\phi}_t r_{t-1} \right) \\
&= \left(\mathbf{P}_{t-1} - \hat{\mathbf{q}}_t \boldsymbol{\Delta\phi}_t^\top \mathbf{P}_{t-1} \right) \boldsymbol{\Delta\Phi}_{t-1}\boldsymbol{\Sigma}_{t-1}^{-1}\mathbf{r}_{t-2} + \hat{\mathbf{q}}_t r_{t-1} \\
&= \hat{\mathbf{w}}_{t-1} + \hat{\mathbf{q}}_t \left(r_{t-1} - \boldsymbol{\Delta\phi}_t^\top \hat{\mathbf{w}}_{t-1} \right).
\end{aligned}$$

In the equality before the last we used the (easily obtained) identity $\hat{\mathbf{q}}_t = \frac{1}{\sigma_{t-1}^2}\mathbf{P}_t\boldsymbol{\Delta\phi}_t$.

A.1.2 Symmetric Parametric GPTD Updates

Let us define

$$\mathbf{Q}_t = \left(\boldsymbol{\Delta\Phi}_t^\top \boldsymbol{\Delta\Phi}_t + \boldsymbol{\Sigma}_t \right)^{-1}.$$

Then, the posterior moments (4.3.11, 4.3.12) are given by

$$\hat{\mathbf{w}}_t = \boldsymbol{\Delta\Phi}_t \mathbf{Q}_t \mathbf{r}_{t-1}, \quad \text{and } \mathbf{P}_t = \mathbf{I} - \boldsymbol{\Delta\Phi}_t \mathbf{Q}_t \boldsymbol{\Delta\Phi}_t^\top.$$

$\mathbf{Q}_t^{-1}$ satisfies the following recursion

$$\mathbf{Q}_t^{-1} = \begin{bmatrix} \mathbf{Q}_{t-1}^{-1} & \boldsymbol{\Delta\Phi}_{t-1}^\top \boldsymbol{\Delta\phi}_t \\ \boldsymbol{\Delta\phi}_t^\top \boldsymbol{\Delta\Phi}_{t-1} & \sigma_{t-1}^2 + \boldsymbol{\Delta\phi}_t^\top \boldsymbol{\Delta\phi}_t \end{bmatrix}.$$

Application of the partitioned matrix inversion formula (see Appendix D.4) results in

$$\mathbf{Q}_t = \begin{bmatrix} \mathbf{Q}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t} \begin{pmatrix} \boldsymbol{g}_t \\ -1 \end{pmatrix} \left(\boldsymbol{g}_t^\top, \quad -1 \right),$$

where

$$\begin{aligned}
\boldsymbol{g}_t &= \mathbf{Q}_{t-1}\boldsymbol{\Delta\Phi}_{t-1}^\top \boldsymbol{\Delta\phi}_t, \\
s_t &= \sigma_{t-1}^2 + \boldsymbol{\Delta\phi}_t^\top \boldsymbol{\Delta\phi}_t - \boldsymbol{g}_t^\top \boldsymbol{\Delta\Phi}_{t-1}^\top \boldsymbol{\Delta\phi}_t = \sigma_{t-1}^2 + \boldsymbol{\Delta\phi}_t^\top \mathbf{P}_{t-1}\boldsymbol{\Delta\phi}_t.
\end{aligned}$$

Let us derive the recursions for $\hat{\mathbf{w}}_t$ and $\mathbf{P}_t$:

$$\begin{aligned}
\hat{\mathbf{w}}_t &= \boldsymbol{\Delta\Phi}_t \mathbf{Q}_t \mathbf{r}_{t-1} \\
&= \left[\boldsymbol{\Delta\Phi}_{t-1}, \quad \boldsymbol{\Delta\phi}_t\right] \left(\begin{bmatrix} \mathbf{Q}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \begin{pmatrix} \boldsymbol{g}_t \\ -1 \end{pmatrix} \left(\boldsymbol{g}_t^\top, \quad -1\right) \right) \begin{pmatrix} \mathbf{r}_{t-2} \\ r_{t-1} \end{pmatrix} \\
&= \hat{\mathbf{w}}_{t-1} - \frac{1}{s_t}\mathbf{p}_t \left(\boldsymbol{g}_t^\top \mathbf{r}_{t-2} - r_{t-1}\right) \\
&= \hat{\mathbf{w}}_{t-1} + \frac{1}{s_t}\mathbf{p}_t d_t,
\end{aligned}$$

where we defined $d_t = r_{t-1} - \boldsymbol{\Delta\phi}_t^\top \hat{\mathbf{w}}_{t-1}$. Similarly,

$$\begin{aligned}
\mathbf{P}_t &= \mathbf{I} - \boldsymbol{\Delta\Phi}_t \mathbf{Q}_t \boldsymbol{\Delta\Phi}_t^\top \\
&= \mathbf{I} - \boldsymbol{\Delta\Phi}_{t-1}\mathbf{Q}_{t-1}\boldsymbol{\Delta\Phi}_{t-1}^\top - \frac{1}{s_t}\left(\boldsymbol{\Delta\Phi}_{t-1}\boldsymbol{g}_t - \boldsymbol{\Delta\phi}_t\right)\left(\boldsymbol{\Delta\Phi}_{t-1}\boldsymbol{g}_t - \boldsymbol{\Delta\phi}_t\right)^\top \\
&= \mathbf{P}_{t-1} - \frac{1}{s_t}\mathbf{p}_t\mathbf{p}_t^\top,
\end{aligned}$$

where we defined $\mathbf{p}_t = \boldsymbol{\Delta\phi}_t - \boldsymbol{\Delta\Phi}_{t-1}\boldsymbol{g}_t = \mathbf{P}_{t-1}\boldsymbol{\Delta\phi}_t$.

A.1.3 Exact Nonparametric GPTD Updates

In the nonparametric case the parameters of the posterior moments are (see Eq. 4.3.18, 4.3.19)

$$\boldsymbol{\alpha}_t = \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{r}_{t-1}, \quad \text{and } \mathbf{C}_t = \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{H}_t,$$

where $\mathbf{Q}_t = \left(\mathbf{H}_t\mathbf{K}_t\mathbf{H}_t^\top + \boldsymbol{\Sigma}_t\right)^{-1}$. The matrices $\mathbf{H}_t$, $\mathbf{K}_t$ and $\boldsymbol{\Sigma}_t$ satisfy simple recursions:

$$\begin{aligned}
\mathbf{H}_t &= \left[\begin{array}{cc} \mathbf{H}_{t-1} & \mathbf{0} \\ \hline \multicolumn{2}{c}{\mathbf{h}_t^\top} \end{array}\right], \quad \text{where } \mathbf{h}_t = (0, \dots, 1, -\gamma)^\top, \\
\mathbf{K}_t &= \begin{bmatrix} \mathbf{K}_{t-1} & \mathbf{k}_{t-1}(\mathbf{x}_t) \\ \mathbf{k}_{t-1}(\mathbf{x}_t)^\top & k(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix}, \quad \text{and } \boldsymbol{\Sigma}_t = \begin{bmatrix} \boldsymbol{\Sigma}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & \sigma_{t-1}^2 \end{bmatrix}.
\end{aligned}$$

These recursions result in a corresponding recursion for $\mathbf{Q}_t^{-1}$:

$$\begin{aligned}
\mathbf{Q}_t^{-1} &= \mathbf{H}_t\mathbf{K}_t\mathbf{H}_t^\top + \mathbf{\Sigma}_t \\
&= \left[\begin{array}{cc} \mathbf{H}_{t-1} & \mathbf{0} \\ \hline \multicolumn{2}{c}{\mathbf{h}_t^\top} \end{array}\right] \begin{bmatrix} \mathbf{K}_{t-1} & \mathbf{k}_{t-1}(\mathbf{x}_t) \\ \mathbf{k}_{t-1}(\mathbf{x}_t)^\top & k(\mathbf{x}_t,\mathbf{x}_t) \end{bmatrix} \left[\begin{array}{c|c} \mathbf{H}_{t-1}^\top & \mathbf{h}_t \\ \mathbf{0}^\top & \end{array}\right] + \begin{bmatrix} \mathbf{\Sigma}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & \sigma_{t-1}^2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{Q}_{t-1}^{-1} & \mathbf{H}_{t-1}\mathbf{\Delta k}_t \\ (\mathbf{H}_{t-1}\mathbf{\Delta k}_t)^\top & \Delta k_{tt} + \sigma_{t-1}^2 \end{bmatrix},
\end{aligned}$$

where we defined

$$\begin{aligned}
&\mathbf{\Delta k}_t \stackrel{\text{def}}{=} \mathbf{K}_{t-1}\mathbf{h}_t = \mathbf{k}_{t-1}(\mathbf{x}_{t-1}) - \gamma\mathbf{k}_{t-1}(\mathbf{x}_t), \\
&\Delta k_{tt} \stackrel{\text{def}}{=} \mathbf{h}_t^\top\mathbf{\Delta k}_t = k(\mathbf{x}_{t-1},\mathbf{x}_{t-1}) - 2\gamma k(\mathbf{x}_{t-1},\mathbf{x}_t) + \gamma^2 k(\mathbf{x}_t,\mathbf{x}_t).
\end{aligned}$$

Using the partitioned matrix inversion formula (see Appendix D.4), we obtain the following expression for $\mathbf{Q}_t$:

$$\mathbf{Q}_t = \frac{1}{s_t}\begin{bmatrix} s_t\mathbf{Q}_{t-1} + \boldsymbol{g}_t\boldsymbol{g}_t^\top & -\boldsymbol{g}_t \\ -\boldsymbol{g}_t^\top & 1 \end{bmatrix},$$

where $\boldsymbol{g}_t = \mathbf{Q}_t\mathbf{H}_{t-1}\mathbf{\Delta k}_t$, and $s_t = \sigma_{t-1}^2 + \Delta k_{tt} - \mathbf{\Delta k}_t^\top\mathbf{C}_{t-1}\mathbf{\Delta k}_t$. Let us define

$$\mathbf{c}_t = \mathbf{h}_t - \begin{pmatrix} \mathbf{H}_{t-1}^\top\boldsymbol{g}_t \\ 0 \end{pmatrix} = \mathbf{h}_t - \begin{pmatrix} \mathbf{C}_{t-1}\mathbf{\Delta k}_t \\ 0 \end{pmatrix}.$$

We are now ready to derive the recursions for $\mathbf{C}_t$ and $\boldsymbol{\alpha}_t$:

$$\begin{aligned}
\mathbf{C}_t &= \mathbf{H}_t^\top\mathbf{Q}_t\mathbf{H}_t \\
&= \frac{1}{s_t}\left[\begin{array}{c|c} \mathbf{H}_{t-1}^\top & \mathbf{h}_t \\ \mathbf{0}^\top & \end{array}\right] \begin{bmatrix} s_t\mathbf{Q}_{t-1} + \boldsymbol{g}_t\boldsymbol{g}_t^\top & -\boldsymbol{g}_t \\ -\boldsymbol{g}_t^\top & 1 \end{bmatrix} \left[\begin{array}{cc} \mathbf{H}_{t-1} & \mathbf{0} \\ \hline \multicolumn{2}{c}{\mathbf{h}_t^\top} \end{array}\right] \\
&= \begin{bmatrix} \mathbf{H}_{t-1}\mathbf{Q}_{t-1}\mathbf{H}_{t-1}^\top & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t}\left(\begin{pmatrix} \mathbf{H}_{t-1}^\top\boldsymbol{g}_t \\ 0 \end{pmatrix} - \mathbf{h}_t\right)\left(\begin{pmatrix} \mathbf{H}_{t-1}^\top\boldsymbol{g}_t \\ 0 \end{pmatrix} - \mathbf{h}_t\right)^\top \\
&= \begin{bmatrix} \mathbf{C}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t}\mathbf{c}_t\mathbf{c}_t^\top.
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{\alpha}_t &= \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{r}_{t-1} \\
&= \frac{1}{s_t} \begin{bmatrix} \mathbf{H}_{t-1}^\top & \Big| \mathbf{h}_t \\ \mathbf{0}^\top & \end{bmatrix} \begin{bmatrix} s_t \mathbf{Q}_{t-1} + \boldsymbol{g}_t \boldsymbol{g}_t^\top & -\boldsymbol{g}_t \\ -\boldsymbol{g}_t^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{r}_{t-2} \\ r_{t-1} \end{bmatrix} \\
&= \begin{pmatrix} \boldsymbol{\alpha}_{t-1} \\ 0 \end{pmatrix} + \frac{1}{s_t} \left(\begin{pmatrix} \mathbf{H}_{t-1}^\top \boldsymbol{g}_t \\ 0 \end{pmatrix} - \mathbf{h}_t \right) \left(\boldsymbol{g}_t^\top \mathbf{r}_{t-2} - r_{t-1} \right) \\
&= \begin{pmatrix} \boldsymbol{\alpha}_{t-1} \\ 0 \end{pmatrix} + \frac{\mathbf{c}_t}{s_t} \left(r_{t-1} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top \boldsymbol{\alpha}_{t-1} \right).
\end{aligned}$$

A.1.4 Sparse Nonparametric GPTD Updates

Here we derive the updates for the reduced posterior mean and covariance parameters

$$\tilde{\boldsymbol{\alpha}}_t = \tilde{\mathbf{H}}_t^\top \tilde{\mathbf{Q}}_t \mathbf{r}_{t-1}, \quad \tilde{\mathbf{C}}_t = \tilde{\mathbf{H}}_t^\top \tilde{\mathbf{Q}}_t \tilde{\mathbf{H}}_t.$$

where

$$\tilde{\mathbf{H}}_t = \mathbf{H}_t \mathbf{A}_t, \quad \tilde{\mathbf{Q}}_t = (\tilde{\mathbf{H}}_t \tilde{\mathbf{K}}_t \tilde{\mathbf{H}}_t^\top + \boldsymbol{\Sigma}_t)^{-1}.$$

At each time-step we will have to consider two possible contingencies: Either $\mathcal{D}_t = \mathcal{D}_{t-1}$, or $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$. Let us handle these in turn.

Case 1: $\mathcal{D}_t = \mathcal{D}_{t-1}$

Hence $\tilde{\mathbf{K}}_t = \tilde{\mathbf{K}}_{t-1}$,

$$\mathbf{A}_t = \begin{bmatrix} \mathbf{A}_{t-1} \\ \boldsymbol{a}_t^\top \end{bmatrix}, \; \tilde{\mathbf{H}}_t = \begin{bmatrix} \tilde{\mathbf{H}}_{t-1} \\ \boldsymbol{a}_{t-1}^\top - \gamma \boldsymbol{a}_t^\top \end{bmatrix}.$$

Defining $\tilde{\mathbf{h}}_t = \boldsymbol{a}_{t-1} - \gamma \boldsymbol{a}_t$, we have

$$\tilde{\mathbf{Q}}_t^{-1} = \begin{bmatrix} \tilde{\mathbf{H}}_{t-1} \\ \tilde{\mathbf{h}}_t^\top \end{bmatrix} \tilde{\mathbf{K}}_{t-1} \left[\tilde{\mathbf{H}}_{t-1}, \tilde{\mathbf{h}}_t \right] + \begin{bmatrix} \boldsymbol{\Sigma}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & \sigma_{t-1}^2 \end{bmatrix}$$

$$= \begin{bmatrix} \tilde{\mathbf{Q}}_{t-1}^{-1} & \tilde{\mathbf{H}}_{t-1} \tilde{\mathbf{K}}_{t-1} \tilde{\mathbf{h}}_t \\ (\tilde{\mathbf{H}}_{t-1} \tilde{\mathbf{K}}_{t-1} \tilde{\mathbf{h}}_t)^\top & \tilde{\mathbf{h}}_t^\top \tilde{\mathbf{K}}_{t-1} \tilde{\mathbf{h}}_t + \sigma_{t-1}^2 \end{bmatrix}.$$

Defining

$$\begin{aligned}
\boldsymbol{\Delta}\tilde{\mathbf{k}}_t &\overset{\text{def}}{=} \tilde{\mathbf{K}}_{t-1} \tilde{\mathbf{h}}_t = \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_{t-1}) - \gamma \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t), \\
\Delta k_{tt} &\overset{\text{def}}{=} \tilde{\mathbf{h}}_t^\top \tilde{\mathbf{K}}_{t-1} \tilde{\mathbf{h}}_t = \tilde{\mathbf{h}}_t^\top \boldsymbol{\Delta}\tilde{\mathbf{k}}_t
\end{aligned}$$

we get

$$\tilde{\mathbf{Q}}_t^{-1} = \begin{bmatrix} \tilde{\mathbf{Q}}_{t-1}^{-1} & \tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t \\ (\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t)^\top & \Delta k_{tt} + \sigma_{t-1}^2 \end{bmatrix} .$$

We obtain $\tilde{\mathbf{Q}}_t$ using the partitioned matrix inversion formula (Appendix D.4).

$$\tilde{\mathbf{Q}}_t = \frac{1}{s_t}\begin{bmatrix} s_t\tilde{\mathbf{Q}}_{t-1} + \tilde{\boldsymbol{g}}_t\tilde{\boldsymbol{g}}_t^\top & -\tilde{\boldsymbol{g}}_t \\ -\tilde{\boldsymbol{g}}_t^\top & 1 \end{bmatrix} ,$$

where $\tilde{\boldsymbol{g}}_t = \tilde{\mathbf{Q}}_{t-1}\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t$, $s_t = \sigma_{t-1}^2 + \Delta k_{tt} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top\tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t$. Let us also define

$$\tilde{\mathbf{c}}_t = \tilde{\mathbf{h}}_t - \tilde{\mathbf{H}}_{t-1}^\top\tilde{\boldsymbol{g}}_t = \tilde{\mathbf{h}}_t - \tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t.$$

Then, $s_t = \sigma_{t-1}^2 - \tilde{\mathbf{c}}_t^\top\boldsymbol{\Delta}\tilde{\mathbf{k}}_t$.

We are now ready to compute $\tilde{\mathbf{C}}_t$:

$$\begin{aligned}
\tilde{\mathbf{C}}_t &= \tilde{\mathbf{H}}_t^\top\tilde{\mathbf{Q}}_t\tilde{\mathbf{H}}_t \\
&= \frac{1}{s_t}\begin{bmatrix}\tilde{\mathbf{H}}_{t-1}^\top, & \tilde{\mathbf{h}}_t\end{bmatrix}\begin{bmatrix} s_t\tilde{\mathbf{Q}}_{t-1} + \tilde{\boldsymbol{g}}_t\tilde{\boldsymbol{g}}_t^\top & -\tilde{\boldsymbol{g}}_t \\ -\tilde{\boldsymbol{g}}_t^\top & 1 \end{bmatrix}\begin{bmatrix}\tilde{\mathbf{H}}_{t-1} \\ \tilde{\mathbf{h}}_t^\top\end{bmatrix} \\
&= \tilde{\mathbf{H}}_{t-1}^\top\tilde{\mathbf{Q}}_{t-1}\tilde{\mathbf{H}}_{t-1} + \frac{1}{s_t}\begin{bmatrix}\tilde{\mathbf{H}}_{t-1}^\top & \tilde{\mathbf{h}}_t\end{bmatrix}\begin{pmatrix}\tilde{\boldsymbol{g}}_t \\ -1\end{pmatrix}\begin{pmatrix}\tilde{\boldsymbol{g}}_t^\top & -1\end{pmatrix}\begin{bmatrix}\tilde{\mathbf{H}}_{t-1} \\ \tilde{\mathbf{h}}_t^\top\end{bmatrix} \\
&= \tilde{\mathbf{C}}_{t-1} + \frac{1}{s_t}\left(\tilde{\mathbf{H}}_{t-1}\tilde{\boldsymbol{g}}_t - \tilde{\mathbf{h}}_t\right)\left(\tilde{\mathbf{H}}_{t-1}\tilde{\boldsymbol{g}}_t - \tilde{\mathbf{h}}_t\right)^\top \\
&= \tilde{\mathbf{C}}_{t-1} + \frac{1}{s_t}\tilde{\mathbf{c}}_t\tilde{\mathbf{c}}_t^\top
\end{aligned}$$

The derivation of $\tilde{\boldsymbol{\alpha}}_t$ proceeds similarly:

$$\begin{aligned}
\tilde{\boldsymbol{\alpha}}_t &= \tilde{\mathbf{H}}_t^\top\tilde{\mathbf{Q}}_t\mathbf{r}_{t-1} \\
&= \frac{1}{s_t}\begin{bmatrix}\tilde{\mathbf{H}}_{t-1}^\top, & \tilde{\mathbf{h}}_t\end{bmatrix}\begin{bmatrix} s_t\tilde{\mathbf{Q}}_{t-1} + \tilde{\boldsymbol{g}}_t\boldsymbol{g}_t^\top & -\boldsymbol{g}_t \\ -\boldsymbol{g}_t^\top & 1 \end{bmatrix}\begin{bmatrix}\mathbf{r}_{t-2} \\ r_{t-1}\end{bmatrix} \\
&= \tilde{\mathbf{H}}_{t-1}^\top\tilde{\mathbf{Q}}_{t-1}\mathbf{r}_{t-2} + \frac{1}{s_t}\begin{bmatrix}\tilde{\mathbf{H}}_{t-1}^\top, & \tilde{\mathbf{h}}_t\end{bmatrix}\begin{pmatrix}\tilde{\boldsymbol{g}}_t \\ -1\end{pmatrix}\begin{pmatrix}\tilde{\boldsymbol{g}}_t^\top & -1\end{pmatrix}\begin{pmatrix}\mathbf{r}_{t-2} \\ r_{t-1}\end{pmatrix} \\
&= \tilde{\boldsymbol{\alpha}}_{t-1} + \frac{1}{s_t}\left(\tilde{\mathbf{H}}_{t-1}\tilde{\boldsymbol{g}}_t - \tilde{\mathbf{h}}_t\right)\left(\tilde{\boldsymbol{g}}_t^\top\mathbf{r}_{t-2} - r_{t-1}\right)^\top \\
&= \tilde{\boldsymbol{\alpha}}_{t-1} + \frac{\tilde{\mathbf{c}}_t}{s_t}\left(r_{t-1} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top\tilde{\boldsymbol{\alpha}}_{t-1}\right).
\end{aligned}$$

Case 2: $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$

Here, $\tilde{\mathbf{K}}_t$ is given by Eq. 2.2.10, which we repeat here for clarity:

$$\tilde{\mathbf{K}}_t = \begin{bmatrix} \tilde{\mathbf{K}}_{t-1} & \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) \\ \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top & k(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix}$$

Furthermore, $\boldsymbol{a}_t = (0, \ldots, 1)^\top$ since the last member added to the dictionary is $\mathbf{x}_t$ itself (and $\boldsymbol{\phi}(\mathbf{x}_t)$ is exactly representable by itself). Therefore

$$\mathbf{A}_t = \begin{bmatrix} \mathbf{A}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix}, \quad \tilde{\mathbf{H}}_t = \begin{bmatrix} \tilde{\mathbf{H}}_{t-1} & \mathbf{0} \\ \boldsymbol{a}_{t-1}^\top & -\gamma \end{bmatrix} = \left[\begin{array}{c} \tilde{\mathbf{H}}_{t-1} \quad \mathbf{0} \\ \hline \tilde{\mathbf{h}}_t^\top \end{array} \right],$$

where we defined

$$\tilde{\mathbf{h}}_t = \begin{pmatrix} \boldsymbol{a}_{t-1} \\ 0 \end{pmatrix} - \gamma \boldsymbol{a}_t = \begin{pmatrix} \boldsymbol{a}_{t-1} \\ -\gamma \end{pmatrix}.$$

The recursion for $\tilde{\mathbf{Q}}_t^{-1}$ is slightly different here:

$$\begin{aligned} \tilde{\mathbf{Q}}_t^{-1} &= \left[\begin{array}{c} \tilde{\mathbf{H}}_{t-1} \quad \mathbf{0} \\ \hline \tilde{\mathbf{h}}_t^\top \end{array} \right] \begin{bmatrix} \tilde{\mathbf{K}}_{t-1} & \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) \\ \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top & k(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix} \left[\begin{array}{c|c} \tilde{\mathbf{H}}_{t-1}^\top & \tilde{\mathbf{h}}_t \\ \mathbf{0}^\top & \end{array} \right] + \begin{bmatrix} \mathbf{\Sigma}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & \sigma_{t-1}^2 \end{bmatrix} \\ &= \begin{bmatrix} \tilde{\mathbf{Q}}_{t-1}^{-1} & \tilde{\mathbf{H}}_{t-1} \mathbf{\Delta} \tilde{\mathbf{k}}_t \\ (\tilde{\mathbf{H}}_{t-1} \mathbf{\Delta} \tilde{\mathbf{k}}_t)^\top & \Delta k_{tt} + \sigma_{t-1}^2 \end{bmatrix}, \end{aligned}$$

where we defined

$$\begin{aligned} \mathbf{\Delta} \tilde{\mathbf{k}}_t &\overset{\text{def}}{=} \tilde{\mathbf{K}}_{t-1} \boldsymbol{a}_{t-1} - \gamma \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) = \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_{t-1}) - \gamma \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t), \\ \Delta k_{tt} &\overset{\text{def}}{=} \boldsymbol{a}_{t-1}^\top \tilde{\mathbf{K}}_{t-1} \boldsymbol{a}_{t-1} - 2\gamma \boldsymbol{a}_{t-1}^\top \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) + \gamma k(\mathbf{x}_t, \mathbf{x}_t) \\ &= \boldsymbol{a}_{t-1}^\top \left(\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_{t-1}) - 2\gamma \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) \right) + \gamma^2 k(\mathbf{x}_t, \mathbf{x}_t). \end{aligned}$$

Denoting $s_t = \sigma_{t-1}^2 + \Delta k_{tt} - \mathbf{\Delta} \tilde{\mathbf{k}}_t^\top \tilde{\mathbf{C}}_{t-1} \mathbf{\Delta} \tilde{\mathbf{k}}_t$ and using again the partitioned matrix inversion formula we get

$$\tilde{\mathbf{Q}}_t = \frac{1}{s_t} \begin{bmatrix} s_t \tilde{\mathbf{Q}}_{t-1} + \tilde{\boldsymbol{g}}_t \tilde{\boldsymbol{g}}_t^\top & -\tilde{\boldsymbol{g}}_t \\ -\tilde{\boldsymbol{g}}_t^\top & 1 \end{bmatrix},$$

with $\tilde{\boldsymbol{g}}_t = \tilde{\mathbf{Q}}_t \tilde{\mathbf{H}}_{t-1} \mathbf{\Delta} \tilde{\mathbf{k}}_t$, as before. Let us define

$$\tilde{\mathbf{c}}_t = \tilde{\mathbf{h}}_t - \begin{pmatrix} \tilde{\mathbf{H}}_{t-1}^\top \tilde{\boldsymbol{g}}_t \\ 0 \end{pmatrix} = \tilde{\mathbf{h}}_t - \begin{pmatrix} \tilde{\mathbf{C}}_{t-1} \mathbf{\Delta} \tilde{\mathbf{k}}_t \\ 0 \end{pmatrix}.$$

We are now ready to compute $\tilde{\mathbf{C}}_t$ and $\tilde{\boldsymbol{\alpha}}_t$:

$$\begin{aligned}
\tilde{\mathbf{C}}_t &= \tilde{\mathbf{H}}_t^\top \tilde{\mathbf{Q}}_t \tilde{\mathbf{H}}_t \\
&= \frac{1}{s_t} \left[\begin{array}{c|c} \tilde{\mathbf{H}}_{t-1}^\top & \tilde{\mathbf{h}}_t \\ \mathbf{0}^\top & \end{array} \right] \left[\begin{array}{cc} s_t \tilde{\mathbf{Q}}_{t-1} + \tilde{\boldsymbol{g}}_t \tilde{\boldsymbol{g}}_t^\top & -\tilde{\boldsymbol{g}}_t \\ -\tilde{\boldsymbol{g}}_t^\top & 1 \end{array} \right] \left[\begin{array}{cc} \tilde{\mathbf{H}}_{t-1} & \mathbf{0} \\ \hline \multicolumn{2}{c}{\tilde{\mathbf{h}}_t^\top} \end{array} \right] \\
&= \begin{bmatrix} \tilde{\mathbf{H}}_{t-1} \tilde{\mathbf{Q}}_{t-1} \tilde{\mathbf{H}}_{t-1}^\top & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t} \left(\begin{pmatrix} \tilde{\mathbf{H}}_{t-1}^\top \tilde{\boldsymbol{g}}_t \\ 0 \end{pmatrix} - \tilde{\mathbf{h}}_t \right) \left(\begin{pmatrix} \tilde{\mathbf{H}}_{t-1}^\top \tilde{\boldsymbol{g}}_t \\ 0 \end{pmatrix} - \tilde{\mathbf{h}}_t \right)^\top \\
&= \begin{bmatrix} \tilde{\mathbf{C}}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t} \tilde{\mathbf{c}}_t \tilde{\mathbf{c}}_t^\top .
\end{aligned}$$

$$\begin{aligned}
\tilde{\boldsymbol{\alpha}}_t &= \tilde{\mathbf{H}}_t^\top \tilde{\mathbf{Q}}_t \mathbf{r}_{t-1} \\
&= \frac{1}{s_t} \left[\begin{array}{c|c} \tilde{\mathbf{H}}_{t-1}^\top & \tilde{\mathbf{h}}_t \\ \mathbf{0}^\top & \end{array} \right] \left[\begin{array}{cc} s_t \tilde{\mathbf{Q}}_{t-1} + \tilde{\boldsymbol{g}}_t \tilde{\boldsymbol{g}}_t^\top & -\tilde{\boldsymbol{g}}_t \\ -\tilde{\boldsymbol{g}}_t^\top & 1 \end{array} \right] \left[\begin{array}{c} \mathbf{r}_{t-2} \\ r_{t-1} \end{array} \right] \\
&= \begin{pmatrix} \tilde{\boldsymbol{\alpha}}_{t-1} \\ 0 \end{pmatrix} + \frac{1}{s_t} \left(\begin{pmatrix} \tilde{\mathbf{H}}_{t-1}^\top \tilde{\boldsymbol{g}}_t \\ 0 \end{pmatrix} - \tilde{\mathbf{h}}_t \right) \left(\boldsymbol{g}_t^\top \mathbf{r}_{t-2} - r_{t-1} \right) \\
&= \begin{pmatrix} \tilde{\boldsymbol{\alpha}}_{t-1} \\ 0 \end{pmatrix} + \frac{\tilde{\mathbf{c}}_t}{s_t} \left(r_{t-1} - \boldsymbol{\Delta} \tilde{\mathbf{k}}_t^\top \tilde{\boldsymbol{\alpha}}_{t-1} \right) .
\end{aligned}$$

A.2 Stochastic Transitions

A.2.1 Parametric Monte-Carlo GPTD Updates

Recall the expressions for the parametric posterior moments (Eq. 4.4.40):

$$\hat{\mathbf{w}}_t = \mathbf{\Delta\Phi}_t \mathbf{Q}_t \mathbf{r}_{t-1}, \quad \mathbf{P}_t = \mathbf{I} - \mathbf{\Delta\Phi}_t \mathbf{Q}_t \mathbf{\Delta\Phi}_t^\top, \quad \text{where } \mathbf{Q}_t = \left(\mathbf{\Delta\Phi}_t^\top \mathbf{\Delta\Phi}_t + \mathbf{\Sigma}_t\right)^{-1},$$

where we used the definition $\mathbf{\Delta\Phi}_t = \mathbf{\Phi}_t \mathbf{H}_t^\top$. In the sequel we will also use the following definitions:

$$\mathbf{h}_t = [0, \ldots, 1, -\gamma]^\top, \quad \text{and } \mathbf{\Delta}\boldsymbol{\phi}_t = \mathbf{\Phi}_t \mathbf{h}_t = \boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t).$$

The matrices $\mathbf{\Delta\Phi}_t$, $\mathbf{\Sigma}_t$ and $\mathbf{Q}_t^{-1}$ may be written recursively as follows:

$$\mathbf{\Delta\Phi}_t = \left[\mathbf{\Delta\Phi}_{t-1}, \quad \mathbf{\Delta}\boldsymbol{\phi}_t\right], \tag{A.2.1}$$

$$\mathbf{\Sigma}_t = \begin{bmatrix} \mathbf{\Sigma}_{t-1} & , -\gamma\sigma_{t-1}^2 \mathbf{u} \\ -\gamma\sigma_{t-1}^2 \mathbf{u}^\top & , \sigma_{t-1}^2 + \gamma^2\sigma_t^2 \end{bmatrix}, \quad \text{and}$$

$$\mathbf{Q}_t^{-1} = \begin{bmatrix} \mathbf{Q}_{t-1}^{-1} & , \mathbf{\Delta\Phi}_{t-1}^\top \mathbf{\Delta}\boldsymbol{\phi}_t - \gamma\sigma_{t-1}^2 \mathbf{u} \\ (\mathbf{\Delta\Phi}_{t-1}^\top \mathbf{\Delta}\boldsymbol{\phi}_t - \gamma\sigma_{t-1}^2 \mathbf{u})^\top & , \mathbf{\Delta}\boldsymbol{\phi}_t^\top \mathbf{\Delta}\boldsymbol{\phi}_t + \sigma_{t-1}^2 + \gamma^2\sigma_t^2 \end{bmatrix}, \tag{A.2.2}$$

where $\mathbf{u} = [0, \ldots, 1]^\top$. Using the partitioned matrix inversion formula (Appendix D.4) we may invert $\mathbf{Q}_t^{-1}$ to obtain

$$\mathbf{Q}_t = \frac{1}{s_t} \begin{bmatrix} s_t \mathbf{Q}_{t-1} + \boldsymbol{g}_t \boldsymbol{g}_t^\top & -\boldsymbol{g}_t \\ -\boldsymbol{g}_t^\top & 1 \end{bmatrix},$$

where

$$\boldsymbol{g}_t = \mathbf{Q}_{t-1}\left(\mathbf{\Delta\Phi}_{t-1}^\top \mathbf{\Delta}\boldsymbol{\phi}_t - \gamma\sigma_{t-1}^2 \mathbf{u}\right)$$
$$s_t = \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \mathbf{\Delta}\boldsymbol{\phi}_t^\top \mathbf{\Delta}\boldsymbol{\phi}_t - \left(\mathbf{\Delta\Phi}_{t-1}^\top \mathbf{\Delta}\boldsymbol{\phi}_t - \gamma\sigma_{t-1}^2 \mathbf{u}\right)^\top \boldsymbol{g}_t.$$

Let us write the expression for $\hat{\mathbf{w}}_t$ from Eq. (4.4.40):

$$\begin{aligned}
\hat{\mathbf{w}}_t &= \boldsymbol{\Delta\Phi}_t \mathbf{Q}_t \mathbf{r}_{t-1} \\
&= \frac{1}{s_t}\left[\boldsymbol{\Delta\Phi}_{t-1}, \boldsymbol{\Delta\phi}_t\right] \begin{bmatrix} s_t\mathbf{Q}_{t-1} + \boldsymbol{g}_t\boldsymbol{g}_t^\top & -\boldsymbol{g}_t \\ -\boldsymbol{g}_t^\top & 1 \end{bmatrix} \begin{pmatrix} \mathbf{r}_{t-2} \\ r_{t-1} \end{pmatrix} \\
&= \boldsymbol{\Delta\Phi}_{t-1}\mathbf{Q}_{t-1}\mathbf{r}_{t-2} + \frac{1}{s_t}\left[\boldsymbol{\Delta\Phi}_{t-1}, \quad \boldsymbol{\Delta\phi}_t\right] \begin{pmatrix} \boldsymbol{g}_t \\ -1 \end{pmatrix} \left(\boldsymbol{g}_t^\top, \quad -1\right) \begin{pmatrix} \mathbf{r}_{t-2} \\ r_{t-1} \end{pmatrix} \\
&= \hat{\mathbf{w}}_{t-1} + \frac{1}{s_t}\left(\boldsymbol{\Delta\Phi}_{t-1}\boldsymbol{g}_t - \boldsymbol{\Delta\phi}_t\right)\left(\boldsymbol{g}_t^\top \mathbf{r}_{t-2} - r_{t-1}\right) \\
&= \hat{\mathbf{w}}_{t-1} + \frac{\mathbf{p}_t}{s_t} d_t,
\end{aligned}$$

where we have defined $d_t = r_{t-1} - \boldsymbol{g}_t^\top \mathbf{r}_{t-2}$ and $\mathbf{p}_t = \boldsymbol{\Delta\phi}_t - \boldsymbol{\Delta\Phi}_{t-1}\boldsymbol{g}_t$.

We treat $\mathbf{P}_t$ similarly:

$$\begin{aligned}
\mathbf{P}_t &= \mathbf{I} - \boldsymbol{\Delta\Phi}_t\mathbf{Q}_t\boldsymbol{\Delta\Phi}_t^\top \\
&= \mathbf{I} - \frac{1}{s_t}\left[\boldsymbol{\Delta\Phi}_{t-1}, \boldsymbol{\Delta\phi}_t\right] \begin{bmatrix} s_t\mathbf{Q}_{t-1} + \boldsymbol{g}_t\boldsymbol{g}_t^\top & -\boldsymbol{g}_t \\ -\boldsymbol{g}_t^\top & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta\Phi}_{t-1}^\top \\ \boldsymbol{\Delta\phi}_t^\top \end{bmatrix} \\
&= \mathbf{I} - \boldsymbol{\Delta\Phi}_{t-1}\mathbf{Q}_{t-1}\boldsymbol{\Delta\Phi}_{t-1}^\top - \frac{1}{s_t}\left[\boldsymbol{\Delta\Phi}_{t-1}, \quad \boldsymbol{\Delta\phi}_t\right] \begin{pmatrix} \boldsymbol{g}_t \\ -1 \end{pmatrix} \left(\boldsymbol{g}_t^\top, \quad -1\right) \begin{bmatrix} \boldsymbol{\Delta\Phi}_{t-1}^\top \\ \boldsymbol{\Delta\phi}_t^\top \end{bmatrix} \\
&= \mathbf{P}_{t-1} - \frac{1}{s_t}\left(\boldsymbol{\Delta\Phi}_{t-1}\boldsymbol{g}_t - \boldsymbol{\Delta\phi}_t\right)\left(\boldsymbol{\Delta\Phi}_{t-1}\boldsymbol{g}_t - \boldsymbol{\Delta\phi}_t\right)^\top \\
&= \mathbf{P}_{t-1} - \frac{1}{s_t}\mathbf{p}_t\mathbf{p}_t^\top.
\end{aligned}$$

We are not done quite yet, since d_t, $\mathbf{p}_t$ and s_t are expressed in terms of several t (or $t-1$) dimensional vectors and matrices, making their computation inefficient. Next, we derive efficient recursive update rules for d_t, $\mathbf{p}_t$ and s_t, that involve, at most, n-dimensional entities. Let us begin with d_t:

$$\begin{aligned}
d_t &= r_{t-1} - \boldsymbol{g}_t^\top \mathbf{r}_{t-2} \\
&= r_{t-1} - \left(\boldsymbol{\Delta\Phi}_{t-1}^\top\boldsymbol{\Delta\phi}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right)^\top \mathbf{Q}_{t-1}\mathbf{r}_{t-2} \\
&= r_{t-1} - \boldsymbol{\Delta\phi}_t^\top\hat{\mathbf{w}}_{t-1} + \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\left[-\boldsymbol{g}_{t-1}^\top, \quad 1\right] \begin{pmatrix} \mathbf{r}_{t-3} \\ r_{t-2} \end{pmatrix} \\
&= \frac{\gamma\sigma_{t-1}^2}{s_{t-1}} d_{t-1} + r_{t-1} - \left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)\right)^\top \hat{\mathbf{w}}_{t-1}.
\end{aligned}$$

Next, we treat $\mathbf{p}_t$:

$$\begin{aligned}
\mathbf{p}_t &= \boldsymbol{\Delta\phi}_t - \boldsymbol{\Delta\Phi}_{t-1}\boldsymbol{g}_t \\
&= \boldsymbol{\Delta\phi}_t - \boldsymbol{\Delta\Phi}_{t-1}\mathbf{Q}_{t-1}\left(\boldsymbol{\Delta\Phi}_{t-1}^\top\boldsymbol{\Delta\phi}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right) \\
&= \mathbf{P}_{t-1}\boldsymbol{\Delta\phi}_t + \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\left[\boldsymbol{\Delta\Phi}_{t-2}, \quad \boldsymbol{\Delta\phi}_{t-1}\right]\begin{bmatrix}-\boldsymbol{g}_{t-1}\\ 1\end{bmatrix} \\
&= \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\mathbf{p}_{t-1} + \mathbf{P}_{t-1}\left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)\right).
\end{aligned}$$

Finally, s_t:

$$s_t = \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \boldsymbol{\Delta\phi}_t^\top\boldsymbol{\Delta\phi}_t - \left(\boldsymbol{\Delta\Phi}_{t-1}^\top\boldsymbol{\Delta\phi}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right)^\top\boldsymbol{g}_t$$

The last term on the r.h.s. is

$$\begin{aligned}
&\left(\boldsymbol{\Delta\Phi}_{t-1}^\top\boldsymbol{\Delta\phi}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right)^\top\boldsymbol{g}_t = \\
&\left(\boldsymbol{\Delta\Phi}_{t-1}^\top\boldsymbol{\Delta\phi}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right)^\top\mathbf{Q}_{t-1}\left(\boldsymbol{\Delta\Phi}_{t-1}^\top\boldsymbol{\Delta\phi}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right) = \\
&\boldsymbol{\Delta\phi}_t^\top\left(\mathbf{I} - \mathbf{P}_{t-1}\right)\boldsymbol{\Delta\phi}_t - 2\gamma\sigma_{t-1}^2\mathbf{u}^\top\mathbf{Q}_{t-1}\boldsymbol{\Delta\Phi}_{t-1}^\top\boldsymbol{\Delta\phi}_t + \gamma^2\sigma_{t-1}^4\mathbf{u}^\top\mathbf{Q}_{t-1}\mathbf{u} = \\
&\boldsymbol{\Delta\phi}_t^\top\left(\mathbf{I} - \mathbf{P}_{t-1}\right)\boldsymbol{\Delta\phi}_t - \frac{2\gamma\sigma_{t-1}^2}{s_{t-1}}\left[-\boldsymbol{g}_{t-1}^\top, \quad 1\right]\boldsymbol{\Delta\Phi}_{t-1}^\top\boldsymbol{\Delta\phi}_t + \gamma^2\sigma_{t-1}^4\mathbf{u}^\top\mathbf{Q}_{t-1}\mathbf{u} = \\
&\boldsymbol{\Delta\phi}_t^\top\left(\mathbf{I} - \mathbf{P}_{t-1}\right)\boldsymbol{\Delta\phi}_t - \frac{2\gamma\sigma_{t-1}^2}{s_{t-1}}\left(\boldsymbol{\Delta\phi}_{t-1} - \boldsymbol{\Delta\Phi}_{t-2}\boldsymbol{g}_{t-1}\right)^\top\boldsymbol{\Delta\phi}_t + \gamma^2\sigma_{t-1}^4\mathbf{u}^\top\mathbf{Q}_{t-1}\mathbf{u} = \\
&\boldsymbol{\Delta\phi}_t^\top\left(\mathbf{I} - \mathbf{P}_{t-1}\right)\boldsymbol{\Delta\phi}_t - \frac{2\gamma\sigma_{t-1}^2}{s_{t-1}}\mathbf{p}_{t-1}^\top\boldsymbol{\Delta\phi}_t + \frac{\gamma^2\sigma_{t-1}^4}{s_{t-1}}
\end{aligned}$$

Assuming that we already computed $\mathbf{p}_t$, we may substitute

$$\mathbf{P}_{t-1}\boldsymbol{\Delta\phi}_t = \mathbf{p}_t - \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\mathbf{p}_{t-1}^\top,$$

Resulting in our final expression for s_t:

$$\begin{aligned}
s_t &= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \boldsymbol{\Delta\phi}_t^\top\mathbf{P}_{t-1}\boldsymbol{\Delta\phi}_t + \frac{2\gamma\sigma_{t-1}^2}{s_{t-1}}\mathbf{p}_{t-1}^\top\boldsymbol{\Delta\phi}_t - \frac{\gamma^2\sigma_{t-1}^4}{s_{t-1}} \\
&= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \left(\mathbf{p}_t + \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\mathbf{p}_{t-1}\right)^\top\left(\boldsymbol{\phi}(\mathbf{x}_{t-1}) - \gamma\boldsymbol{\phi}(\mathbf{x}_t)\right) - \frac{\gamma^2\sigma_{t-1}^4}{s_{t-1}}.
\end{aligned}$$

Throughout the derivations above we repeatedly made use of $\mathbf{I}-\mathbf{P}_{t-1} = \boldsymbol{\Phi}_{t-1}\mathbf{H}_{t-1}^\top\mathbf{Q}_{t-1}\mathbf{H}_{t-1}\boldsymbol{\Phi}_{t-1}^\top$, $\mathbf{Q}_{t-1}\mathbf{u} = [-\boldsymbol{g}_{t-1}^\top, 1]^\top$, and the recursive expressions for $\mathbf{H}_{t-1}$, $\mathbf{Q}_{t-1}$, $\boldsymbol{\Phi}_{t-1}$ and $\mathbf{r}_{t-1}$.

A.2.2 Nonparametric Monte-Carlo GPTD

Recall the expressions for the nonparametric posterior mean and variance (Eq. 4.3.19):

$$\hat{V}_t(\mathbf{x}) = \mathbf{k}_t(\mathbf{x})^\top \boldsymbol{\alpha}_t, \quad \text{and } P_t(\mathbf{x},\mathbf{x}') = k(\mathbf{x},\mathbf{x}') - \mathbf{k}_t(\mathbf{x})^\top \mathbf{C}_t \mathbf{k}_t(\mathbf{x}'),$$

respectively, with $\boldsymbol{\alpha}_t = \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{r}_{t-1}$, $\mathbf{C}_t = \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{H}_t$ and $\mathbf{Q}_t = \left(\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \boldsymbol{\Sigma}_t\right)^{-1}$. Let us write recursive expressions for $\mathbf{H}_t$, $\boldsymbol{\Sigma}_t$ and $\mathbf{Q}_t^{-1}$. Define $\mathbf{u} = (0,\ldots,0,1)^\top$ and $\mathbf{h}_t = (\mathbf{u}^\top, -\gamma)^\top = (0,\ldots,1,-\gamma)^\top$, then,

$$\mathbf{H}_t = \left[\begin{array}{c} \mathbf{H}_{t-1},\ \mathbf{0} \\ \hline \mathbf{h}_t^\top \end{array}\right], \quad \mathbf{K}_t = \begin{bmatrix} \mathbf{K}_{t-1} & \mathbf{k}_{t-1}(\mathbf{x}_t) \\ \mathbf{k}_{t-1}(\mathbf{x}_t)^\top & k(\mathbf{x}_t,\mathbf{x}_t) \end{bmatrix},$$
$$\boldsymbol{\Sigma}_t = \begin{bmatrix} \boldsymbol{\Sigma}_{t-1} & -\gamma\sigma_{t-1}^2 \mathbf{u} \\ -\gamma\sigma_{t-1}^2 \mathbf{u}^\top & \sigma_{t-1}^2 + \gamma^2\sigma_t^2 \end{bmatrix}.$$

Substituting these into the expression for $\mathbf{Q}_t^{-1}$ we get,

$$\mathbf{Q}_t^{-1} = \mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \boldsymbol{\Sigma}_t = \begin{bmatrix} \mathbf{Q}_{t-1}^{-1} & \mathbf{H}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t - \gamma\sigma_{t-1}^2\mathbf{u} \\ (\mathbf{H}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t - \gamma\sigma_{t-1}^2\mathbf{u})^\top & \Delta k_{tt} + \sigma_{t-1}^2 + \gamma^2\sigma_t^2 \end{bmatrix},$$

where we made use of the definitions

$$\boldsymbol{\Delta}\mathbf{k}_t \overset{\text{def}}{=} \mathbf{k}_{t-1}(\mathbf{x}_{t-1}) - \gamma\mathbf{k}_{t-1}(\mathbf{x}_t),$$
$$\Delta k_{tt} \overset{\text{def}}{=} k(\mathbf{x}_{t-1},\mathbf{x}_{t-1}) - 2\gamma k(\mathbf{x}_{t-1},\mathbf{x}_t) + \gamma^2 k(\mathbf{x}_t,\mathbf{x}_t).$$

$\mathbf{Q}_t$ may now be obtained using the partitioned matrix inversion formula (see Appendix D.4):

$$\mathbf{Q}_t = \frac{1}{s_t}\begin{bmatrix} s_t\mathbf{Q}_{t-1} + \boldsymbol{g}_t\boldsymbol{g}_t^\top & -\boldsymbol{g}_t \\ -\boldsymbol{g}_t^\top & 1 \end{bmatrix},$$

where $\boldsymbol{g}_t = \mathbf{Q}_{t-1}\left(\mathbf{H}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right)$, and $s_t = \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \Delta k_{tt} - \boldsymbol{g}_t^\top\left(\mathbf{H}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right)$. Let us define

$$\mathbf{c}_t = \mathbf{h}_t - \begin{pmatrix} \mathbf{H}_{t-1}^\top \boldsymbol{g}_t \\ 0 \end{pmatrix}, \quad \text{and } d_t = r_{t-1} - \boldsymbol{g}_t^\top \mathbf{r}_{t-2}.$$

We are now ready to derive the recursions for $\mathbf{C}_t$ and $\boldsymbol{\alpha}_t$:

$$\begin{aligned}
\mathbf{C}_t &= \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{H}_t \\
&= \frac{1}{s_t}\left[\begin{array}{c|c} \mathbf{H}_{t-1}^\top & \mathbf{h}_t \\ \mathbf{0}^\top & \end{array}\right] \begin{bmatrix} s_t\mathbf{Q}_{t-1} + \boldsymbol{g}_t\boldsymbol{g}_t^\top & -\boldsymbol{g}_t \\ -\boldsymbol{g}_t^\top & 1 \end{bmatrix} \left[\begin{array}{cc} \mathbf{H}_{t-1} & \mathbf{0} \\ \hline \multicolumn{2}{c}{\mathbf{h}_t^\top} \end{array}\right] \\
&= \begin{bmatrix} \mathbf{H}_{t-1}\mathbf{Q}_{t-1}\mathbf{H}_{t-1}^\top & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t}\left(\begin{pmatrix} \mathbf{H}_{t-1}^\top \boldsymbol{g}_t \\ 0 \end{pmatrix} - \mathbf{h}_t\right)\left(\begin{pmatrix} \mathbf{H}_{t-1}^\top \boldsymbol{g}_t \\ 0 \end{pmatrix} - \mathbf{h}_t\right)^\top \\
&= \begin{bmatrix} \mathbf{C}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t}\mathbf{c}_t\mathbf{c}_t^\top .
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{\alpha}_t &= \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{r}_{t-1} \\
&= \frac{1}{s_t}\left[\begin{array}{c|c} \mathbf{H}_{t-1}^\top & \mathbf{h}_t \\ \mathbf{0}^\top & \end{array}\right] \begin{bmatrix} s_t\mathbf{Q}_{t-1} + \boldsymbol{g}_t\boldsymbol{g}_t^\top & -\boldsymbol{g}_t \\ -\boldsymbol{g}_t^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{r}_{t-2} \\ r_{t-1} \end{bmatrix} \\
&= \begin{pmatrix} \boldsymbol{\alpha}_{t-1} \\ 0 \end{pmatrix} + \frac{1}{s_t}\left(\begin{pmatrix} \mathbf{H}_{t-1}^\top \boldsymbol{g}_t \\ 0 \end{pmatrix} - \mathbf{h}_t\right)\left(\boldsymbol{g}_t^\top \mathbf{r}_{t-2} - r_{t-1}\right) \\
&= \begin{pmatrix} \boldsymbol{\alpha}_{t-1} \\ 0 \end{pmatrix} + \frac{\mathbf{c}_t}{s_t} d_t .
\end{aligned}$$

We are not done quite yet, since d_t, $\mathbf{c}_t$ and s_t are not explicitly given in terms of current-time-step quantities, as in the case of deterministic transitions. We show next that, similarly to the parametric updates derived above, d_t, $\mathbf{c}_t$ and s_t may be recursively updated without requiring any additional bookkeeping, apart of maintaining the sequence of states seen so far. Let us begin with d_t:

$$\begin{aligned}
d_t &= r_{t-1} - \boldsymbol{g}_t^\top \mathbf{r}_{t-2} \\
&= r_{t-1} - \left(\mathbf{H}_{t-1}\Delta\mathbf{k}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right)^\top \mathbf{Q}_{t-1}\mathbf{r}_{t-2} \\
&= r_{t-1} - \Delta\mathbf{k}_t^\top \boldsymbol{\alpha}_{t-1} - \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\left(\boldsymbol{g}_{t-1}^\top \mathbf{r}_{t-3} - r_{t-2}\right) \\
&= \frac{\gamma\sigma_{t-1}^2}{s_{t-1}} d_{t-1} + r_{t-1} - \Delta\mathbf{k}_t^\top \boldsymbol{\alpha}_{t-1} .
\end{aligned}$$

Turning to $\mathbf{c}_t$, first notice that

$$\begin{aligned}
\mathbf{H}_t^\top \mathbf{Q}_t \mathbf{u} &= \frac{1}{s_t}\left[\begin{array}{c|c} \mathbf{H}_{t-1}^\top & \mathbf{h}_t \\ \mathbf{0}^\top & \end{array}\right]\begin{bmatrix} s_t\mathbf{Q}_{t-1} + \boldsymbol{g}_t\boldsymbol{g}_t^\top & -\boldsymbol{g}_t \\ -\boldsymbol{g}_t^\top & 1 \end{bmatrix}\mathbf{u} \\
&= \frac{1}{s_t}\left[\begin{array}{c|c} \mathbf{H}_{t-1}^\top & \mathbf{h}_t \\ \mathbf{0}^\top & \end{array}\right]\begin{pmatrix} -\boldsymbol{g}_t \\ 1 \end{pmatrix} \\
&= \frac{1}{s_t}\left(\mathbf{h}_t - \begin{pmatrix} \mathbf{H}_{t-1}^\top \boldsymbol{g}_t \\ 0 \end{pmatrix}\right) \\
&= \frac{\mathbf{c}_t}{s_t}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathbf{c}_t &= \mathbf{h}_t - \begin{pmatrix} \mathbf{H}_{t-1}^\top \boldsymbol{g}_t \\ 0 \end{pmatrix} \\
&= \mathbf{h}_t - \begin{pmatrix} \mathbf{H}_{t-1}^\top \mathbf{Q}_{t-1}\left(\mathbf{H}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right) \\ 0 \end{pmatrix} \\
&= \mathbf{h}_t - \begin{pmatrix} \mathbf{C}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t - \gamma\sigma_{t-1}^2\mathbf{H}_{t-1}^\top\mathbf{Q}_{t-1}\mathbf{u} \\ 0 \end{pmatrix} \\
&= \mathbf{h}_t - \begin{pmatrix} \mathbf{C}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t - \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\mathbf{c}_{t-1} \\ 0 \end{pmatrix} \\
&= \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\begin{pmatrix} \mathbf{c}_{t-1} \\ 0 \end{pmatrix} + \mathbf{h}_t - \begin{pmatrix} \mathbf{C}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t \\ 0 \end{pmatrix}.
\end{aligned}$$

Finally, s_t:

$$\begin{aligned}
s_t &= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \Delta k_{tt} - \boldsymbol{g}_t^\top\left(\mathbf{H}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right) \\
&= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \Delta k_{tt} - \left(\mathbf{H}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right)^\top\mathbf{Q}_{t-1}\left(\mathbf{H}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right) \\
&= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \Delta k_{tt} - \boldsymbol{\Delta}\mathbf{k}_t^\top\mathbf{C}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t + 2\gamma\sigma_{t-1}^2\mathbf{u}^\top\mathbf{Q}_{t-1}\mathbf{H}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t - \frac{\gamma^2\sigma_{t-1}^4}{s_{t-1}} \\
&= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \Delta k_{tt} - \boldsymbol{\Delta}\mathbf{k}_t^\top\mathbf{C}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t + \frac{2\gamma\sigma_{t-1}^2}{s_{t-1}}\mathbf{c}_{t-1}^\top\boldsymbol{\Delta}\mathbf{k}_t - \frac{\gamma^2\sigma_{t-1}^4}{s_{t-1}}.
\end{aligned}$$

A.2.3 Sparse Nonparametric Monte-Carlo GPTD

At each time step the current sampled state $\mathbf{x}_t$ may either be left out of the dictionary, in which case $\mathcal{D}_t = \mathcal{D}_{t-1}$, or added to it, in which case $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$, as determined by the sparsification criterion. Let us begin with the updates for the first case.

Case 1: $\mathcal{D}_t = \mathcal{D}_{t-1}$

Since the dictionary remains unchanged, $\tilde{\mathbf{K}}_t = \tilde{\mathbf{K}}_{t-1}$. Defining $\tilde{\mathbf{h}}_t = \boldsymbol{a}_{t-1} - \gamma \boldsymbol{a}_t$ and $\mathbf{u} = [0, \ldots, 0, 1]^\top$, we have

$$\mathbf{A}_t = \begin{bmatrix} \mathbf{A}_{t-1} \\ \boldsymbol{a}_t^\top \end{bmatrix}, \; \tilde{\mathbf{H}}_t = \begin{bmatrix} \tilde{\mathbf{H}}_{t-1} \\ \tilde{\mathbf{h}}_t^\top \end{bmatrix}, \; \boldsymbol{\Sigma}_t = \begin{bmatrix} \boldsymbol{\Sigma}_{t-1} & -\gamma\sigma_{t-1}^2 \mathbf{u} \\ -\gamma\sigma_{t-1}^2 \mathbf{u}^\top & \sigma_{t-1}^2 + \gamma^2\sigma_t^2 \end{bmatrix}.$$

Consequently, we may obtain a recursive formula for $\tilde{\mathbf{Q}}_t^{-1}$ (see Eq. 4.3.27):

$$\begin{aligned} \tilde{\mathbf{Q}}_t^{-1} &= \begin{bmatrix} \tilde{\mathbf{H}}_{t-1} \\ \tilde{\mathbf{h}}_t^\top \end{bmatrix} \tilde{\mathbf{K}}_{t-1} \begin{bmatrix} \tilde{\mathbf{H}}_{t-1}, & \tilde{\mathbf{h}}_t \end{bmatrix} + \begin{bmatrix} \boldsymbol{\Sigma}_{t-1} & -\gamma\sigma_{t-1}^2 \mathbf{u} \\ -\gamma\sigma_{t-1}^2 \mathbf{u}^\top & \sigma_{t-1}^2 + \gamma^2\sigma_t^2 \end{bmatrix} \\ &= \begin{bmatrix} \tilde{\mathbf{Q}}_{t-1}^{-1} & \tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma_{t-1}^2\mathbf{u} \\ (\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma_{t-1}^2\mathbf{u})^\top & , \Delta k_{tt} + \sigma_{t-1}^2 + \gamma^2\sigma_t^2 \end{bmatrix}, \end{aligned}$$

where in the last equality we used the definition $\boldsymbol{\Delta}\tilde{\mathbf{k}}_t \overset{\text{def}}{=} \tilde{\mathbf{K}}_{t-1}\tilde{\mathbf{h}}_t = \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_{t-1}) - \gamma\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$, and $\Delta k_{tt} = \tilde{\mathbf{h}}_t^\top \boldsymbol{\Delta}\tilde{\mathbf{k}}_t$. We may now invert $\tilde{\mathbf{Q}}_t^{-1}$ using the partitioned matrix inversion formula:

$$\tilde{\mathbf{Q}}_t = \frac{1}{s_t} \begin{bmatrix} s_t\tilde{\mathbf{Q}}_{t-1} + \tilde{\boldsymbol{g}}_t\tilde{\boldsymbol{g}}_t^\top & -\tilde{\boldsymbol{g}}_t \\ -\tilde{\boldsymbol{g}}_t^\top & 1 \end{bmatrix},$$

where $\tilde{\boldsymbol{g}}_t = \tilde{\mathbf{Q}}_{t-1}\left(\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma^2\mathbf{u}\right)$, and $s_t = (1+\gamma^2)\sigma^2 + \Delta k_{tt} - \tilde{\boldsymbol{g}}_t^\top\left(\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma^2\mathbf{u}\right)$. Let us define $\tilde{\mathbf{c}}_t = \left(\tilde{\mathbf{h}}_t - \tilde{\mathbf{H}}_{t-1}\tilde{\boldsymbol{g}}_t\right)$. The update of the covariance parameter matrix $\tilde{\mathbf{C}}_t$ is:

$$\begin{aligned} \tilde{\mathbf{C}}_t &= \tilde{\mathbf{H}}_t^\top \tilde{\mathbf{Q}} \tilde{\mathbf{H}}_t \\ &= \frac{1}{s_t} \begin{bmatrix} \tilde{\mathbf{H}}_{t-1}^\top, & \tilde{\mathbf{h}}_t \end{bmatrix} \begin{bmatrix} s_t\tilde{\mathbf{Q}}_{t-1} + \tilde{\boldsymbol{g}}_t\tilde{\boldsymbol{g}}_t^\top & -\tilde{\boldsymbol{g}}_t \\ -\tilde{\boldsymbol{g}}_t^\top & 1 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{H}}_{t-1} \\ \tilde{\mathbf{h}}_t^\top \end{bmatrix} \\ &= \tilde{\mathbf{H}}_{t-1}^\top \tilde{\mathbf{Q}}_{t-1} \tilde{\mathbf{H}}_{t-1} + \frac{1}{s_t} \begin{bmatrix} \tilde{\mathbf{H}}_{t-1}^\top & \tilde{\mathbf{h}}_t \end{bmatrix} \begin{pmatrix} \tilde{\boldsymbol{g}}_t \\ -1 \end{pmatrix} \begin{pmatrix} \tilde{\boldsymbol{g}}_t^\top & -1 \end{pmatrix} \begin{bmatrix} \tilde{\mathbf{H}}_{t-1} \\ \tilde{\mathbf{h}}_t^\top \end{bmatrix} \\ &= \tilde{\mathbf{C}}_{t-1} + \frac{1}{s_t} \left(\tilde{\mathbf{H}}_{t-1}\tilde{\boldsymbol{g}}_t - \tilde{\mathbf{h}}_t\right) \left(\tilde{\mathbf{H}}_{t-1}\tilde{\boldsymbol{g}}_t - \tilde{\mathbf{h}}_t\right)^\top \\ &= \tilde{\mathbf{C}}_{t-1} + \frac{1}{s_t}\tilde{\mathbf{c}}_t\tilde{\mathbf{c}}_t^\top. \end{aligned}$$

Next, we compute $\tilde{\boldsymbol{\alpha}}_t$:

$$\begin{aligned}
\tilde{\boldsymbol{\alpha}}_t &= \tilde{\mathbf{H}}_t^\top \tilde{\mathbf{Q}}_t \mathbf{r}_{t-1} \\
&= \frac{1}{s_t}\left[\tilde{\mathbf{H}}_{t-1}^\top, \tilde{\mathbf{h}}_t\right] \begin{bmatrix} s_t\tilde{\mathbf{Q}}_{t-1} + \tilde{\boldsymbol{g}}_t\tilde{\boldsymbol{g}}_t^\top & , -\tilde{\boldsymbol{g}}_t \\ -\tilde{\boldsymbol{g}}_t^\top & , 1 \end{bmatrix} \begin{pmatrix} \mathbf{r}_{t-2} \\ r_{t-1} \end{pmatrix} \\
&= \tilde{\boldsymbol{\alpha}}_{t-1} + \frac{1}{s_t}(\tilde{\mathbf{H}}_{t-1}^\top \tilde{\boldsymbol{g}}_t - \tilde{\mathbf{h}}_t)(\tilde{\boldsymbol{g}}_t^\top \mathbf{r}_{t-2} - r_{t-1}) \\
&= \tilde{\boldsymbol{\alpha}}_{t-1} + \frac{\tilde{\mathbf{c}}_t}{s_t} d_t,
\end{aligned}$$

where we have defined

$$d_t = r_{t-1} - \tilde{\boldsymbol{g}}_t^\top \mathbf{r}_{t-2}.$$

Note that $\tilde{\boldsymbol{g}}_t$ is a $(t-1)\times 1$ vector; we would therefore like to eliminate it from our updates, since we aim at keeping the memory and time requirements of each update independent of t. Due to the special form of the matrix $\boldsymbol{\Sigma}_t$ we can derive simple recursive formulas for $\tilde{\mathbf{c}}_t$, d_t and s_t, thus eliminating $\tilde{\boldsymbol{g}}_t$ from the updates. Let us begin with $\tilde{\mathbf{c}}_t$:

$$\begin{aligned}
\tilde{\mathbf{c}}_t &= \tilde{\mathbf{h}}_t - \tilde{\mathbf{H}}_{t-1}^\top \tilde{\boldsymbol{g}}_t \\
&= \tilde{\mathbf{h}}_t - \tilde{\mathbf{H}}_{t-1}^\top \tilde{\mathbf{Q}}_{t-1}\left(\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right) \\
&= \tilde{\mathbf{h}}_t - \tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t + \gamma\sigma_{t-1}^2 \tilde{\mathbf{H}}_{t-1}\tilde{\mathbf{Q}}_{t-1}\mathbf{u} \\
&= \tilde{\mathbf{h}}_t - \tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t + \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\left(\tilde{\mathbf{h}}_{t-1} - \tilde{\mathbf{H}}_{t-2}\tilde{\boldsymbol{g}}_{t-1}\right) \\
&= \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\tilde{\mathbf{c}}_{t-1} + \tilde{\mathbf{h}}_t - \tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t.
\end{aligned}$$

Turning to d_t:

$$\begin{aligned}
d_t &= r_{t-1} - \tilde{\boldsymbol{g}}_t^\top \mathbf{r}_{t-2} \\
&= r_{t-1} - \left(\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right)^\top \tilde{\mathbf{Q}}_{t-1}\mathbf{r}_{t-2} \\
&= r_{t-1} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top \tilde{\boldsymbol{\alpha}}_{t-1} - \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\left(\tilde{\boldsymbol{g}}_{t-1}^\top \mathbf{r}_{t-3} - r_{t-2}\right) \\
&= \frac{\gamma\sigma_{t-1}^2}{s_{t-1}} d_{t-1} + r_{t-1} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top \tilde{\boldsymbol{\alpha}}_{t-1}
\end{aligned}$$

Finally, s_t:

$$\begin{aligned}
s_t &= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \Delta k_{tt} - \tilde{\boldsymbol{g}}_t^\top \left(\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right) \\
&= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \Delta k_{tt} - \left(\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right)^\top \tilde{\mathbf{Q}}_{t-1}\left(\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right) \\
&= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \Delta k_{tt} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top\tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t + 2\gamma\sigma_{t-1}^2\mathbf{u}^\top\tilde{\mathbf{Q}}_{t-1}\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \frac{\gamma^2\sigma_{t-1}^4}{s_{t-1}} \\
&= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \tilde{\mathbf{h}}_t^\top\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top\tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t + \frac{2\gamma\sigma_{t-1}^2}{s_{t-1}}\tilde{\mathbf{c}}_{t-1}^\top\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \frac{\gamma^2\sigma_{t-1}^4}{s_{t-1}}
\end{aligned}$$

Recall that $\tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t$ was computed for the $\tilde{\mathbf{c}}_t$ update. Given $\tilde{\mathbf{c}}_t$ we may substitute $\tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t = \frac{\gamma\sigma^2}{s_{t-1}}\tilde{\mathbf{c}}_{t-1} - \tilde{\mathbf{c}}_t + \tilde{\mathbf{h}}_t$, resulting in

$$\begin{aligned}
s_t &= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \tilde{\mathbf{h}}_t^\top\boldsymbol{\Delta}\tilde{\mathbf{k}}_t \\
&\quad - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top\left(\frac{\gamma\sigma^2}{s_{t-1}}\tilde{\mathbf{c}}_{t-1} - \tilde{\mathbf{c}}_t + \tilde{\mathbf{h}}_t\right) + \frac{2\gamma\sigma^2}{s_{t-1}}\tilde{\mathbf{c}}_{t-1}^\top\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \frac{\gamma^2\sigma_{t-1}^4}{s_{t-1}} \\
&= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top\left(\tilde{\mathbf{c}}_t + \frac{\gamma\sigma^2}{s_{t-1}}\tilde{\mathbf{c}}_{t-1}\right) - \frac{\gamma^2\sigma_{t-1}^4}{s_{t-1}}.
\end{aligned}$$

Case 2: $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$

Here, $\tilde{\mathbf{K}}_t$ is given by Eq. 2.2.10, which we repeat here for clarity:

$$\tilde{\mathbf{K}}_t = \begin{bmatrix} \tilde{\mathbf{K}}_{t-1} & \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) \\ \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top & k(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix}$$

Furthermore, $\boldsymbol{a}_t = (0, \ldots, 1)^\top$ since the last member added to the dictionary is $\mathbf{x}_t$ itself (and $\boldsymbol{\phi}(\mathbf{x}_t)$ is exactly representable by itself). Therefore

$$\mathbf{A}_t = \begin{bmatrix} \mathbf{A}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix}, \quad \tilde{\mathbf{H}}_t = \begin{bmatrix} \tilde{\mathbf{H}}_{t-1} & \mathbf{0} \\ \boldsymbol{a}_{t-1}^\top & -\gamma \end{bmatrix} = \left[\begin{array}{c} \tilde{\mathbf{H}}_{t-1} \quad \mathbf{0} \\ \hline \tilde{\mathbf{h}}_t^\top \end{array}\right],$$

where we defined

$$\tilde{\mathbf{h}}_t = \begin{pmatrix} \boldsymbol{a}_{t-1} \\ 0 \end{pmatrix} - \gamma\boldsymbol{a}_t = \begin{pmatrix} \boldsymbol{a}_{t-1} \\ -\gamma \end{pmatrix}.$$

The recursion for $\tilde{\mathbf{Q}}_t^{-1}$ is given by:

$$\begin{aligned}
\tilde{\mathbf{Q}}_t^{-1} &= \left[\begin{array}{c} \tilde{\mathbf{H}}_{t-1} \quad \mathbf{0} \\ \hline \tilde{\mathbf{h}}_t^\top \end{array}\right] \begin{bmatrix} \tilde{\mathbf{K}}_{t-1} & \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) \\ \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top & k(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix} \left[\begin{array}{c|c} \tilde{\mathbf{H}}_{t-1}^\top & \tilde{\mathbf{h}}_t \\ \mathbf{0}^\top & \end{array}\right] + \begin{bmatrix} \boldsymbol{\Sigma}_{t-1} & -\gamma\sigma_{t-1}^2\mathbf{u} \\ -\gamma\sigma_{t-1}^2\mathbf{u}^\top & \sigma_{t-1}^2 + \gamma^2\sigma_t^2 \end{bmatrix} \\
&= \begin{bmatrix} \tilde{\mathbf{Q}}_{t-1}^{-1} & \tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t \\ (\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t)^\top & \Delta k_{tt} + \sigma_{t-1}^2 + \gamma^2\sigma_t^2 \end{bmatrix},
\end{aligned}$$

where we defined

$$\begin{aligned}
\mathbf{\Delta}\tilde{\mathbf{k}}_t &\stackrel{\text{def}}{=} \tilde{\mathbf{K}}_{t-1}\boldsymbol{a}_{t-1} - \gamma\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) = \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_{t-1}) - \gamma\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t),\\
\Delta k_{tt} &\stackrel{\text{def}}{=} \boldsymbol{a}_{t-1}^\top\tilde{\mathbf{K}}_{t-1}\boldsymbol{a}_{t-1} - 2\gamma\boldsymbol{a}_{t-1}^\top\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) + \gamma^2 k(\mathbf{x}_t,\mathbf{x}_t)\\
&= \boldsymbol{a}_{t-1}^\top\left(\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_{t-1}) - 2\gamma\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)\right) + \gamma^2 k(\mathbf{x}_t,\mathbf{x}_t).
\end{aligned}$$

Defining, as before, $\tilde{\boldsymbol{g}}_t = \tilde{\mathbf{Q}}_{t-1}\left(\tilde{\mathbf{H}}_{t-1}\mathbf{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma^2\mathbf{u}\right)$ and $s_t = \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \Delta k_{tt} - \tilde{\boldsymbol{g}}_t^\top\left(\tilde{\mathbf{H}}_{t-1}\mathbf{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma^2\mathbf{u}\right)$, and using again the partitioned matrix inversion formula we get

$$\tilde{\mathbf{Q}}_t = \frac{1}{s_t}\begin{bmatrix} s_t\tilde{\mathbf{Q}}_{t-1} + \tilde{\boldsymbol{g}}_t\tilde{\boldsymbol{g}}_t^\top & -\tilde{\boldsymbol{g}}_t \\ -\tilde{\boldsymbol{g}}_t^\top & 1 \end{bmatrix}.$$

Let us define

$$\tilde{\mathbf{c}}_t = \tilde{\mathbf{h}}_t - \begin{pmatrix}\tilde{\mathbf{H}}_{t-1}^\top\tilde{\boldsymbol{g}}_t \\ 0\end{pmatrix}, \quad \text{and } d_t = r_{t-1} - \tilde{\boldsymbol{g}}_t^\top\mathbf{r}_{t-2}.$$

We are now ready to compute $\tilde{\mathbf{C}}_t$ and $\tilde{\boldsymbol{\alpha}}_t$:

$$\begin{aligned}
\tilde{\mathbf{C}}_t &= \tilde{\mathbf{H}}_t^\top\tilde{\mathbf{Q}}_t\tilde{\mathbf{H}}_t\\
&= \frac{1}{s_t}\left[\begin{array}{c|c} \tilde{\mathbf{H}}_{t-1}^\top & \tilde{\mathbf{h}}_t \\ \mathbf{0}^\top & \end{array}\right]\begin{bmatrix} s_t\tilde{\mathbf{Q}}_{t-1} + \tilde{\boldsymbol{g}}_t\tilde{\boldsymbol{g}}_t^\top & -\tilde{\boldsymbol{g}}_t \\ -\tilde{\boldsymbol{g}}_t^\top & 1 \end{bmatrix}\left[\begin{array}{cc} \tilde{\mathbf{H}}_{t-1} & \mathbf{0} \\ \hline \multicolumn{2}{c}{\tilde{\mathbf{h}}_t^\top} \end{array}\right]\\
&= \begin{bmatrix} \tilde{\mathbf{H}}_{t-1}\tilde{\mathbf{Q}}_{t-1}\tilde{\mathbf{H}}_{t-1}^\top & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t}\left(\begin{pmatrix}\tilde{\mathbf{H}}_{t-1}^\top\tilde{\boldsymbol{g}}_t \\ 0\end{pmatrix} - \tilde{\mathbf{h}}_t\right)\left(\begin{pmatrix}\tilde{\mathbf{H}}_{t-1}^\top\tilde{\boldsymbol{g}}_t \\ 0\end{pmatrix} - \tilde{\mathbf{h}}_t\right)^\top\\
&= \begin{bmatrix} \tilde{\mathbf{C}}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t}\tilde{\mathbf{c}}_t\tilde{\mathbf{c}}_t^\top.
\end{aligned}$$

$$\begin{aligned}
\tilde{\boldsymbol{\alpha}}_t &= \tilde{\mathbf{H}}_t^\top\tilde{\mathbf{Q}}_t\mathbf{r}_{t-1}\\
&= \frac{1}{s_t}\left[\begin{array}{c|c} \tilde{\mathbf{H}}_{t-1}^\top & \tilde{\mathbf{h}}_t \\ \mathbf{0}^\top & \end{array}\right]\begin{bmatrix} s_t\tilde{\mathbf{Q}}_{t-1} + \tilde{\boldsymbol{g}}_t\tilde{\boldsymbol{g}}_t^\top & -\tilde{\boldsymbol{g}}_t \\ -\tilde{\boldsymbol{g}}_t^\top & 1 \end{bmatrix}\begin{bmatrix}\mathbf{r}_{t-2} \\ r_{t-1}\end{bmatrix}\\
&= \begin{pmatrix}\tilde{\boldsymbol{\alpha}}_{t-1} \\ 0\end{pmatrix} + \frac{1}{s_t}\left(\begin{pmatrix}\tilde{\mathbf{H}}_{t-1}^\top\tilde{\boldsymbol{g}}_t \\ 0\end{pmatrix} - \tilde{\mathbf{h}}_t\right)\left(\tilde{\boldsymbol{g}}_t^\top\mathbf{r}_{t-2} - r_{t-1}\right)\\
&= \begin{pmatrix}\tilde{\boldsymbol{\alpha}}_{t-1} \\ 0\end{pmatrix} + \frac{\tilde{\mathbf{c}}_t}{s_t}d_t.
\end{aligned}$$

As above, we still need to derive recursions for d_t, $\tilde{\mathbf{c}}_t$ and s_t. Let us begin with d_t:

$$\begin{aligned}
d_t &= r_{t-1} - \tilde{\boldsymbol{g}}_t^\top \mathbf{r}_{t-2} \\
&= r_{t-1} - \left(\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right)^\top \mathbf{Q}_{t-1}\mathbf{r}_{t-2} \\
&= r_{t-1} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top \tilde{\boldsymbol{\alpha}}_{t-1} - \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\left(\tilde{\boldsymbol{g}}_{t-1}^\top \mathbf{r}_{t-3} - r_{t-2}\right) \\
&= \frac{\gamma\sigma_{t-1}^2}{s_{t-1}} d_{t-1} + r_{t-1} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top \tilde{\boldsymbol{\alpha}}_{t-1}.
\end{aligned}$$

Turning to $\tilde{\mathbf{c}}_t$, first notice that

$$\begin{aligned}
\tilde{\mathbf{H}}_t^\top \mathbf{Q}_t \mathbf{u} &= \frac{1}{s_t}\left[\begin{array}{c|c} \tilde{\mathbf{H}}_{t-1}^\top & \tilde{\mathbf{h}}_t \\ \mathbf{0}^\top & \end{array}\right]\begin{bmatrix} s_t\mathbf{Q}_{t-1} + \tilde{\boldsymbol{g}}_t\tilde{\boldsymbol{g}}_t^\top & -\tilde{\boldsymbol{g}}_t \\ -\tilde{\boldsymbol{g}}_t^\top & 1 \end{bmatrix}\mathbf{u} \\
&= \frac{1}{s_t}\left[\begin{array}{c|c} \tilde{\mathbf{H}}_{t-1}^\top & \tilde{\mathbf{h}}_t \\ \mathbf{0}^\top & \end{array}\right]\begin{pmatrix} -\tilde{\boldsymbol{g}}_t \\ 1 \end{pmatrix} \\
&= \frac{1}{s_t}\left(\tilde{\mathbf{h}}_t - \begin{pmatrix} \tilde{\mathbf{H}}_{t-1}^\top \tilde{\boldsymbol{g}}_t \\ 0 \end{pmatrix}\right) \\
&= \frac{\tilde{\mathbf{c}}_t}{s_t}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\tilde{\mathbf{c}}_t &= \tilde{\mathbf{h}}_t - \begin{pmatrix} \tilde{\mathbf{H}}_{t-1}^\top \tilde{\boldsymbol{g}}_t \\ 0 \end{pmatrix} \\
&= \tilde{\mathbf{h}}_t - \begin{pmatrix} \tilde{\mathbf{H}}_{t-1}^\top \mathbf{Q}_{t-1}\left(\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right) \\ 0 \end{pmatrix} \\
&= \tilde{\mathbf{h}}_t - \begin{pmatrix} \tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma_{t-1}^2\tilde{\mathbf{H}}_{t-1}^\top \mathbf{Q}_{t-1}\mathbf{u} \\ 0 \end{pmatrix} \\
&= \tilde{\mathbf{h}}_t - \begin{pmatrix} \tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\tilde{\mathbf{c}}_{t-1} \\ 0 \end{pmatrix} \\
&= \frac{\gamma\sigma_{t-1}^2}{s_{t-1}}\begin{pmatrix} \tilde{\mathbf{c}}_{t-1} \\ 0 \end{pmatrix} + \tilde{\mathbf{h}}_t - \begin{pmatrix} \tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t \\ 0 \end{pmatrix}.
\end{aligned}$$

Finally, s_t:

$$\begin{aligned}
s_t &= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \Delta k_{tt} - \tilde{\boldsymbol{g}}_t^\top \left(\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right) \\
&= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \Delta k_{tt} - \left(\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right)^\top \mathbf{Q}_{t-1}\left(\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \gamma\sigma_{t-1}^2\mathbf{u}\right) \\
&= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \Delta k_{tt} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top\tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t + 2\gamma\sigma_{t-1}^2\mathbf{u}^\top\mathbf{Q}_{t-1}\tilde{\mathbf{H}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \frac{\gamma^2\sigma_{t-1}^4}{s_{t-1}} \\
&= \sigma_{t-1}^2 + \gamma^2\sigma_t^2 + \Delta k_{tt} - \boldsymbol{\Delta}\tilde{\mathbf{k}}_t^\top\tilde{\mathbf{C}}_{t-1}\boldsymbol{\Delta}\tilde{\mathbf{k}}_t + \frac{2\gamma\sigma_{t-1}^2}{s_{t-1}}\tilde{\mathbf{c}}_{t-1}^\top\boldsymbol{\Delta}\tilde{\mathbf{k}}_t - \frac{\gamma^2\sigma_{t-1}^4}{s_{t-1}}.
\end{aligned}$$

Appendix B

TD(1) as a ML Gradient Algorithm

In what follows, we show that TD(1) may be derived as a gradient ascent method for maximizing the log-likelihood of the parametric MC-GPTD model, with a fixed noise variance σ^2. This should not come as a surprise, as we have already shown that LSTD(1) is a ML variant of the MC-GPTD algorithm. We assume that we are in the episodic setting and that $\mathbf{x}_t$ is the last state in the current episode. $\mathbf{H}_{t+1}$ is therefore an invertible $(t+1) \times (t+1)$ matrix, given by Eq. 4.3.9. Recall the expression for the log-likelihood:

$$\log \Pr(R_t = \mathbf{r}_t | W) \propto -\left(\mathbf{H}_{t+1}\mathbf{\Phi}_t^\top W - \mathbf{r}_t\right)^\top \mathbf{\Sigma}_{t+1}^{-1} \left(\mathbf{H}_{t+1}\mathbf{\Phi}_t^\top W - \mathbf{r}_t\right).$$

Fixing an initial guess for W by setting $W = \hat{\mathbf{w}}_0$ (In the ML framework we no longer consider the weight vector to be random) and taking the gradient of this expression w.r.t. $\hat{\mathbf{w}}_0$, we can write down the gradient-based update rule, using $\mathbf{\Sigma}_{t+1} \propto \mathbf{H}_{t+1}\mathbf{H}_{t+1}^\top$, as follows.

$$\begin{aligned} \Delta\hat{\mathbf{w}} &= -\eta \mathbf{\Phi}_t \mathbf{H}_{t+1}^\top \left(\mathbf{H}_{t+1}\mathbf{H}_{t+1}^\top\right)^{-1} \left(\mathbf{H}_{t+1}\mathbf{\Phi}_t^\top \hat{\mathbf{w}}_0 - \mathbf{r}_t\right) \\ \hat{\mathbf{w}}_{t+1} &= \hat{\mathbf{w}}_0 + \Delta\hat{\mathbf{w}} \end{aligned} \tag{B.0.1}$$

This rule makes all of its updates based on the initial value estimates induced by $\hat{\mathbf{w}}_0$, waiting until the end of the learning episode before replacing $\hat{\mathbf{w}}_0$ with $\hat{\mathbf{w}}_{t+1}$.

The first part of the expression for $\Delta\hat{\mathbf{w}}$ is

$$\begin{aligned}
\boldsymbol{\Phi}_t\mathbf{H}_{t+1}^\top\left(\mathbf{H}_{t+1}\mathbf{H}_{t+1}^\top\right)^{-1} &= \boldsymbol{\Phi}_t\mathbf{H}_{t+1}^{-1} \\
&= \begin{bmatrix} \vdots & & \vdots \\ \boldsymbol{\phi}(\mathbf{x}_0)\,, & \dots & ,\boldsymbol{\phi}(\mathbf{x}_t) \\ \vdots & & \vdots \end{bmatrix}
\begin{bmatrix} 1 & \gamma & \gamma^2 & \dots & \gamma^t \\ 0 & 1 & \gamma & \dots & \gamma^{t-1} \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \\
&= \begin{bmatrix} \vdots & \vdots & & \vdots \\ \boldsymbol{\phi}(\mathbf{x}_0)\,, & \gamma\boldsymbol{\phi}(\mathbf{x}_0)+\boldsymbol{\phi}(\mathbf{x}_1)\,, & \dots & ,\sum_{i=0}^{t}\gamma^{t-i}\boldsymbol{\phi}(\mathbf{x}_i) \\ \vdots & \vdots & & \vdots \end{bmatrix} \\
&= \mathbf{Z}_{t+1}^{(1)}, \qquad \text{(B.0.2)}
\end{aligned}$$

where $\mathbf{Z}_{t+1}^{(1)}$ is the eligibility matrix for $\lambda = 1$, defined in Eq. (4.5.57). The rightmost parenthesized term in the update is simply a vector of temporal differences:

$$\mathbf{H}_{t+1}\boldsymbol{\Phi}_t^\top\hat{\mathbf{w}}_0 - \mathbf{r}_t = \begin{pmatrix} \hat{V}_0(\mathbf{x}_0) - \gamma\hat{V}_0(\mathbf{x}_1) - r_0 \\ \hat{V}_0(\mathbf{x}_1) - \gamma\hat{V}_0(\mathbf{x}_2) - r_1 \\ \vdots \\ \hat{V}_0(\mathbf{x}_{t-1}) - \gamma\hat{V}_0(\mathbf{x}_t) - r_{t-1} \\ \hat{V}_0(\mathbf{x}_t) - r_t \end{pmatrix}.$$

Overall, the update of Eq. (B.0.1) may be written as

$$\hat{\mathbf{w}}_t = \hat{\mathbf{w}}_0 - \eta\sum_{i=0}^{t}\mathbf{z}_i\left(\hat{V}_0(\mathbf{x}_i) - \gamma\hat{V}_0(\mathbf{x}_{i+1}) - r_i\right), \quad \text{with } \hat{V}_0(\mathbf{x}_t) = 0, \qquad \text{(B.0.3)}$$

$\mathbf{z}_i$ being the i'th column of $\mathbf{Z}_t^{(1)}$, which is also the TD(1) eligibility vector at time-step i. $\hat{V}_0(\mathbf{x}_{t+1}) = 0$ as $\mathbf{x}_{t+1}$ is a zero-reward absorbing terminal state. The update equation (B.0.3) is precisely the batch (off-line) form of the TD(1) algorithm for linear architectures (see Bertsekas & Tsitsiklis, 1996, 6.3.2, 6.3.3, and Sutton & Barto, 1998).

Appendix C

Proofs

C.1 Proof of Lemma 1.3.2

Lemma. *If* $\mathbf{A}$ *is an* $n \times m$ *matrix,* $\mathbf{B}$ *an* $m \times n$ *matrix, and* $\mathbf{BA} + \mathbf{I}$ *non-singular, then* $\mathbf{AB} + \mathbf{I}$ *is also non-singular, and* [1]

$$\mathbf{A}\left(\mathbf{BA} + \mathbf{I}\right)^{-1} = \left(\mathbf{AB} + \mathbf{I}\right)^{-1}\mathbf{A}.$$

Proof Assume to the contrary that $\mathbf{AB} + \mathbf{I}$ is singular, then there exists a vector $\mathbf{y} \neq \mathbf{0}$ such that

$$(\mathbf{AB} + \mathbf{I})\mathbf{y} = \mathbf{0}. \tag{C.1.1}$$

Left-multiply this by $\mathbf{B}$ to obtain

$$\mathbf{B}(\mathbf{AB} + \mathbf{I})\mathbf{y} = (\mathbf{BA} + \mathbf{I})\mathbf{By} = \mathbf{0}.$$

Now, $\mathbf{By} \neq \mathbf{0}$, for otherwise we would have from Eq. (C.1.1) that $\mathbf{y} = \mathbf{0}$. Therefore, for $\mathbf{x} \overset{\text{def}}{=} \mathbf{By} \neq \mathbf{0}$ we have

$$(\mathbf{BA} + \mathbf{I})\mathbf{x} = \mathbf{0},$$

which means that $(\mathbf{BA} + \mathbf{I})$ is singular, resulting in a contradiction.

The second part is proved as follows:

$$\begin{aligned}\mathbf{A}\left(\mathbf{BA} + \mathbf{I}\right)^{-1} &= (\mathbf{AB} + \mathbf{I})^{-1}(\mathbf{AB} + \mathbf{I})\mathbf{A}\left(\mathbf{BA} + \mathbf{I}\right)^{-1} \\ &= (\mathbf{AB} + \mathbf{I})^{-1}\mathbf{A}(\mathbf{BA} + \mathbf{I})(\mathbf{BA} + \mathbf{I})^{-1} \\ &= (\mathbf{AB} + \mathbf{I})^{-1}\mathbf{A}\end{aligned}$$

[1] Note that $\mathbf{I}$ on the l.h.s. is the $m \times m$ identity matrix, while on the r.h.s. it is the $n \times n$ identity matrix.

C.2

Proposition C.2.1. *In the parametric model with deterministic transitions*

$$\frac{\sigma_{t-1}^2}{s_t}\mathbf{p}_t = \mathbf{Cov}\left[W, V(\mathbf{x}_{t-1}) - \gamma V(\mathbf{x}_t) - R(\mathbf{x}_{t-1})|R_{t-1}\right].$$

Proof First, note that the $R(\mathbf{x}_{t-1})$ term contributes nothing, as it is being conditioned upon. Therefore, we need only compute

$$\begin{aligned}
\mathbf{Cov}\left[W, V(\mathbf{x}_{t-1}) - \gamma V(\mathbf{x}_t)|R_{t-1}\right] &= \mathbf{Cov}\left[W, \mathbf{h}_t^\top \mathbf{\Phi}_t^\top W|R_{t-1}\right] \\
&= \mathbf{Cov}\left[W, W|R_{t-1}\right]\mathbf{\Phi}_t \mathbf{h}_t \\
&= \mathbf{P}_t \mathbf{\Delta}\boldsymbol{\phi}_t \\
&= \left(\mathbf{P}_{t-1} - \frac{1}{s_t}\mathbf{p}_t\mathbf{p}_t^\top\right)\mathbf{\Delta}\boldsymbol{\phi}_t \\
&= \frac{1}{s_t}\left(s_t\mathbf{p}_t - \mathbf{p}_t(s_t - \sigma_{t-1}^2)\right) \\
&= \frac{\sigma_{t-1}^2}{s_t}\mathbf{p}_t.
\end{aligned}$$

□

In the third equality we used the definition of $\mathbf{P}_t$, in the fourth the recursive update for $\mathbf{P}_t$, and in the fifth the definitions of $\mathbf{p}_t$ and s_t.

C.3

Proposition C.3.1. *In the nonparametric model with deterministic transitions*

$$\frac{\sigma_{t-1}^2}{s_t}\mathbf{c}_t^\top \mathbf{k}_t(\mathbf{x}) = \mathbf{Cov}\left[V(\mathbf{x}), V(\mathbf{x}_{t-1}) - \gamma V(\mathbf{x}_t) - R(\mathbf{x}_{t-1})|R_{t-1}\right].$$

Proof First, note that the $R(\mathbf{x}_{t-1})$ term contributes nothing, as it is being conditioned upon. Therefore, we need only compute

$$\begin{aligned}
\mathbf{Cov}\left[V(\mathbf{x}), V(\mathbf{x}_{t-1}) - \gamma V(\mathbf{x}_t) | R_{t-1}\right] &= \mathbf{Cov}\left[V(\mathbf{x}), V_t | R_{t-1}\right] \mathbf{h}_t \\
&= \mathbf{h}_t^\top \left(\mathbf{k}_t(\mathbf{x}) - \mathbf{K}_t \mathbf{C}_t \mathbf{k}_t(\mathbf{x})\right) \\
&= \mathbf{h}_t^\top \left(\mathbf{I} - \mathbf{K}_t \left(\begin{bmatrix} \mathbf{C}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t}\mathbf{c}_t\mathbf{c}_t^\top\right)\right) \mathbf{k}_t(\mathbf{x}) \\
&= \left(\mathbf{c}_t^\top - \frac{1}{s_t}(\mathbf{k}_t(\mathbf{x}_{t-1}) - \gamma \mathbf{k}_t(\mathbf{x}_t))^\top \mathbf{c}_t \mathbf{c}_t^\top\right) \mathbf{k}_t(\mathbf{x}) \\
&= \frac{1}{s_t}\left(s_t - (\mathbf{k}_t(\mathbf{x}_{t-1}) - \gamma \mathbf{k}_t(\mathbf{x}_t))^\top \mathbf{c}_t\right) \mathbf{c}_t^\top \mathbf{k}_t(\mathbf{x}) \\
&= \frac{\sigma_{t-1}^2}{s_t} \mathbf{c}_t^\top \mathbf{k}_t(\mathbf{x}).
\end{aligned}$$

□

In the third equality we used the recursive formula for $\mathbf{C}_t$, in the fourth the definitions of $\mathbf{c}_t$, and in the sixth we used the definition of s_t and the identity $(\mathbf{k}_t(\mathbf{x}_{t-1}) - \gamma \mathbf{k}_t(\mathbf{x}_t))^\top \mathbf{c}_t = \Delta k_{tt} - \mathbf{\Delta}\tilde{\mathbf{k}}_t^\top \mathbf{C}_{t-1} \mathbf{\Delta}\tilde{\mathbf{k}}_t$.

Appendix D

Mathematical Formulae

In the sequel read Pr as the probability density function of its argument, unless that argument is countable, in which case Pr denotes probability and $\int$ denotes a sum.

D.1 Bayes' Rule

Let X and Y be random variables or vectors, then

$$\Pr(X|Y) = \frac{\Pr(Y|X)\Pr(X)}{\Pr(Y)} = \frac{\Pr(Y|X)\Pr(X)}{\int dX' \Pr(Y|X')\Pr(X')}.$$

D.2 The Multivariate Normal Distribution

Let X be an n dimensional random vector. It is said that X is normally distributed (or alternatively, that its components are jointly normally distributed) as $\mathcal{N}\{\mathbf{m}, \mathbf{\Sigma}\}$ if its probability density function satisfies

$$\Pr(\mathrm{X}) = (2\pi)^{-n/2}|\mathbf{\Sigma}|^{-1/2}\exp\left(-\tfrac{1}{2}(X-\mathbf{m})^\top \mathbf{\Sigma}^{-1}(X-\mathbf{m})\right).$$

D.3 Conditional Expectation and Covariance Formulae

Let X, Y, Z be random variables or vectors, then

$$\begin{aligned}\mathbf{E}[X] &= \mathbf{E}_Y\left[\mathbf{E}[X|Y]\right],\\ \mathbf{Cov}[X,Z] &= \mathbf{E}_Y\left[\mathbf{Cov}[X,Z|Y]\right] + \mathbf{Cov}_Y\left[\mathbf{E}[X|Y], \mathbf{E}[Z|Y]\right].\end{aligned}$$

Specializing the last equation to X=Z scalars, we get the conditional variance formula:

$$\mathbf{Var}[X] = \mathbf{E}_Y\left[\mathbf{Var}[X|Y]\right] + \mathbf{Var}_Y\left[\mathbf{E}[X|Y]\right].$$

D.4 Matrix Inversion Formulae

Matrix Inversion Lemma (Sherman-Morrison-Woodbury)

Let $\mathbf{U}$ be a square invertible matrix, then

$$(\mathbf{U} + \beta\mathbf{X}\mathbf{Y})^{-1} = \mathbf{U}^{-1} - \mathbf{U}^{-1}\mathbf{X}\left(\beta\mathbf{I} + \mathbf{Y}\mathbf{U}^{-1}\mathbf{X}\right)^{-1}\mathbf{Y}\mathbf{U}^{-1}.$$

Partitioned Covariance Matrix Inverse Formula

Let $\mathbf{K}_t$ be a $t \times t$ (symmetric and positive-definite) covariance matrix, partitioned as

$$\mathbf{K}_t = \begin{bmatrix} \mathbf{K}_{t-1} & \mathbf{k}_t \\ \mathbf{k}_t^\top & k_{tt} \end{bmatrix}.$$

Then

$$\mathbf{K}_t^{-1} = \begin{bmatrix} \mathbf{K}_{t-1}^{-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t}\begin{bmatrix} \mathbf{K}_{t-1}^{-1}\mathbf{k}_t \\ -1 \end{bmatrix}\begin{bmatrix} \mathbf{k}_t^\top\mathbf{K}_{t-1}^{-1} & -1 \end{bmatrix},$$

where $s_t = k_{tt} - \mathbf{k}_t^\top\mathbf{K}_{t-1}^{-1}\mathbf{k}_t$. s_t is called the *Schur complement* of k_{tt} in $\mathbf{K}_t$. If $X = (X_1, \ldots, X_t)^\top$ is a jointly Gaussian random vector, the self-covariance of which satisfies $\mathbf{Cov}[X] = \mathbf{K}_t$, then by the Gauss-Markov theorem (Theorem 1.3.1), s_t satisfies

$$s_t = \mathbf{Var}[X_t|X_1, \ldots, X_{t-1}].$$

(Scharf, 1991, Chapter 2).

Bibliography

Aizerman, M., Braverman, E., & Rozonoer, L. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, *25*, 821–837.

Anthony, M., & Bartlett, P. (1999). *Neural network learning; theoretical foundations.* Cambridge University Press.

Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. *International Conference on Machine Learning* (pp. 30–37).

Bellman, R. (1957). *Dynamic programming.* Princeton University Press.

Berry, D., & Fristedt, B. (1985). *Bandit problems: Sequential allocation of experiments.* Chapman and Hall.

Bertsekas, D. (1995). *Dynamic programming and optimal control.* Athena Scientific.

Bertsekas, D., & Tsitsiklis, J. (1996). *Neuro-dynamic programming.* Athena Scientific.

Boser, B. E., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. *Computational Learning Theory* (pp. 144–152).

Boyan, J. (1999a). Least-squares temporal difference learning. *Proc. 16th International Conference on Machine Learning* (pp. 49–56). Morgan Kaufmann, San Francisco, CA.

Boyan, J. (1999b). Technical update: Least-squares temporal difference learning.

Boyan, J., & Moore, A. (1995). Generalization in reinforcement learning: Safely approximating the value function. *Advances in Neural Information Processing Systems 7* (pp. 369–376). Cambridge, MA: MIT Press.

Bradtke, S., & Barto, A. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, *22*, 33–57.

Brown, M., Grundy, W., Lin, D., Cristianini, N., Sugnet, C., Furey, T., Ares, M., & Haussler, D. (2000). Knowledge-based analysis of microarray gene expression data by using suport vector machines. *Proceedings of the National Academy of Sciences* (pp. 262–267).

Burges, C. (1996). Simplified support vector decision rules. *International Conference on Machine Learning* (pp. 71–77).

Burges, C., & Schölkopf, B. (1997). Improving the accuracy and speed of support vector machines. *Advances in Neural Information Processing Systems.* MIT Press.

Collins, M., & Duffy, N. (2001). Convolution kernels for natural language. *Advances in Neural Information Processing Systems 14* (pp. 625–632).

Collobert, R., & Bengio, S. (2001). SVMTorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, *1*, 143–160.

Cox, D., & O'Sullivan, F. (1990). Asymptotic analysis of penalized likelihood and related estimators. *The Annals of Statistics*, *18*, 1676–1695.

Crites, R., & Barto, A. (1996). Improving elevator performance using reinforcement learning. *Advances in Neural Information Processing Systems* (pp. 1017–1023). MIT Press.

Csató, L., & Opper, M. (2001). Sparse representation for Gaussian process models. *Advances in Neural Information Processing Systems 13*.

Csató, L., & Opper, M. (2002). Sparse on-line Gaussian processes. *Neural Computation*, *14*, 641–668.

Cucker, F., & Smale, S. (2001). On the mathematical foundations of learning. *Bulletin of the American Mathematical Society*, *39:1*, 1–49.

Dayan, P. (1992). The convergence of TD(λ) for general λ. *Machine Learning*, *8*, 341–362.

Dayan, P., & Sejnowski, T. (1994). TD(λ) converges with probability 1. *Machine Learning*, *14 (3)*, 295–301.

Dearden, R., Friedman, N., & Russell, S. (1998). Bayesian Q-learning. *Proc. of the Fifteenth National Conference on Artificial Intelligence*.

DeCoste, D., & Schölkopf, B. (2002). Training invariant support vector machines. *Machine Learning*, *46*, 161–190.

Downs, T., Gates, K., & Masters, A. (2001). Exact simplification of support vector solutions. *Journal of Machine Learning Research*, *2*, 293–297.

Dzeroski, S., Raedt, L. D., & Blockeel, H. (1998). Relational reinforcement learning. *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 136–143).

Engel, Y., & Mannor, S. (2001). Learning embedded maps of Markov processes. *Proc. of the Eighteenth International Conference on Machine Learning*.

Engel, Y., & Mannor, S. (2005). Collaborative temporal difference learning with Gaussian processes. *Submitted to the Twenty-Second International Conference on Machine Learning*.

Engel, Y., Mannor, S., & Meir, R. (2002). Sparse online greedy support vector regression. *13th European Conference on Machine Learning*.

Engel, Y., Mannor, S., & Meir, R. (2003). Bayes meets Bellman: The Gaussian process approach to temporal difference learning. *Proc. of the 20th International Conference on Machine Learning*.

Engel, Y., Mannor, S., & Meir, R. (2004). The kernel recursive least squares algorithm. *IEEE Transactions on Signal Processing*, *52*, 2275–2285.

Engel, Y., Mannor, S., & Meir, R. (2005). Reinforcement learning with Gaussian processes. *Submitted to the Twenty-Second International Conference on Machine Learning.*

Even-Dar, E., Mannor, S., & Mansour, Y. (2003). Action elimination and stopping conditions for reinforcement learning. *Proc. of the 20th International Conference on Machine Learning.*

Evgeniou, T., Pontil, M., & Poggio, T. (2000). Regularization networks and support vector machines. *Advances in Computational Mathematics*, *13(1)*, 1–50.

Feynman, R. (1988). http://www.enc.org/features/calendar/unit/0,1819,243,00.shtm.

Fine, S., & Scheinberg, K. (2001). Efficient SVM training using low-rank kernel representation. *JMLR (Special Issue on Kernel Methods)*, 243–264.

Friedman, J. (1991). Multivariate adaptive regression splines. *Annals of Statistics*, *19(1)*, 1–141.

Gibbs, M., & MacKay, D. (1997). *Efficient implementation of Gaussian processes* (Technical Report). Department of Physics, Cavendish Laboratory, Cambridge University.

Girosi, F., Jones, M., & Poggio, T. (1995). Regularization theory and neural networks architectures. *Neural Computation*, *7*, 219–269.

Gordon, G. (1996). Stable fitted reinforcement learning. *Advances in Neural Information Processing Systems* (pp. 1052–1058). MIT Press.

Graepel, T. (2003). Solving noisy linear operator equations by gaussian processes: Application to ordinary and partial differential equations. *Proceedings of the 20th International Conference on Machine Learning* (pp. 234–241). AAAI Press.

Haykin, S. (1996). *Adaptive filter theory.* Prentice Hall. 3rd edition.

Herbrich, R. (2002). *Learning Kernel Classifiers.* Cambridge, MA: MIT Press.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. *Proceedings of the Tenth European Conference on Machine Learning* (pp. 137–142).

Kaelbling, L., Littman, M., , & Cassandra, A. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*, 99–134.

Kaelbling, L. P. (1993). *Learning in embedded systems.* MIT Press.

Kailath, T., Sayed, A., & Hassibi, B. (2000). *Linear estimation.* Prentice Hall.

Kalman, R. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME – Journal of Basic Engineering*, 35–45.

Kimeldorf, G., & Wahba, G. (1971). A correspondence between Bayesian estimation on stochastic processes and smoothing by splines. *Ann. Math. Statist.*, *41(2)*, 495–502.

Konda, V., & Tsitsiklis, J. (2000). Actor-critic algorithms.

Kondor, R. I., & Jebara, T. (2003). A kernel between sets of vectors. *Machine Learning, Proceedings of the Twentieth International Conference* (pp. 361–368).

König, H. (1986). *Eigenvalue distribution of compact operators.* Basel: Birkhauser Verlag.

Lagoudakis, M., Parr, R., & Littman, M. (2002). Least-squares methods in reinforcement learning for control. *SETN* (pp. 249–260).

LeCun, Y., Jackel, L. D., Bottou, L., Brunot, A., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Muller, U. A., Sackinger, E., Simard, P., & Vapnik, V. (1995). Comparison of learning algorithms for handwritten digit recognition. *International Conference on Artificial Neural Networks* (pp. 53–60). Paris: EC2 & Cie.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. *Proceedings of the 11th International Conference on Machine Learning (ICML-94)* (pp. 157–163). New Brunswick, NJ: Morgan Kaufmann.

Ljung, L. (1999). *System identification: Theory for the user.* New Jersey: Prentice Hall. Second edition.

M. Littman, R. S., & Singh, S. (2002). Predictive representations of state. *Advances in Neural Information Processing Systems 14 (NIPS)* (pp. 1555–1561).

Mackay, D. (1997). *Gaussian processes: A replacement for supervised neural networks?* (Technical Report). Cavendish Laboratory, Cambridge University.

Mackey, M., & Glass, L. (1977). Oscillation and chaos in physiological control systems. *Science*, *197*, 287–289.

Mallat, S. (1998). *A wavelet tour of signal processing.* New York: Academic Press.

Malzahn, D., & Opper, M. (2001). Learning curves for gaussian processes regression: A framework for good approximations. *Advances in Neural Information Processing Systems 13* (pp. 273–279). MIT Press.

Meir, R., & Zhang, T. (2003). Generalization error bounds for Bayesian mixture algorithms. *Journal of Machine Learning Research*, *4*, 839–860.

Mercer, J. (1909). Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society*, *A, 209*, 415–446.

Müller, K.-R., Smola, A., Rätsch, G., Schölkopf, B., Kohlmorgen, J., & Vapnik, V. (1997). Using support vector machines for time series prediction. *Proceedings ICANN'97, International Conference on Artificial Neural Networks* (pp. 999–1004). Berlin: Springer.

Munos, R. (2000). A study of reinforcement learning in the continuous case by the means of viscosity solutions. *Machine Learning*, *40(3)*, 265–299.

Munos, R. (2003). Error bounds for approximate policy iteration. *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA* (pp. 560–567). AAAI Press.

Natarajan, B. (1995). Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, *24(2)*, 227–234.

Nedic, A., & Bertsekas, D. P. (2003). Least squares policy evaluation algorithms with linear function approximation. *J. of Discrete Event Systems*, *13*, 79–110.

Opper, M., & Vivarelli, F. (1999). General bounds on bayes errors for regression with gaussian processes. *Advances in Neural Information Processing Systems 11* (pp. 302–308).

Osuna, E., & Girosi, F. (1998). Reducing the run-time complexity of support vector machines. *International Conference on Pattern Recognition.*

Puterman, M. (1994). *Markov decision processes: Discrete stochastic dynamic programming.* John Wiley & Sons.

Rasmussen, C., & Kuss, M. (2004). Gaussian processes in reinforcement learning. *Advances in Neural Information Processing Systems 16.* Cambridge, MA: MIT Press.

Robert, C. (1994). *The Bayesian choice.* New York: Springer.

Sayed, A., & Kailath, T. (1994). A state-space approach to adaptive RLS filtering. *IEEE Signal Processing Magazine*, *11*, 18–60.

Scharf, L. (1991). *Statistical signal processing.* Addison-Wesley.

Schölkopf, B., Mika, S., Burges, C., Knirsch, P., Mller, K.-R., Rtsch, G., & Smola, A. (1999). Input space vs. feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, *10*, 1000–1017.

Schölkopf, B., & Smola, A. (2002). *Learning with Kernels.* Cambridge, MA: MIT Press.

Schölkopf, B., Smola, A., & Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, *10*, 1299–1319.

Schraudolph, N., Dayan, P., & Sejnowski, T. (1994). Temporal difference learning of position evaluation in the game of Go. *Advances in Neural Information Processing Systems* (pp. 817–824). Morgan Kaufmann Publishers, Inc.

Sebald, D., & Bucklew, J. (2000). Support vector machine techniques for nonlinear equalization. *IEEE Transactions on Signal Processing*, *48*, 3217–3226.

Seeger, M. (2003). *Bayesian Gaussian process models: PAC-Bayesian generalisation error bounds and sparse approximations.* Doctoral dissertation, University of Edinburgh.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis.* Cambridge, England: Cambridge University Press.

Singh, S., Jaakkola, T., & Jordan, M. (1995). Reinforcement learning with soft state aggregation. *Advances in Neural Information Processing Systems 7* (pp. 361–368). MIT Press.

Singh, S. P., & Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, *22(1).*

Smola, A., & Bartlett, P. (2001). Sparse greedy Gaussian process regression. *Advances in Neural Information Processing Systems 13* (pp. 619–625). MIT Press.

Smola, A., & Schölkopf, B. (2000). Sparse greedy matrix approximation for machine learning. *Proc. 17th International Conference on Machine Learning* (pp. 911–918). Morgan Kaufmann, San Francisco, CA.

Sobel, M. (1982). The variance of discounted Markov decision processes. *Journal of Applied Probability*, *19*, 794–802.

Solak, E., Murray-Smith, R., Leithead, W., Leith, D., & Rasmussen, C. (2003). Derivative observations in Gaussian process models of dynamic systems. *Advances in Neural Information Processing Systems 15* (pp. 1033–1040). Cambridge, MA: MIT Press.

Sollich, P. (2002). Bayesian methods for support vector machines: Evidence and predictive class probabilities. *Machine Learning*, *46*, 21–52.

Sollich, P., & Williams, C. K. I. (2005). Using the equivalent kernel to understand gaussian process regression. *Advances in Neural Information Processing Systems 17* (pp. 1313–1320). Cambridge, MA: MIT Press.

Sutton, R. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction.* MIT Press.

Syed, N., Liu, H., & Sung, K. (1999). Incremental learning with support vector machines. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99).*

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Comm. ACM*, *38*, 58–68.

Tikhonov, A. N., & Arsenin, V. Y. (1977). *Solutions of ill-posed problems.* Washington, D.C.: W. H. Winston.

Tipping, M. (2001a). Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, *1*, 211–244.

Tipping, M. (2001b). Sparse kernel principal component analysis. *Advances in Neural Information Processing Systems 135* (pp. 633–639). MIT Press.

Tsitsiklis, J., & Van Roy, B. (1996). *An analysis of temporal-difference learning with function approximation* (Technical Report LIDS-P-2322). MIT.

Turing, A. (1950). Computing machinery and intelligence. *Mind*, *59 (N.S. 236)*, 433–460.

van der Vaart, A., & Wellner, J. (1996). *Weak Convergence and Empirical Processes.* New York: Springer Verlag.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory.* New York: Springer Verlag.

Vapnik, V. (1998). *Statistical Learning Theory.* New York: Wiley Interscience.

Vapnik, V., Golowich, S., & Smola, A. (1997). Support vector method for function approximation, regression estimation, and signal processing. *Advances in Neural Information Processing Systems* (pp. 281–287).

Vidyasagar, M. (2003). Recent advances in statistical learning theory. In J. Suykens, G. Horvath, S. Basu, C. Micchelli and J. Vandewalle (Eds.), *Advances in learning theory: Methods, models and applications, NATO science series III: Computer & systems sciences*, vol. 190, 357–374. Amsterdam: IOS Press.

Vincent, P., & Bengio, Y. (2002). Kernel matching pursuit. *Machine Learning*, *48*, 165–187.

Watkins, C. (1989). *Learning from delayed rewards.* Doctoral dissertation, King's College, Cambridge.

Watkins, C. (1999). *Dynamic alignment kernels* (Technical Report CSD-TR-98-11). UL Royal Holloway.

Weigend, A. S., & Gershenfeld, N. A. (Eds.). (1994). *Time series prediction: Forecasting the future and understanding the past.* Reading, MA: Addison-Wesley.

Werbos, P. (1990). Consistency of HDP applied to simple reinforcement learning problems. *Neural Networks*, *3*, 179–189.

Williams, C. (1999). Prediction with Gaussian processes: From linear regression to linear prediction and beyond. In M. I. Jordan (Ed.), *Learning and inference in graphical models.* Kluwer.

Williams, C., & Seeger, M. (2001). Using the Nyström method to speed up kernel machines. *Advances in Neural Information Processing Systems 13* (pp. 682–688).

Williams, R. J., & Baird, L. C. (1993). *Tight performance bounds on greedy policies based on imperfect value functions* (Technical Report NU-CCS-93-14). Northeastern University, College of Computer Science.

Zhang, T. (2003). Sequential greedy approximation for certain convex optimization problems. *IEEE Transactions on Information Theory*, *49*, 682–691.