

https://dspace.org/

Representation discovery using a fixed basis in reinforcement learning

Wookey, Dean Stephen

2016

http://hdl.handle.net/10539/21642

Wookey, Dean Stephen (2016) Representation discovery using a fixed basis in reinforcement learning, University of Witwatersrand, Johannesburg, <http://wiredspace.wits.ac.za/handle/10539/21642>

Downloaded from a DSpace repository. https://dspace.org/

Representation Discovery Using a Fixed Basis in Reinforcement Learning

Dean Stephen Wookey
Supervised by Prof. George Konidaris and Prof. Clint van Alten

A thesis presented for the degree of
Doctor of Philosophy



School of Computer Science and Applied Mathematics
University of the Witwatersrand
South Africa

26 August 2016

Abstract

In the reinforcement learning paradigm, an agent learns by interacting with its environment. At each state, the agent receives a numerical reward. Its goal is to maximise the discounted sum of future rewards. One way it can do this is through learning a value function; a function which maps states to the discounted sum of future rewards. With an accurate value function and a model of the environment, the agent can take the optimal action in each state. In practice, however, the value function is approximated, and performance depends on the quality of the approximation. Linear function approximation is a commonly used approximation scheme, where the value function is represented as a weighted sum of basis functions or features. In continuous state environments, there are infinitely many such features to choose from, introducing the new problem of *feature selection*. Existing algorithms such as OMP-TD are slow to converge, scale poorly to high dimensional spaces, and have not been generalised to the online learning case. We introduce heuristic methods for reducing the search space in high dimensions that significantly reduce computational costs and also act as regularisers. We extend these methods and introduce *feature regularisation* for incremental feature selection in the batch learning case, and show that introducing a smoothness prior is effective with our SSOMP-TD and STOMP-TD algorithms. Finally we generalise OMP-TD and our algorithms to the online case and evaluate them empirically.

Declaration

I, Dean Wookey, hereby declare the contents of this thesis to be my own work. This thesis is submitted for the degree of Doctor of Philosophy at the University of the Witwatersrand. This work has not been submitted to any other university, or for any other degree.

Signature _____ Date 26/08/2016

Acknowledgements

The completion of this research has not been easy. I would not have been able to do it without the help and support of my family and friends who have always encouraged me to keep pushing through the hard times.

I would like to thank Pravesh Ranchod for his constant optimism and ideas. He has always pushed me to pursue my ideas for which I am grateful for.

Clint van Alten has always looked out for my best interests. He has encouraged me, given me confidence, and helped me find solutions to some difficult problems.

Max Rabkin and Michael Mitchley, I thank you for listening to my ideas and offering suggestions. Your different backgrounds and ways of thinking benefited me and my research.

Steven James and Jeremy Lai Hong, you guys have stuck with me, encouraged me, and pushed me to be better. I am lucky to call you friends.

Warwick Masson and Craig Bester, thank you for supporting and participating in my initiatives. I have appreciated our conversations about research as well.

Shun Pillay, sorry for breaking the cluster so many times. You have been super supportive of the me and my initiatives, and I appreciate it.

Shashilan Singh, is always there when you need someone to talk to. He is always willing and able to get things done, and he steps way outside his role to improve computer science.

Federick Lim, you have been a true friend, and your support has been greatly appreciated.

When things were at their most difficult I could always count on Casey Bartels's love, support and encouragement to carry me through. Thank you for all the time you have spent listening to my ideas, checking and helping me with my maths and being there for me when nothing was going right. You are awesome and I love you.

Finally, I would like to thank George Konidaris. I could not have been luckier to have a supervisor like George. George has been an inspiration to me and many others in computer science, lending his considerable expertise to people he is barely met over email and skype. His optimism when things were bad, his knowledge when I was unsure, and his insights and ideas have been invaluable to me. I feel I owe him a great deal. Without George none of this would have been possible. Thank you George for being the best supervisor. Thank you for being there when I needed you. Thank you for supporting my initiatives. Thank you for helping all your students at Wits. You, and your expertise, make a huge difference here.

Contents

Abstract i

Declaration ii

Acknowledgements iii

1 Introduction 1

2 Background and Related Work 3

2.1 Introduction . . . 3
2.2 Reinforcement Learning . . . 3
2.2.1 Linear Function Approximation . . . 5
2.2.2 Basis Functions . . . 6
2.2.3 Learning Algorithms . . . 14
2.3 Feature Selection . . . 17
2.3.1 Matching Pursuit . . . 17
2.3.2 OMP-TD . . . 18
2.3.3 Related Work . . . 20
2.4 Conclusion . . . 21

3 Heuristics for Dictionary Size Reduction 23

3.1 Introduction . . . 23
3.2 State Variable Dependencies . . . 23
3.3 Dictionary Restriction . . . 25
3.3.1 Frequency-ordered Dictionaries . . . 26
3.4 Regularised Fourier Correlation Search . . . 27
3.5 Algorithms . . . 28
3.6 Experiments . . . 30
3.6.1 Domains . . . 31
3.6.2 Methods . . . 32
3.7 Discussion . . . 34
3.8 Conclusion . . . 36

4 Regularised Feature Selection 38

4.1 Regularisation . . . 39
4.2 Feature Regularisation . . . 39
4.2.1 Value Function Smoothness . . . 40
4.2.2 Smoothness Regularised Feature Selection Methods . . . 45
4.2.3 ROMP-TD and LSTD-RP . . . 47

4.3 Experiments . . . 49
4.3.1 Domains . . . 49
4.3.2 Results . . . 49
4.4 Summary and Conclusions . . . 53

5 Online Feature Selection **54**
5.1 Introduction . . . 54
5.2 Contraction for Q . . . 55
5.2.1 QBEBFs . . . 57
5.3 Exponentially Weighted Correlation . . . 60
5.4 Online Feature Selection Algorithms . . . 61
5.5 Experiments . . . 62
5.5.1 Methods . . . 62
5.5.2 Domains . . . 64
5.5.3 Results . . . 64
5.6 Conclusion . . . 66

6 Conclusion and Future Work **69**

7 Appendix **71**
7.1 Bounding α with Scaling . . . 71

References **72**

Chapter 1

Introduction

Reinforcement learning is a machine learning paradigm in which an agent's objectives are implicitly defined by numerical rewards given to it by the environment. At each state the agent selects an action, receiving both the next state and reward resulting from that action. The agent's goal is to choose actions such that the discounted total numerical reward it receives is maximised. It does this by learning a policy, a stochastic mapping of states to actions.

One way to learn a policy is through learning a value function. Value functions map states to the expected discounted total numerical reward that can be received from those states. With a model of the environment which predicts the outcome of taking actions, the value function can be used to evaluate the expected outcome of each action. Choosing the best action then is simply a matter of evaluating each action and choosing the one which has the largest expected discounted total numerical reward.

When a model is not available, an action-value function can be learned instead. Action-value functions map (state, action) pairs to the expected discounted total numerical reward for taking that action in that state. They can similarly be used to choose the best action to take in each state.

Unfortunately both methods require accurate values at each state. When the states are discrete and small in number, the value mappings can be stored in a table. However in many applications of interest, state spaces are either too large to store explicitly, or are continuous. In these cases some form of approximation must be used.

Linear function approximation is one approximation scheme which is commonly used in reinforcement learning. It combines basis functions—real valued functions of the environment state variables—in a linear fashion with a set of associated weights. The basis set determines the class of functions that can be represented, and normally consists of a set of orthogonal basis functions, usually a fixed basis set containing functions up to a certain accuracy (or order) of approximation.

Linear function approximation works well when the number of variables (or *dimensions*) comprising the state is low, but unfortunately many tasks of interest have a large number of state variables. Since the number of basis functions in a linear function approximation scales exponentially in the number of state variables, there is a combinatorial explosion of basis functions in high-dimensional state spaces.

Fortunately, many tasks are not inherently difficult, and only require a relatively small subset of basis functions to approximate well. Choosing this subset is challenging as there are infinitely many basis functions to choose from. Even when the set of candidates is restricted to a fixed basis set of a certain order, a linear search through this set (or *dictionary*) is infeasible in high dimensional domains because the dictionary itself grows exponentially in the number of dimensions.

A common approach for avoiding this problem involves making an *independence assumption* with regards to the way state variables interact [Stone *et al.* 2006]. If state variables are assumed to contribute to the value function independently, then large numbers of basis functions involving interactions of state variables can be discarded, resulting in basis sets which scale linearly with dimension. While making this assumption often produces good results, there is often little theoretical grounding for making it, and doing so blindly can be dangerous. This method of reducing the basis function set size has been used effectively in Sutton and Barto [1998] and Stone *et al.* [2006], but without a reason to believe state variables are independent, it is possible that important features are discarded and the agent's performance may suffer.

Feature selection and construction aim to solve these problems by automatically selecting or constructing good basis functions to use in linear function approximation. Recent work in *feature selection* [Kolter and Ng 2009; Johns *et al.* 2010; Painter-Wakefield and Parr 2012; Petrik *et al.* 2010; Ghavamzadeh *et*

al. 2011; Geist *et al.* 2012] and *construction* [Ghavamzadeh *et al.* 2010; Sun *et al.* 2011; Parr *et al.* 2007; Mahadevan and Liu 2010] select or construct basis functions specifically for each domain, and are typically used in conjunction with a batch learning method [Bradtke and Barto 1996; Lagoudakis and Parr 2003; Ernst *et al.* 2005] to fit the weights of the linear function approximation.

Of particular interest are greedy selection methods such as temporal difference[1] orthogonal matching pursuit (OMP-TD), and orthogonal matching pursuit with Bellman residual minimisation (OMP-BRM) [Painter-Wakefield and Parr 2012], which aim to select basis functions predicted to cause the largest reduction in approximation error post selection.

These methods use a set of samples collected from the environment, both to inform selection of new basis functions from a dictionary of candidates as well as to approximate the value function. The result of either method is a tailored set of basis functions and their respective weights specific to the current domain/task.

In many cases dictionary search methods such as OMP-TD become impractical as the dictionary grows too large to search in even linear time. We explore methods for restricting the set of Fourier basis functions which enter the dictionary in Chapter 3, thereby reducing the computational cost of selecting features. We show that using frequency as a dictionary restriction heuristic is effective at reducing the dictionary size and also acts as a regulariser.

Even when the number of candidates in the dictionary is low however, current selection metrics tend to select many poor basis functions leading to slow convergence. One way to improve selection is to introduce a prior. The smoothness prior is related to frequency in the Fourier basis, but is more general and can be used to bias selection towards basis functions which are smoother. This works because value functions tend to be smooth in general. In Chapter 4, we introduce two algorithms based on OMP-TD which use the smoothness prior—Smoothness Scaled OMP-TD and Smooth Tikhonov OMP-TD—and empirically evaluate them.

The methods discussed so far deal with value functions in the batch setting. In the online setting it is not immediately clear that using correlation to the Bellman error as a selection metric is theoretically sound. Furthermore the value function is constantly changing as the agent learns new policies. Since the Bellman error is dependent on the current policy, it is not clear what the Bellman error is at any one time. In Chapter 5 we extend the theoretical results of Parr *et al.* [2007] to the case of action-value functions, showing that correlations to the Bellman errors in each action-value function can be used as feature selection metrics. We also tackle the problem of changing policies, and empirically evaluate our algorithms on six benchmark domains with the Fourier basis.

These contributions provide a collection of practical tools for feature selection in high-dimensional reinforcement learning problems, both online and in the batch setting.

[1] Temporal difference (TD) methods use numeric differences in the values of successive states to learn. They are further described in Section 2.2.3.1.

Chapter 2

Background and Related Work

2.1 Introduction

Reinforcement learning, described in Section 2.2, is a machine learning paradigm central to the topic of this thesis: improving feature selection for linear function approximation in reinforcement learning, both online and offline with fixed basis sets. Within the reinforcement learning paradigm, one way learning takes place is through learning a value function, a function which maps environment states to returns. Our goal is to improve reinforcement learning algorithms that use linear function approximation (Section 2.2.1) to approximate the value function.

Linear function approximation is commonly used in reinforcement learning, where it is normally paired with a fixed basis set such as those described in Sections 2.2.2.1, 2.2.2.2, 2.2.2.3 and less commonly 2.2.2.4. Learning the weights associated with linear function approximations can be achieved in many ways, however we use Sarsa(λ) (Section 2.2.3.3) in the online setting, and least squares temporal difference (LSTD) [Bradtke and Barto 1996] (Section 2.2.3.4) in the offline setting. Although we do not use least squares policy iteration (LSPI) [Lagoudakis and Parr 2003] (Section 2.2.3.5), we include it as a way to do policy improvement.

The primary focus of this work is feature selection, which we discuss in Section 2.3. In the context of reinforcement learning with linear function approximation, feature selection tackles the problem of determining which basis functions to include in the approximation. One algorithm, orthogonal matching pursuit applied to the reinforcement learning setting (OMP-TD) [Painter-Wakefield and Parr 2012], is discussed in Section 2.3.2. This algorithm is based on the matching pursuit (MP) [Mallat and Zhang 1993] algorithm discussed in Section 2.3.1. Many of the algorithms presented in this thesis are based on OMP-TD.

The final section, Section 2.3.3, of Chapter 2 discusses the relevant related work. In particular it discusses related feature selection methods and one alternative to feature selection: feature construction. The theory behind Bellman error basis functions (BEBFs) [Parr *et al.* 2007], in Section 2.3.3.5, form the basis for the OMP-TD algorithm as well as our work on online feature selection in Chapter 5.

2.2 Reinforcement Learning

A stochastic process has the Markov property if knowledge of states prior to the current state is unnecessary to predict future states. A reinforcement learning task that satisfies the Markov property is typically formalised as a Markov decision process (MDP) [Sutton and Barto 1998]. An MDP is defined as a tuple,

$$M = (S, A, P, R), \tag{2.1}$$

where $S = \{s_1, s_2, ..., s_n\}$ is the set of all possible states, $A = \{a_1, a_2, ..., a_m\}$ is the set of all actions, P is the transition probability model where $P^a_{ss'}$ is the probability of moving from state s to state s' with action a, and R is a real valued reward function where $r = R(s, a, s')$ is the reward for taking action a in state s and ending up in state s' [Sutton and Barto 1998]. We assume that the state consists of a vector of real values, i.e., $s = [x_1, ..., x_d]$ where each x_j is scaled to be between 0 and 1 and d is the total number of dimensions. Let s_t and a_t denote the state and action taken at time t, and r_t denote the reward given when entering state s_t at time t.

We can think of an MDP as a decision making process where an agent moves between states in an environment by selecting and executing actions. The agent only knows in which state it is, what actions

are available to it, and the rewards it has received. The agent does not have the transition probability model, P, meaning that before taking an action, the resulting next state s' is unknown. The reward function R is also unknown. To traverse the environment, an agent executes one of the available actions and receives a numerical reward from the environment upon entering the next state (which may or may not be the same state). This process is repeated until the agent enters a terminal state.

A policy, π, is a stochastic mapping of states to actions where the probability of choosing a particular action a in state s is given by $\pi(s, a)$. In the case of a deterministic policy, π is a direct mapping of states to actions $\pi : S \to A$ and $\pi(s)$ is the action chosen in state s [Lagoudakis and Parr 2003]. The probability $P^{\pi}_{ss'}$ an agent transitions from a state s to another state s' with policy π is given as $P^{\pi}_{ss'} = \sum_a \pi(s, a) P^a_{ss'}$.

The goal of reinforcement learning is to find an optimal policy π^*, maximising the return $\mathcal{R}$ (the discounted sum of future rewards) from each state s_t

$$\mathcal{R}_t = \sum_{i=0}^{\infty} \gamma^i r_{i+t+1}, \tag{2.2}$$

where $\gamma \in [0, 1)$ is the discount parameter which reduces the weight of future rewards.

The agent represents the expected return from a state following a policy π using a *value function* $V^{\pi}(s)$ which satisfies:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{R_t | s = s_t\} \\ &= \sum_a \pi(s, a) \sum_{s'} P^a_{ss'} \left[R(s, a, s') + \gamma V^{\pi}(s') \right]. \end{aligned} \tag{2.3}$$

For a given $V^{\pi}(s)$, if P is known, the value of taking each action at a particular state s can be calculated.

$$V^{\pi}(s|a) = \sum_{s'} P^a_{ss'} \left[R(s, a, s') + \gamma V^{\pi}(s') \right]. \tag{2.4}$$

If P is not known, the action-value function, $Q(s, a)$ satisfying

$$\begin{aligned} Q^{\pi}(s, a) &= E_{\pi}\{R_t | s = s_t, a = a_t\} \\ &= \sum_{s'} P^a_{ss'} \left[R(s, a, s') + \gamma \sum_{a'} \pi(s', a') Q^{\pi}(s', a') \right], \end{aligned} \tag{2.5}$$

can be used to determine the value of taking a particular action a in state s.

The policy is typically a function of the value function, where the probability of taking each action a in state s is based on the immediate reward and expected value of the resulting states $\sum_{s'} P^a_{ss'}[R(s, a, s') + \gamma V^{\pi}(s')] = Q^{\pi}(s, a)$. One such policy is the *greedy* policy which chooses the action with highest value at each step,

$$\pi(s) = \arg\max_{a \in A} \sum_{s'} P^a_{ss'} \left[R(s, a, s') + \gamma V^{\pi}(s') \right], \tag{2.6}$$

or

$$\pi(s) = \arg\max_{a \in A} Q^{\pi}(s, a). \tag{2.7}$$

Given a policy, π_t, and access to a policy evaluation method that returns an accurate value function $V^{\pi}(s)$ for any policy π, an improved policy π_{t+1} can be obtained from the greedy policy over $V^{\pi_t}(s)$.

$$\pi_t(s) = \arg\max_{a \in A} \sum_{s'} P^a_{ss'} V^{\pi_{t-1}}(s'), \tag{2.8}$$

or

$$\pi_t(s) = \arg\max_{a \in A} Q^{\pi_{t-1}}(s, a). \tag{2.9}$$

This process of repeatedly evaluating the policy and improving it is known as policy iteration, and is guaranteed to converge to the optimal policy π^*.

Although the greedy policy appears to be the best policy to use, it is often not. When learning online, the greedy policy does not explore the environment, and as a result may never accurately represent the value of some states because it never selects suboptimal actions in order to learn about them. States previously thought to be bad could turn out to be good if the right actions are taken later on, or if the rewards are not deterministic and the state has not been visited often enough to learn the expected reward at that state. If the policy explores too much, the rewards the agent receives will be low. The trade-off between exploiting (choosing the highest valued actions) and exploration must be balanced to achieve good learning rates and overall performance.

A simple policy that both explores and exploits is the ϵ-greedy policy. The ϵ-greedy policy takes a parameter $\epsilon \in [0, 1]$ which determines the probability $(1 - \epsilon)$ of selecting the action with the largest value; alternatively, it selects a random action. Ties are broken randomly in the case where there are multiple optimal actions. $\epsilon = 1$ results in the random policy and $\epsilon = 0$ results in the greedy policy.

2.2.0.1 Bellman Equation

The value function V^* for an optimal policy satisfies the *Bellman equation* or *Bellman optimality equation*

$$V^*(s) = \max_{a \in A} \sum_{s'} P^a_{ss'} \left[R(s, a, s') + \gamma V^*(s') \right], \tag{2.10}$$

where the optimal value of a state is the expected return at that state following the optimal policy.

The *Bellman operator* T^π for a policy π operating on a value function V is defined as

$$(T^\pi V)(s) = \sum_a \sum_{s'} \pi(s, a) P^a_{ss'} \left[R(s, a, s') + \gamma V(s') \right]. \tag{2.11}$$

T^*, the Bellman operator for the optimal policy is defined as

$$(T^* V)(s) = \max_{a \in A} \sum_{s'} P^a_{ss'} \left[R(s, a, s') + \gamma V(s') \right]. \tag{2.12}$$

It follows then that the Bellman equation can be written as $T^\pi V^\pi = V^\pi$.

The *Bellman error* or *residual Bellman error* B^π with respect to policy π at a state s is given as

$$B^\pi(s) = (T^\pi V)(s) - V(s). \tag{2.13}$$

2.2.1 Linear Function Approximation

In many domains it is not possible to represent the value function exactly due to infinitely large or continuous state spaces. Instead some form of function approximation must be used. Linear function approximation is a popular choice because of the simplicity of its update rules and because its error surface has no local minima [Konidaris *et al.* 2011].

The general form of a linear function approximation is a weighted sum of a set of basis functions $\Phi = [\phi_1(s), \phi_2(s), .., \phi_n(s)]$ with weights $\boldsymbol{w} = [w_1, w_2, ..., w_n]^T$.

$$\begin{aligned} V(s) &= w_1\phi_1(s) + w_2\phi_2(s) + ... + w_n\phi_n(s) \\ &= \sum_{i=1}^{m} w_i \phi_i(s) \\ &= \boldsymbol{\Phi}(s)\boldsymbol{w}, \end{aligned} \tag{2.14}$$

where each $\phi_i(s) \in \mathbb{R}$ is a real valued function of the state. Weight values w_i are chosen and updated by the reinforcement learning algorithm in such a way that the true value function is approximated by the weighted sum of basis functions at each state.

In the case of a finite number of discrete states, value functions for states can be stored in tabular form. An alternate form is that of a linear function approximation with a basis function for each state such that,

$$\phi_i(s) = \begin{cases} 1 \text{ if } s = s_i \\ 0 \text{ otherwise.} \end{cases} \tag{2.15}$$

Hence, representing the value function as a weighted sum of basis functions has the practical benefit that any algorithms designed for use with linear function approximation automatically work for domains where a tabular representation is preferred.

2.2.2 Basis Functions

When using linear function approximation, the choice of basis functions can have a huge impact on performance. Among the options available, fixed bases – predefined sets of basis functions of size determined by the number of dimensions and an order parameter – are popular with practitioners due to their low number of parameters required and relative ease of use. The challenges associated with fixed bases come in the form of trial and error experimentation on the task domain, with the practitioner testing different basis sets and their parameters until satisfactory performance is achieved. These tests are time consuming and often impractical for real world problems.

We now described some of the popular basis function schemes. We use a synthetic function (shown in Figure 2.1) and approximate it with each type of basis discussed.

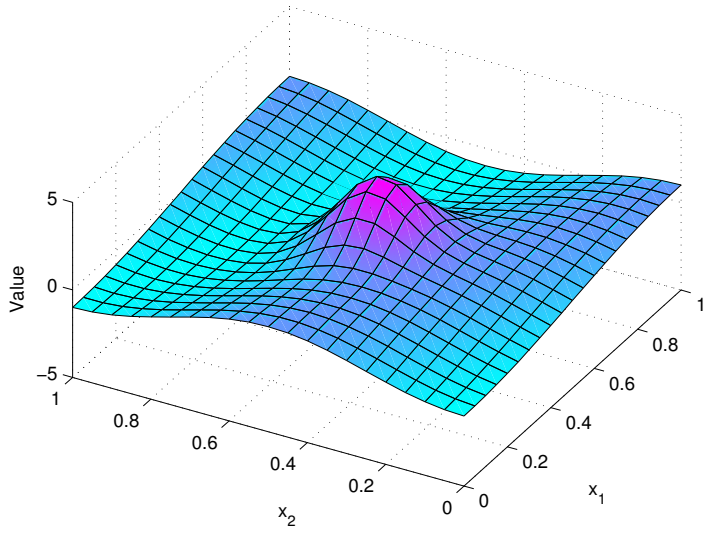


Figure 2.1: Example function to approximate: $-cos(\pi(x_1 + 2x_2)) + \frac{1}{\sqrt{0.02\pi}} e^{\frac{-((0.5-x_1)^2+(0.5-x_2)^2)}{0.02}}$ made up of both a Fourier basis function and a radial basis function (RBF).

2.2.2.1 Tilings

Tilings or CMACs are a set of basis functions which discretise the state space into a set of tiles, rectangular d-dimensional regions. We only consider the case where the tiles do not overlap in this work, although tiling schemes where tiles exist in independent dimensions, as well as schemes where tilings of different grid sizes are overlayed, are common.

Each tile is a basis function which outputs a value of 1 when the state falls within that tile's rectangular region and 0 otherwise. A tile may contain one (where every tile only contains one state it is also known as a tabular basis) and up to infinitely many states. When a tile has value 1 (is active), no other tiles are active (non-overlapping tiles), and hence the value at a state is determined solely by this tile's weight value, $V(s) = w_i\phi_i(s)$ where $\phi_i(s) = 1$ and $\phi_j(s) = 0\ \forall j \neq i$. This means that states that activate a particular tile have the same value. Each basis function $\phi_i(s)$ is given by:

$$\begin{aligned}\phi_i(s) = 1 \text{ if } \forall j \in [0, ..., d], &\begin{cases} x_j \in \left(\frac{c_{i,j}}{n+1}, \frac{c_{i,j}+1}{n+1}\right], c_{i,j} \neq 0 \\ x_j \in \left[\frac{c_{i,j}}{n+1}, \frac{c_{i,j}+1}{n+1}\right], c_{i,j} = 0 \end{cases} \\ = 0 \text{ otherwise,}\end{aligned} \tag{2.16}$$

where $\mathbf{c}_i = [c_{i,1}, ..., c_{i,d}], c_{i,j} \in [0, ..., n]$ is a parameter vector specifying the bounds of the tile, d is the number of dimensions in the state space, and n is the order of approximation, giving $(n+1)$ tiles per dimension for a total of $(n+1)^d$ tiles.

Tilings have the advantage that the value at any state can be calculated in O(1) time and similarly updated in time independent of the total number of tiles. The resulting value function however is piecewise constant, and a trade-off must be made between the granularity of the approximation and the time taken to learn the value of each tile, as well as other practical concerns such as the memory required to store each tile's value.

Despite the shortcoming of tilings, they are immensely popular among RL practitioners due to their ease of use and respectable results. An example of the tiling basis is shown in 2.2 where the function in Figure 2.1 is approximated by a 3x3 tiling.

2.2.2.2 Radial Basis Functions

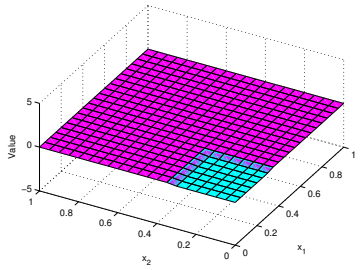
A radial basis function (RBF) is a Gaussian with variance $\sigma^2 > 0 \in \mathbb{R}$ and center $b_i \in \mathbb{R}^d$ denoted by $\phi_i(s) = \frac{1}{\sigma^{d/2}\pi^{d/4}} e^{-||b_i - s||^2/2\sigma^2}$, where each b_{ij} gives the ith RBF's center position in dimension $j \in [1, ..., d]$. Typically, for an order n RBF approximation, $n+1$ RBFs are tiled evenly across each dimension with an additional constant function for a total of $(n+1)^d+1$ basis functions. The RBF centers b_i in each dimension are further parametrized, with $b_{ij} = (c_{ij})/n$, where $c_i = [c_{i1}, c_{i2}, ..., c_{id}], c_{ij} \in [0, ..., n]$. The entire tiling can be obtained by varying the c parameter vector, and while σ can vary, we use $\sigma^2 = 1/(2(n+1)^2 - 4(n+1)^2 + 2(n+1))$ as described in Benoudjit and Verleysen [2003].

For example, a 2 dimensional state space with a 2nd order RBF basis would contain the following terms:

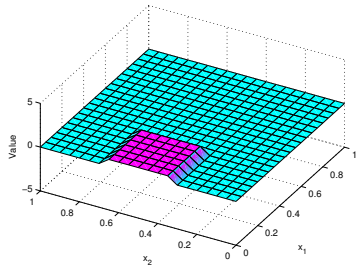
$$\mathbf{\Phi} = \begin{pmatrix} \frac{1}{\sqrt{4\pi}} e^{\frac{-((0-x_1)^2+(0-x_2)^2)}{4}} \\ \frac{1}{\sqrt{4\pi}} e^{\frac{-((1-x_1)^2+(0-x_2)^2)}{4}} \\ \frac{1}{\sqrt{4\pi}} e^{\frac{-((0-x_1)^2+(1-x_2)^2)}{4}} \\ \frac{1}{\sqrt{4\pi}} e^{\frac{-((1-x_1)^2+(1-x_2)^2)}{4}} \end{pmatrix}. \tag{2.17}$$

A constant basis function $\phi_0 = 1$ may also be added to the approximation. Adding this term is not always necessary, but can be beneficial.

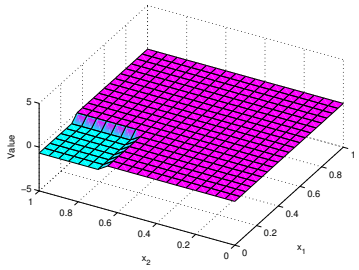
Both the $\boldsymbol{c}$ and σ^2 parameters can be varied, where smaller σ^2 could be used with centres $\boldsymbol{c}$ that correspond to areas of the state space where the value function is less smooth, giving greater resolution in those areas. Prior knowledge about the domain or its value function might lead the practitioner to make such decisions about the distribution of basis functions to achieve better performance. In practice however, the value function is not known a priori which means optimisations at this level are best suited to the learning algorithm itself. Blindly increasing the resolution in areas of the state space could reduce the performance of the algorithm by decreasing the generalisation between states.



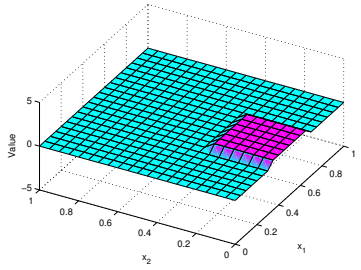
(a) $\phi(s) = 1$ if $x_1 \in [0, 0.33]$ and $x_2 \in [0, 0.33]$, 0 otherwise. $w = -0.12$



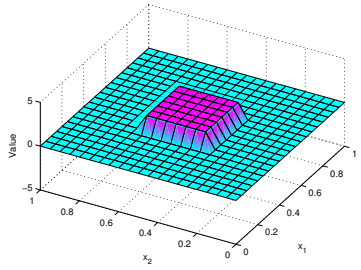
(b) $\phi(s) = 1$ if $x_1 \in [0, 0.33]$ and $x_2 \in (0.33, 0.66]$, 0 otherwise. $w = 0.76$



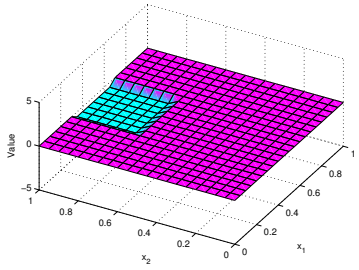
(c) $\phi(s) = 1$ if $x_1 \in [0, 0.33]$ and $x_2 \in (0.66, 1]$, 0 otherwise. $w = -0.69$



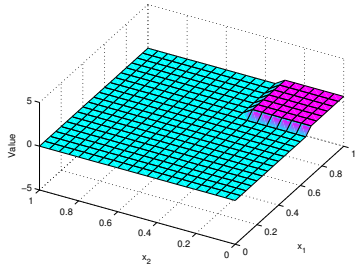
(d) $\phi(s) = 1$ if $x_1 \in (0.33, 0.66]$ and $x_2 \in [0, 0.33]$, 0 otherwise. $w = 0.7$



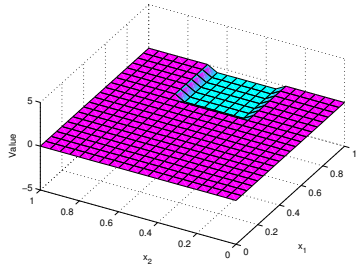
(e) $\phi(s) = 1$ if $x_1, x_2 \in (0.33, 0.66]$, 0 otherwise. $w = 1.74$



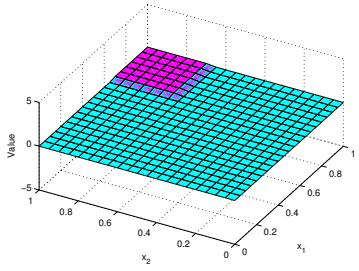
(f) $\phi(s) = 1$ if $x_1 \in (0.33, 0.66]$ and $x_2 \in (0.66, 1]$, 0 otherwise. $w = -0.55$



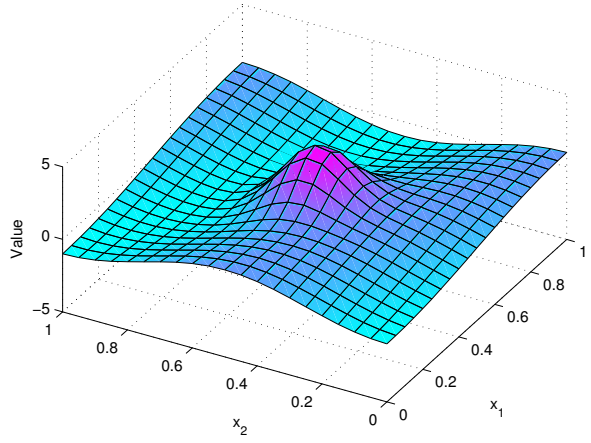
(g) $\phi(s) = 1$ if $x_1 \in (0.66, 1]$ and $x_2 \in [0, 0.33]$, 0 otherwise. $w = 0.69$



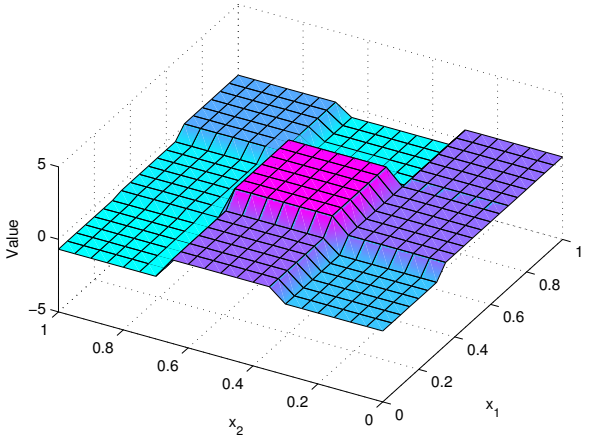
(h) $\phi(s) = 1$ if $x_1 \in (0.66, 1]$ and $x_2 \in (0.33, 0.66]$, 0 otherwise. $w = -0.62$



(i) $\phi(s) = 1$ if $x_1 \in (0.66, 1]$ and $x_2 \in (0.66, 1]$, 0 otherwise. $w = 0.12$



(j) Original function in Figure 2.1 for comparison.



(k) The complete tiling approximation which is the weighted sum of ϕ's in Figures a to i.

Figure 2.2: A 3x3 tiling approximation of the function in Figure 2.1

Generalisation between states can greatly enhance performance, which is a key side benefit of function approximation. RBFs are able to generalise between states locally which gives them an advantage over tilings which have limited generalisation capabilities. Tilings only generalise between states that fall in one tile, with each state getting the same value, and sharing none of the information learned in neighbouring tiles. Whilst tilings generally do not overlap, RBFs do, which often gives the learning algorithm a starting point which could accelerate learning. The effective overlap is not very large as most of the power of the RBF lies at its centre, but it does mean that RBFs are able to generalise between states where tilings would not. The limited range of generalisation often leads to relatively slow initial performance when compared to basis functions with a larger scope [Konidaris *et al.* 2011].

RBFs have some advantages over tilings such as their smoothness and generalisation capabilities, but computational cost is not one of them. Not only is each individual function comprised of complex operations, but every RBF has a non-zero value everywhere in the state space which means that to calculate the state value at a state, the value of every single RBF basis function must be computed. In many cases the cost can be reduced since the value of many basis functions will be close to 0 at a state and may be approximated by 0, however, the cost is much greater than that of tilings even with these optimisations.

An example of RBF approximation is shown in Figure 2.3 where the function in Figure 2.1 is approximated by 9 RBFs. The RBFs form an evenly spaced grid with $n = 2$ giving 3 RBFs per dimension.

2.2.2.3 Fourier Basis Functions

The Fourier basis [Konidaris *et al.* 2011] is a simple yet reliable basis which requires the choice of only one parameter, the order of approximation. For a given order n with d state variables, each basis function ϕ_i is of the following form:

$$\phi_i(s) = cos(\pi \boldsymbol{c}_i \cdot s), \tag{2.18}$$

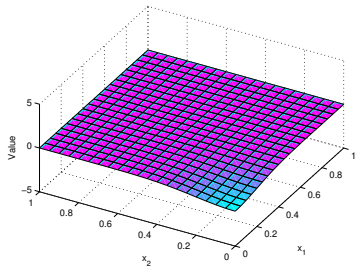
where $\boldsymbol{c}_i$ is a vector of d integer elements $\boldsymbol{c}_i = [c_{i,1}, ..., c_{i,d}], c_{i,j} \in [0, ..., n]$. Each basis function is completely determined by its $\boldsymbol{c}$ and the entire set of basis functions is obtained by creating a basis function for every valid construction of $\boldsymbol{c}$. This gives a total of $(n+1)^d$ basis functions for a given order of n.

For example, a 2 dimensional state space with a 2nd order Fourier basis would contain the following terms:

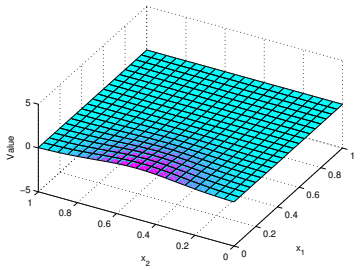
$$\boldsymbol{\Phi} = \begin{pmatrix} 1 \\ cos(\pi x_1) \\ cos(\pi x_2) \\ cos(\pi x_1 + \pi x_2) \\ cos(2\pi x_1) \\ cos(2\pi x_2) \\ cos(2\pi x_1 + \pi x_2) \\ cos(\pi x_1 + 2\pi x_2) \\ cos(2\pi x_1 + 2\pi x_2) \end{pmatrix}. \tag{2.19}$$

Unlike tilings and RBFs, the Fourier basis is a truly global basis. Every Fourier basis function must be evaluated at each state, multiplied by its weight and summed to get the value at that state. This allows for far greater generalisation than that of tilings and RBFs. Another advantage is the computational speed of calculating the value at a specific state. While not as computationally efficient as tilings, the Fourier basis is relatively easy to calculate when compared to RBFs and comes with the advantage that often less basis terms are needed because they generalise better [Konidaris *et al.* 2011].

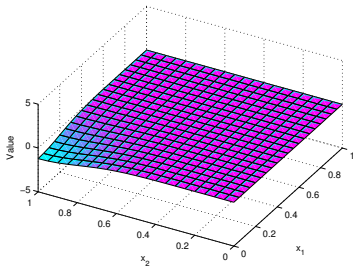
The Fourier basis has been shown to work well on a large number of benchmark domains which makes it a good choice for domains in which little is known before learning commences [Konidaris *et al.* 2011].



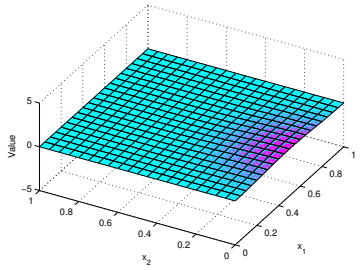
(a) $c = [0, 0]$ and $\sigma^2 = 0.042$. $w = -0.52$



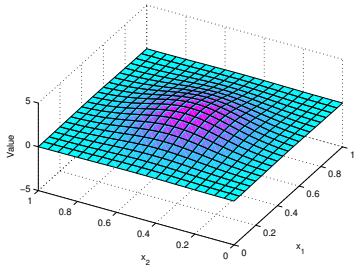
(b) $c = [0, 0.5]$ and $\sigma^2 = 0.042$. $w = 0.41$



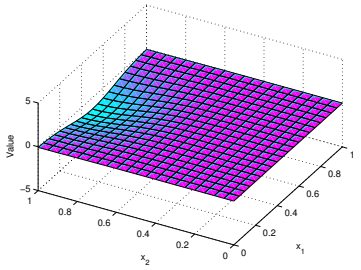
(c) $c = [0, 1]$ and $\sigma^2 = 0.042$. $w = -0.63$



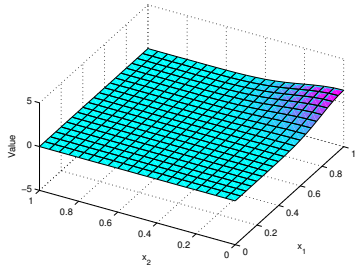
(d) $c = [0.5, 0]$ and $\sigma^2 = 0.042$. $w = 0.19$



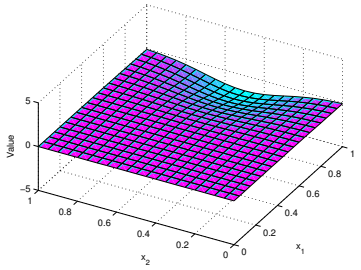
(e) $c = [0.5, 0.5]$ and $\sigma^2 = 0.042$. $w = 0.99$



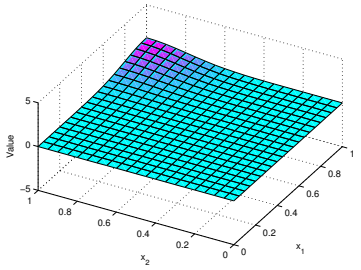
(f) $c = [0.5, 1]$ and $\sigma^2 = 0.042$. $w = -0.67$



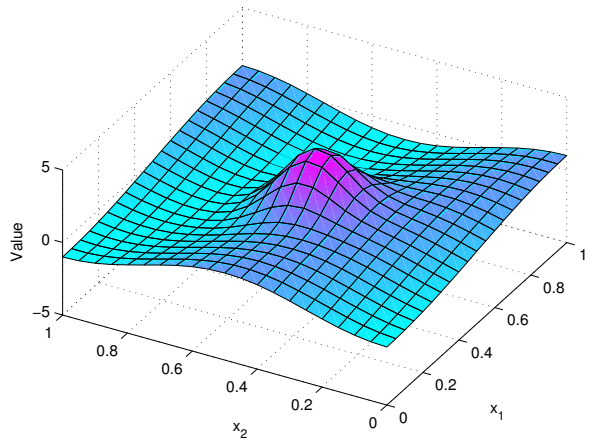
(g) $c = [1, 0]$ and $\sigma^2 = 0.042$. $w = 0.74$



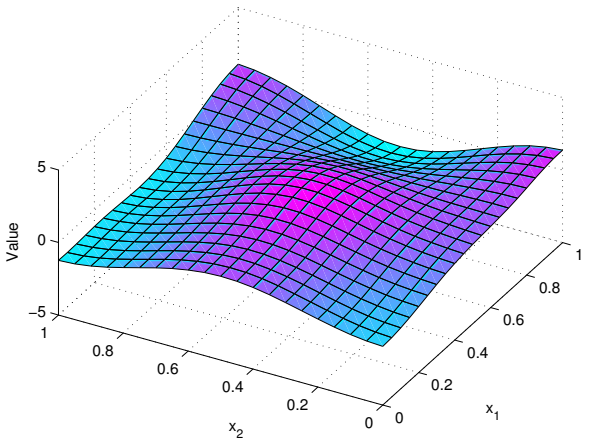
(h) $c = [1, 0.5]$ and $\sigma^2 = 0.042$. $w = -0.89$



(i) $c = [1, 1]$ and $\sigma^2 = 0.042$. $w = 0.64$



(j) Original function in Figure 2.1 for comparison.



(k) The complete RBF approximation which is the weighted sum of ϕ's in Figures a to i.

Figure 2.3: An order 3 approximation of the function in Figure 2.1 with RBFs.

Empirical results on many domains show good learning rates and policy convergence in an online setting. This could be because low-order basis terms represent the overall shape of the value function early on, while high-order terms fill in the details later. To enhance this behaviour in practice, learning rates on each basis function are often scaled with a scaling parameter, $\alpha_i = \alpha_1 / ||\boldsymbol{c}_i||_2$ for basis function ϕ_i, where $\alpha_1 = 1$ corresponds to the constant term. Low order terms have higher learning rates thereby learning the general shape of the value function first, after which high order terms add detail.

An example of Fourier basis function approximation is shown in Figure 2.4 where the function in Figure 2.1 is approximated by an order 2 Fourier basis set.

2.2.2.4 Polynomial Basis Functions

Given a state s with d state variables (d dimensions), $s = [x_1, ..., x_d]$ and an order n, each polynomial basis function is defined as:

$$\phi_i(s) = x_1^{c_{i,1}} \times x_2^{c_{i,2}} \times ... \times x_d^{c_{i,d}} = \prod_{j=1}^{d} x_j^{c_{i,j}}, \tag{2.20}$$

where $\boldsymbol{c}_i$ is a vector of d integer elements $\boldsymbol{c}_i = [c_{i,1}, ..., c_{i,d}]$ and $c_{i,j} \in [0, ..., n]$.

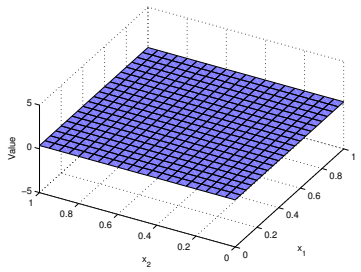
For example, a 2 dimensional state space with a 2nd order polynomial basis would contain the following terms:

$$\boldsymbol{\Phi} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ x_1 x_2 \\ x_1^2 \\ x_2^2 \\ x_1^2 x_2 \\ x_1 x_2^2 \\ x_1^2 x_2^2 \end{pmatrix}. \tag{2.21}$$

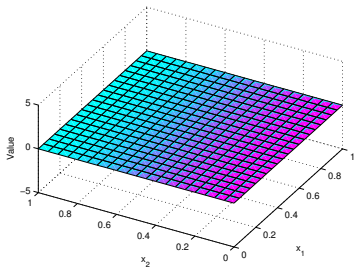
The polynomial basis is an easy to understand yet problematic basis for reinforcement learning. It is a global basis like the Fourier basis, but has unconstrained output where the Fourier basis' output is constrained on $[-1, 1]$. Even when the state variables lie between 0 and 1, restricting the output of the polynomial basis functions to lie between 0 and 1, the basis remains problematic due to its highly volatile nature. Relatively large and frequent swings in the value of each basis function lead to large coefficients in the function approximation. This can be a big problem, especially in online learning applications where new states are encountered regularly. These new states will often start with extremely high or low values which are a poor approximation to the true value, and will influence the behaviour of the learning agent significantly. The coefficients do not uniformly converge to stable values resulting in a value function that is in constant flux. To counteract this, learning rates have to be small, resulting in poor performance.

The polynomial basis is also difficult to compute accurately and efficiently. The exponent terms in each basis function are expensive to compute, and for high orders of approximation, computational accuracy becomes a problem. These factors make the polynomial basis a poor choice for function approximation in RL, though it is still often used for demonstrations due to its simplicity.

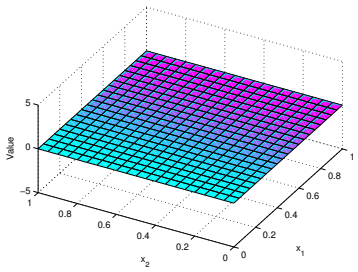
An example of polynomial function approximation is shown in Figure 2.5 where the function in Figure 2.1 is approximated by an order 2 polynomial basis set.



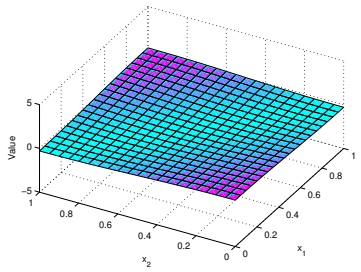
(a) $c = [0, 0]$. $w = 0.38$



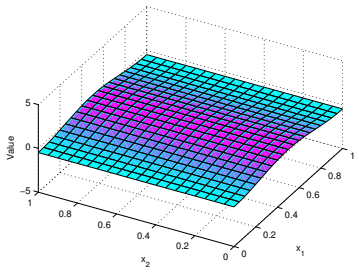
(b) $c = [0, 1]$. $w = 0$



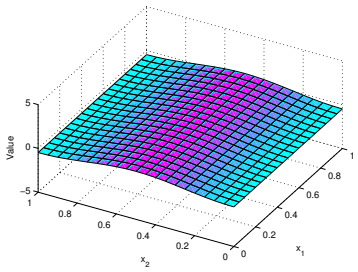
(c) $c = [1, 0]$. $w = 0$



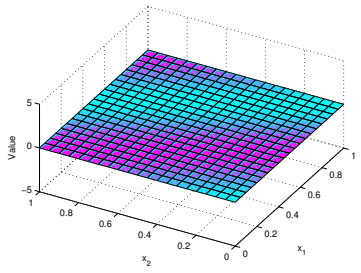
(d) $c = [1, 1]$. $w = 0.29$



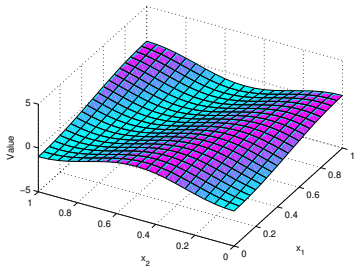
(e) $c = [2, 0]$. $w = -0.48$



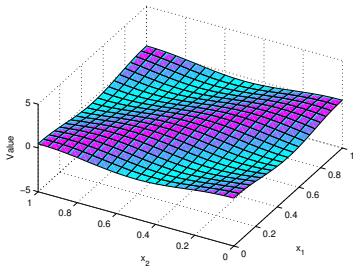
(f) $c = [0, 2]$. $w = -0.48$



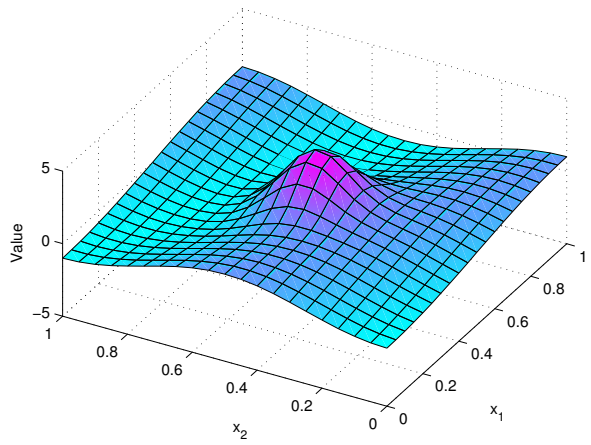
(g) $c = [2, 1]$. $w = 0$



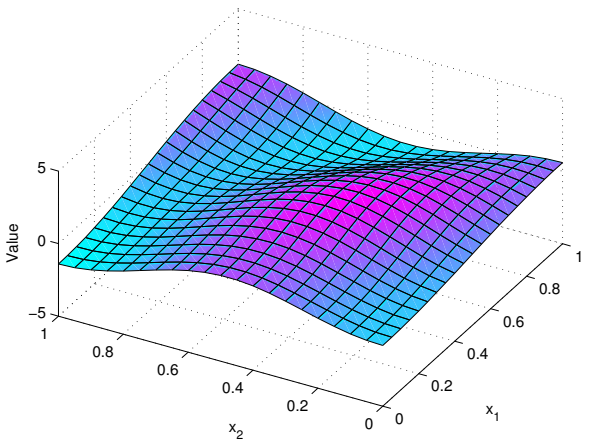
(h) $c = [1, 2]$. $w = -1$



(i) $c = [2, 2]$. $w = 0.47$

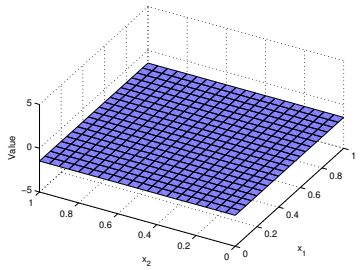


(j) Original function in Figure 2.1 for comparison.

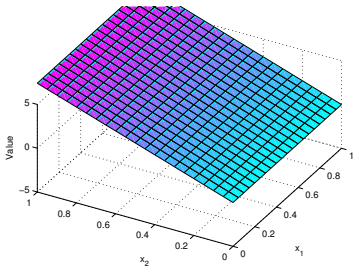


(k) The complete Fourier approximation which is the weighted sum of ϕ's in Figures a to i.

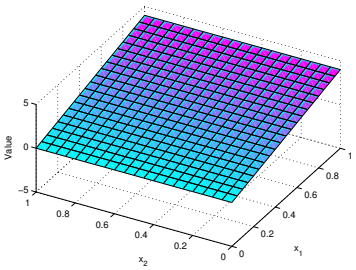
Figure 2.4: An order 2 approximation of the function in Figure 2.1 with Fourier basis functions.



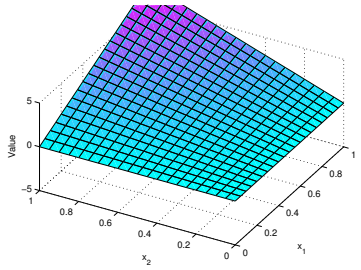
(a) $c = [0, 0]$. $w = -1.47$



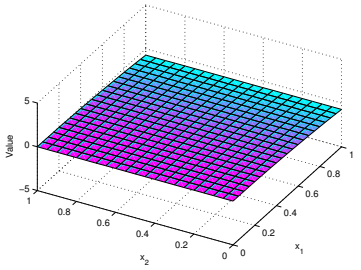
(b) $c = [0, 1]$. $w = 7.43$



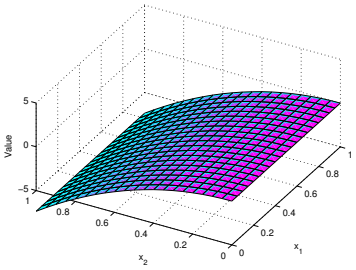
(c) $c = [1, 0]$. $w = 3.99$



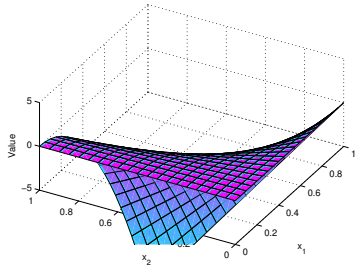
(d) $c = [1, 1]$. $w = 7.43$



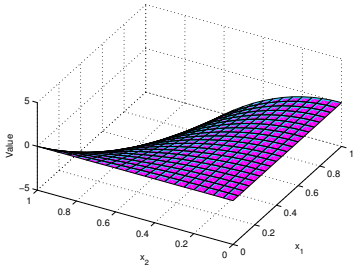
(e) $c = [2, 0]$. $w = -0.72$



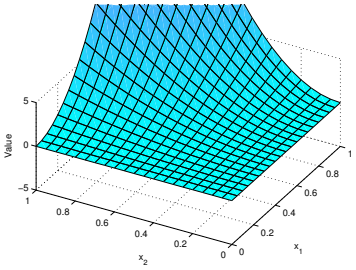
(f) $c = [0, 2]$. $w = -7.37$



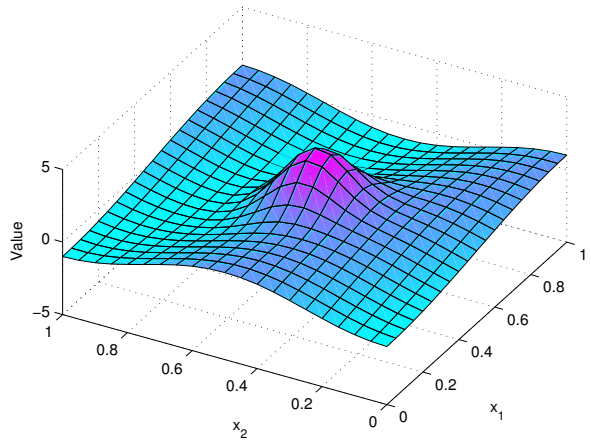
(g) $c = [2, 1]$. $w = -27.37$



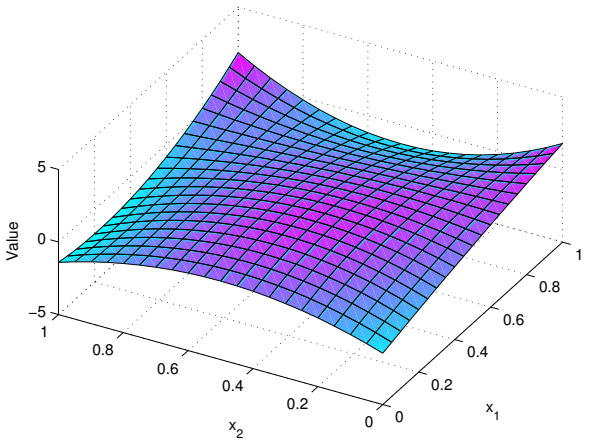
(h) $c = [1, 2]$. $w = -14.08$



(i) $c = [2, 2]$. $w = 34.02$



(j) Original function in Figure 2.1 for comparison.



(k) The complete polynomial approximation which is the weighted sum of ϕ's in Figures a to i. Notice the high w values for the basis functions.

Figure 2.5: An order 2 approximation of the function in Figure 2.1 with polynomial basis functions.

2.2.3 Learning Algorithms

2.2.3.1 Temporal Difference Methods

Given a sample trajectory consisting of states s_t, sampled at fixed time intervals t, temporal difference (TD) methods aim to minimise the temporal difference error δ_t in the value function V for each state s_t, where δ_t is defined as:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t), \tag{2.22}$$

and in particular, when V is approximated with a linear function approximation, as:

$$\delta_t = r_{t+1} + \gamma \Phi(s_{t+1})\boldsymbol{w} - \Phi(s_t)\boldsymbol{w}. \tag{2.23}$$

The TD error is the sampled version of the Bellman error in Equation (2.13), and hence its expected value is the Bellman error.

TD methods attempt to learn the expected return at a state from samples, using estimated values of the value function at the current (s_t) and next (s_{t+1}) states in the sample. The simplest TD algorithm is TD(0) which uses the following update rule for each sample in an episode:

$$\begin{aligned} V(s_t) &= V(s_t) + \alpha\delta_t \\ \boldsymbol{w} &= \boldsymbol{w} + \alpha\delta_t \Phi(s_t)^T, \end{aligned} \tag{2.24}$$

where $\alpha \in (0, 1]$ is a learning rate parameter.

2.2.3.2 Sarsa

Sarsa is a TD algorithm for action-value (Q) learning. It gets its name from the 5-tuple of events $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ that occur during a transition from one state and action pair to another [Sutton and Barto 1998]. The update rule for Sarsa is similar to that of TD(0) with δ_t defined as:

$$\begin{aligned} \delta_t &= r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \\ &= r_{t+1} + \gamma \Phi(s_t, a_t)\boldsymbol{w} - \Phi(s_t, a_t)\boldsymbol{w}, \end{aligned} \tag{2.25}$$

and the Sarsa update rule as:

$$\begin{aligned} Q(s_t, a_t) &= Q(s_t, a_t) + \alpha\delta_t \\ \boldsymbol{w} &= \boldsymbol{w} + \alpha\delta\Phi(s_t, a_t)^T. \end{aligned} \tag{2.26}$$

At each step, the algorithm chooses an action using its policy π. Usually π is dependent on the approximate values of each action. For example, the $\epsilon-$greedy policy chooses the best action with a certain probability, and a random action otherwise. In this case, the value of taking each action is evaluated using the Q function at the current state s_t. The policy then selects the action that results in the highest value with a certain probability, else it selects a random action. Once an action a_t has been selected and performed, the environment returns the reward r_{t+1} received as well as the following state s_{t+1}. The algorithm then decides what action a_{t+1} it will take next using the same policy, and using this tuple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, an update to the Q function is performed using Equation (2.26).

The algorithm is shown in Algorithm 1.

2.2.3.3 Sarsa(λ)

Sarsa(λ) [Sutton and Barto 1998] is a well understood online reinforcement learning algorithm that is widely used because of its linear time complexity and good empirical results in a variety of domains.

Algorithm 1 Sarsa

Input:
$Q(s, a)$ initialised arbitrarily,
Policy $\pi(s, a)$,
$\alpha \in (0, 1]$,
$\gamma \in [0, 1]$
Output:
$Q(s, a)$.

$s \leftarrow$ environment initialised state
$a \leftarrow$ action selected with policy π at state s
repeat
    Take action a, observe reward r and state s'
    $a' \leftarrow$ action selected with policy π at state s'
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
    $Q(s, a) \leftarrow Q(s, a) + \alpha\delta$
    $s \leftarrow s'$; $a \leftarrow a'$
until s is a terminal state

The intuition behind Sarsa(λ) is that states visited on the way to the current state deserve some of the reward for getting to it. Sarsa(λ) assigns each state visited an importance value based on how long ago that state was visited. This importance value, or eligibility trace, represents how eligible that state is to undergo learning for the currently received reward. The eligibility value for each state decays based on λ, the trace-decay parameter, whenever a state transition takes place and increases when that state is visited. When a new state is visited and a reward received, each previously visited state gets updated by an amount proportional to its eligibility value. States that occurred long ago get updated by a small amount, whereas recently visited states get a larger update. The value of a state is therefore affected by the rewards received in future states, which often speeds up the learning process.

The eligibility trace $\vec{e} \in \mathbb{R}^{1\times n}$ is a real valued vector with the same dimensions as Φ. On each state transition from a state s the eligibility trace is updated with the following formula (we only show the linear function approximation case since it can easily be presented as a set of update rules):

$$\vec{e}_t = \gamma\lambda\vec{e}_{t-1} + \Phi(s_t, a_t). \tag{2.27}$$

The update for Q's parameter vector $\boldsymbol{w}$ then becomes:

$$\boldsymbol{w} = \boldsymbol{w} + \alpha\delta_t(\vec{e}_t)^T, \tag{2.28}$$

where $Q(s, a) = \Phi(s, a)\boldsymbol{w}$.

2.2.3.4 Least Squares Temporal Difference

Least Squares Temporal Difference (LSTD) [Bradtke and Barto 1996] is an algorithm for solving for the set of weights $\boldsymbol{w} = [w_1, ..., w_n]^T$ corresponding to a set of basis functions $\Phi = [\phi_1, ..., \phi_n]$ which minimises the projected Bellman error given a set of samples, $\vec{s}, \vec{r}, \vec{s'}$, where $\vec{s} = [s_1, ..., s_m]^T$ is a vector of start states, $\vec{s'} = [s'_1, ..., s'_m]^T$ is a vector of corresponding resulting states, and $\vec{r} = [r_1, ..., r_m]^T$ is a vector of rewards, where r_i is the reward received upon arriving at state s'_i from state s_i. $\gamma \in [0, 1)$ is the discount factor.

We define the following:

$$\phi(\vec{s}) = \begin{bmatrix} \phi(s_1) \\ \phi(s_2) \\ . \\ . \\ . \\ \phi(s_m) \end{bmatrix} \in \mathbb{R}^m \tag{2.29}$$

$$\Phi(s) = [\phi_1(s), \phi_2(s), ..., \phi_n(s)] \in \mathbb{R}^{1\times n}, \tag{2.30}$$

and

$$\Phi(\vec{s}) = [\phi_1(\vec{s}), \phi_2(\vec{s}), ..., \phi_n(\vec{s})] \in \mathbb{R}^{m\times n}. \tag{2.31}$$

Then the goal is to find a $\boldsymbol{w}$ which minimises the projected Bellman error using projection matrix $\Pi = \Phi(s)(\Phi(s)^T\Phi(s))^{-1}\Phi(s)^T$ which projects it onto the space spanned by $\Phi(s)$:

$$\begin{aligned}
\bar{V} &= \Pi T(\bar{V}) \\
\Phi(s)\boldsymbol{w} &= \Pi(\vec{r} + \gamma\Phi(s')\boldsymbol{w}) \\
\Phi(s)\boldsymbol{w} &= \Phi(s)(\Phi(s)^T\Phi(s))^{-1}\Phi(s)^T(\vec{r} + \gamma\Phi(s')\boldsymbol{w}) \\
\boldsymbol{w} &= (\Phi(s)^T\Phi(s))^{-1}\Phi(s)^T(\vec{r} + \gamma\Phi(s')\boldsymbol{w}) \\
(\Phi(s)^T\Phi(s))\boldsymbol{w} &= \Phi(s)^T(\vec{r} + \gamma\Phi(s')\boldsymbol{w}) \\
(\Phi(s)^T\Phi(s))\boldsymbol{w} &= \Phi(s)^T\vec{r} + \gamma\Phi(s)^T\Phi(s')\boldsymbol{w} \\
(\Phi(s)^T\Phi(s))\boldsymbol{w} - \gamma\Phi(s)^T\Phi(s')\boldsymbol{w} &= \Phi(s)^T\vec{r} \\
\Phi(s)^T(\Phi(s) - \gamma\Phi(s'))\boldsymbol{w} &= \Phi(s)^T\vec{r} \\
\boldsymbol{w} &= (\Phi(s)^T(\Phi(s) - \gamma\Phi(s')))^{-1}\Phi(s)^T\vec{r},
\end{aligned} \tag{2.32}$$

where $\bar{V} = \Phi(s)\boldsymbol{w}$ is the approximation of the value function V. The $\boldsymbol{w}$ in Equation (2.32) are the weights which minimise the projected Bellman error.

In the case where L_2 regularisation is required, a small scalar regularisation parameter λ is chosen and added to the diagonal of the A matrix, where $A = \Phi(s)^T(\Phi(s) - \gamma\Phi(s'))$. More precisely, the L_2 regularised solution for $\boldsymbol{w}$ is:

$$\boldsymbol{w} = (\lambda I + \Phi(s)^T(\Phi(s) - \gamma\Phi(s')))^{-1}\Phi(s)^T\vec{r}. \tag{2.33}$$

2.2.3.5 Least Squares Policy Iteration

Least Squares Policy Iteration (LSPI) is an offline algorithm which, given a set of samples consisting of state transitions and rewards received, attempts to calculate an optimal value function [Lagoudakis and Parr 2003]. The key features of the algorithm are that it is off-policy, model free and makes efficient use of samples. These features mean that samples can be collected from an arbitrary source, allowing it to learn an estimated optimal value function without a model [Lagoudakis and Parr 2003].

LSPI is free from additional parameters such as learning rates which often require a great amount of tuning before good results can be obtained. There are some disadvantages to LSPI however. The algorithm is offline unlike Sarsa(λ), which means samples must be collected first before learning can take place [Lagoudakis and Parr 2003]. Another disadvantage is that LSPI is not well suited to non-stationary environments (environments with changing value functions). Multi-agent domains are one example where this is relevant. Agents can learn to expect other agents to behave in a certain manner, but as the other agents are learning, their behaviour is also changing.

LSPI and LSTD are both batch algorithms that use different methods to solve for an approximate solution to the Bellman equation. They are the least-squares counterparts to the online temporal difference

algorithms which learn using stochastic gradient descent. Unlike LSTD which is used to learn a value function for a fixed policy, LSPI aims to learn the value function for the optimal policy. In this work we do not tackle the problem of learning an optimal value function, and instead focus on the problem of policy evaluation, and hence use LSTD. LSPI could be used in the case where learning the optimal value function is required, however it can be unstable and often requires many more samples than LSTD.

2.3 Feature Selection

When using linear function approximation on high-dimensional domains with a fixed basis, the number of basis terms increases rapidly as the number of state variables grows, making computation of these basis terms infeasible. In order to use this approximation method in practice, a subset of these basis functions must be selected. The problem of which functions to select and how to select them is a major challenge in reinforcement learning. Many current techniques are able to choose good subsets of basis functions in low-dimensional domains, but due to practical concerns, become infeasible as the dimensionality increases. Some of these methods are described in the following subsections.

2.3.1 Matching Pursuit

Matching pursuit (MP) was first introduced by Mallat and Zhang [1993] where it was used in the context of signal processing. The machine learning perspective and extensions to the basic algorithm are described in Vincent and Bengio [2002]. Matching pursuit attempts to find a set of basis functions and associated weights to approximate a function given samples of it using linear function approximation. It keeps two sets of basis functions, those that are in the approximation and those that are candidates for selection to be added to the approximation. The candidates reside in a dictionary where they are selected iteratively and added to the approximation set with associated weights.

2.3.1.1 Basic Matching Pursuit

Given a set of input samples $\vec{x} = [x_1, x_2, ..., x_k]^T$, $x_i \in \mathbb{R}^d$, and corresponding observations $\vec{y} = [y_1, y_2, ..., y_k]^T$, $y_i \in \mathbb{R}$, of an unknown function $f(x)$, the goal of the matching pursuit algorithm is to build up an approximation of this function of the form

$$\begin{aligned} f_N(x) &= \sum_{n=1}^{N} \alpha_n \phi_n(x) \\ &= \Phi(x)\boldsymbol{w}, \end{aligned} \tag{2.34}$$

where $\boldsymbol{w} = [\alpha_1, \alpha_2, ..., \alpha_N]^T$ is the column vector of weights associated with the row vector of basis functions $\Phi(x) = [\phi_1(x), \phi_2(x), ..., \phi_N(x)]$. It does this by selecting functions $\phi_i(x)$ from a dictionary D, and finding the associated weights α_i in an incremental fashion. The residual error of the function is defined as the difference between the current approximation and the output observations:

$$\vec{R} = \vec{y} - \Phi(\vec{x})\boldsymbol{w}. \tag{2.35}$$

The basic matching pursuit algorithm repeatedly chooses basis functions from a dictionary to add to the function approximation, each time choosing the basis function which reduces the residual the most using least squares error [Vincent and Bengio 2002]. At the nth step, the residual $\vec{R}_{n-1}$ is reduced by choosing a basis function $\phi_n(x)$ from the dictionary D and an associated weight α_n such that

$$\vec{R}_n = \vec{R}_{n-1} - \alpha_n \phi_n(\vec{x}), \tag{2.36}$$

where $\vec{R}_0 = \vec{y}$. The function $\phi_n(x)$ is chosen in such a way as to minimise the error in the function approximation in a least squares sense. That is, $||\vec{R}_n||^2$ should be minimised:

$$(\phi_n, \alpha_n) = \arg \min_{\phi \in D, \alpha \in \mathbb{R}} ||\vec{R}_{n-1} - \alpha\phi(\vec{x})||^2. \tag{2.37}$$

In general it is not possible to find an optimal set of functions $\Phi \subset D$ and associated weights $\boldsymbol{w}$ using this greedy approach of iteratively selecting the the best function $\phi(x)$ and the corresponding best α. However when the $\phi(x)$ in D are all mutually orthogonal, the greedy approach becomes equivalent to matching pursuit with pre-fitting which is a variant which maintains optimal α values for each selected ϕ_i as new ϕs are added.

To find the optimal α for a given $\phi(x)$, we have

$$\begin{aligned} 0 &= \frac{\partial ||\vec{R}_{n-1} - \alpha\phi(\vec{x})||^2}{\partial \alpha} \\ 0 &= -2\langle \vec{R}_{n-1}, \phi(\vec{x}) \rangle + 2\alpha ||\phi(\vec{x})||^2 \\ \alpha &= \frac{\langle \vec{R}_{n-1}, \phi(\vec{x}) \rangle}{||\phi(\vec{x})||^2}, \end{aligned} \tag{2.38}$$

where the inner product is defined as $\langle \vec{x}, \vec{y} \rangle = \sum_i x_i y_i$ and the norm as $||\vec{x}||^2 = \langle \vec{x}, \vec{x} \rangle$.

Substituting the optimal α in, the optimal $\phi(x)$ is the one that minimises the following equation:

$$\begin{aligned} ||\vec{R}_{n-1} - \alpha\phi(\vec{x})||^2 &= \left|\left| \vec{R}_{n-1} - \frac{\langle \vec{R}_{n-1}, \phi(\vec{x}) \rangle}{||\phi(\vec{x})||^2} \phi(\vec{x}) \right|\right|^2 \\ &= ||\vec{R}_{n-1}||^2 - \left(\frac{\langle \vec{R}_{n-1}, \phi(\vec{x}) \rangle}{||\phi(\vec{x})||} \right)^2, \end{aligned} \tag{2.39}$$

which can be minimised by maximising $\rho = \left| \frac{\langle \vec{R}_{n-1}, \phi(\vec{x}) \rangle}{||\phi(\vec{x})||} \right|$. This results in a linear search through the dictionary functions to find which one gives the maximum ρ. ρ can also be viewed as the correlation between the basis function and the residual, with a higher correlation leading to a higher reduction in the least squares error.

2.3.2 OMP-TD

In the RL setting, MP can be used to select basis functions for value function approximation. In the RL setting however, the target function (the value function) is unknown, so MP cannot be used as is. One variation of MP applied to the RL setting, orthogonal matching pursuit with temporal differences (OMP-TD), uses samples of the Bellman error as the residual, $\vec{R}_{k+1} = \boldsymbol{r} + \gamma\Phi(\boldsymbol{s}')\boldsymbol{w} - \Phi(\boldsymbol{s})\boldsymbol{w} = \vec{B}_{k+1}$ and maintains optimal weight values $w_1, ..., w_k$ using LSTD after each basis function is added. At each iteration, the residual is recalculated and a basis function selected from the dictionary which has maximum correlation

$$\rho = \left| \frac{\langle \vec{B}_{n-1}, \phi(\vec{x}) \rangle}{||\phi(\vec{x})||} \right| \tag{2.40}$$

with the residual as in MP. The orthogonal part of OMP-TD comes from the fact that the weights are recalculated at each iteration.

The OMP-TD algorithm uses the fact that the Bellman error B of a value function V with respect to a policy π gives a rough indication of where the value function V^π lies relative to V. Adding a basis function to the approximation set which is sufficiently correlated to the Bellman error is guaranteed to improve the approximation set. Unfortunately, the Bellman error can only be sampled leading to errors

in the correlation calculation. In addition, there are no guarantees that a better improvement could not be achieved by selecting a different basis function, only that one sufficiently correlated to the Bellman error will lead to an improvement.

OMP-TD compares favourably with competitor methods such as LARS-TD (Section 2.3.3.3) in both computational efficiency and performance [Painter-Wakefield and Parr 2012].

The OMP-TD algorithm is given in Algorithm 2. The dictionary is given as $\Phi = [\phi_1, ..., \phi_k]$, and the selected basis functions are stored as an index set I such that Φ_I is the set of functions in the approximation, and $\Phi_{\bar{I}}$ is the set of functions that remain in the dictionary to be selected. The *Select* function returns a set of indices of basis functions to be added to the function approximation, and in the case of standard OMP-TD simply returns the index of the most correlated basis function:

$$J \leftarrow \{\arg\max_{i \in \bar{I}} \rho_i\}. \tag{2.41}$$

The β parameter is an optional stopping condition which stops basis function selection when the correlation of the most correlated basis function falls below β.

Algorithm 2 OMP-TD

Input:
$\Phi = [\phi_1, ..., \phi_k]$, $\boldsymbol{s} = [s_1, \ldots, s_n]^T$, $\boldsymbol{s'} = [s'_1, \ldots, s'_n]^T$, $\boldsymbol{r} \in \mathbb{R}^n$, $\gamma \in [0, 1)$, $\beta \in \mathbb{R}$,
where $\phi_i(\boldsymbol{s}) = [\phi_i(s_1), \ldots, \phi_i(s_n)]^T$ and $\Phi(\boldsymbol{s}) \in \mathbb{R}^{n \times k} : \Phi(\boldsymbol{s})_{ij} = \phi_j(s_i)$.
Output:
Approximation weights w.

$I \leftarrow \{\}$
$\bar{I} \leftarrow \{1, \ldots, k\}$
$\boldsymbol{w} \leftarrow \mathbf{0}$
repeat
  $\vec{B} \leftarrow \boldsymbol{r} + \gamma\Phi(\boldsymbol{s'})\boldsymbol{w} - \Phi(\boldsymbol{s})\boldsymbol{w}$
  $\rho_i \leftarrow \frac{<\vec{B}, \phi_i(\boldsymbol{s})>}{||\phi_i(\boldsymbol{s})||}$ for all $i \in \bar{I}$
  $J \leftarrow Select(\boldsymbol{\rho}, \bar{I})$
  if $||\boldsymbol{\rho}_J|| > \beta$ **then**
    $I \leftarrow I \cup \{J\}$
    $\bar{I} \leftarrow \bar{I} \backslash \{J\}$
    $\boldsymbol{w}_I \leftarrow (\Phi_I(\boldsymbol{s})^T\Phi_I(\boldsymbol{s}) - \gamma\Phi_I(\boldsymbol{s})^T\Phi_I(\boldsymbol{s'}))^{-1}\Phi_I(\boldsymbol{s})^T\boldsymbol{r}$
  end if
until $||\boldsymbol{\rho}_J|| \leq \beta$ **or** $\bar{I} = \{\}$

The dictionary in OMP-TD can contain any set of basis functions, but since the correlation search in Matching Pursuit is linear in the number of basis functions in the dictionary, the size of the set of basis functions chosen can be a limiting factor. If the set is large enough, which is often the case in high-dimensional domains, even a linear search may be practically infeasible. A common choice for dictionary functions is to use a fixed basis set such as the Fourier basis, or RBFs. These basis sets are simple to use, can approximate a wide range of continuous functions, and the meaning of the selected functions can be investigated more easily than constructed basis functions[1] since the basis sets are generally well understood.

[1]Constructed basis functions are composed of other functions designed either by a practitioner, or by an algorithm. They are usually more complex than functions from fixed basis sets, and hence their role in the resulting approximation can be more difficult to understand.

2.3.3 Related Work

2.3.3.1 Introduction

The related work in this section describes alternative feature selection methods as well as construction methods. We discuss two other approaches for feature selection, incremental feature dependency discovery (iFDD) [Geramifard *et al.* 2011] which identifies interactions between state variables and includes tiles to cater for those interactions, and feature selection through L_1 regularisation Kolter and Ng [2009], which narrows the set of features using L_1 regularisation.

Although we do not tackle basis function construction, feature selection and construction methods are strongly related. We include background on some construction methods such as proto-value functions (PVFs) [Mahadevan 2005], Bellman error basis functions (BEBFs) [Parr *et al.* 2007], and predictive projections [Sprague 2009]. BEBFs in particular are important in this work as the OMP-TD algorithm which we base our work on uses BEBFs as a target for feature selection.

2.3.3.2 Incremental Feature Dependency Discovery

Incremental Feature Dependency Detection (iFDD) [Geramifard *et al.* 2011] is an algorithm for incrementally increasing the representational power of a linear function approximation. The initial set of basis functions consists of binary features, each dependent on one dimension of the state space. When learning, areas of high Bellman error are identified and pairs of these features are combined and added to the basis set.

This allows the learning algorithm to learn quickly as 1-dimensional basis terms are active in many more states than higher dimensional terms. After the initial learning has taken place, more specific terms relying on multiple dimensions are added to increase the representational power of the approximation where it is needed most. The process adds increasingly complex terms to the approximation until learning completes.

iFDD can be used with any online value-based reinforcement learning method that relies on binary features [Geramifard *et al.* 2011]. Most commonly, Sarsa(λ) with CMAC tilings are used. The algorithm begins with an initial set of tiles, $\phi(s, a)$, each of which only cares about one of the dimensions of s. At each step in the chosen learning algorithm, the TD error is obtained,

$$\delta_t = r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_s). \tag{2.42}$$

All active features $\phi(s, a) = 1$ in state s are contributors to the error δ_t and thus their pairwise combinations will also be active in s. A potential value ψ is stored for each pair, and increased $\psi = \psi + |\delta_t|$ each time the pair is active simultaneously. When the potential for a pair reaches a certain user-defined threshold, ξ, the pair is combined to form one basis function and added to the function approximation. The combination function can then be combined with other functions to form more complex functions.

2.3.3.3 Feature Selection through L_1 Regularisation

While Geramifard *et al.* [2011] use a fairly simple method to select new features, the algorithm presented in Kolter and Ng [2009] starts instead with an enriched set of basis functions in the approximation, and focuses on solving the problem of over-fitting with these additional basis functions. Their algorithm, called LARS-TD, uses L_1 regularisation to focus the learned weights on a sparse subset of the initial set of basis terms. The terms given non-zero weights are considered to be selected and could be extracted from the approximation for computational efficiency. In higher-dimensional domains, the size of the basis set grows exponentially with the number of dimensions, making LARS-TD infeasible as the algorithm is linear in the number of basis functions. Like LSPI, LARS-TD is part of the Least Squares

Temporal Difference (LSTD) family of algorithms, meaning it is sample efficient at the cost of being offline and computationally expensive [Kolter and Ng 2009].

Johns *et al.* [2010] also tackle feature selection using L_1 regularisation with their algorithms: LC-MPI and LARQ. Their approach is based on linear complimentary problem (LCP) theory, and offers benefits over LARS-TD such as warm starts (learning can begin with starting points taken from similar problems) and stronger theoretical results [Johns *et al.* 2010]. This method however shares many of the same strengths and weaknesses as LSTD.

2.3.3.4 Proto-value Functions

Proto-value Functions (PVFs) [Mahadevan 2005] are constructed basis functions learned from the topology of the state space as opposed to the rewards experienced. PVFs are based on the idea of constructing a graph of state transitions, and then, using that graph, constructing basis functions which span all possible value functions for that state space [Mahadevan 2005].

2.3.3.5 Bellman Error Basis Functions

Bellman Error Basis Functions (BEBFs) are basis functions derived from the Bellman error of a previous set of basis functions [Parr *et al.* 2007]. After each step of learning the Bellman error is approximated from the sample data and added to the function approximation as a new basis function. The error is then recalculated with this new basis function and the process repeated until the error becomes sufficiently small. The Bellman error is given in Equation (2.13).

BEBFs and in fact any functions with sufficiently high correlation to the Bellman residual are guaranteed to improve the bound on the distance of the fixed point of the current value function $\hat{V}$ and the true value function V^* [Parr *et al.* 2007; Painter-Wakefield and Parr 2012]. BEBFs themselves however are often impractical to construct, and must be approximated.

2.3.3.6 Predictive Projections

The predictive projections algorithm reduces the number of dimensions in domains by attempting to find a low-dimensional state representation that still contains much of the useful information the higher-dimensional original state representation did. The low-dimensional representation is attained by discovering projections which can be used to make accurate predictions of future states [Sprague 2009]. From there, normal reinforcement learning techniques and function approximation can be used as the new projected features are of lower dimensionality than the original features, but still encode many of the important characteristics of the state space. In low-dimensional state spaces, full fixed basis sets can be used without the practical concerns (computation and memory requirements of large basis function sets) of their high dimensional counterparts.

Unfortunately the transform to a low dimensional space requires optimising a non-convex function, and since the transform is linear, the transformed state space may not contain all the information necessary to learn well.

2.4 Conclusion

In reinforcement learning, function approximation is often required to deal with large or continuous state spaces. Linear function approximation is one type of function approximation, however it introduces new problems such as how to select the basis functions or features that the approximation is built from. When there are few dimensions, a naive technique for selecting these features is to use many more than

necessary. Due to computational concerns, this does not work in high dimensions, as the number of features would be too large.

Feature selection in reinforcement learning is an important field of research as domains of interest are often high dimensional. Of the feature selection algorithms available, OMP-TD is simple to use, works relatively well in practice and has a solid theoretical foundation. Unfortunately, in high dimensions, searching a dictionary of basis functions in linear time can be too slow, as the dictionary it searches grows exponentially in the number of dimensions. In the following chapters we investigate solutions to this problem, as well as other issues such as improving the selection metrics of OMP-TD and dealing with the online setting.

Chapter 3

Heuristics for Dictionary Size Reduction

3.1 Introduction

Scaling to high dimensional state spaces is a major challenge in reinforcement learning. In particular, dictionary based feature selection algorithms such as OMP-TD use fixed basis sets in their dictionaries, performing a linear search through the basis functions each time one is added. When the state dimensions are high, fixed basis sets can become too large for even a linear search as the number of basis functions grows exponentially with the number of dimensions.

One way to solve this problem is to restrict the size of the dictionary in some intelligent way that preserves the important basis functions so that they may be selected by the selection algorithm. Since it is generally not possible to determine which basis functions will be important beforehand (the selection process would be redundant if this were the case), a dictionary restriction algorithm would instead have to add a set of promising functions to the dictionary. A larger set would have more chance of containing the most important functions but would also come with increased computational costs.

We present a method for adding a relatively small set of promising Fourier basis functions to the dictionary based on an intuitive ordering. We then modify the OMP-TD selection process to take this ordering into account, using it as a regularisation method. Both methods achieve good empirical results, with the dictionary restriction method beating a full dictionary search with OMP-TD on some domains.

3.2 State Variable Dependencies

A central concern of reinforcement learning in continuous domains is approximating the value function effectively, with the most common approach being linear function approximation. The reinforcement learning algorithm learns the value function by adjusting the weights of a set of basis functions. The accuracy of the approximation is dependent on both the learning algorithm as well as the basis functions selected. An ideal basis function set would be able to represent the optimal value function perfectly, however obtaining such a set is often not possible. Instead, a generic basis is often used which is able to represent a wide range of value functions, and approximate most with reasonable accuracy. Recall the linear function approximation of the value function is defined as a weighted sum of basis functions $\Phi(s) = [\phi_1(s), \phi_2(s), ..., \phi_n(s)]$ with weights $\boldsymbol{w} = [w_1, w_2, ..., w_n]^T$:

$$\begin{aligned} \hat{V}^\pi(s) &= w_1\phi_1(s) + w_2\phi_2(s) + ... + w_n\phi_n(s) \\ &= \sum_{i=1}^{m} w_i\phi_i(s) \\ &= \boldsymbol{\Phi}(s)\boldsymbol{w}, \end{aligned} \tag{3.1}$$

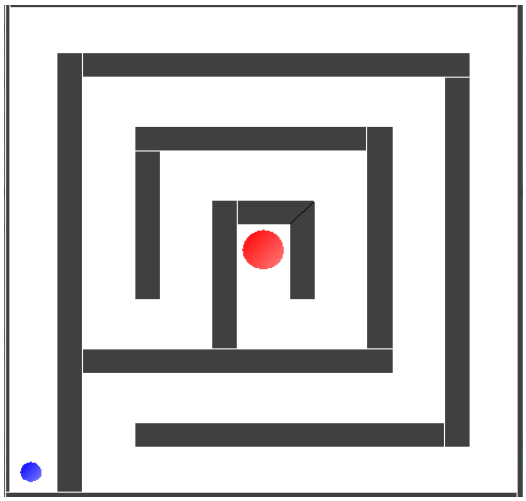
where the goal is to approximate the value function $V^\pi(s)$ for a particular policy π. The true $V^\pi(s)$ is unknown to the learning algorithm, which must learn an approximation $\hat{V}^\pi(s)$, but since the error $V_E^\pi = V^\pi - \hat{V}^\pi$ is unknown, the easiest way to reduce it is to use a larger basis set which is able to represent more functions.

The Fourier basis described in Section 2.2.2.3 is defined as a set of cosine functions of the form:

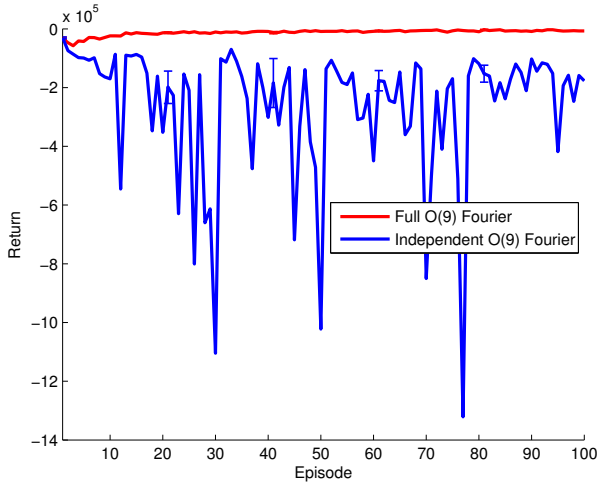
$$\phi_i(s) = cos(\pi \boldsymbol{c}_i \cdot s), \tag{3.2}$$

where $\boldsymbol{c}_i = [c_{i,1}, c_{i,2}, ..., c_{i,d}], c_{i,j} \in [0, ..., n]$ is an adjustable parameter vector, d is the number of dimensions, and n is the order of approximation.

A full set of Fourier basis functions for an order n approximation consists of each basis function generated from unique permutations of the $\boldsymbol{c}$ parameter vector. A major problem with fixed basis sets like the Fourier basis is that the size of the set scales exponentially with the number of dimensions in the state space. The resulting combinatorial explosion forces practitioners to sacrifice representational capability and discard large numbers of basis functions. One technique for doing this makes an *independence assumption*, where state variables are assumed to contribute to the value function independently. In the case of the Fourier basis, these independent basis functions are defined by a $\boldsymbol{c}$ parameter vector containing no more than one non-zero value, hence each basis function's value is only influenced by one state variable at most. For a given order n in a state space of d dimensions, a fixed basis like the Fourier basis using the independence assumption would only require $nd + 1$ basis functions, instead of the full $(n + 1)^d$, removing the combinatorial explosion since the number of basis functions is now linear in the dimensionality of the state space.

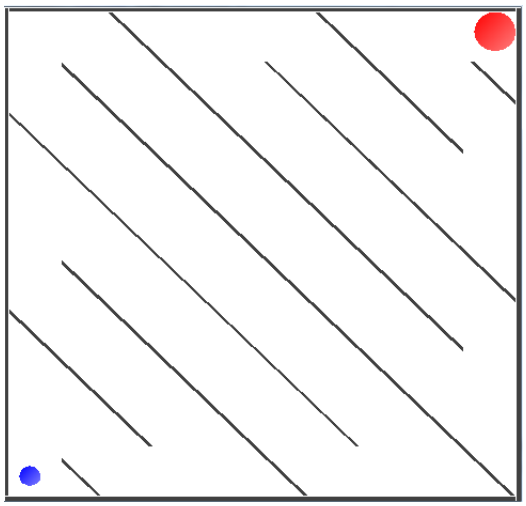


(a) Pinball domain b. Position state variables are clearly not independent since the optimal action is often dependent on both the x and y positions.

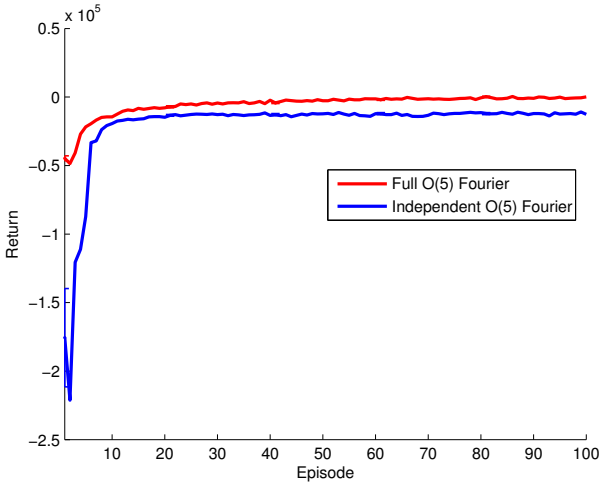


(b) The full order 9 Fourier approximation consisting of a set of 10000 basis functions is tested against the independent set of 37 basis functions on pinball domain b to the left.

Figure 3.1: In this pinball domain, the independent basis set performs dismally while the full set performs well.



(a) Pinball domain c. Position state variables are clearly not independent since the optimal action is often dependent on both the x and y positions.



(b) The full order 5 Fourier approximation consisting of a set of 1296 basis functions is tested against the independent set of 21 basis functions on pinball domain c to the left.

Figure 3.2: The full set of basis functions learns faster and converges to a better solution.

The independence assumption can be effective in many domains, but it is easy to construct domains which require dependent terms. The experimental results in Figures 3.1 and 3.2 on two toy pinball domain configurations, show that learning is possible without dependent terms, however, to achieve peak performance, they can be necessary. Pinball domain b's (Figure 3.1) value function is clearly difficult to represent using only independent Fourier basis functions, while in pinball domain c (Figure 3.2) the independent set performed fairly well. All experiments were performed using Sarsa(λ) ($\gamma = 1$, $\lambda = 0.95$, $\epsilon = 0.1$, $\alpha = 1$) with an ϵ-greedy policy. An adaptive learning rate bounding algorithm described in Dabney and Barto [2012] was used to set the α value with a minor modification described in the appendix. An order 9 Fourier approximation was used in the first domain because the independent set performed poorly enough at lower orders that the order was raised to reduce the running time of these experiments.

The results in Figure 3.2 show that with a relatively small number of basis functions (the independent set), a good value function can be learned. Adding the full set, up to a given order adds many more basis functions than may be necessary and often only a small number of these get high weight values while the rest have insignificant contributions. Identifying the basis functions with large weights can give insights into the nature of the problem as well as allow basis functions with smaller weights to be discarded with little effect on the value function. Identifying these important basis functions before they are added to the set used in function approximation is a more difficult problem. Solving it would mean improved performance in high dimensional domains where we can start with the independent basis functions—which grow only linearly with d—and incrementally add dependencies as necessary. Such an approach would keep the computational complexity down whilst at the same time providing good performance.

3.3 Dictionary Restriction

There are two main techniques for adding functions to a value function approximation. One of these is feature construction, where new features are constructed that improve the function approximation's representational capabilities. The other is selection; basis functions are stored in a dictionary from which they are selected based on certain criteria such as their correlation to the Bellman residual [Painter-Wakefield and Parr 2012].

While the feature construction methods can provide better theoretical guarantees, they do not always perform well and there are practical problems which make these methods difficult to use. Constructed features are themselves approximations to some ideal target function, which is determined by using a large set of samples. An example of this is Bellman Error Basis Functions (BEBFs) [Parr *et al.* 2007], where a large amount of sample data is used to approximate the Bellman error, and the resulting approximated error function is added to the set of features used in the value function approximation. These features are more expensive to use since they are often composed of many functions themselves, and their creation usually requires a large amount of data. Once constructed, they are also difficult to interpret due to their complex nature, making it difficult to gain insights into the important dynamics of the domain. Fixed basis sets such as the Fourier basis on the other hand, lend themselves to analysis as the basis functions themselves are generally well understood.

Feature selection methods like OMP-TD usually select basis functions from a dictionary populated by a fixed basis set such as the Fourier basis. A fixed basis set avoids the expensive construction step and removes the additional layers of approximation. Although dictionaries can be populated with any basis functions, the usual case is to populate them with a single basis function type and retain the advantages of using a fixed basis such as orthogonality and interpretability. The disadvantage of these methods is the cost of searching through the dictionary and, while less expensive than learning with the full set of basis functions, still comes with considerable costs when the dictionary is large. Since fixed basis sets scale exponentially in the number of dimensions, using one as a dictionary can become impractical as the dimensionality grows.

We explore a method of restricting the size of the dictionary as a way to use dictionary search methods in high dimensional domains. An alternative approach reserved for future work is to leave the dictionary unchanged, but to devise methods for searching it in sublinear time.

3.3.1 Frequency-ordered Dictionaries

Restricting the dictionary size is a simple method of making dictionary search methods practical in high dimensional domains. The key is to make sure important basis functions are included so that they may be added to the approximation. Unfortunately it is often impossible to know which basis functions will be important. Heuristics can be used instead to decide which basis functions should form part of the dictionary. Our heuristics use properties of Fourier basis functions, but the method of updating a dictionary is not specific to any basis function type.

The Fourier basis consists of cosine functions of varying frequencies. The lowest frequency function, $\boldsymbol{c} = [0, ..., 0]$, is a constant and is usually important, whereas the highest frequency function for an order n approximation is $\boldsymbol{c} = [n, ..., n]$, and is usually not very important. Here the importance is measured by how large a weight the term is given after learning. Our belief is that low frequency terms are generally more important than high frequency terms as high frequency terms are more likely to fit noise and low frequency terms are more likely to fit the signal. This belief is supported by Occam's razor and regularisation theory (see Chapter 4), which suggests that simpler (corresponding to low frequency) functions are more favourable than complex (corresponding to high frequency) ones [MacKay 1992].This can also be seen with the Fourier basis in online learning settings where each basis function $\phi_i(s)$ has its learning rate scaled by $1/||\boldsymbol{c}_i||$, where $\boldsymbol{c}_i$ is that basis function's parameter vector [Konidaris *et al.* 2011]. This gives high frequency basis functions a lower learning rate which leads to improved performance.

A simple method for restricting dictionary sizes is a two tiered approach. The dictionary is broken up into two parts, D and $\bar{D}$. The basis functions in D are eligible for selection by the feature selection algorithm, while the remaining basis functions (dictionary candidates) in $\bar{D}$ can be added to the dictionary as needed. This leads to computationally efficient dictionary searches at the possible cost of not being able to find the best basis functions. If the dictionary candidates are added to the dictionary in an intelligent way, the risk can be minimised.

Using the heuristic that low frequency Fourier basis functions are better, a simple method for updating D adds a small number of dictionary candidates at each step, adding basis functions ϕ with low frequency $||\boldsymbol{c}||$ first, then those with higher $||\boldsymbol{c}||$. Each time a basis function is selected from D and added to the approximation set Φ, a batch of d new basis functions $\phi_i, ..., \phi_{i+d-1} \in \bar{D}$ are chosen and removed from the dictionary candidates $\bar{D}$ and added to the dictionary D. These terms are the d lowest frequency dictionary candidates remaining. Let $\boldsymbol{c}_i$ be the parameter vector of Fourier basis function ϕ_i, the ith basis function added to the dictionary, then $\forall \phi_j \in D \cup \Phi, ||\boldsymbol{c}_j|| \leq ||\boldsymbol{c}_i||$. This method ensures a relatively slow dictionary growth rate[1], while adding basis functions believed to be of highest importance first. The dictionary update algorithm is shown in Algorithm 3. A second algorithm which randomly populates the dictionary with candidates is shown in Algorithm 4 and is later used in the experiments as a control.

While it is possible for the domain's value function to require high frequency basis functions to approximate well, this is often not the case in practice. Despite this, one would still expect an algorithm with a larger dictionary to perform better given that it has more choices. In practice the experimental results in Section 3.6 tell a different story.

[1]Choosing d as the number of basis functions to select is a heuristic with scales linearly with the dimensionality of the domain, and gives the selection algorithm more choice when the domain is more complex (higher dimensional).

Algorithm 3 Frequency-ordered Dictionary Update

Input:
Current dictionary D,
Potential functions, $\bar{D} = \{\phi_1, ..., \phi_k\}$,
State dimensions $d \in \mathbb{Z}^+$,
Output:
Updated dictionary D.

if not sorted($\bar{D}$) **then**
    sort($\bar{D}$) in ascending order of $||\boldsymbol{c}||$ value
end if
for $i = 1$ to d **do**
    $D \leftarrow D \cup \bar{D}_1$ where $\bar{D}_i$ is the ith basis function in $\bar{D}$ after sorting.
    $\bar{D} \leftarrow \bar{D} \backslash \bar{D}_1$, remove the basis function that was just added to D.
end for

Algorithm 4 Random Dictionary Update

Input:
Current dictionary D,
Potential functions, $\bar{D} = \{\phi_1, ..., \phi_k\}$,
State dimensions $d \in \mathbb{Z}^+$,
Output:
Updated dictionary D.

shuffle($\bar{D}$)
for $i = 1$ to d **do**
    $D \leftarrow D \cup \bar{D}_1$ where $\bar{D}_i$ is the ith basis function in $\bar{D}$ after shuffling.
    $\bar{D} \leftarrow \bar{D} \backslash \bar{D}_1$, remove the basis function that was just added to D.
end for

3.4 Regularised Fourier Correlation Search

At each iteration of the basic matching pursuit algorithm [Mallat and Zhang 1993], a function is selected from a dictionary D which, when added to the current set of functions, minimises the residual least squares error. The selected function at each step is the one that has the largest correlation with the residual,

$$\rho_i = \left| \frac{\langle \vec{R}, \phi_i(\vec{s}) \rangle}{||\phi_i(\vec{s})||} \right|, \tag{3.3}$$

where ρ_i is the correlation of basis function $\phi_i(s)$ to the residual $\vec{R}$ on the set of samples $\vec{s}$.

The OMP-TD algorithm [Painter-Wakefield and Parr 2012] shown in Algorithm 2, uses orthogonal matching pursuit (OMP) to select basis functions from a dictionary, and calculates their weights with LSTD. In the reinforcement learning setting, the target function V^π, is unknown, so the Bellman error $B^\pi = T^\pi V - V$ is used as the residual error. As with basic matching pursuit, the basis function in the dictionary with the highest correlation to the residual error is added to the approximation set at each iteration. However in Painter-Wakefield and Parr [2012], a slightly different correlation calculation is used,

$$\bar{\rho}_i = \left| \frac{\langle \vec{B}, \phi_i(\vec{s}) \rangle}{\sqrt{N}} \right|, \tag{3.4}$$

where N is the number of samples in $\vec{s}$ and the other values are as before. Under certain conditions

the formulas are equivalent, but not in general. We experimented with both correlation calculations and found that there was no significant difference in the performance of OMP-TD using the different formulas. In this work we use the correlation defined in Equation 3.3 with $\vec{R} = \vec{B}$.

Adding a basis function which has sufficiently high correlation with the Bellman error guarantees an improvement in the bound on the μ-weighted distance $||V^\pi - \Pi_\mu V^\pi||_\mu$ between the projection of the optimal value function V^π into the basis set Φ, and the optimal value function for policy V^π [Painter-Wakefield and Parr 2012], where μ is the stationary distribution and π is the policy. This says that with the addition of a highly correlated basis function, the basis set can approximate V^π better, even though V^π is unknown.

The most correlated function is not necessarily the best function to add at each iteration. The theory gives a lower bound on the improvement, however there could be better basis functions with low correlation to the Bellman error in the dictionary. Overfitting is another concern in practical applications, where the most correlated basis function may be overfitting the Bellman error. A common technique for solving overfitting problems is to use some form of regularisation where basis functions more likely in the prior distribution are favoured. Even if overfitting is not a problem, regularisation using a prior can improve the selection metric if the prior is good.

We use the frequency of the Fourier basis functions to address both concerns of selecting poor basis functions and overfitting. With the Fourier basis, high frequency terms are more likely to overfit data, and are generally less useful than low frequency terms. Much like the previous heuristic in Section 3.3.1, we would like to prioritise the low frequency terms. To do this we regularise the correlation by dividing through by the norm of the Fourier basis function's $\boldsymbol{c}$ value, $||\boldsymbol{c}||$, where the new regularised correlation is defined as:

$$\rho_i' = \left| \frac{\langle \vec{B}, \phi_i(\vec{s}) \rangle}{||\phi_i(\vec{s})||||\boldsymbol{c}_{\phi_i}||} \right|. \tag{3.5}$$

Using ρ_i' as the selection metric in OMP-TD prioritises low frequency basis functions while allowing high frequency basis functions to be selected if they are important enough.

Regularising the selection process on its own is not a solution to scaling up to high dimensional spaces but the results in Section 3.6 show that this direction is promising. Future work that looks at combining this regularised approach with a fast dictionary search could be an answer to scaling up.

3.5 Algorithms

For our experiments we use a more general algorithmic framework shown in Algorithm 5 as opposed to the original OMP-TD algorithm shown in Algorithm 2 in order to facilitate the inclusion of different dictionary update and correlation functions. The main difference is the introduction of a second dictionary $\bar{D}$ which stores candidate basis functions to be added to the original dictionary D. Algorithm 6 decides which update method to use, either adding the entire candidate set $\bar{D}$ to the dictionary D immediately, or using heuristic methods to restrict the size of D. The second difference on line 25 generalises the selection process, allowing different correlation calculations to be used. The original OMP-TD algorithm can be obtained by initialising the dictionary with all the required basis functions and skipping the calls to Algorithm 6 (Update Dictionary) on line 20. The correlation calculation for OMP-TD is the one found in Equation (2.40).

The general OMP-TD algorithm, Algorithm 5, can be explained as follows: first samples are collected using a fixed policy by placing the agent in the environment and allowing it to take actions using its policy. Next, an initial set of basis functions Φ is chosen, as well as a dictionary D and a secondary dictionary $\bar{D}$. LSTD is then run using the samples with the basis set Φ. Once complete, the Bellman error is calculated. Basis functions are removed from $\bar{D}$ and added to D according to the dictionary update algorithm, Algorithm 6. The most correlated basis function to the Bellman error in D is found

and removed using the desired correlation equation (Equations (2.40) and (3.5) for example). That basis function is added to Φ and LSTD run again. This process continues until D and $\bar{D}$ are empty, or until the desired number of basis functions has been reached, or when the correlation of the most correlated basis function does not meet a user defined threshold β.

Algorithm 5 General OMP-TD

Input:
Dictionary $D = \{\}$,
Dictionary candidates $\bar{D} = \{\phi_1, ..., \phi_k\}$,
Initial approximation set Φ,
Samples, each consisting of a state s, next state s', and reward r,
$\vec{s} = [s_1, ..., s_m]^T$,
$\vec{s'} = [s'_1, ..., s'_m]^T$,
$\vec{r} \in \mathbb{R}^{m \times 1} : \vec{r} = [r_1, ..., r_m]^T$,
State dimensions $d \in \mathbb{Z}^+$,
Approximation order $n \in \mathbb{Z}^+$,
Discount $\gamma \in [0, 1)$,
Stopping condition $\beta \in \mathbb{R}$.
Output:
Approximation weights $\boldsymbol{w}$ and corresponding feature set Φ.

$\boldsymbol{w} \leftarrow \vec{0}$ (Feature set weights)
Initialise feature set Φ (set of independent basis functions if using the Fourier basis, or the constant $\phi(s) = 1$ if using RBFs).
repeat
  $w \leftarrow (\Phi(\vec{s})^T \Phi(\vec{s}) - \gamma \Phi(\vec{s})^T \Phi(\vec{s'}))^{-1} \Phi(\vec{s})^T \vec{r}$ (Calculate weights)
  call Algorithm 6 (Update Dictionary) **with** D, $\bar{D}$ and d
  $m \leftarrow 0$
  $\phi_{max} \leftarrow \varnothing$
  $residualError \leftarrow R + \gamma \Phi(s')\boldsymbol{w} - \Phi(s)\boldsymbol{w}$ (Calculate Bellman error)
  for each $\phi \in D$ **do**
    $v \leftarrow$ get correlation for ϕ with states s and residual $residualError$
    if $v \geq m$ **then**
      $m \leftarrow v$
      $\phi_{max} \leftarrow \phi$
    end if
  end for
  if $m > \beta$ **then**
    $\Phi \leftarrow \Phi \cup \{\phi_{max}\}$
    $D \leftarrow D \setminus \{\phi_{max}\}$
  end if
until $m \leq \beta$ **or** $D = \{\}$

For our experiments we do not use a β value to act as a stopping condition purely for demonstration purposes. We run the main loop a fixed number of times, adding a basis function each time and recording the error. We do this to compare the algorithm variations at the level of selected basis function quality. In practice a β value would have to be chosen experimentally as in Painter-Wakefield and Parr [2012].

We use the notation $\Phi(s)$ to denote the $m \times k$ matrix created when the k basis functions are evaluated on the m states in s, $\Phi_{i,j} = \phi_j(s_i), j \in [1, \ldots, k], i \in [1, \ldots, m]$.

In the update dictionary algorithm, Algorithm 6, all the candidates are added to the dictionary immediately for the full experiment, whereas they are added gradually in the restricted dictionary methods.

Algorithm 6 Update Dictionary

Input:
Dictionary D,
$\bar{D} = \{\phi_1, \ldots, \phi_k\}$, //Candidate dictionary set
$d \in \mathbb{Z}^+$ //State dimensions
Output:
Updated dictionary D

if Full Dictionary or Frequency-regularised Dictionary **then**
    $D \leftarrow D \cup \bar{D}$
    $\bar{D} \leftarrow \{\}$
end if
if Frequency-ordered Dictionary **then**
    call Algorithm 3 **with** D, $\bar{D}$ and d
end if
if Random Dictionary **then**
    call Algorithm 4 **with** D, $\bar{D}$ and d
end if

3.6 Experiments

The aim of these experiments is to test the performance of the presented feature selection algorithms against the original OMP-TD algorithm presented in Painter-Wakefield and Parr [2012]. We use similar sample collection and error metrics as those presented in Painter-Wakefield and Parr [2012] and like Painter-Wakefield and Parr [2012], we do not consider policy improvement, but instead only focus on learning an approximate value function for a fixed policy using samples collected with that policy.

In the experiments evaluating each of the methods described in Section 3.6.2 the following general procedure was used:

- The policy π to evaluate was chosen.
- Samples were collected from the domain by placing the agent in a valid state, taking an action using π, and observing the resulting state and reward. The state, resulting state, and reward formed a single sample. This process was repeated from the resulting state until termination. This ensured that the sample distribution approximated the stationary distribution. In chainwalk, only two samples were collected before moving the agent to a new state.
- The true value $V^\pi(s)$ for the policy π was calculated by doing a Monte Carlo rollout following π for each state s in the collected samples. If the environment and π were deterministic, only one rollout was required, otherwise a large number of trajectories starting at s had to be sampled and the returns averaged to get $V^\pi(s)$. In the chainwalk domain, the value function was calculated with dynamic programming techniques.
- Each method's approximation set was initialised with some number of independent basis functions shown in Table 3.1. The remaining basis functions were either added to the dictionary incrementally, or all at once depending on the method.
- For each method, the general OMP-TD algorithm (Algorithm 5) was run using the appropriate correlation calculation and dictionary update method for that method. The samples used for learning were a portion of the collected samples. The remaining samples were used for testing. After

a basis function was selected and the weights fitted, the error in the approximation was calculated using the true values obtained using rollouts with the test samples. The average error over 100 experiments was then plotted against the number of basis functions in the approximation.

3.6.1 Domains

3.6.1.1 50 State ChainWalk

The 50 state chainwalk domain [Lagoudakis and Parr 2003] (similar to the 4 state chainwalk domain in Figure 3.3 (a)) is a 1 dimensional domain with 50 states. There are rewards at states 10 and 41 and no terminal states. Upon entering a reward state, the agent receives a reward of 1, otherwise it receives a reward of 0. The agent can either go right or left. Each move succeeds with probability 0.9. If the move fails, then the agent is moved in the opposite direction. If the agent would move off the chain for any reason, it stays in its current location instead. Since the number of states is low, no function approximation is necessary, however for demonstration purposes we scaled the states between 0 and 1 for function approximation. The policy used was the optimal policy.

3.6.1.2 Mountain Car

The mountain car task shown in Figure 3.3 (b) is a simple two dimensional task in which an underpowered car must get to the top of one side of a valley. Since the car is underpowered, it cannot achieve this by simply driving up one side. To get to the top, it needs to drive up the other side first, then using the potential energy it has stored by doing this, drive up the other side. The agent is presented with 2 state variables, its position and its velocity. At each time step where the agent is in the valley, it receives a reward of -1. When it reaches the top, it receives a reward of 0 and the episode terminates. The agent has 3 actions it can take at each time step: forward full throttle, reverse full throttle and neutral [Sutton and Barto 1998]. We used the policy of accelerating in the direction of the current velocity or going forward full throttle if at rest. We used this policy since it is guaranteed to terminate [Painter-Wakefield and Parr 2012].

3.6.1.3 Mountain Car 3D

Mountain car 3d [Taylor *et al.* 2008] in Figure 3.3 (c) is a 4 dimensional version of the mountain car domain. We used a similar policy to the 2 dimensional case where the car always accelerates in the direction of maximum velocity, or in the closest direction towards the goal if stationary.

3.6.1.4 Pinball

The pinball task shown in Figure 3.3 (d) consists of a ball, a play area which may contain obstacles, and a hole, which is the goal of the task. The learning agent can change the ball's velocity in each direction (x and y) with a reward of -5, or simply let the ball continue with a reward of -1. The goal of the task is to maneuver the ball into the hole which gives a reward of 10000 and ends the episode. Since collisions with obstacles are elastic, the agent may choose to use these collisions to help achieve the goal [Konidaris and Barto 2009].

Pinball has four continuous dimensions, the ball's position in the x and y dimensions and its velocity $\dot{x}$ and $\dot{y}$ in those dimensions, that fully represent the ball's state in the play area. Even four dimensions can be a challenge for reinforcement learning methods which makes pinball a reasonable step up from mountain car. The play area is fully configurable which allows for the creation of specific tests, setting the difficulty of the domain exactly as desired. We used the configuration in Figure 3.3 (d) in our ex-

periments, as the multiple routes to the goal meant that agents with better value functions could perform noticeably better.

The policy was the greedy policy over a value function learned using Sarsa(λ) ($\gamma = 0.99, \lambda = 0.95, \epsilon = 0.01$) with an $O(5)$ Fourier basis and an automatic learning rate adjusting method [Dabney and Barto 2012].

3.6.1.5 RC Car

The Remote Control Car domain [Geramifard 2013] shown in Figure 3.3 (e) is a 4 dimensional continuous domain with 9 actions in which a remote control car must be maneuvered to within a distance of 0.1 meters from the goal. The 9 possible actions are derived from combinations of 2 primitive action types: steering $\in \{\texttt{left}, \texttt{straight}, \texttt{right}\}$ and acceleration $\in \{\texttt{forward}, \texttt{coast}, \texttt{backward}\}$. The state is given by the x and y position of the car, its heading θ in radians and its velocity v. If the car collides with any of the walls in the 2×3 meter room it is in, its velocity is set to 0, otherwise the domain is frictionless. At each step (every 0.1 seconds) a reward of -1 is given if the agent is not at the terminal goal state or 100 if it is. The policy π used simulates all possible actions at each state and chooses the one which minimises $\delta + 2o$, where δ is the Euclidean distance to the goal and o is the orthogonal distance to the goal along the current trajectory of the car.

3.6.1.6 Acrobot

The acrobot domain [Sutton and Barto 1998] shown in Figure 3.3 (f) is a 4 dimensional domain where an agent controls the torque of the middle joint of a 2 link pendulum. The state variables consist of two angles, θ_1 and θ_2 shown in Figure 3.3 (f), as well as their angular velocities $\dot{\theta}_1$ and $\dot{\theta}_2$. The goal is to raise the tip of the pendulum a certain height. The policy π applies torque in the direction of the joints if they are moving in the same direction, else it reverses it. Occasionally this policy does not terminate. We discarded samples where this occurred.

3.6.2 Methods

In our experiments, the aim was to test how well each method's selected set of basis functions performed relative to the other method's selected sets. For each trial, sample training and test data was collected, and all algorithms were tested on the same data. After each algorithm added a basis function to the approximation set, the root mean square error was calculated for that algorithm using estimates of the true value obtained using rollouts for each of the states in the training set. 100 trials were used for each domain and the results averaged. A small amount of L_2 regularisation (0.01) was added in the LSTD step for all methods tested.

All algorithms tested are based on the algorithm in Algorithm 5, where the update dictionary (Algorithm 6) and the correlation calculation steps are responsible for the differences. For each algorithm, an initial basis set of independent Fourier basis functions up to the order n given in Table 3.1 was added to their initial set of basis functions Φ. The remaining functions consisting of interaction terms up to order n were added to a list of dictionary candidates, $\bar{D}$. These candidates were either added incrementally to the dictionaries D or all at once depending on the method.

The following methods are investigated in our experiments:

OMP-TD: The OMP-TD method's dictionary was initialised with all candidate basis functions. The selection metric used was the unregularised correlation to the Bellman error, Equation (2.40). This is the standard OMP-TD method introduced in Painter-Wakefield and Parr [2012]. All remaining methods are variants of OMP-TD using Algorithm 5.

Frequency-ordered: The frequency-ordered dictionary approach restricts the size of the dictionary by

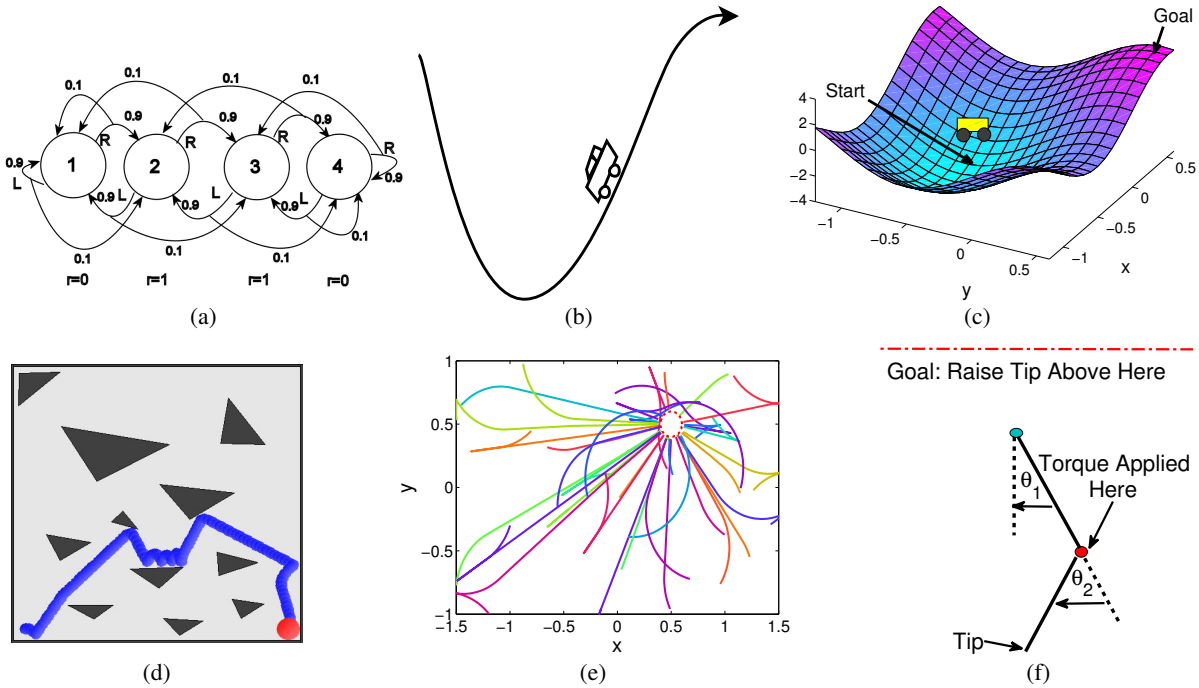


Figure 3.3: The domains used in the dictionary restriction experiments. A 4 state chainwalk with rewards at states 2 and 3 (a). Mountain car 2d (b). Mountain car 3d (c). The pinball configuration used in our experiments (configuration a) with sample trajectory (d). RC Car sample trajectories with random starts using our hand coded policy (e). Acrobot (f).

adding to an initially empty dictionary gradually using Algorithm 3. Before the correlations are calculated in each step, d Fourier basis functions are added to the dictionary from the candidate set in order of their $||\boldsymbol{c}||$ value. This has the primary effect of restricting the dictionary size to reduce computational costs with the additional side benefit of acting as a regularisation method. The selection metric used was the unregularised correlation to the Bellman error, Equation (2.40).

Random Dictionary: The random dictionary method is similar to the frequency-ordered method, instead using Algorithm 4. Starting with an empty dictionary, d random basis functions from the candidate set are added to it before a basis function is selected each time. The selection metric used was the unregularised correlation to the Bellman error, Equation (2.40).

Frequency-regularised: The frequency-ordered method was combined with a full dictionary to produce the frequency-regularised method. Instead of a strict ordering of basis functions based on their frequency, basis functions are selected not only by their correlation to the Bellman residual, but also by their frequency. This method encourages the selection of low frequency basis functions, but also allows the selection of high frequency basis functions if they are highly correlated to the Bellman residual. The frequency-regularised method's dictionary was initialised with all candidate basis functions. The selection metric used was the regularised correlation to the Bellman error, Equation (3.5).

Independence Assumption: This method used only those independent basis functions already in its approximation set. No new basis functions were selected.

Random Selection: The random selection method adds the entire set of candidate basis functions to its dictionary. It randomly selects one at each step.

The number of basis functions used and the domain parameters are given in Table 3.1. The results are shown in Figure 3.4 and the dictionary sizes are shown in Figure 3.5, where the OMP-TD, frequency-regularised and random selection methods are all full dictionary methods and the random dictionary and

Domain	γ	Max Order	Initial BFs	Total BFs	Training Samples	Test Samples	L_2 Regularisation
50 State Chainwalk	0.8	300	1	301	500	1000	0.01
Mountain Car	0.99	10	21	121	5000	10000	0.01
Mountain Car 3D	0.99	5	21	1296	20000	10000	0.01
Pinball	0.99	5	21	1296	20000	10000	0.01
RC Car	0.99	5	21	1296	20000	10000	0.01
Acrobot	0.99	5	21	1296	20000	10000	0.01

Table 3.1: Discount, initial dictionary and approximation set, samples collected and L_2 regularisation for each dictionary restriction experiment using the Fourier basis.

frequency-ordered are restricted dictionary methods.

The vertical axis shows the root mean square error with respect to the true value function for the policy π, V^π calculated with Monte Carlo simulations.

3.7 Discussion

In these experiments the standard OMP-TD algorithm was tested against the presented methods in 6 domains. Figure 3.4 shows how well the selected set of basis functions performed for each method for increasingly large basis function set sizes. Figure 3.5 shows the size of the dictionary D each method could select from. The full dictionary method only removes basis functions from the dictionary, so it its size decreases over time. The other methods add basis functions from the dictionary candidate set $\bar{D}$ and remove one basis function from the dictionary each time a basis function is selected. In the case of mountain car Figure 3.5 (b), the lines meet up when the dictionary candidate set $\bar{D}$ is empty.

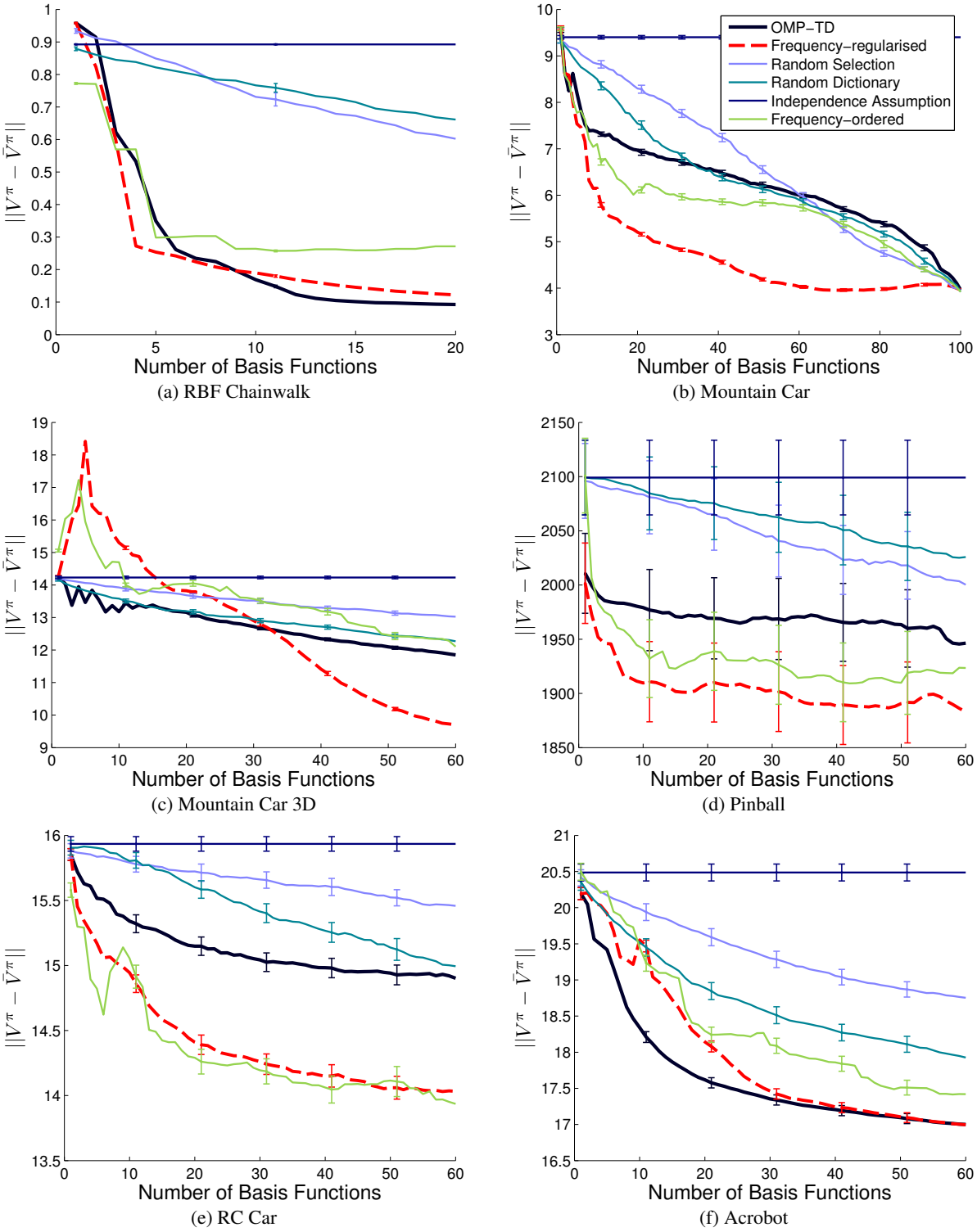
OMP-TD performed the best on acrobot and chainwalk, but only by a narrow margin. We expected OMP-TD using a full dictionary to outperform all the methods with reduced dictionary sizes. In practice however this was not always the case. Sometimes reducing the choices improved the performance. The frequency-ordered dictionary method outperformed standard OMP-TD on some domains with a smaller dictionary by reducing the number of choices available to the OMP-TD algorithm. The intended benefit of this is the reduced computational cost associated with dictionary searches, however there was also an unintended benefit where selection from the reduced dictionary actually improved results in some domains. This is a surprising, but good result.

The frequency-regularised method performed the best in general. In mountain car it was by far the best method, and it performed well in all other domains. In the mountain car 3d domain however, there is a large initial spike. This is most likely due to overfitting by the LSTD algorithm. With more basis functions it quickly overtakes the rest of the methods. Overall the strength of this method indicates that prioritising low frequency basis functions is a promising direction.

Random selection always performed poorly as expected, however the random dictionary approach performed significantly better than pure random selection. While the features in its dictionary were random, it could still choose the best one to add at each step.

In the chainwalk experiment, the number of basis functions in the dictionary was always 1 making the selection deterministic. For 1-dimensional domains, it may be a good idea to initialise the dictionary with more basis functions, although linear searches with 1-dimensional basis sets should usually be fast enough for most applications.

As a way to handle high dimensional domains, the frequency-ordering heuristic not only dramatically reduces the size of the dictionary but can also outperform the full dictionary method which is an unexpected



(a) RBF Chainwalk

(b) Mountain Car

(c) Mountain Car 3D

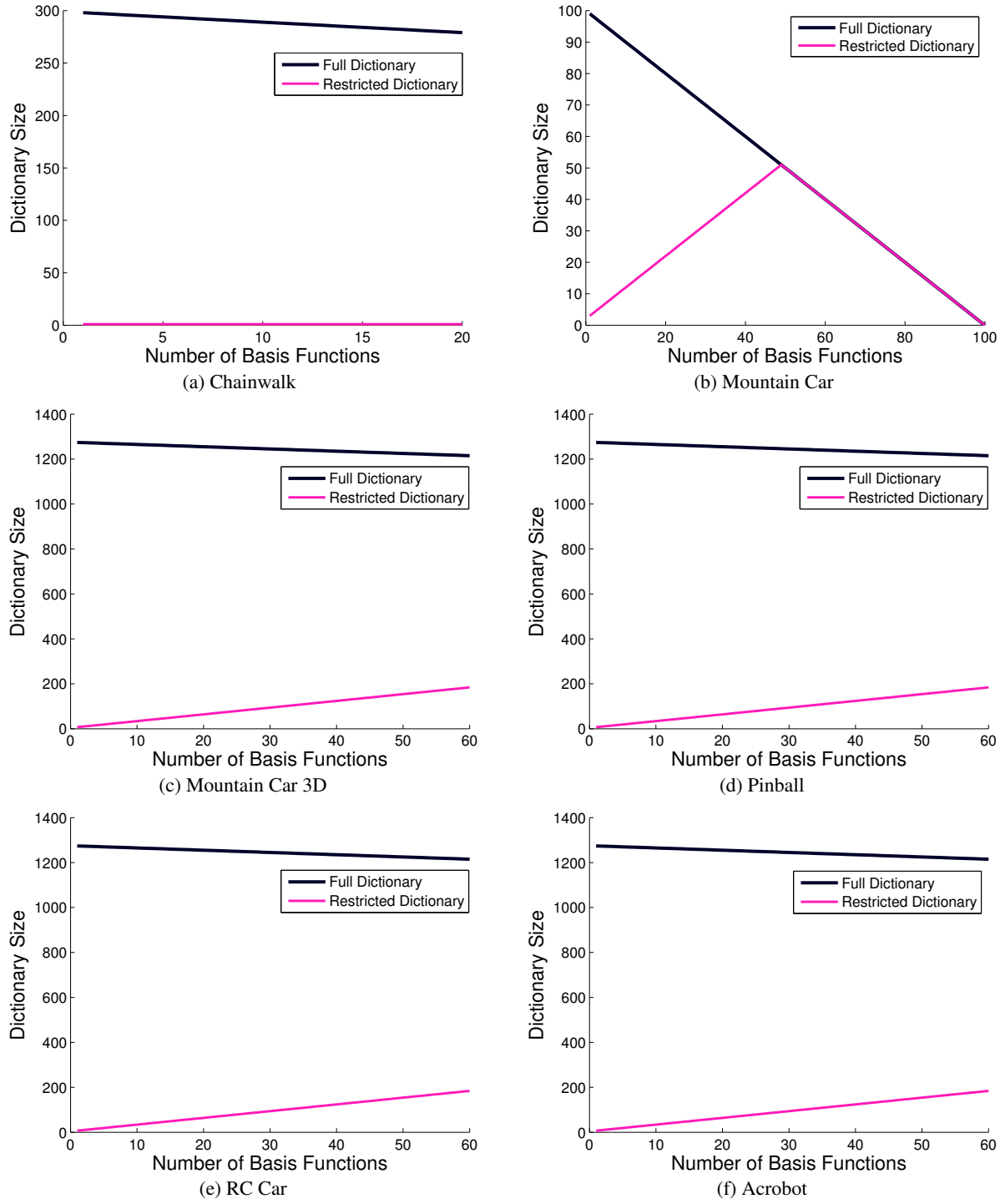
(d) Pinball

(e) RC Car

(f) Acrobot

Figure 3.4: Approximation error for the Fourier basis on the chainwalk, mountain car, mountain car 3d, pinball, RC car and acrobot domains (a-f).

result. The performance of the method is almost as good as the best method tested (the frequency-regularised method), which suggests that in high dimensional domains, using the frequency-ordered dictionary heuristic with OMP-TD is a good way to find important dependent terms without the high computational costs, and the results may even be better.



(a) Chainwalk (b) Mountain Car

(c) Mountain Car 3D (d) Pinball

(e) RC Car (f) Acrobot

Figure 3.5: Dictionary sizes for the full dictionary methods and the restricted dictionary methods on chainwalk, mountain car, mountain car 3d, pinball, RC car and acrobot domains (a-f).

3.8 Conclusion

The frequency-ordered method was tested against a random dictionary restriction algorithm and the original OMP-TD algorithm on a full dictionary of basis functions. The results show that the frequency-ordered method works well, even outperforming OMP-TD in some domains, despite it being able to chose the same features as the frequency-ordered method. More choices reduced performance as poor features were often chosen for their correlation to the training data.

The success of the frequency-ordered method led to the creation of a frequency-regularised OMP-TD method for selecting features from a full dictionary of Fourier basis functions. This method performed the best in general on the test domains with the same computational complexity as the original OMP-TD algorithm. Although the method does not scale to high dimensional state spaces any better than OMP-TD, it does show that the simple heuristic of prioritising the basis functions with low frequencies is effective.

The results imply that simply using the Bellman residual as a target for feature selection is not enough. The addition of a regularisation step significantly improves results which suggests similar methods should be further explored. In the following chapter, Chapter 4, we generalise the methods presented in this chapter, and show that $||\boldsymbol{c}||$ is related to a measure of smoothness.

Chapter 4

Regularised Feature Selection

In the batch policy evaluation setting, samples are collected from interactions with the environment using a fixed policy. A value function is learned using these samples, and another iteration of sample collection can begin using an improved policy based on the new value function. Value function accuracy is critical, and when using a linear function approximation to approximate it, the set of basis functions used has a large impact. Not only is the best set domain specific, but between iterations of policy evaluation and improvement, the value function changes, likely requiring a new set of basis functions to approximate it well. Automatic feature selection methods aim to solve the problem of selecting these basis function sets.

Of the existing feature selection algorithms for batch reinforcement learning, greedy selection methods such as orthogonal matching pursuit TD (OMP-TD), and orthogonal matching pursuit with Bellman residual minimisation (OMP-BRM) [Painter-Wakefield and Parr 2012] are of particular interest. These methods are often more computationally efficient than feature construction methods, and perform well in practice. They work by iteratively selecting basis functions predicted to cause the largest reduction in approximation error post selection.

Although greedy selection of basis functions using these methods does lead to steadily improving approximations, the goal is to improve the approximation as rapidly as possible. OMP-TD often performs better in practice than OMP-BRM [Painter-Wakefield and Parr 2012], but even so can be slow to converge for large discount factors [Mahadevan and Liu 2010]. In our experiments, values as low as $\gamma = 0.8$ reduced performance noticeably on some domains. This problem occurs because the approximation improvement bounds for selecting a feature are linked to γ, with small γ giving better bounds.

With large γ, however, value functions of environments with sparse reward functions and local transitions tend to be smooth. This is due to the Bellman equation, where the value of a state is equal to the sum of the discounted expected value of the next state and the expected reward. The values between consecutive states change little with sparse reward functions, and if those consecutive states are close in distance sufficiently often, the value function as a whole can be expected to be smooth. Many domains based on physical systems exhibit spatially local transitions [Jonschkowski and Brock 2013], and sparse reward functions are common, making smoothness a reliably exploitable attribute of value functions.

We introduce *feature regularisation* as a way to enhance performance during feature selection by giving preference to basis functions according to prior expectations of the value function. In standard regression, regularisation is commonly used to avoid overfitting, where simpler (smoother) solutions—solutions with smaller weights—are preferred. This regularisation is usually applied in the final fitting of weights with a predetermined set of basis functions. In the incremental feature selection setting, however, regularisation can be applied to the selection of individual basis functions as they are selected. Such regularisation could be used to reduce overfitting, or to introduce a prior, but unlike regularising the fit of the entire function, weight-based regularisation is inappropriate since incremental feature selection methods usually choose basis functions that have the largest weights and therefore highest impact.

Tikhonov regularisation [Tikhonov 1963] is one way to incorporate domain knowledge such as value function smoothness into feature selection. We present the general form of this regularisation in Section 4.1 and use it to form the smooth Tikhonov OMP-TD (STOMP-TD) algorithm in Section 4.2.2.1, which uses a smoothness regulariser on the selection step of the OMP-TD algorithm. We also present an almost parameter free heuristic algorithm utilising smoothness: smoothness scaled OMP-TD (SSOMP-TD) in Section 4.2.2.2 which is related to the heuristic algorithms in Chapter 3.

We show empirically that basis function smoothness is an effective regulariser in many reinforcement learning domains, and compare these two methods against methods with similar goals, regularised OMP-

TD (ROMP-TD) [Needell and Vershynin 2009] and least squares TD with random projections (LSTD-RP) [Ghavamzadeh *et al.* 2010]. We chose these methods because ROMP-TD focuses on the problem of overfitting in feature selection, and LSTD-RP deals with the problem of large feature spaces by projecting them onto a smaller space.

In Section 4.3 we empirically evaluate feature regularisation across six benchmark domains using two different types of basis, showing that feature regularisation can significantly lower prediction error. SSOMP-TD in particular performs reliably well with no increase in computational complexity and an easily chosen parameter.

4.1 Regularisation

A common solution in least squares regression to the problem of ill-posed problems and overfitting is regularisation. Given a matrix X and target vector $\boldsymbol{y}$, the problem is to find a vector $\boldsymbol{w}$ which minimises the residual subject to the regulariser $\lambda U(\boldsymbol{w})$, equivalent to minimising

$$||\boldsymbol{y} - X\boldsymbol{w}||^2 + \lambda U(\boldsymbol{w}) \tag{4.1}$$

for some $\lambda \in \mathbb{R}^+$.

An often desirable side effect of regularisation is a sparse solution [Kolter and Ng 2009], which can be achieved directly with L_1 regularisation by imposing a sparsity inducing regulariser $U(\boldsymbol{w}) = ||\boldsymbol{w}||_1$. L_2 regularisation instead introduces a constraint U which favours certain solutions or makes a problem well-posed. L_2 regularisation has the advantage that solutions can be found analytically where L_1 regularisation often requires linear programming techniques. The most common form of constraint for L_2 regularisation, $U(\boldsymbol{w}) = ||\Gamma\boldsymbol{w}||^2$, is known as Tikhonov regularisation [Tikhonov 1963] or ridge regression, where Γ is a suitable regularisation matrix (usually the identity matrix which encourages solutions with small weight norms). While it is common to choose a U which prefers low weights, U can also be chosen as a measure of the solution's smoothness. This expresses preference for simpler solutions that are less likely to overfit the data, and works in the incremental feature selection setting where basis functions with large weights are explicitly selected for.

One class of smoothness measures[1][2] for a function f acting on a state vector s is given by

$$U_m(f) = \int (f^{(m)}(s))^2 ds, \tag{4.2}$$

where $f^{(m)}$ is the mth derivative of f with $m = 2$ being a common and intuitive choice [Wahba 1990] since it gives the total squared rate of change in slope. In the case of linear function approximation, where $X \in \mathbb{R}^{q \times k}$ is a matrix of basis functions, $\phi_1, \ldots, \phi_k$, evaluated at sampled states $s_1, \ldots, s_q$ with $X_{ij} = \phi_j(s_i)$, the smoothness of the underlying function can be expressed independently of X as $U_m(\boldsymbol{\phi} \cdot \boldsymbol{w})$.

4.2 Feature Regularisation

With OMP-TD, each basis function selected is guaranteed to improve the approximation provided it is sufficiently correlated with the Bellman error basis function (BEBF) [Parr *et al.* 2007], the basis function equal to the Bellman error. While correlation to the Bellman error is a reliable metric for feature selection,

[1] There are many ways to measure smoothness, many of which exhibit similarities, for example the total variation norm is similar to U_1, the absolute value is taken instead of squaring the function's derivative. Our smoothness measure was chosen for its intuitive simplicity and because it allows the derivation of analytic solutions.

[2] This is not the same as the mathematical definition of 'smooth', used to describe a infinitely differentiable function. Smoothness here is a relative term, where one function can be 'smoother' than another.

in practice OMP-TD can be slow to reduce prediction error when the value function does not resemble the reward function [Mahadevan and Liu 2010]. Using the Neumann series expansion of the value function $V = R + \gamma PR + \gamma^2 P^2 R + ...$ it is easy to see that the value function looks less like the reward function as γ increases away from 0. This is important because the first k BEBFs which OMP-TD aims to select span the space of the first k terms of the Neumann series [Parr *et al.* 2008]. Even values as low as $\gamma = 0.9$ can lead to slow convergence in practice [Mahadevan and Liu 2010]. Since OMP-TD chooses a basis function which approximates the BEBF at each step, it can be even slower to converge.

One way to improve performance could be to choose basis functions less likely to overfit the BEBF, a problem when there are insufficient samples and many basis functions to choose from. This problem is particularly acute in the reinforcement learning setting where dimensionality can be high and interaction costly. Another possibility is to introduce prior knowledge to augment selection. The smoothness prior can be used when the underlying value function is expected to be smooth. We show in Section 4.2.1 that there is good reason to believe that value functions are smooth in general, making the smoothness prior a powerful tool for feature selection in reinforcement learning.

4.2.1 Value Function Smoothness

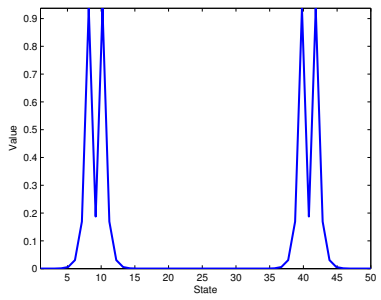
The idea that value functions are smooth in general is not new. Mahadevan and Maggioni [2007] note that value functions are typically smooth in Euclidean space. Their proto-value functions (PVFs), however, aim to take advantage of smoothness in the state graph, the graph of the state space with edges between states which can transition to each other. This type of smoothness derives from the fact that the value of a state is a function of "neighbouring" state values and associated rewards, where a state's neighbours are those reachable through transitions. Smoothness in Euclidean space is related, and coincides when neighbouring states are close in *distance* sufficiently often in conjunction with smooth or sparse reward functions. These conditions are often easy to fulfil as spatially local transitions are frequent in domains based on physical interactions [Jonschkowski and Brock 2013] and sparse/smooth reward functions are common.

Value function smoothness is also related to the discount factor γ by the Bellman equation, since the value of each state is a sum of γ-discounted values of future states. Values of γ closer to 1 lead to increased smoothness in the value function along trajectories through the state space as successive values change less. However, this can lead to larger differences in values between states which are close in space but not close in the state graph.

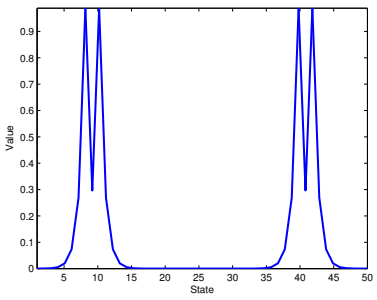
This can be seen in Figure 4.1 and Figure 4.2 where the chainwalk and negative mountain car value functions have been plotted against γ using the same policies used in the experiments in Section 4.3. In mountain car with $\gamma = 0.99$, the policy follows the smooth spiral path down to the goal (position ≥ 0.5). The reward $= 0$ received at the goal is propagated along the spiral trajectory to the start position at $(0, 0)$. Although the path following π is smoothest as $\gamma \to 1$, the value function as a whole is smoothest around $\gamma = 0.9$ due to sharp discontinuities forming for larger γ. Chainwalk gets smoother as $\gamma \to 1$.

These observations of value function smoothness naturally lead to a smoothness prior for feature selection. While value functions are not necessarily smooth in Euclidean state space, smoothness is common, and unlike smoothness over the state graph, introducing a smoothness prior in Euclidean space does not require specific domain knowledge or data. In the case of OMP-TD with a fixed basis, the smoothness of each function in the dictionary $U_m(\phi_i)$ can be pre-calculated independently of the state graph since each ϕ_i is known. Using the Frobenius tensor norm, $||A|| = \sqrt{\sum_{j_1,j_2,\ldots,j_l} A^2_{j_1,j_2,\ldots,j_l}}$, where A is any order$-l$ tensor, we derive the smoothness measures $U_m(\phi_i)$ using Equation (4.2) in the general case where $m \geq 1$ for both the RBF and Fourier bases. For an RBF basis function ϕ_i in d dimensions with variance σ and arbitrary center position, the smoothness, derived in Section 4.2.1.2 is:

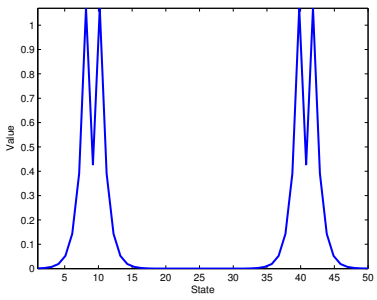
$$U^2_m(\phi) = \frac{\prod_{j=1}^{m}(d + 2(j-1))}{2^m \sigma^{2m}}. \tag{4.3}$$



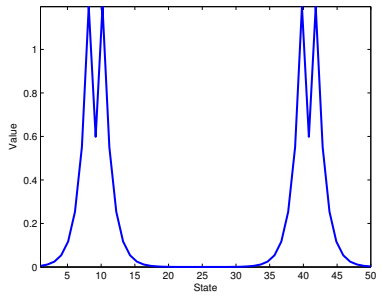
(a) $\gamma = 0.2$, $U_1 \approx 11.25$, $U_2 \approx 13020.75$, $U_3 \approx 19217463.74$



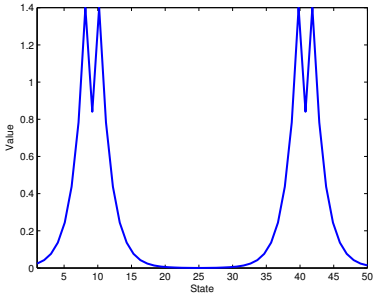
(b) $\gamma = 0.3$, $U_1 \approx 10.09$, $U_2 \approx 11129.71$, $U_3 \approx 16077380.24$



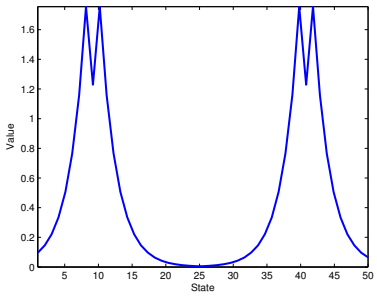
(c) $\gamma = 0.4$, $U_1 \approx 8.61$, $U_2 \approx 8838.87$, $U_3 \approx 12372098.55$



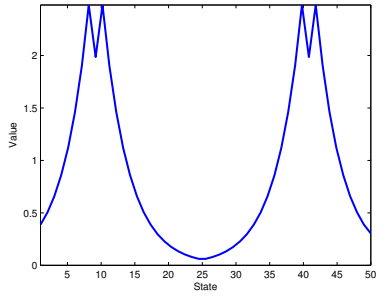
(d) $\gamma = 0.5$, $U_1 \approx 6.94$, $U_2 \approx 6420.01$, $U_3 \approx 8606565.02$



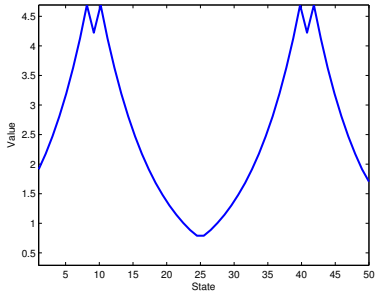
(e) $\gamma = 0.6$, $U_1 \approx 5.17$, $U_2 \approx 4136.82$, $U_3 \approx 5237358.85$



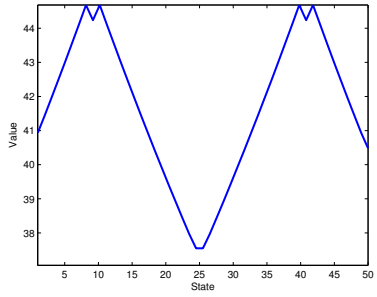
(f) $\gamma = 0.7$, $U_1 \approx 3.43$, $U_2 \approx 2221.06$, $U_3 \approx 2609861.32$



(g) $\gamma = 0.8$, $U_1 \approx 1.82$, $U_2 \approx 858.98$, $U_3 \approx 915355.58$



(h) $\gamma = 0.9$, $U_1 \approx 0.55$, $U_2 \approx 157.11$, $U_3 \approx 144922.68$



(i) $\gamma = 0.99$, $U_1 \approx 0.01$, $U_2 \approx 1.11$, $U_3 \approx 844.36$

Figure 4.1: 50-state chainwalk value functions for different values of γ (a-i). As γ increases, the value function gets smoother. Smoothness values were calculated numerically after normalising.

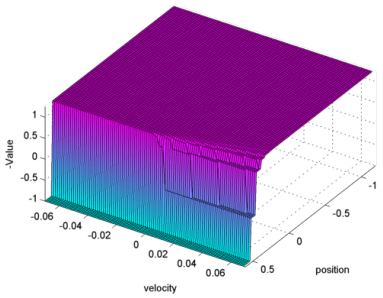
For a Fourier basis function ϕ_i with parameter vector c_i, the smoothness, derived in Section 4.2.1.1 is:

$$U_m^2(\phi_i) = \frac{||c_i||^{2m}\pi^{2m}}{2}. \tag{4.4}$$

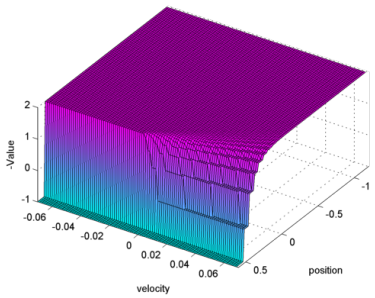
This is similar to the form of the frequency regularisation used in Chapter 3, where each Fourier basis function ϕ_i was regularised by $||c_i||$.

4.2.1.1 Fourier Smoothness Derivation

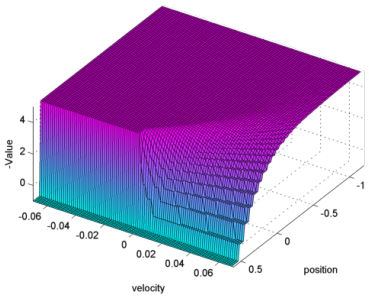
Given a Fourier basis function $\phi(s) = cos(\pi c \cdot s)$ in d dimensions with parameter vector $c = [c_1, c_2, ..., c_d]$ defined on the interval $[0, 1]$, where $s = [x_1, x_2, ..., x_d]$ and $\phi(s)'$ is the derivative of ϕ with respect to s,



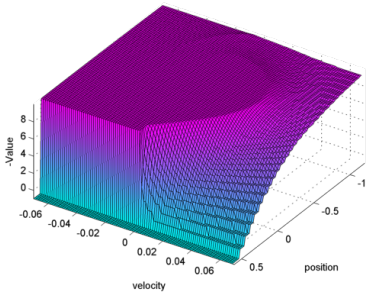
(a) $\gamma = 0.2$, $U_1 \approx 0.013$, $U_2 \approx 60.62$, $U_3 \approx 463331.15$



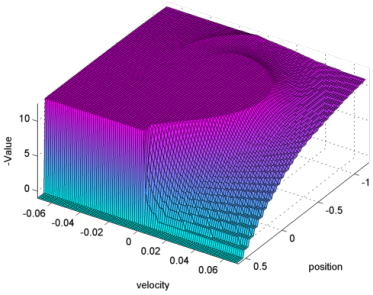
(b) $\gamma = 0.5$, $U_1 \approx 0.0085$, $U_2 \approx 39.01$, $U_3 \approx 298420.47$



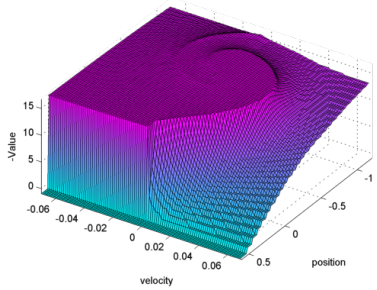
(c) $\gamma = 0.8$, $U_1 \approx 0.005$, $U_2 \approx 23.2$, $U_3 \approx 179096.6$



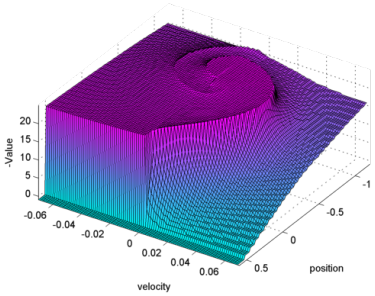
(d) $\gamma = 0.9$, $U_1 \approx 0.0043$, $U_2 \approx 20.38$, $U_3 \approx 158177.3$



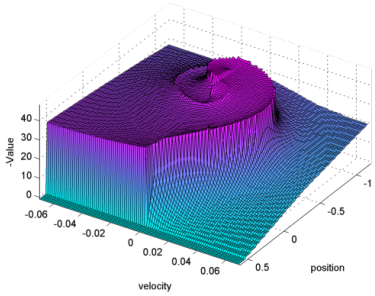
(e) $\gamma = 0.92$, $U_1 \approx 0.0043$, $U_2 \approx 20.45$, $U_3 \approx 159049.59$



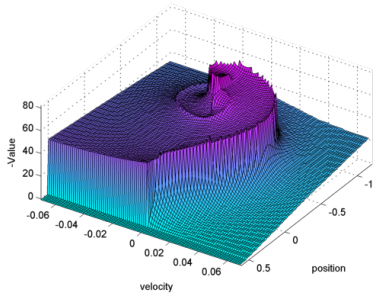
(f) $\gamma = 0.94$, $U_1 \approx 0.0045$, $U_2 \approx 21.23$, $U_3 \approx 166418.9$



(g) $\gamma = 0.96$, $U_1 \approx 0.005$, $U_2 \approx 24.03$, $U_3 \approx 195469.17$



(h) $\gamma = 0.98$, $U_1 \approx 0.0068$, $U_2 \approx 35.97$, $U_3 \approx 336816.36$



(i) $\gamma = 0.99$, $U_1 \approx 0.0095$, $U_2 \approx 56.75$, $U_3 \approx 595234.81$

Figure 4.2: Negative mountain car value functions for different values of γ (a-i). As γ increases, the value function gets smoother until $\gamma \approx 0.9$ and then rougher again. Smoothness values were calculated numerically after normalising.

the smoothness measure for $m = 1$, $U_1(\phi)$ is as follows:

$$\begin{aligned} U_1^2(\phi) &= \int_0^1 ||\phi(s)'||^2 ds \\ &= \int_0^1 \left|\left|\left(\frac{\partial cos(\pi c \cdot s)}{\partial x_1} \quad \frac{\partial cos(\pi c \cdot s)}{\partial x_2} \quad \dots \quad \frac{\partial cos(\pi c \cdot s)}{\partial x_d}\right)\right|\right|^2 dx_1 dx_2 ... dx_d \\ &= \int_0^1 (c_1^2 + c_2^2 + ... + c_n^2)\pi^2 sin(\pi c \cdot s)^2 dx_1 dx_2 ... dx_d \\ &= \int_0^1 ||c||^2 \pi^2 \frac{(1 - cos(2\pi c \cdot s))}{2} dx_1 dx_2 ... dx_d. \end{aligned} \tag{4.5}$$

Since c is a vector of integer values, $sin(2\pi \sum_{i=1}^{d} c_i) = 0$, and hence the expression simplifies to:

$$U_1^2(\phi) = \frac{||c||^2 \pi^2}{2}. \tag{4.6}$$

For $U_2(\phi)$, the smoothness is

$$\begin{aligned}
U_2^2(\phi) &= \int_0^1 ||\phi(s)''||^2 ds \\
&= \int_0^1 \left|\left|\left|\begin{pmatrix} \frac{\partial^2 cos(\pi c \cdot s)}{\partial x_1^2} & \frac{\partial^2 cos(\pi c \cdot s)}{\partial x_2 \partial x_1} & \dots & \frac{\partial^2 cos(\pi c \cdot s)}{\partial x_n \partial x_1} \\ \frac{\partial^2 cos(\pi c \cdot s)}{\partial x_1 \partial x_2} & \frac{\partial^2 cos(\pi c \cdot s)}{\partial x_2^2} & \dots & \frac{\partial^2 cos(\pi c \cdot s)}{\partial x_n \partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 cos(\pi c \cdot s)}{\partial x_1 \partial x_n} & \frac{\partial^2 cos(\pi c \cdot s)}{\partial x_2 \partial x_n} & \dots & \frac{\partial^2 cos(\pi c \cdot s)}{\partial x_n^2} \end{pmatrix}\right|\right|\right|^2 dx_1 dx_2 ... dx_d \\
&= \int_0^1 (c_1^4 + 2c_1^2 c_2^2 + ... + c_n^4)\pi^4 cos(\pi c \cdot s)^2 dx_1 dx_2 ... dx_d \\
&= \int_0^1 (c_1^2 + c_2^2 + ... + c_n^2)^2 \pi^4 cos(\pi c \cdot s)^2 dx_1 dx_2 ... dx_d \\
&= \int_0^1 ||c||^4 \pi^4 cos(\pi c \cdot s)^2 dx_1 dx_2 ... dx_d \\
&= \int_0^1 ||c||^4 \pi^4 \frac{(cos(2\pi c \cdot s) + 1)}{2} dx_1 dx_2 ... dx_d,
\end{aligned} \tag{4.7}$$

and since c is a vector of integer values, $sin(2\pi \sum_{i=1}^d c_i) = 0$, and hence the expression simplifies to:

$$U_2^2(\phi) = \frac{||c||^4 \pi^4}{2}. \tag{4.8}$$

Finally, the smoothness $U_3(\phi)$ is

$$\begin{aligned}
U_3^2 &= \int_0^1 ||\phi(s)'''||^2 ds \\
&= \int_0^1 (c_1^6 + c_1^4 c_2^2 + c_1^2 c_2^2 c_3^2 + ... + c_n^6)\pi^6 sin(\pi c \cdot s)^2 dx_1 dx_2 ... dx_d \\
&= \int_0^1 (c_1^2 + c_2^2 + ... + c_n^2)^3 \pi^6 sin(\pi c \cdot s)^2 dx_1 dx_2 ... dx_d \\
&= \int_0^1 ||c||^6 \pi^6 \frac{(1 - cos(2\pi c \cdot s))}{2} dx_1 dx_2 ... dx_d,
\end{aligned} \tag{4.9}$$

and since c is a vector of integer values, $sin(2\pi \sum_{i=1}^d c_i) = 0$, and hence the expression simplifies to:

$$U_3^2(\phi) = \frac{||c||^6 \pi^6}{2}. \tag{4.10}$$

From the first 3 derivatives a pattern emerges giving the general form of the smoothness parameter for a given $m > 0, m \in \mathbb{Z}$:

$$U_m^2(\phi) = \frac{||c||^{2m} \pi^{2m}}{2}. \tag{4.11}$$

4.2.1.2 RBF Smoothness Derivation

For a radial basis function,

$$\phi(s) = \frac{e^{\frac{-||c-s||^2}{2\sigma^2}}}{\pi^{d/4}\sigma^{d/2}}, \tag{4.12}$$

in d dimensions with parameter vector $c = [c_1, c_2, ..., c_d]$ and variance σ defined on the interval $(-\infty, \infty)$, where $s = [x_1, x_2, ..., x_d]$ and $\phi(s)'$ is the derivative of ϕ with respect to s.

When taking the integral with respect to s from $-\infty$ to ∞, the location of the RBF center becomes irrelevant so c can be set to 0. The smoothness measure for $m = 1$, $U_1(\phi)$ is as follows:

$$\begin{aligned}
U_1^2(\phi) &= \int_{-\infty}^{\infty} ||\phi^{(1)}(s)||^2 dx_1 dx_2 ... dx_d \\
&= \int_{-\infty}^{\infty} \left|\left| \begin{pmatrix} \frac{\partial \phi(s)}{\partial x_1} & \frac{\partial \phi(s)}{\partial x_2} & \cdots & \frac{\partial \phi(s)}{\partial x_d} \end{pmatrix} \right|\right|^2 dx_1 dx_2 ... dx_d \\
&= \int_{-\infty}^{\infty} \sum_{i=1}^{d} \left(\frac{x_i^2}{\pi^{d/2}\sigma^{d+4}} e^{\frac{-||c-s||^2}{\sigma^2}} \right) ds \\
&= \frac{1}{\pi^{d/2}\sigma^{d+4}} \left(\sum_{i=1}^{d} \int_{-\infty}^{\infty} x_i^2 e^{-x_i^2/\sigma^2} dx_i \prod_{j=1, j\neq i}^{d} \int_{-\infty}^{\infty} e^{-x_j^2/\sigma^2} dx_j \right) \\
&= \frac{1}{\pi^{d/2}\sigma^{d+4}} \left(\sum_{i=1}^{d} \sigma^3 \int_{-\infty}^{\infty} t^2 e^{-t^2} dt \prod_{j=1, j\neq i}^{d} \sigma \int_{-\infty}^{\infty} e^{-t^2} dt \right) \\
&= \frac{1}{\pi^{d/2}\sigma^{d+4}} \left(d\sigma^3 \int_{-\infty}^{\infty} t^2 e^{-t^2} dt \left(\sigma \int_{-\infty}^{\infty} e^{-t^2} dt \right)^{d-1} \right) \\
&= \frac{d\sigma^{d+2}}{\pi^{d/2}\sigma^{d+4}} \frac{\sqrt{\pi}}{2} (\sqrt{\pi})^{d-1} \\
&= \frac{d}{2\sigma^2}.
\end{aligned} \tag{4.13}$$

The smoothness measure for $m = 2$, $U_2(\phi)$ is derived as follows:

When $i = j$,

$$\begin{aligned}
\frac{\partial^2 \phi(s)}{\partial x_i x_j} &= \frac{\partial^2 \phi(s)}{\partial x_i^2} \\
&= \left(\frac{x_i^2}{\sigma^4} - \frac{1}{\sigma^2} \right) \frac{e^{-(x_1^2+...+x_d^2)/2\sigma^2}}{(\pi\sigma^2)^{d/4}} \\
&= \frac{1}{\pi^{d/4}\sigma^{d/2+4}} (x_i^2 - \sigma^2) e^{-(x_1^2+...+x_d^2)/2\sigma^2},
\end{aligned} \tag{4.14}$$

and when $i \neq j$,

$$\begin{aligned}
\frac{\partial^2 \phi(s)}{\partial x_i x_j} &= \frac{x_i x_j}{\sigma^4} \frac{e^{-(x_1^2+...+x_d^2)/2\sigma^2}}{(\pi\sigma^2)^{d/4}} \\
&= \frac{x_i x_j}{\pi^{d/4}\sigma^{d/2+4}} e^{-(x_1^2+...+x_d^2)/2\sigma^2},
\end{aligned} \tag{4.15}$$

then:

$$
\begin{aligned}
U_2^2(\phi) &= \int_{-\infty}^{\infty} ||\phi^{(2)}(s)||^2 dx_1 dx_2 ... dx_d \\
&= \int_{-\infty}^{\infty} \left|\left| \begin{pmatrix} \frac{\partial^2\phi(s)}{\partial x_1^2} & \frac{\partial^2\phi(s)}{\partial x_2 \partial x_1} & \cdots & \frac{\partial^2\phi(s)}{\partial x_n \partial x_1} \\ \frac{\partial^2\phi(s)}{\partial x_1 \partial x_2} & \frac{\partial^2\phi(s)}{\partial x_2^2} & \cdots & \frac{\partial^2\phi(s)}{\partial x_n \partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2\phi(s)}{\partial x_1 \partial x_n} & \frac{\partial^2\phi(s)}{\partial x_2 \partial x_n} & \cdots & \frac{\partial^2\phi(s)}{\partial x_n^2} \end{pmatrix} \right|\right|^2 dx_1 dx_2 ... dx_d \\
&= \frac{1}{\pi^{d/2}\sigma^{d+8}} \int_{-\infty}^{\infty} \left(\sum_{i=1}^{d} (x_i^2 - \sigma^2)^2 + \sum_{i=1}^{d} \sum_{j=1, i\neq j}^{d} x_i^2 x_j^2 \right) e^{-(x_1^2 + ... + x_d^2)/\sigma^2} dx_1 dx_2 ... dx_d \\
&= \frac{1}{\pi^{d/2}\sigma^{d+8}} \left\{ \left(d\sigma^5 \int_{-\infty}^{\infty} (t^2 - 1)^2 e^{-t^2} dt \right) \left(\sigma \int_{-\infty}^{\infty} e^{-t^2} dt \right)^{d-1} \right. \\
&\quad \left. + d(d-1) \left(\sigma^3 \int_{-\infty}^{\infty} t^2 e^{-t^2} dt \right)^2 \left(\sigma \int_{-\infty}^{\infty} e^{-t^2} dt \right)^{d-2} \right\} \\
&= \frac{1}{\pi^{d/2}\sigma^{d+4}} \left\{ \left(3d \frac{\sqrt{\pi}}{4} \right) \pi^{(d-1)/2} + d(d-1) \frac{\pi}{4} \pi^{(d-2)/2} \right\} \\
&= \frac{1}{\sigma^4} \left\{ \frac{3d}{4} + \frac{d(d-1)\pi}{4} \right\} \\
&= \frac{3d + d(d-1)}{4\sigma^4} \\
&= \frac{d(d+2)}{4\sigma^4}.
\end{aligned}
\tag{4.16}
$$

After the 4th derivative a pattern emerges and the general form for $m > 0, m \in \mathbb{Z}$ is:

$$
U_m^2(\phi_i) = \frac{\prod_{i=1}^{m} (d + 2(i-1))}{2^m \sigma^{2m}}. \tag{4.17}
$$

4.2.2 Smoothness Regularised Feature Selection Methods

We propose two new methods which integrate the smoothness regulariser into the selection process: STOMP-TD in Section 4.2.2.1 which uses the standard Tikhonov regularisation formulation, and SSOMP-TD in Section 4.2.2.2 which directly modifies the basis function correlation formula.

The algorithm for each method follows the same steps as OMP-TD. Given a set of samples obtained using a fixed policy, and a dictionary of basis functions, an approximation set of basis functions is formed. This is initialised with at least the constant function. Weights are fitted to the approximation set using LSTD, and the resulting Bellman residual calculated. The correlation ρ of the Bellman residual to each basis function in the dictionary is calculated. This correlation is then modified by STOMP-TD (Equation (4.22)) or SSOMP-TD (Equation (4.23)) and the single most correlated basis function is selected and added to the approximation set. The weights are again calculated by LSTD and the process repeated until the Bellman residual is sufficiently small.

The algorithms can be prevented from selecting redundant features by using orthogonal basis sets in the dictionary, however even with redundant features, their correlation to the Bellman error will be small. This is because LSTD is the orthogonal projection of TV into the approximation set, and consequently Parr *et al.* [2007] show that the Bellman residual $(TV - V)$ is orthogonal to the span of the approximation set.

4.2.2.1 Smooth Tikhonov OMP-TD

Tikhonov regularisation (ridge regression) is a common type of regularisation where the regularisation term λU is added linearly to the minimisation problem. For feature selection we are only interested in finding the single most correlated basis function, and hence the problem becomes finding the basis function ϕ_i and associated weight w_i which minimise the regularised problem at step k:

$$(\phi_i, w_i) = \arg \min_{\phi \in D, w \in \mathbb{R}} ||\vec{B}_{k-1} - w\phi||^2 + \lambda U_m(w\phi), \tag{4.18}$$

where $U_m : C^1(\mathbb{R}^d, \mathbb{R}) \to \mathbb{R}$ is a suitable regularisation function and $\lambda \in \mathbb{R}$ controls the harshness of the regularisation with $\lambda = 0$ giving the unregularised case. Here $C^1(\mathbb{R}^d, \mathbb{R})$ is the class of continuously differentiable functions from $\mathbb{R}^d$ to $\mathbb{R}$. We use the notation $||\vec{x}||^2$ to denote the vector norm, where $||\vec{x}||^2 = \sum_i x_i^2$ and the notation $\langle \vec{x}, \vec{y} \rangle$ for the inner product of vectors $\vec{x}$ and $\vec{y}$, where $\langle \vec{x}, \vec{y} \rangle = \sum_i x_i y_i$.

The regularised correlation ρ' for a basis function ϕ can be derived by first solving for the optimal w for the given ϕ using the smoothing regularisation constraint U:

$$\begin{aligned} 0 &= \frac{\partial ||\vec{B}_{n-1} - w\phi||^2}{\partial w} + \frac{\partial \lambda ||wU||^2}{\partial w} \\ 0 &= -2\langle \vec{B}_{n-1}, \phi \rangle + 2w||\phi||^2 + 2\lambda w U^2 \\ w &= \frac{\langle \vec{B}_{n-1}, \phi \rangle}{||\phi||^2 + \lambda U^2}. \end{aligned} \tag{4.19}$$

Substituting the optimal w into the minimisation problem, the best $\phi \in D$ is the one that minimises the following equation:

$$\begin{aligned} ||\vec{B}_{n-1} - w\phi||^2 + \lambda ||wU||^2 &= \left|\left| \vec{B}_{n-1} - \frac{\langle \vec{B}_{n-1}, \phi \rangle}{||\phi||^2 + \lambda U^2} \phi \right|\right|^2 + \lambda \left|\left| \frac{U \langle \vec{B}_{n-1}, \phi \rangle}{||\phi||^2 + \lambda U^2} \right|\right|^2 \\ &= ||\vec{B}_{n-1}||^2 - 2 \left(\frac{\langle \vec{B}_{n-1}, \phi \rangle}{||\phi||^2 + \lambda U^2} \right) \langle \vec{B}_{n-1}, \phi \rangle \\ &\quad + \left(\frac{\langle \vec{B}_{n-1}, \phi \rangle}{||\phi||^2 + \lambda U^2} \right)^2 ||\phi||^2 + \lambda \left|\left| \frac{U \langle \vec{B}_{n-1}, \phi \rangle}{||\phi||^2 + \lambda U^2} \right|\right|^2 \\ &= ||\vec{B}_{n-1}||^2 - 2 \left(\frac{\langle \vec{B}_{n-1}, \phi \rangle^2}{||\phi||^2 + \lambda U^2} \right) \\ &\quad + \left(\frac{\langle \vec{B}_{n-1}, \phi \rangle}{||\phi||^2 + \lambda U^2} \right)^2 ||\phi||^2 + \lambda U^2 \left(\frac{\langle \vec{B}_{n-1}, \phi \rangle}{||\phi||^2 + \lambda U^2} \right)^2 \\ &= ||\vec{B}_{n-1}||^2 - \frac{\langle \vec{B}_{n-1}, \phi \rangle^2}{||\phi||^2 + \lambda U^2}. \end{aligned} \tag{4.20}$$

Therefore in order to minimise Equation (4.18), the regularised correlation,

$$\rho' = \left| \frac{\langle \vec{B}_{n-1}, \phi \rangle}{\sqrt{||\phi||^2 + \lambda U^2}} \right|, \tag{4.21}$$

must be maximised.

We use the smoothing regularisation functions in equations (4.3) and (4.4) and find the solution to Equation (4.18) by searching the dictionary for the basis function $\phi_i \in D$ with largest regularised correlation ρ'_i where,

$$\rho'_i = \left| \frac{\langle \vec{B}_{k-1}, \phi_i \rangle}{\sqrt{||\phi_i||^2 + \lambda U_m(\phi_i)}} \right|. \tag{4.22}$$

Using ρ'_i in place of ρ_i (Equation (2.40)) in the OMP-TD algorithm results in our Smooth Tikhonov OMP-TD (STOMP-TD) algorithm.

4.2.2.2 Smoothness Scaled OMP-TD

The STOMP-TD algorithm introduces a new parameter λ. Methods for finding appropriate values for λ exist [Wahba 1990], however we propose a parameter free heuristic approach which is less powerful than STOMP-TD but simpler to use in practice, called Smoothness Scaled OMP-TD (SSOMP-TD). As with STOMP-TD, the SSOMP-TD algorithm favours basis functions which are both smooth and highly correlated instead of focusing on correlation alone. In order to achieve this, the correlation ρ_i ρ_i (Equation (2.40)) at step k of each basis function $\phi_i \in D$ is scaled by the square root of its smoothness $U_m(\phi_i)$, giving the regularised correlation ρ''_i:

$$\rho''_i = \left| \frac{\langle \vec{B}_{k-1}, \phi_i \rangle}{||\phi_i|| \sqrt{U_m(\phi_i)}} \right| = \frac{\rho_i}{\sqrt{U_m(\phi_i)}}, \tag{4.23}$$

where smooth basis functions have a low $U_m(\phi_i)$, increasing the correlation, and erratic/rough basis functions have a high $U_m(\phi_i)$, particularly in the case where $m = 2$. To avoid division by 0, all basis functions with a smoothness of 0 can be added to the approximation up front.

STOMP-TD and SSOMP-TD are similar as SSOMP-TD can be thought of as equivalent STOMP-TD with a large λ. Another interesting relation is to the regularisation used in Chapter 3, which is equivalent to SSOMP-TD with $m = 2$ when using the Fourier basis. Using this fact, the frequency-ordered method and the frequency-regularised method can be generalised to use smoothness, and become applicable to more than just the Fourier basis. In fact, the frequency-regularised method is equivalent to SSOMP-TD with $m = 2$ using the Fourier basis.

SSOMP-TD's performance is strongly related to the smoothness of V^* as shown in Figure 4.3 where the performance of OMP-TD and SSOMP-TD $m = 2$ are plotted against the discount factor γ (as a way of controlling the function's smoothness). In chainwalk, the error in SSOMP-TD steadily decreases along with the smoothness as γ increases, except at the end where it slightly increases. In mountain car, the best performance of SSOMP-TD is reached when the value function is smoothest.

Both STOMP-TD and SSOMP-TD only modify the correlation calculation without increasing complexity and hence the running times are the same as OMP-TD's $O(Kq)$ for selection, and $O(k^3 + qk^2)$ for fitting with LSTD, where K is the size of the dictionary, q is the number of samples and k is the size of the approximation set.

4.2.3 ROMP-TD and LSTD-RP

We compare our methods to two methods with similar goals; the temporal difference version of the regularised orthogonal matching pursuit algorithm (ROMP) [Needell and Vershynin 2009]: ROMP-TD, and the least squares temporal difference with random projections algorithm (LSTD-RP) [Ghavamzadeh *et al.* 2010].

An adaptation to the OMP algorithm, ROMP has not yet been applied in the reinforcement learning setting to the best of our knowledge. We introduce the ROMP-TD variant of the ROMP algorithm here, and while the theoretical guarantees of the ROMP algorithm may still apply in the TD case, we leave the proofs of such for a more complete study of the ROMP-TD algorithm. We include it as the only other algorithm we know of which employs regularisation in the selection step of OMP. Unlike our OMP-TD based methods, ROMP-TD selects a *set* of basis functions to add to the approximation at each step instead of a single function. The selected set J_0 fulfils two criteria: 1) the correlations of each basis function in the set are comparable,[3] and 2) the magnitude or energy $||\rho|_{J_0}||^2$ of the correlations in the set

[3]The correlation ρ_i of basis function ϕ_i is comparable to the correlation ρ_j of ϕ_j if $|\rho_i| \leq 2|\rho_j|$ and $|\rho_j| \leq 2|\rho_i|$.

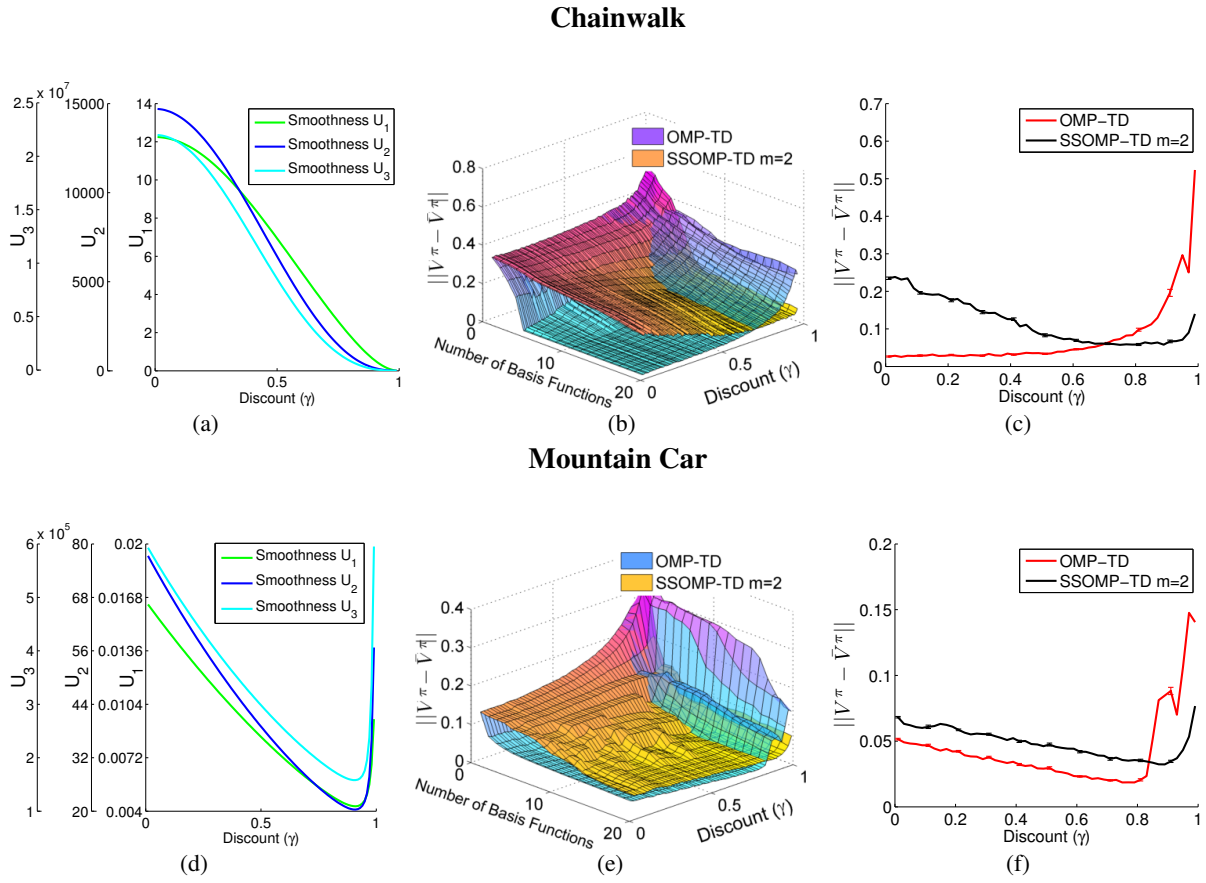


Figure 4.3: The effect of discount factor γ on the normalised value function smoothness in chainwalk and mountain car (a) and (d). The normalised root mean squared error of OMP-TD and SSOMP-TD in relation to γ on the chainwalk (b,c) and mountain car domains (e,f) after each of the first 20 basis functions are selected (b,e) and after exactly 20 basis functions have been selected (c,f).

is the largest amongst all sets which fulfil criteria 1, where $\rho|_{J_0}$ is the vector of correlations for the basis functions in set J_0 . This forces the correlation of basis functions in the solution to be closer to uniform and thus more regular [Needell and Vershynin 2009]. Intuitively, it is better to have a set of good basis functions rather than one highly correlated basis function which may overfit the data. The selection step of the ROMP-TD algorithm is decoupled from the OMP fitting step and can be inserted into OMP-TD without modification.

LSTD-RP deals with large feature sets differently. Instead of selecting features so that the set used in the fitting step is computationally manageable, LSTD-RP projects the entire set of features using a random projection matrix onto a low dimensional space. The low dimensional space results in a significant reduction in the size of the matrices used in LSTD, making it possible to compute value functions despite the curse of dimensionality.

The running time of the ROMP selection step is of the same order as OMP-TD's selection step: $O(Kq)$ where K is the number of basis functions in the dictionary and q is the number of samples.

LSTD-RP has the advantage that it is not incremental and the LSTD calculation only has to be done once in $O(k^3 + qkK)$ where q is the number of samples, K is the size of the dictionary and k is the size of the low dimensional space which is the size of the approximation sets of the feature selection methods. Predicting the value of a state comes with a cost of $O(Kk)$ instead of $O(k)$ for feature selection methods.

4.3 Experiments

Our goal is to compare the selection methods based on the ordering they impose on their candidate basis functions. We therefore show the performance of each method against the number of basis functions selected instead of the stopping condition parameter (β) described in the OMP-TD algorithm [Painter-Wakefield and Parr 2012]. Any setting of β would correspond to truncating the results at some point. Our experimental design is similar to that used in Painter-Wakefield and Parr [2012]. For ROMP-TD, where sets of basis functions are selected,[4] we added basis functions from the set one at a time in descending order of correlation magnitude to make the methods comparable. For LSTD-RP, we projected the entire dictionary available to the other algorithms such that the matrices used in the fitting step of each method were the same size. We used the same amount of L_2 regularisation as the other methods in the fit.

In the RBF experiments, the approximation set contained the constant function $\phi_1 = 1$ at the start. The dictionary contained basis function sets up to a maximum order. For example, with a maximum order of 4, the RBF basis sets of orders $n = 1, 2, 3$ and 4 would all be added to the dictionary, with each set using a different variation parameter $\sigma^2 = 1/(2(n+1)^3 - 4(n+1)^2 + 2(n+1))$ depending on its order, n. This would give the selection algorithm the choice between wider (more general/smoother) and thinner (more specific/sharper) basis functions where appropriate.

In the Fourier basis experiments, the approximation set contained independent basis functions up to a given order at the start. The dictionary contained all basis functions up to a maximum order. For the chain walk experiments, where all basis functions are independent, only the constant basis function was added to the approximation set at the start.

4.3.1 Domains

The domains and policies used are the same as those in Chapter 3 shown in Figure 3.3: 50 state chain-walk, mountain car, mountain car 3d, pinball (with configuration (a)), RC car and acrobot.

In each domain, training and test samples were collected using a fixed deterministic policy. In chainwalk, pairs of samples were collected by following the optimal policy from a random state to the next state. In the other domains, the given policy was followed from a random state until termination. The error in the approximation was obtained by comparing the approximate value $\bar{V}^\pi$ to the true value V^π using the root mean squared error. In chainwalk, V^π was calculated using dynamic programming. In the other domains, V^π was calculated using a single rollout (deterministic policy).

The experiment setup for each domain with the RBF basis is given in Table 4.1 and with the Fourier basis in Table 4.2. We note that the high regularisation for some of the RBF experiments was due to unstable behavior in OMP-TD and sometimes ROMP-TD. Painter-Wakefield and Parr [2012] also note this behavior. For the STOMP-TD experiments, we tried on average 10 different values of λ and only showed the best performing out of those. We tried to choose a reasonable value but did not spend significant time optimising. A poor choice of λ significantly affected the results. Results were averaged over 100 trials. For each experiment we used a small amount of L_2 regularisation at first, and increased it where there was significant overfitting. We repeated this as necessary.

4.3.2 Results

The experimental results are shown in Figure 4.4 and Figure 4.5. They show that standard OMP-TD performs well in general, but is beaten by SSOMP-TD and STOMP-TD on all but the Fourier acrobot experiment. Interestingly, with the settings of γ selected, the requirement for guaranteed improvement (the angle between ϕ and the BEBF must be less than $cos^{-1}(\gamma)$ [Parr *et al.* 2007]) was seldom, if ever, satisfied. Despite this, OMP-TD performs well, suggesting the existence of stronger theoretical results.

[4] We did not restrict the size of the selected set using the optional parameter ψ.

Domain	γ	**Max Order**	**Initial BFs**	**Total BFs**	**Training Samples**	**Test Samples**	L_2 **Regularisation**
50 State Chainwalk	0.8	49	1	1276	500	1000	0.01
Mountain Car	0.99	9	1	385	5000	10000	0.1
Mountain Car 3D	0.99	4	1	978	20000	10000	1000.0
Pinball	0.99	4	1	978	20000	10000	0.1
RC Car	0.99	4	1	978	20000	10000	2500.0
Acrobot	0.99	4	1	978	20000	10000	5000.0

Table 4.1: Discount, initial dictionary and approximation set, samples collected and L_2 regularisation for each experiment using the RBF basis.

Domain	γ	**Max Order**	**Initial BFs**	**Total BFs**	**Training Samples**	**Test Samples**	L_2 **Regularisation**
50 State Chainwalk	0.8	300	1	301	500	1000	0.01
Mountain Car	0.99	10	21	121	5000	10000	0.01
Mountain Car 3D	0.99	5	21	1296	20000	10000	0.01
Pinball	0.99	5	21	1296	20000	10000	0.01
RC Car	0.99	5	21	1296	20000	10000	0.01
Acrobot	0.99	5	21	1296	20000	10000	0.01

Table 4.2: Discount, initial dictionary and approximation set, samples collected and L_2 regularisation for each experiment using the Fourier basis.

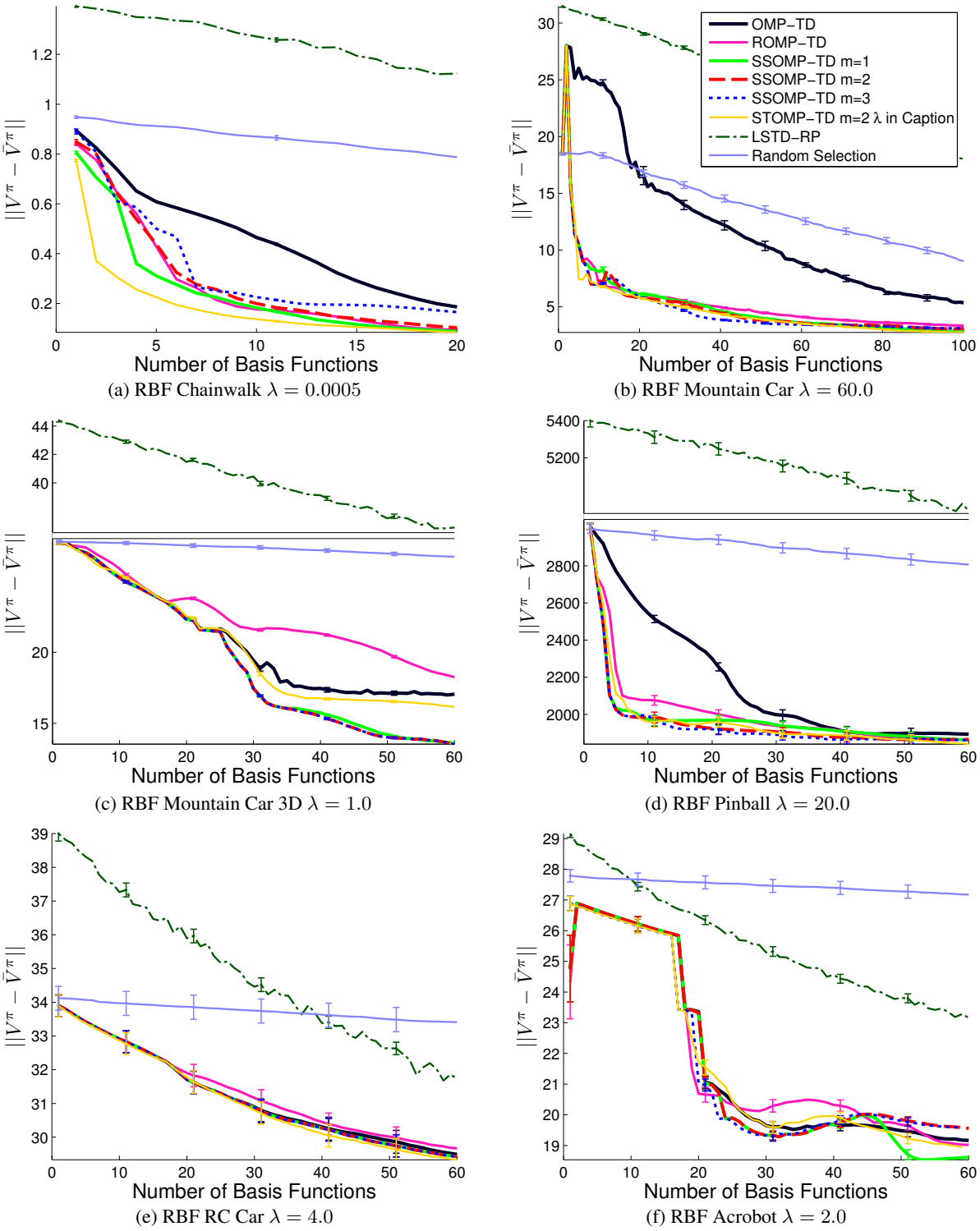
The high regularisation for the RBF experiments was due to unstable behavior in the LSTD fitting step. Interestingly SSOMP-TD and STOMP-TD did not suffer from this unstable behavior.

The SSOMP-TD algorithms performed reliably well, with the smoothing regulariser U_1 performing the worst of the three on average, and U_2 most often performing the best, or tied for best. This result is comforting, since $m = 2$ makes the most sense intuitively, where the smoothness is the total squared change in the derivative i.e. it describes how quickly the slope changes. SSOMP-TD performed its worst on acrobot with both basis sets. This could indicate that the underlying value function of our policy is not smooth. The algorithm seems to be at least as good as the others, if not better on the remaining domains taken as a whole.

The STOMP-TD algorithm performed well across all domains, however selecting an appropriate λ was difficult, and some settings resulted in performance significantly worse than OMP-TD. It is more powerful than SSOMP-TD as it can be adapted to use any prior, but it is also more difficult to use. In practice, we would use SSOMP-TD as choosing the m parameter is significantly easier than choosing λ, and the results are comparable for the two methods.

ROMP-TD was never the best performer, and in multiple experiments performed the worst. ROMP-TD primarily tackles the problem of overfitting, and it is possible there were too many samples for overfitting to occur. This suggests that the effectiveness of SSOMP-TD and STOMP-TD is primarily due to the smoothness of the value function in these experiments. Interestingly in the Fourier pinball experiment, selecting basis functions randomly outperformed ROMP-TD. It is possible that selecting batches of basis functions was too conservative and the other methods could take advantage of the changing residual after each selection. Even randomly chosen functions were better than a set of functions selected to approximate one residual well.

LSTD-RP performed poorly in all domains, only beating the random selection method in two of the RBF experiments. Its best performance with the Fourier basis was in mountain car, which was the only



(a) RBF Chainwalk $\lambda = 0.0005$

(b) RBF Mountain Car $\lambda = 60.0$

(c) RBF Mountain Car 3D $\lambda = 1.0$

(d) RBF Pinball $\lambda = 20.0$

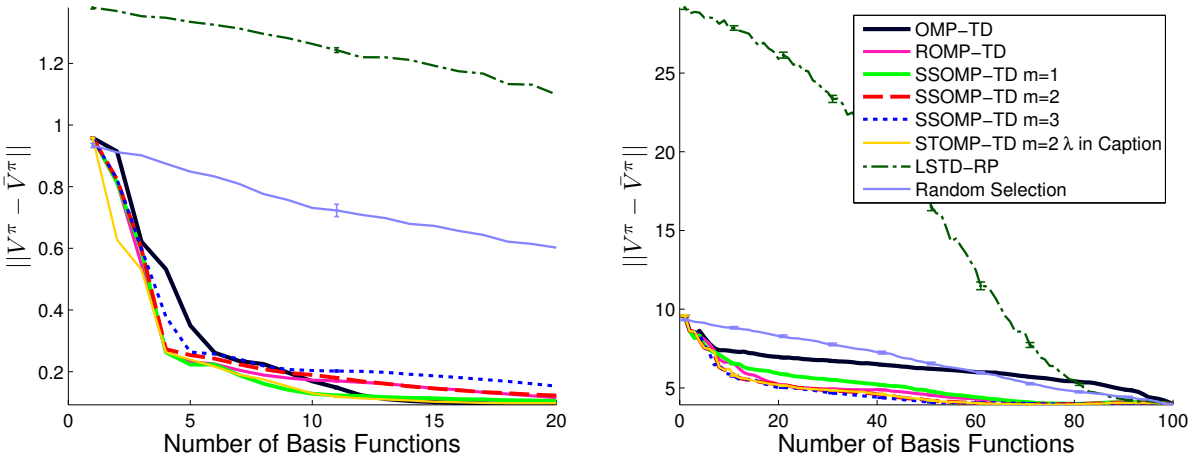
(e) RBF RC Car $\lambda = 4.0$

(f) RBF Acrobot $\lambda = 2.0$

Figure 4.4: Experimental results for the RBF basis on the chainwalk, mountain car, mountain car 3d, pinball, rc car and acrobot domains (a-f).

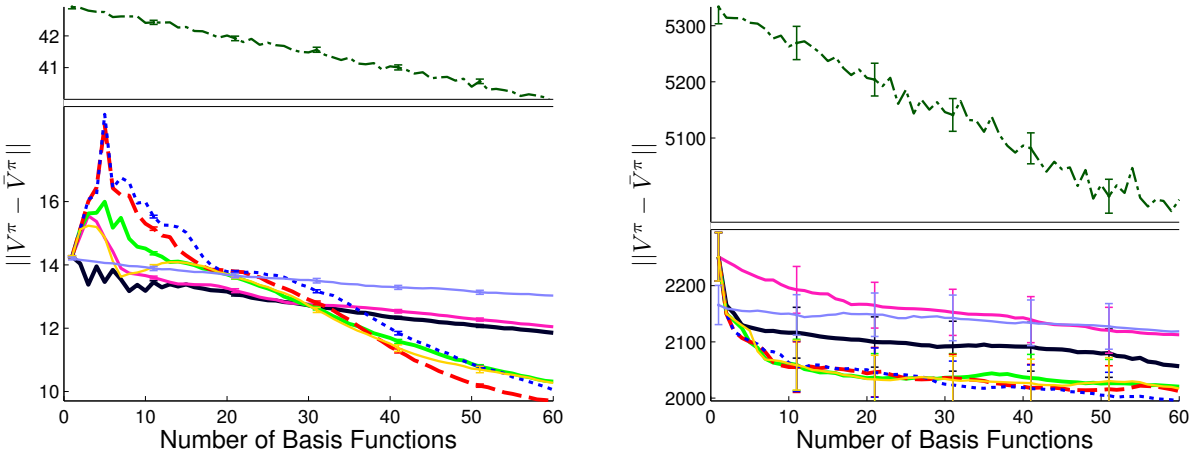
experiment where every basis function in the dictionary was selected. In that experiment its performance matched that of the other methods near the end, indicating that LSTD-RP is negatively affected by poor basis functions in the dictionary, which dilute the resulting projection.

The spikes in the graphs, particularly in Figure 4.5 (c) the mountain car 3d domain with Fourier basis functions, are the result of fitting problems with LSTD. We verified this by plotting ΠV^π, the projection



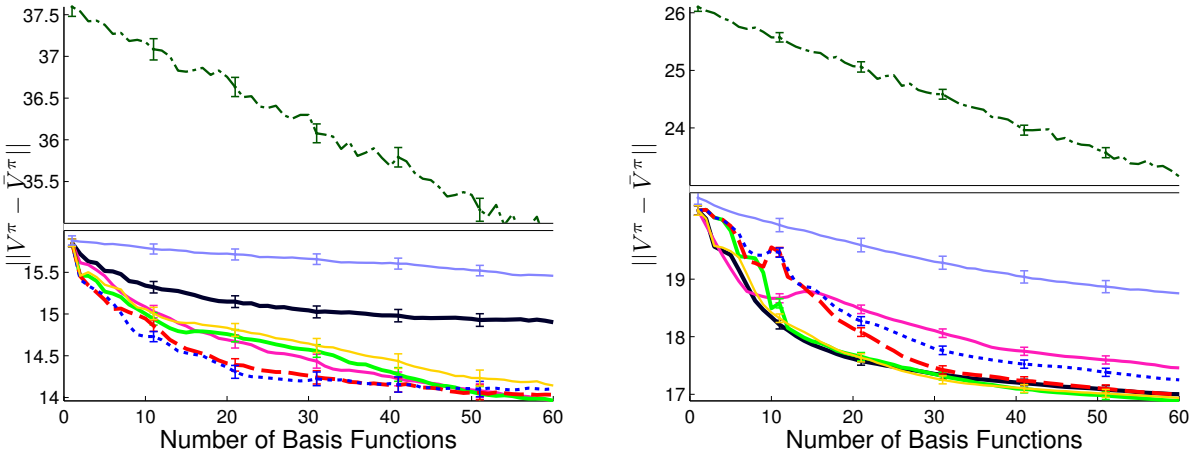
(a) Fourier Chainwalk $\lambda = 0.00001$

(b) Fourier Mountain Car $\lambda = 10.0$



(c) Fourier Mountain Car 3D $\lambda = 10.0$

(d) Fourier Pinball $\lambda = 10.0$



(e) Fourier RC Car $\lambda = 10.0$

(f) Fourier Acrobot $\lambda = 0.1$

Figure 4.5: Experimental results for the Fourier basis on the chainwalk, mountain car, mountain car 3d, pinball, rc car and acrobot domains (a-f).

of the true value function into the basis set of each method. The results for mountain car 3d domain with the Fourier basis can be seen in Figure 4.6. As expected, the error for each method decreased monotonically and the relative performance of the methods was similar.

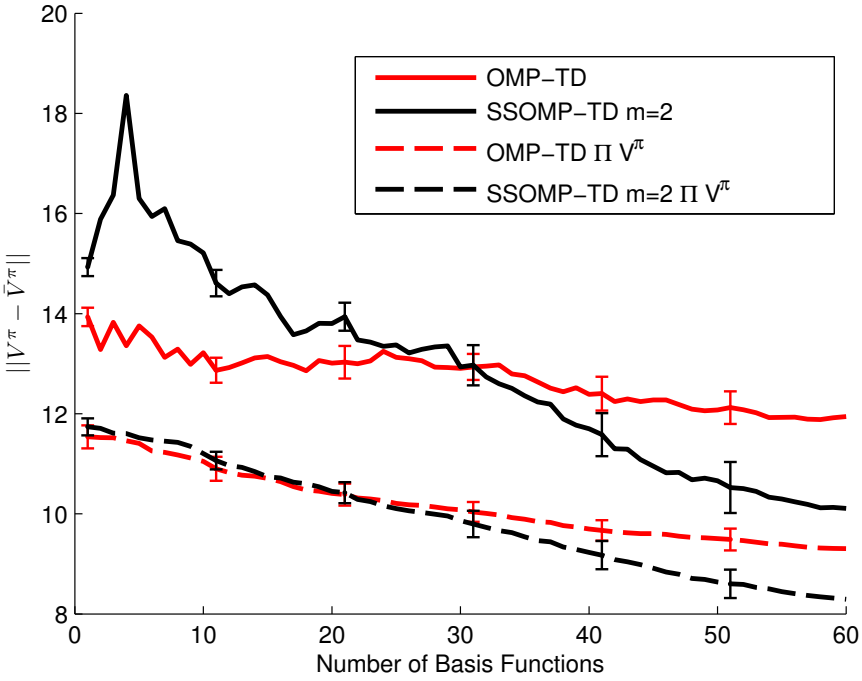


Figure 4.6: LSTD error and least squares error in mountain car 3d with the Fourier basis.

4.4 Summary and Conclusions

Current greedy feature selection algorithms such as OMP-TD are not able to take advantage of knowledge about the underlying value function and simply select basis functions which best fit the sample data. This may lead to the selection of more basis functions than necessary. The introduction of prior knowledge can improve performance. One such prior is the smoothness prior which can be integrated into the selection process via regularisation.

We have presented three ways to regularise feature selection in reinforcement learning, ROMP-TD, STOMP-TD and SSOMP-TD. Our results using these methods show that feature regularisation is a simple yet powerful tool for improving performance without significant computational overhead. Further, the smoothness prior generalises the intuition that wide/low frequency basis functions are better when only a few basis functions can be selected, and shows continued empirical improvements well beyond the first few basis functions.

Chapter 5

Online Feature Selection

5.1 Introduction

Online learning algorithms such as Sarsa(λ) use action-value functions $Q(s, a)$ instead of value functions $V(s)$. While you can think of $Q(s, a)$ as a single function dependent on both the parameters s and a, in practice there is a single $Q^a(s)$ function for each action a in the discrete action case. When each Q^a is optimal ($Q^a = Q^{*a}$), determining the best action to take in a state s is simply a matter of evaluating each Q^{*a} at that state, and choosing the action relating to the one with the largest value. When the Q are not optimal, performance depends on how well the actions with the largest action-value at each state $\arg\max_a Q^a(s)$ correspond to the actions with the largest optimal action-value $\arg\max_a Q^{*a}(s)$ at those states. What matters is that the correct action is chosen at each state, and if the agent's policy is to choose the action with the largest value at each state, then many action-value function sets could generate optimal policies with incorrect values.

Solving for action-value functions that have the same best action at each state as the optimal action-value functions is difficult. We usually assume that the best action-value function Q^a for a specific action a is the one that minimises the $\mu\pi_a$-weighted L_2 distance

$$||Q^{*a} - Q^a||_{\mu\pi_a} = \sqrt{\sum_s \mu(s)\pi(s, a)(Q^{*a}(s) - Q^a(s))}, \tag{5.1}$$

between it and the optimal action-value function Q^{*a}, where $\mu\pi_a$ is a weighting that favours states visited more often. Here, μ is the stationary distribution of the probability transition model defined in Section 5.2 and $\pi(s, a)$ is the probability of selecting action a in state s using policy π. The action-value functions that minimise that distance are the optimal value functions themselves, however action-value functions even a small distance away can have poor policies. In practice this is usually not a problem, and minimising the weighted L_2 distance $||Q^{*a} - Q^a||_{\mu\pi_a}$ remains the goal of many learning algorithms in reinforcement learning.

As with value functions V, linear function approximation is commonly used to approximate action-value Q functions. Unfortunately much of the theory relating to value functions has not been explicitly extended to the action-value case. In particular, Bellman error basis functions [Parr *et al.* 2007] (BEBFs) are specific to value functions approximated with a single linear function approximator.

In online settings, where learning takes place while the agent interacts with the environment, there are some additional questions:

- How should basis functions be selected?
- How do we deal with a non-stationary policy/value function?
- When do we select basis functions?

We tackle the problem of basis function selection by extending the theory of BEBFs to the action-value Q case. In particular we introduce Q Bellman error basis functions (QBEBFs) and show that they retain many of the theoretical guarantees as BEBFs.

The problem of non-stationary policies/value functions occurs because when learning online, policies based on the current value of states (ϵ-greedy for example) change as the agent learns. This is a problem for feature selection algorithms, as a changing policy means that the value function the agent is trying to

learn is also changing. This problem is particularly acute for feature selection algorithms, where the best features to choose are the ones that best fit the value function.

Our solution stores estimates of the correlation to the QBEBF using an exponentially weighted average with a weighting tied to the learning rate of Sarsa(λ). This gives correlation estimates to each QBEBF that change as the QBEBF/policy changes.

Finally, we use a simple strategy of selecting basis functions at the end of each episode, which provides an effective way to measure the effectiveness of the BEBF and QBEBF feature selection metrics.

We evaluate QBEBFs and BEBFs empirically across 6 domains using Sarsa(λ), with results showing that both selection methods work well in the online setting.

5.2 Contraction for Q

The theory behind why BEBFs improve the set of basis functions approximating the value function V relies on that fact that the Bellman operator is a contraction in the μ-weighted norm, where μ is the stationary distribution of the transition probability model P. In order to show a similar result for action-value functions Q, a similar contraction result is required. To show this we require Jensen's inequality.

Theorem 5.2.1 (Jensen's Inequality)**.** *Let f be a real function, convex on the interval I. Let $x_1, \ldots, x_n$ be points in I with associated non-negative weights $b_1, \ldots, b_n$ where $\sum_i b_i > 0$, then $f((\sum_i b_i x_i)/\sum_i b_i) \leq (\sum_i b_i f(x_i))/\sum_i b_i$.*

In particular, we note that the function $f(x) = x^2$ is convex.

We recall the following. Let P^a be the transition matrix for action a, where $P^a_{ss'}$ is the probability of transitioning from state s to state s' with action a. Let P^π be the transition matrix for policy π where $P^\pi_{ss'}$ is the probability of transitioning from state s to state s' using an action selected by π, then $P^\pi_{ss'} = \sum_a \pi(s,a) P^a_{ss'}$.

Let μ be the stationary distribution of P^π, i.e. $\mu P^\pi = \mu \Leftrightarrow \sum_s \mu(s) P^\pi_{ss'} = \mu(s')$. Let $R(s,a)$ be the expected reward for taking action a in state s, that is $R(s,a) = \sum^a_{ss'} P^a_{ss'} R(s,a,s')$. The value function V^π for a policy π satisfies the Bellman equation, that is:

$$V^\pi(s) = \sum_a \pi(s,a) Q^\pi(s,a), \tag{5.2}$$

where $Q^\pi(s,a)$ is the solution to the Bellman equation for Q:

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'} P^a_{ss'} V^\pi(s'). \tag{5.3}$$

The Bellman operator for policy π, T^π, is defined as:

$$(T^\pi V)(s) = \sum_a \pi(s,a) T^\pi_Q Q(s,a). \tag{5.4}$$

We define the Bellman operator for Q, T^π_Q as:

$$(T^\pi_Q Q)(s,a) = R(s,a) + \gamma \sum_{s'} P^a_{ss'} V(s'). \tag{5.5}$$

T^π for any policy π is a contraction in the maximum norm:

$$||T^\pi V_1 - T^\pi V_2||_\infty \leq \gamma ||V_1 - V_2||_\infty, \tag{5.6}$$

and is also a contraction in μ-weighted L_2 norm weighted by the stationary distribution μ [Van Roy 1998]:

$$||T^\pi V_1 - T^\pi V_2||_\mu \leq \gamma ||V_1 - V_2||_\mu, \tag{5.7}$$

where the weighted inner product between two vectors f and g with weighting μ is defined as:

$$\langle f, g \rangle_\mu = \sum_s \mu(s) f(s) g(s), \tag{5.8}$$

and the μ-weighted norm of f as:

$$||f||_\mu = \sqrt{\langle f, f \rangle_\mu}. \tag{5.9}$$

A similar contraction property can be shown for Q under weightings $\mu\pi_a$, where

$$\langle f, g \rangle_{\mu\pi_a} = \sum_s \mu(s)\pi(s,a) f(s) g(s). \tag{5.10}$$

Theorem 5.2.2. *Given a policy π, transition matrix P^π, and μ, the stationary distribution of P^π, for any value functions V_1 and V_2 and corresponding Q functions, the following inequalities hold:*

$$||T^\pi V_1 - T^\pi V_2||_\mu^2 \leq \sum_a ||T_Q^\pi Q_1(\cdot, a) - T_Q^\pi Q_2(\cdot, a)||_{\mu\pi_a}^2 \leq \gamma^2 ||V_1 - V_2||_\mu^2 \leq \gamma^2 \sum_a ||Q_1(\cdot, a) - Q_2(\cdot, a)||_{\mu\pi_a}^2. \tag{5.11}$$

In particular

$$||T^\pi V_1 - T^\pi V_2||_\mu^2 \leq \gamma^2 ||V_1 - V_2||_\mu^2, \tag{5.12}$$

shows that T^π is a contraction in the μ-weighted norm, and

$$\sum_a ||T_Q^\pi Q_1(\cdot, a) - T_Q^\pi Q_2(\cdot, a)||_{\mu\pi_a}^2 \leq \gamma^2 \sum_a ||Q_1(\cdot, a) - Q_2(\cdot, a)||_{\mu\pi_a}^2 \tag{5.13}$$

shows that a similar contraction property applies to T_Q^π.

Proof. The proof follows and extends the proof given by Van Roy [1998] using Jensen's inequality and the properties of the stationary distribution.

$$\begin{aligned}
||T^\pi V_1 - T^\pi V_2||_\mu^2 &= \left\| \sum_a \pi(s,a) T_Q^\pi Q_1(s,a) - \sum_a \pi(s,a) T_Q^\pi Q_1(s,a) \right\|_\mu^2 \\
&= \sum_s \mu(s) \left(\sum_a \pi(s,a) (T_Q^\pi Q_1(s,a) - T_Q^\pi Q_1(s,a)) \right)^2 \\
&\leq \sum_s \mu(s) \sum_a \pi(s,a) \left(T_Q^\pi Q_1(s,a) - T_Q^\pi Q_1(s,a) \right)^2 \text{ by Jensen's} \\
&= \sum_a ||T_Q^\pi Q_1(\cdot, a) - T_Q^\pi Q_2(\cdot, a)||_{\mu\pi_a}^2
\end{aligned}$$

$$
\begin{aligned}
&\sum_a ||T_Q^\pi Q_1(\cdot,a) - T_Q^\pi Q_2(\cdot,a)||^2_{\mu\pi_a} \\
&= \sum_s \sum_a \mu(s)\pi(s,a)\left(R(s,a) + \gamma P^a_{ss'} V_1(s') - R(s,a) - \gamma P^a_{ss'} V_2(s')\right)^2 \\
&= \sum_s \sum_a \mu(s)\pi(s,a)\left(\gamma P^a_{ss'} V_1(s') - \gamma P^a_{ss'} V_2(s')\right)^2 \\
&\leq \sum_s \sum_a \mu(s)\pi(s,a) P^a_{ss'}\left(\gamma V_1(s') - \gamma V_2(s')\right)^2 \text{ by Jensen's} \\
&= \sum_s \mu(s) P^\pi_{ss'}\left(\gamma V_1(s') - \gamma V_2(s')\right)^2 \text{ since } \textstyle\sum_a \pi(s,a) P^a_{ss'} = P^\pi_{ss'} \\
&= \sum_s \mu(s')\left(\gamma V_1(s') - \gamma V_2(s')\right)^2 \text{ by stationarity} \\
&= \gamma^2 ||V_1 - V_2||^2_\mu
\end{aligned}
$$

$$
\begin{aligned}
\gamma^2 ||V_1 - V_2||^2_\mu &= \gamma^2 \sum_s \mu(s) \left(\sum_a \pi(s,a)(Q_1(s,a) - Q_1(s,a)) \right)^2 \\
&\leq \gamma^2 \sum_s \mu(s) \sum_a \pi(s,a) \left(Q_1(s,a) - Q_1(s,a)\right)^2 \text{ by Jensen's} \\
&= \gamma^2 \sum_a ||Q_1(\cdot,a) - Q_2(\cdot,a)||^2_{\mu\pi_a}.
\end{aligned}
$$

□

The action-value contraction property $\sum_a ||T_Q^\pi Q_1(\cdot,a) - T_Q^\pi Q_2(\cdot,a)||^2_{\mu\pi_a} \leq \gamma^2 \sum_a ||Q_1(\cdot,a) - Q_2(\cdot,a)||^2_{\mu\pi_a}$ says that simultaneous applications of the Bellman operator for Q, T_Q^π, bring the Q values as a **whole** closer (in L_2 distance with the $\mu\pi_a$ weighting) to their Q^π values. Since it is a contraction, eventually all the $Q(\cdot,a)$ will converge to the $Q_\pi(\cdot,a)$.

It is worth noting that some $Q(\cdot,a)$ may move away from their $Q^\pi(\cdot,a)$ with a simultaneous application of T_Q^π, but at least one will move closer. If $Q(\cdot,a_k) = Q^\pi(\cdot,a_k)$ and $Q(\cdot,a_i) \neq Q^\pi(\cdot,a_i), \forall a_i \in A, i \neq k$ for example, an application of T_Q^π could move $Q(\cdot,a_k)$ away from $Q^\pi(\cdot,a_k)$. The contraction property however, tells us that there will be some improvement, so at least one of the remaining $Q(\cdot,a_i)$ must move closer to its $Q^\pi(\cdot,a_i)$.

5.2.1 QBEBFs

In this section we present Q Bellman error basis functions (QBEBFs), an extension of Bellman error basis functions [Parr *et al.* 2007] (BEBFs) for the case when the value function V is represented as a π-weighted sum of action-value functions Q, which are in turn represented as a weighted sum of basis functions.

When the value function V is represented as a single weighted sum of basis functions and fitted using an algorithm that solves for the fixed point $V = \Pi_\mu T^\pi V$, adding a BEBF $\phi = T^\pi V - V$ (the basis function equal to the Bellman error) to the approximation set for V is guaranteed to reduce the μ-weighted L_2 distance $||V^\pi - \Pi_\mu V^\pi||_\mu$ between V^π and $\Pi_\mu V^\pi$, where Π_μ is the μ-weighted L_2 projection into the span of Φ, the set of orthogonal basis functions used to approximate V. That is, if we have an algorithm that solves for the weights in a linear function approximation for V such as LSTD, then adding a BEBF to the approximation of V will allow this approximation set to better approximate V^π, even though V^π is unknown.

In the action-value case, this theoretical result no longer holds. This is because value functions represented as a weighted sum of basis functions can only represent a subset of the functions that action-value functions can given the same set of basis functions. Questions arise as to whether the action-value

functions can already approximate all or part of the BEBF, and whether the BEBF's contribution to the action-value functions improves their ability to represent V^π.

We present QBEBFs, BEBFs for each action-value function instead of one global BEBF, and extend the theory of BEBFs given in Parr *et al.* [2007] to QBEBFs.

We define the fixed point of V in the action-value case where $V(s) = \sum_a \pi(s,a)Q(s,a)$ to be

$$V = \sum_a \pi(s,a)\Pi^a_{\mu\pi} T^\pi_Q Q(s,a), \tag{5.14}$$

where $\Pi^a_{\mu\pi}$ is the $\mu\pi_a$-weighted L_2 projection into the span of Φ^a, the set of orthogonal basis functions specific to action a.

Given a set Φ^a of k QBEBFs specific to Q^a, we define the $(k+1)^{th}$ QBEBF as $\phi^a_{k+1}(s) = T^\pi_Q Q(s,a) - Q(s,a)$, where the $(k+1)^{th}$ BEBF ϕ_{k+1} is equal to the π-weighted sum of QBEBFs $\phi_{k+1}(s) = \sum_a \pi(s,a)\phi^a_{k+1}$.

Lemma 5.2.3. *If V is the fixed point solution of Equation* (5.14) *using basis sets $\Phi^a = \{\phi^a_1, \ldots, \phi^a_k\}$, then each QBEBF ϕ^a_{k+1} is orthogonal to the span of Φ^a under the $\mu\pi_a$ weighting.*

Proof. This follows directly from the definition of the fixed point of V in Equation (5.14), where each $Q(\cdot,a)$ is the $\mu\pi_a$-weighted orthogonal projection of $T^\pi_Q Q(\cdot,a)$ into the span of Φ^a. □

Theorem 5.2.4. *Let $V = \sum_a \pi(s,a)Q(s,a)$ be the π-weighted sum of Q functions, each of which is the fixed point solution weighted by $\mu(s)\pi(s,a)$ using the sequence of QBEBFs: $\phi^a_1, \ldots, \phi^a_k$. If*

$$\sqrt{\sum_a ||Q^\pi(\cdot,a) - Q(\cdot,a)||^2_{\mu\pi_a}} - \sqrt{\sum_a ||Q^\pi(\cdot,a) - T^\pi_Q Q(\cdot,a)||^2_{\mu\pi_a}} = x,$$

then for the new BEBF, ϕ_{k+1}, composed of QBEBFs ϕ^a_{k+1} with $\Phi^{a'} = \{\Phi^a, \phi^a_{k+1}\}$ and corresponding $\Pi'^a_{\mu\pi}$, the improvement in the approximation is

$$\sqrt{\sum_a ||Q^\pi(\cdot,a) - \Pi^a_{\mu\pi} Q^\pi(\cdot,a)||^2_{\mu\pi_a}} - \sqrt{\sum_a ||Q^\pi(\cdot,a) - \Pi'^a_{\mu\pi} Q^\pi(\cdot,a)||^2_{\mu\pi_a}} = y \geq x.$$

Proof. The proof is a straightforward adaptation of the proof presented in Parr *et al.* [2007].

The contraction property of T^π_Q guarantees $x > 0$ for $V \neq V^\pi$.

Using the identity

$$c - d = \frac{c^2 - d^2}{c + d}, \quad c + d \neq 0, \tag{5.15}$$

we can expand x,

$$\begin{aligned} x &= \sqrt{\sum_a ||Q^\pi(\cdot,a) - Q(\cdot,a)||^2_{\mu\pi_a}} - \sqrt{\sum_a ||Q^\pi(\cdot,a) - T^\pi_Q Q(\cdot,a)||^2_{\mu\pi_a}} \\ &= \frac{\sum_a ||Q^\pi(\cdot,a) - Q(\cdot,a)||^2_{\mu\pi_a} - \sum_a ||Q^\pi(\cdot,a) - T^\pi_Q Q(\cdot,a)||^2_{\mu\pi_a}}{\sqrt{\sum_a ||Q^\pi(\cdot,a) - Q(\cdot,a)||^2_{\mu\pi_a}} + \sqrt{\sum_a ||Q^\pi(\cdot,a) - T^\pi_Q Q(\cdot,a)||^2_{\mu\pi_a}}}, \end{aligned} \tag{5.16}$$

and expand y,

$$\begin{aligned} y &= \sqrt{\sum_a ||Q^\pi(\cdot,a) - \Pi^a_{\mu\pi} Q^\pi(\cdot,a)||^2_{\mu\pi_a}} - \sqrt{\sum_a ||Q^\pi(\cdot,a) - \Pi'^a_{\mu\pi} Q^\pi(\cdot,a)||^2_{\mu\pi_a}} \\ &= \frac{\sum_a ||Q^\pi(\cdot,a) - \Pi^a_{\mu\pi} Q^\pi(\cdot,a)||^2_{\mu\pi_a} - \sum_a ||Q^\pi(\cdot,a) - \Pi'^a_{\mu\pi} Q^\pi(\cdot,a)||^2_{\mu\pi_a}}{\sqrt{\sum_a ||Q^\pi(\cdot,a) - \Pi^a_{\mu\pi} Q^\pi(\cdot,a)||^2_{\mu\pi_a}} + \sqrt{\sum_a ||Q^\pi(\cdot,a) - \Pi'^a_{\mu\pi} Q^\pi(\cdot,a)||^2_{\mu\pi_a}}}. \end{aligned} \tag{5.17}$$

Since the denominator in Equation (5.16) is greater than or equal to the denominator in Equation (5.17), all that remains is to show that the numerator in Equation (5.16) is less than or equal to the numerator in Equation (5.17) to show that $y \geq x$.

By the orthogonality of the basis,

$$\Pi^a_{\mu\pi} Q^\pi(\cdot, a) = \sum_{i=1}^{k} w_i^a \phi_i^a,$$

and

$$\Pi'^a_{\mu\pi} Q^\pi(\cdot, a) = \sum_{i=1}^{k+1} w_i^a \phi_i^a.$$

For $Q(s,a) = \sum_{i=1}^{k} \alpha_i^a \phi_i^a$,

$$Q^\pi(s,a) - Q(s,a) = \sum_{i=1}^{k} (w_i^a - \alpha_i^a)\phi_i^a + \sum_{i=k+1}^{n} w_i^a \phi_i^a.$$

Since for each s, Φ^a is an orthogonal basis:

$$||Q^\pi(\cdot,a) - Q(\cdot,a)||^2_{\mu\pi_a} = \sum_{i=1}^{k} (w_i^a - \alpha_i^a)^2 ||\phi_i^a||^2_{\mu\pi_a} + \sum_{i=k+1}^{n} (w_i^a)^2 ||\phi_i^a||^2_{\mu\pi_a}, \tag{5.18}$$

and as $\phi^a_{k+1} = T^\pi_Q Q(\cdot,a) - Q(\cdot,a)$,

$$\begin{aligned} ||Q^\pi(\cdot,a) - T^\pi_Q Q(\cdot,a)||^2_{\mu\pi_a} &= ||Q^\pi(\cdot,a) - Q(\cdot,a) - \phi^a_{k+1}||^2_{\mu\pi_a} \\ &= \sum_{i=1}^{k} (w_i^a - \alpha_i^a)^2 ||\phi_i^a||^2_{\mu\pi_a} + (w^a_{k+1} - 1)^2 ||\phi^a_{k+1}||^2_{\mu\pi_a} \\ &+ \sum_{i=k+2}^{n} (w_i^a)^2 ||\phi_i^a||^2_{\mu\pi_a}. \end{aligned} \tag{5.19}$$

The numerator of x can therefore be written as

$$\begin{aligned} &\sum_a ||Q^\pi(\cdot,a) - Q(\cdot,a)||^2_{\mu\pi_a} - \sum_a ||Q^\pi(\cdot,a) - T^\pi_Q Q(\cdot,a)||^2_{\mu\pi_a} \\ &= \sum_a \left((w^a_{k+1})^2 - (w^a_{k+1} - 1)^2 \right) ||\phi^a_{k+1}||^2_{\mu\pi_a}. \end{aligned} \tag{5.20}$$

Similarly, the numerator of y is

$$\begin{aligned} &\sum_a ||Q^\pi(\cdot,a) - \Pi^a_{\mu\pi} Q^\pi(\cdot,a)||^2_{\mu\pi_a} - \sum_a ||Q^\pi(\cdot,a) - \Pi'^a_{\mu\pi} Q^\pi(\cdot,a)||^2_{\mu\pi_a} \\ &= \sum_a (w^a_{k+1})^2 ||\phi^a_{k+1}||^2_{\mu\pi_a}. \end{aligned} \tag{5.21}$$

Comparing equations (5.20) and (5.21), it is easy to see that the numerator of x is less than or equal to the numerator of y. Hence the result follows.

□

Theorem 5.2.4 says that if we simultaneously add the QBEBF, ϕ^a_{k+1}, for each action a to the appropriate approximation set Φ^a, then the new approximation sets, $\Phi^{a'} = [\Phi^a, \phi^a_{k+1}]$, will be able to approximate their $Q^\pi(., a)$ better (under the $\mu\pi_a$-weighted norm).

In practice QBEBFs must often be approximated. A similar result to that of BEBFs can likely be obtained where a basis function sufficiently correlated to the BEBF would also guarantee improvement. In the case of QBEBFs, each approximate QBEBF would need to be sufficiently correlated to the QBEBF it is approximating. We leave this result to future work.

5.3 Exponentially Weighted Correlation

Since the policy in an online setting is non-stationary, the Bellman error changes as learning takes place. Keeping an approximation of the Bellman error itself is not practical. Instead, we can store an estimate of the correlation to the Bellman error for each basis function.

The correlation $\rho_i = \langle \phi_i, \vec{B}_t \rangle / ||\phi_i||$ of a basis function ϕ_i to the Bellman error, or Bellman error for Q, $\vec{B}_t = [B_0, \dots, B_t]$ evaluated at states $s_0, \dots, s_t$ is derived by solving

$$\begin{aligned}(\phi_i, w_i) &= \arg \min_{\phi \in D, w \in \mathbb{R}} ||\vec{B}_t - w\phi||^2 \\ &= \arg \min_{\phi \in D, w \in \mathbb{R}} \frac{1}{t+1} \sum_{j=0}^{t} (B_j - w\phi(s_j))^2,\end{aligned} \tag{5.22}$$

where the ϕ_i that minimises Equation (5.22) is the one with the largest ρ_i. In the online setting, recent samples of the Bellman error should be weighted higher as they are more accurate samples of the current Bellman error.

Given samples of a function f, $f(s_0), \dots, f(s_t)$, the exponentially weighted average F_t of the first $t+1$ samples with step size α is

$$\begin{aligned}F_t &= F_{t-1} + \alpha(f(s_t) - F_{t-1}) \\ &= \sum_{j=0}^{t} \alpha(1-\alpha)^{t-j} f(s_j).\end{aligned} \tag{5.23}$$

This weighting gives larger weights to more recent samples, with the impact of older samples decaying at an exponential rate for constant $\alpha \in (0, 1)$.

If $f(s_i) = (B_i - w\phi(s_i))^2$, then the exponentially weighted version of Equation (5.22) with step size α is

$$\begin{aligned}(\phi_i, w_i) &= \arg \min_{\phi \in D, w \in \mathbb{R}} \sum_{j=0}^{t} \alpha(1-\alpha)^{t-j} (B_j - w\phi(s_j))^2 \\ &= \arg \min_{\phi \in D, w \in \mathbb{R}} ||\vec{B}_t - w\phi||^2_{\alpha_t},\end{aligned} \tag{5.24}$$

where $||.||_{\alpha_t}$ is the corresponding weighted norm.

Given a basis function ϕ, the optimal weight w for that ϕ using weighting α_t is

$$\begin{aligned}0 &= \frac{\partial ||\vec{B}_t - w\phi||^2_{\alpha_t}}{\partial w} \\ 0 &= -2\langle \vec{B}_t, \phi \rangle_{\alpha_t} + 2w||\phi||^2_{\alpha_t} \\ w &= \frac{\langle \vec{B}_t, \phi \rangle_{\alpha_t}}{||\phi||^2_{\alpha_t}}.\end{aligned} \tag{5.25}$$

Substituting the optimal w in, the best ϕ is the one that minimises the following equation:

$$\begin{aligned} ||\vec{B}_t - w\phi||^2_{\alpha_t} &= \left|\left|\vec{B}_t - \frac{\langle \vec{B}_t, \phi \rangle_{\alpha_t}}{||\phi||^2_{\alpha_t}}\phi\right|\right|^2_{\alpha_t} \\ &= ||\vec{B}_t||^2_{\alpha_t} - \left(\frac{\langle \vec{B}_t, \phi \rangle_{\alpha_t}}{||\phi||_{\alpha_t}}\right)^2, \end{aligned} \tag{5.26}$$

which can be minimised by maximising

$$\rho_{\alpha_t} = \left|\frac{\langle \vec{B}_t, \phi \rangle_{\alpha_t}}{||\phi||_{\alpha_t}}\right|. \tag{5.27}$$

To keep an online estimate of $\rho_{i\alpha_t}$ for each basis function ϕ_i, we break the numerator and denominator into separate parts and use the update rule formulation of the exponentially weighted average:

$$\rho_{i\alpha_t} = \frac{\rho^{num}_{i\alpha_t}}{\sqrt{\rho^{den}_{i\alpha_t}}}, \tag{5.28}$$

where

$$\rho^{num}_{i\alpha_t} = \rho^{num}_{i\alpha_{t-1}} + \alpha(B_t\phi(s_t) - \rho^{num}_{i\alpha_{t-1}}) \tag{5.29}$$

and

$$\rho^{den}_{i\alpha_t} = \rho^{den}_{i\alpha_{t-1}} + \alpha(\phi(s_t)^2 - \rho^{den}_{i\alpha_{t-1}}). \tag{5.30}$$

The update equations in Equations (5.29) and (5.30) give us a way to update the correlation estimates, $\rho_{i\alpha}$ in Equation (5.28), of basis functions ϕ_i iteratively. In the online setting, as each new sample of the Bellman error arrives, each basis function in the dictionary can be updated with a new estimate of its correlation to the Bellman error. In this way, the $\rho_{i\alpha}$ form moving estimates of the correlation to the Bellman error as it changes. Choosing the step size α is critical. Too low and the estimates will not adapt quickly enough. Too high and the estimates will only take into account the most recent samples of the Bellman error, not covering the state space sufficiently. A logical choice could be to choose the step size as the algorithm's learning rate. That way, the correlation estimates ρ_{α_i} change at the same rate as the value or action-value functions do.

5.4 Online Feature Selection Algorithms

We present two algorithms for online feature selection with action-value functions based on Sarsa(λ). The first uses QBEBFs (Algorithm 7) and the second uses BEBFs (Algorithm 8). The algorithms differ from Sarsa(λ) in that basis functions are added to the approximation at the end of each episode, based on samples collected with Sarsa(λ). Both algorithms maintain dictionaries of candidate basis functions, and update the correlations to the Bellman error of the basis functions within, selecting the basis functions with the largest regularised or unregularised correlation to the Bellman error at the end of each episode.

QBEBF Sarsa(λ) in Algorithm 7 maintains a dictionary for each action (line 4). Each time an action a is taken and a sample (s, a, r, s') obtained, only the dictionary relating to action a is updated (line 24). The online correlations of each basis function therefore estimate the correlation to the QBEBF for their associated action. Since the samples are collected on policy, the correlation estimates approximate the $\mu\pi_a$-weighted correlation. At the end of each episode the basis function most correlated to the QBEBF for that action is added to the Q function for that action (line 29).

BEBF Sarsa(λ) in Algorithm 8 maintains a single dictionary (line 4), and uses every sample obtained to update the correlations of every basis function in the dictionary (line 24). This results in an approximation

Domain	γ	λ	Initial Order	Initial BFs	Max Order	Dictionary BFs
Mountain Car	0.999	0.9	4	9	20	432
Mountain Car 3D	0.999	0.9	2	9	10	14632
Pinball Normal	0.999	0.9	2	9	10	14632
Pinball Diagonal	0.999	0.9	4	17	10	14624
RC Car	0.999	0.9	10	41	10	14600
Acrobot	0.999	0.9	2	9	10	14632

Table 5.1: Discount γ, eligibility trace decay λ, number of basis functions initially in the approximation set, initial dictionary size for each experiment using the Fourier basis.

of the μ-weighted correlation to the BEBF. At the end of each episode the most correlated basis function is added to the Q for each action (line 29). Unlike QBEBFs, we were not able to prove that adding a BEBF to each Q^a would improve the representation.

The correlation update algorithm in Algorithm 9 updates an online estimate of the correlations of each basis function in the given dictionary to the Bellman error. Correlations are stored in 2 parts, the numerator ϕ_i^{num} and the square of the denominator ϕ_i^{den}, and are updated using an α-weighted average (Section 5.3). The algorithm first calculates the Bellman error for the sample (lines 9 to 13) and then updates the estimates for ϕ_i^{num} and ϕ_i^{den} for each basis function in the dictionary D.

At the end of each episode, basis functions are selected in both algorithms with optional smoothness regularisation (Algorithm 10). The algorithm simply loops through the basis functions in the dictionary, calculates its estimated regularised or unregularised correlation, and returns the most correlated basis function.

One alternative to selecting basis functions at the end of each episode is to automatically select a basis function once its correlation has reached a certain threshold. Unfortunately choosing such a threshold is difficult, with poor choices leading to the selection of the entire dictionary or no selection at all. We avoid the problem of when best to select basis functions in order to demonstrate the performance of the QBEBF and BEBF selection metrics independently.

In our experiments we used an automatic learning rate adjusting method [Dabney and Barto 2012] which we combined with the α scaling from the Fourier basis [Konidaris *et al.* 2011]. More details on combining the two are contained in the appendix.

5.5 Experiments

In the experiments we compared different basis function selection methods in the online setting using Sarsa(λ) as the learning algorithm. For each method we initialised each Q function with an order n independent Fourier basis where n is given in Table 5.1 under 'Initial Order'. All methods learned using Sarsa(λ) with the only difference being their feature selection method. At the end of each episode up until episode 250, each method added up to one basis function to each of its Q functions, and the return for that episode recorded. The returns for each episode were averaged over 500 runs and plotted in Figure 5.2. The parameters used in each experiment are given in Table 4.2. We used Sarsa(λ) with an ϵ-greedy policy ($\epsilon = 0.01$).

5.5.1 Methods

BEBF: The BEBF method used Algorithm 8 to select its features, where the samples collected for each action are combined and used to select a single basis function ϕ which is added to the Q functions for each of the actions.

Algorithm 7 QBEBF Sarsa(λ)

Input:
Q initialised arbitrarily,
Set of all actions $A = \{a_1, ..., a_k\}$,
Dictionaries D_i, $i \in A$,
Policy $\pi(s, a)$ derived from Q (epsilon greedy for example),
$\gamma \in [0, 1)$,
$\lambda \in [0, 1)$,
Output:
Q.

$\alpha = 1$
repeat {for each episode}
  $\vec{e} = \vec{0}$
  $s \leftarrow$ environment initialised state
  $a \leftarrow$ action selected with policy π at state s
  repeat {for each step in episode}
    Take action a, observe reward r and state s'
    $a' \leftarrow$ action selected with policy π at state s'
    $\vec{e} \leftarrow \lambda\vec{e}$
    $\vec{e} \leftarrow$ Update eligibility trace
    $\alpha \leftarrow$ Update learning rate
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
    $Q \leftarrow Q + \alpha\delta\vec{e}$
    Call Algorithm 9 with $D_a, s, a, r, s', \pi, \gamma, \alpha$ (Update Correlations)
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is a terminal state
  for each $a'' \in A$ **do**
    $\phi \leftarrow$ call Algorithm 10 with $D_{a''}$ (Select Most Correlated)
    add ϕ to Q for action a''
  end for
until

Regularised BEBF: The same as the BEBF method, however SSOMP-TD regularisation is added to the correlation calculation in order to determine which basis function is selected and added to each Q function.

Random: The random method had a separate dictionary for each action. At the end of each episode a random basis function was selected from each dictionary and added to the corresponding Q function. This method served as a control.

QBEBF: The QBEBF method used Algorithm 7 to select its features. Each of the Q functions has its own dictionary. The correlations of basis functions in those dictionaries are only updated when a sample using the action which corresponds to that Q function is obtained. At the end of each episode, the most correlated basis function from each of the dictionaries is added to the corresponding Q function.

Regularised QBEBF: The same as the QBEBF method, however SSOMP-TD regularisation is added to the correlation calculation in order to determine which basis functions are selected and added to each Q function.

No Selection: This method selected no additional basis functions.

Algorithm 8 BEBF Sarsa(λ)

Input:
Q initialised arbitrarily,
Set of all actions $A = \{a_1, ..., a_k\}$,
Dictionary D,
Policy $\pi(s, a)$ derived from Q (epsilon greedy for example),
$\gamma \in [0, 1)$,
$\lambda \in [0, 1)$
Output:
Q.

$\alpha = 1$
repeat {for each episode}
  $\vec{e} = \vec{0}$
  $s \leftarrow$ environment initialised state
  $a \leftarrow$ action selected with policy π at state s
  repeat {for each step in episode}
    Take action a, observe reward r and state s'
    $a' \leftarrow$ action selected with policy π at state s'
    $\vec{e} \leftarrow \lambda\vec{e}$
    $\vec{e} \leftarrow$ Update eligibility trace
    $\alpha \leftarrow$ Update learning rate
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
    $Q \leftarrow Q + \alpha\delta\vec{e}$
    Call Algorithm 9 with $D, s, a, r, s', \pi, \gamma, \alpha$ (Update Correlations)
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is a terminal state
  $\phi \leftarrow$ call Algorithm 10 with D (Select Most Correlated)
  for each $a'' \in A$ **do**
    add ϕ to Q for action a''
  end for
until

5.5.2 Domains

We used the same domains as in Chapter 3 with the exception of chainwalk, which we substituted for a variant of the pinball domain which has diagonal barriers. The domains: mountain car, mountain car 3d, pinball with standard configuration, pinball with diagonal configuration, RC car and acrobot, are shown in Figure 5.1.

5.5.3 Results

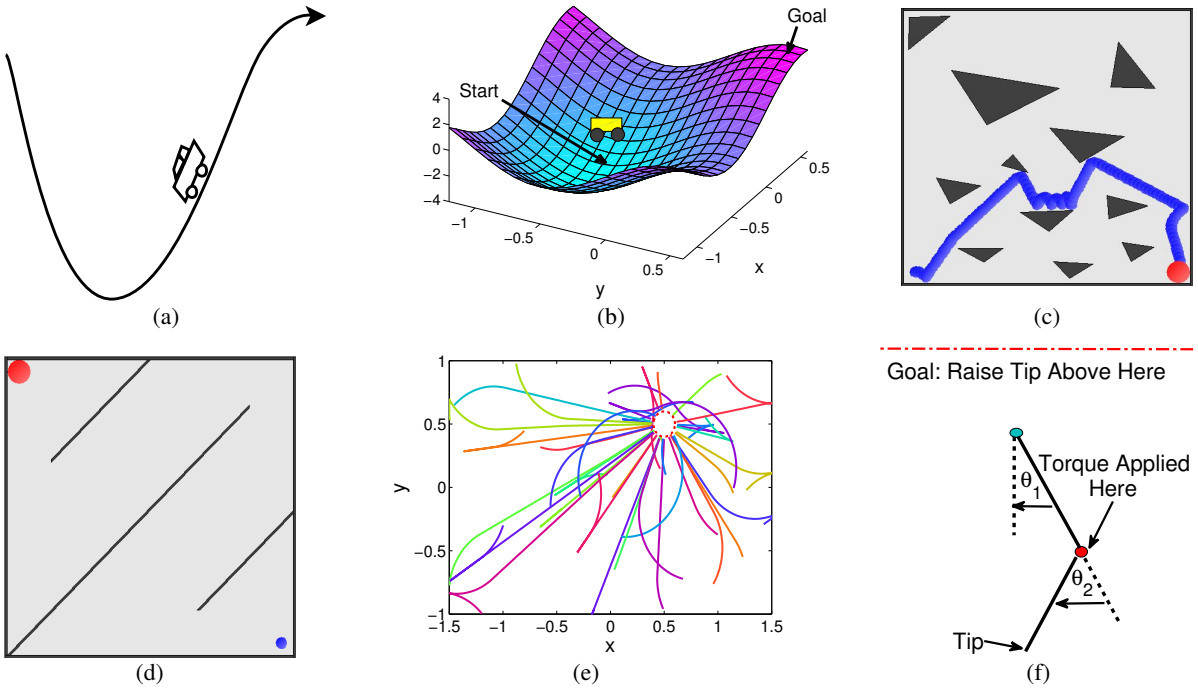
The results are shown in Figure 5.2. Of particular interest are the QBEBF and BEBF methods, which always perform better than the random method by the end of the experiment. These methods appear to be selecting good basis functions, with QBEBFs performing better on 3d mountain car and acrobot, and BEBFs performing better on mountain car and diagonal pinball. This suggests that it may be possible to prove that the BEBF method is equally effective in the action-value case.

The big exception is the RC car experiment. 'No Selection' outperformed all the other methods, even at lower orders of approximation (2, 4, etc.) This is likely because a poor value function approximation does not necessarily imply poor performance. In fact, improving the approximation can make the performance

Algorithm 9 Update Correlations

Input:
Dictionary D
Sample (s, a, r, s'),π,γ,α
Each $Q(s, a)$ after learning from sample
Set of all actions A ,
Output:
Updated dictionary D_a.

$TQ \leftarrow r$
for each $a'' \in A$ **do**
  $TQ \leftarrow TQ + \pi(s, a'')\gamma Q(s', a'')$
end for
$B \leftarrow TQ - Q(s, a)$ (Estimate of Bellman error at state s)
for each $\phi_i \in D$ **do**
  $\rho_i^{num} \leftarrow \rho_i^{num} + \alpha(\phi(s)B - \rho_i^{num})$
  $\rho_i^{den} \leftarrow \rho_i^{den} + \alpha(\phi(s)^2 - \rho_i^{den})$
end for



(a) (b) (c) (d) (e) (f)

Figure 5.1: The domains used. Mountain car 2d (a). Mountain car 3d (b). The standard pinball configuration with sample trajectory (c). The diagonal pinball configuration (d). RC Car sample trajectories with random starts using our hand coded policy (e). Acrobot (f).

worse. It is more important that the best action has the largest Q value, rather than the Q values be correct.

The regularised method's performance varied a lot. In some instances (mountain car and acrobot), the regularised methods performed the best. In others (mountain car 3d, diagonal pinball and normal pinball), they were worse than random. It is unclear why this is the case. It could be a similar problem to that experienced in RC car, where an improved value function does not necessarily translate to an improved policy. It could also be due to the α-scaling used by the Fourier basis. α-scaling reduces the learning rate for high frequency basis functions, acting as a regulariser. It may be that choosing a QBEBF is

Algorithm 10 Select Most Correlated

Input:
Dictionary D,
Output:
m, the most correlated ϕ in D

best $\leftarrow -\infty$
for each $\phi_i \in D$ **do**
  if Smoothness Regularised **then**
    corr $\leftarrow \rho_i^{num} / \sqrt{\rho_i^{den} U_2(\phi)}$
  else
    corr $\leftarrow \rho_i^{num} / \sqrt{\rho_i^{den}}$
  end if
  if |corr| $>$ |best| **then**
    best $\leftarrow$ corr
    $m \leftarrow \phi$
  end if
end for
$D \leftarrow D \backslash \{m\}$

more likely to pay off in some domains, as the α-scaling reduces potential negative effects of poor high frequency basis functions. This may point to future work in the batch learning setting, where instead of regularising the fit with ridge regression or similar, some smoothness inducing regularisation could be used.

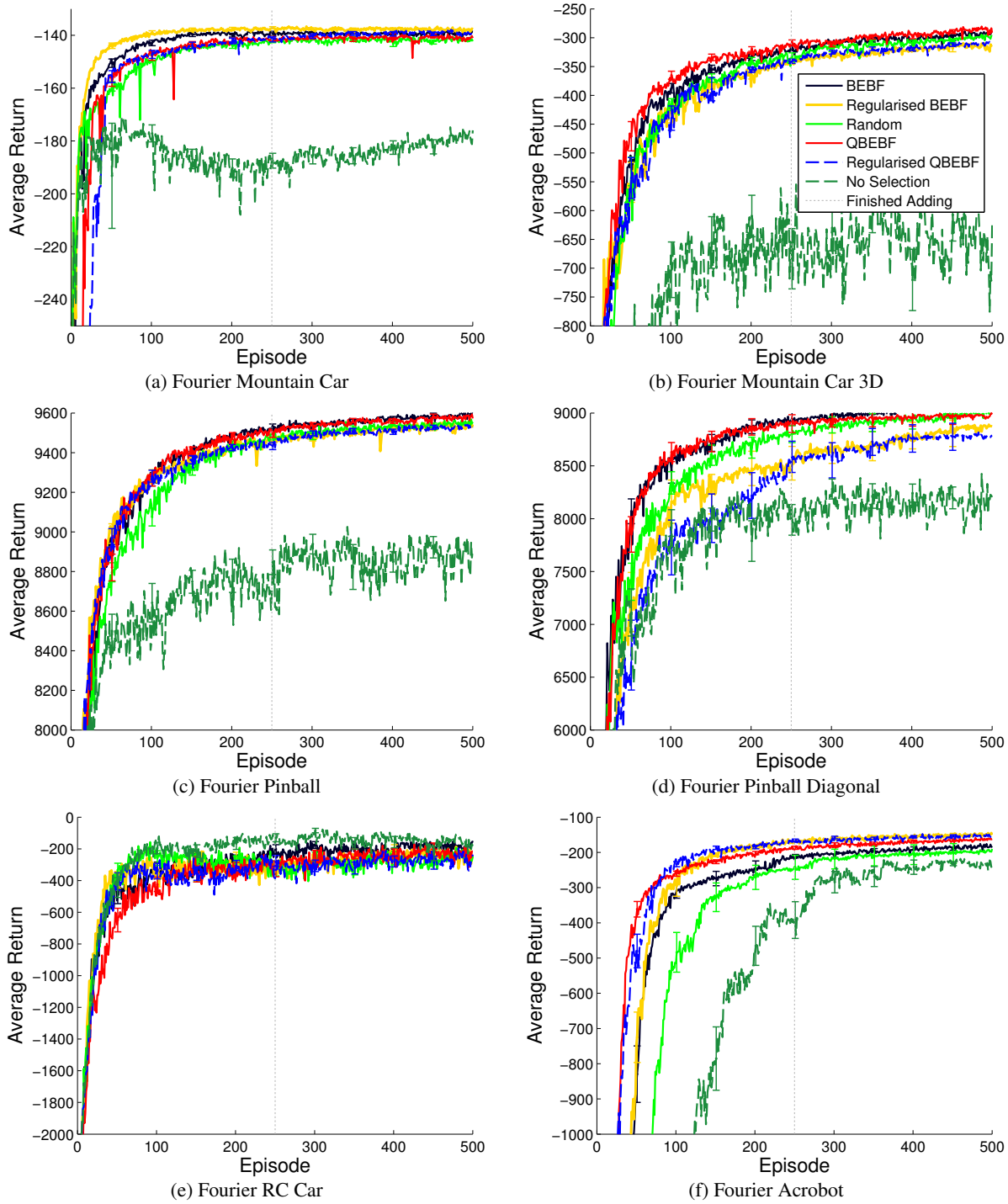
5.6 Conclusion

Online learning algorithms such as Sarsa(λ) use action-value functions. Like value functions, action-value functions are often represented with linear function approximators. Feature selection in this case is difficult, as new feature selection metrics are required, and online learning means policies can change as the agent learns.

Greedy batch algorithms such as OMP-TD use correlation to the Bellman error as their selection metric. This works because Bellman error basis functions (BEBFs), basis functions equal to the Bellman error, are guaranteed to improve the representational power of a linear function approximation for the value function. In the online case, with action-value functions instead of value functions, the theory behind this selection metric no longer holds. To prove similar theory for action-value functions, we showed a new contraction result for the Bellman operator for Q, and used it to prove that Q Bellman error basis functions (QBEBFs) have similar theoretical properties to BEBFs. In particular, adding a QBEBF to each action-value function simultaneously, guarantees an improvement in the representation.

Since representing QBEBFs, like BEBFs, is difficult, we approximated the QBEBFs by choosing the single most correlated basis function in a dictionary to approximate it. Normally, calculating the correlation would require a set of samples of the Bellman error sampled with a fixed policy, however in the online setting, it is often not practical to stop learning to collect these samples. We used an exponential weighted average instead to maintain approximate correlations of each basis function to the current BEBF or QBEBF.

Future work in online feature selection could include extending the theory of QBEBFs to include all the theoretical results in Parr *et al.* [2007], as well as attempting to find the relationship between BEBFs and QBEBFs, and when the use of each is appropriate.



(a) Fourier Mountain Car

(b) Fourier Mountain Car 3D

(c) Fourier Pinball

(d) Fourier Pinball Diagonal

(e) Fourier RC Car

(f) Fourier Acrobot

Figure 5.2: Online feature selection experimental results for the Fourier basis on the mountain car, mountain car 3d, pinball, diagonal pinball, RC car and acrobot domains (a-f).

In our experiments, we selected the most correlated basis functions at the end of each episode, however methods for deciding when to select basis functions would be an important step towards dealing with non-episodic domains as well as domains where selecting at the end of the episode may be too slow or too fast. We also combined a learning rate adjusting algorithm [Dabney and Barto 2012] with the exponential weighted average of correlations. While it makes sense to do so, there may be room for improvement.

The experiments highlight the difficulties of representing policies with value functions, where selecting

a basis function to improve the value function could lead to a worse policy. Despite this, the results with our Sarsa(λ) BEBF and Sarsa(λ) QBEBF algorithms showed that both methods are able to select relevant basis functions and improve performance, outperforming random selection in each of the experiments.

Chapter 6

Conclusion and Future Work

As reinforcement learning techniques move beyond theoretical problems to the real world, many difficult challenges present themselves. The problems faced include smaller data sets, larger state spaces, hidden state variables and learning in real time. Many of these problems are far more challenging than current methods are able to deal with, largely due to the high numbers of dimensions in their state spaces. Finding practical solutions to dealing with these state spaces is of key importance when solving real-life problems. Function approximation is an important tool to meet these challenges, particularly for value function representation in large or continuous state spaces.

Linear function approximation is a popular function approximation scheme used primarily because of its theoretical properties but also because of its simplicity and the ease of interpretation, especially when compared to rival methods like neural networks. This feature of linear function approximation is further enhanced when using a well understood fixed basis set.

Fixed basis sets work well in low dimensions, but practical problems arise when these sets are used to approximate higher-dimensional state spaces as they scale exponentially with the number of dimensions. Even a relatively low order approximation quickly grows out of hand as the the number of dimensions increases, creating a combinatorial explosion. Storing weights and doing calculations with this massive amount of basis functions is prohibitively difficult and is better avoided altogether.

Fortunately, the use of a full fixed basis set is often not necessary as many value functions can be represented by a relatively small subset of the full set. Choosing this set by hand is often difficult and impractical. As such, feature selection methods which automatically select features have received significant attention lately.

OMP-TD [Painter-Wakefield and Parr 2012] is a greedy feature selection algorithm which searches through a dictionary of basis functions and adds them to the linear function approximation. Determining which basis function to select is a difficult problem however, and often many poor basis functions are chosen, increasing the size of the approximation set unnecessarily and slowing convergence.

We presented a dictionary restriction method for scaling dictionary search methods such as OMP-TD up to high dimensional state spaces when using the Fourier basis. This simple method restricts the number of basis functions that OMP-TD is able to search through by populating the dictionary with only the most likely candidates. This is achieved using the heuristic that low frequency basis functions are more important. We later show that low frequency can also mean smooth, and that the dictionary restriction method can be used with any set of basis functions. Although many basis functions never become eligible for selection, our results indicate that the set of functions that are eligible contain many of the important basis functions for compactly representing the value function. Despite not being able to select from the full set of basis functions, the restricted dictionary with OMP-TD outperformed the full dictionary OMP-TD on some domains.

To improve the selection metric itself, we introduced two algorithms, Smoothness Scaled OMP-TD (SSOMP-TD) and Smooth Tikhonov OMP-TD (STOMP-TD) which modify the selection metric of OMP-TD, biasing selection towards smooth basis functions. Smooth basis functions are generally more likely to fit the underlying value function as it itself is often smooth. This often leads to improved performance, which we demonstrate across six benchmark domains. SSOMP-TD in particular works reliably well and has only one easily tunable parameter.

These methods work well in the batch setting, where samples are collected offline with a fixed policy. In the online setting where learning takes place as the agent moves through the environment, determining which basis function to select is a more difficult problem. We extend the theory behind OMP-TD's se-

lection metric to the action-value case and tackle the problem of non-stationary policies. Our algorithms Sarsa(λ) QBEBF and Sarsa(λ) BEBF successfully perform feature selection online despite these difficulties. While there is room for improvement, QBEBFs lay the ground-work for future online feature selection algorithms.

One possible direction for future study is to combine STOMP-TD and SSOMP-TD with state space restructuring methods such as Predictive Projections [Sprague 2009], which projects the original state space in such a way that neighbouring states in the state graph are closer together in the projected state space. There is also potential to use the smoothness prior in dictionary construction, to either reduce its size significantly in high dimensional domains, or to improve the quality of basis functions in a dictionary of fixed size. Randomly selected RBFs would be a natural choice, where a bias towards smooth RBFs could be included easily.

In our research, we extensively explored searching full dictionaries in sublinear time, however our algorithms were only approximate and not accurate enough to be effective. One algorithm was a branch and bound algorithm, using properties of the Fourier basis to find a lower bound on the highest correlation of a set of basis functions. The upper bound could be found approximately by filtering out higher frequencies in the data. Using these two bounds, an approximate $O(logn)$ algorithm could be used to find the most correlated basis function in high dimensional and high order fixed basis sets. A successful version of this algorithm would be an immediate improvement of OMP-TD where much larger dictionaries could be searched with reduced computational cost.

In this research only OMP-TD was used as a test-bed on which the dictionary and correlation modifications were tested. With the successes in OMP-TD, OMP-BRM may also benefit from these methods or similar. It may also be worth presenting these algorithms in the context of a framework like LSPI as in Parr *et al.* [2007].

Of the contributions presented, the online feature selection method and QBEBF theory hold the most promise for future work. The success of the dictionary restriction heuristic is also exciting. It not only achieved the goal of being a practical method for basis function selection in high-dimensional spaces, but it also outperformed a full dictionary search in some empirical tests using the same selection metric. Despite the success, there remains significant room for improvement and future work.

Chapter 7

Appendix

7.1 Bounding α with Scaling

When using the Fourier basis with methods that require a learning rate parameter α, it is common practice to scale the α value for each individual basis function. For the Fourier basis, the scaling parameter θ_i corresponding to the basis function ϕ_i is defined as:

$$\theta_i = \frac{1}{||\boldsymbol{c}_i||}, \tag{7.1}$$

where $\boldsymbol{c}_i$ is the parameter vector given in Equation (2.18). To avoid division by 0, let $\theta_0 = 1$ where ϕ_0 is the constant term parametrised by $\boldsymbol{c}_0 = [0, ..., 0]$.

Let α_t represent the global learning rate at time t, then the scaled learning rate for ϕ_i, $\vec{\alpha}_{t,i}$ is given as:

$$\vec{\alpha}_{t,i} = \alpha_t \theta_i, \tag{7.2}$$

The α bounding algorithm described in Dabney and Barto [2012] starts with an initial learning rate α and decreases it gradually, attempting to keep α at its largest possible value that does not cause divergence. The unscaled α bounding algorithm updates α before the weights are updated at each time step according to the formula:

$$\begin{aligned} \alpha_t &= \min[\alpha_{t-1}, [e_t^T(\gamma\phi_{t-1} - \phi_t)]^{-1}] \\ \alpha_0 &= 1.0, \end{aligned} \tag{7.3}$$

where e_t is the eligibility trace vector at time t. This update only occurs when $e_t^T(\gamma\phi_{t-1} - \phi_t) > 0$, otherwise $\alpha_t = \alpha_{t-1}$.

In gradient descent methods using a single learning rate, the parameters are updated using the formula,

$$w_{t+1} = w_t + \alpha_t \delta_t e_t, \tag{7.4}$$

where δ_t is the error at time t. In the scaled learning rate case, the formula becomes:

$$w_{t+1} = w_t + \alpha_t \delta_t (\theta \cdot e_t), \tag{7.5}$$

where θ is a constant vector of scaling parameters each corresponding to a particular ϕ_i according to Equation (7.1). Replacing $\theta \cdot e_t$ with e'_t gives us

$$w_{t+1} = w_t + \alpha_t \delta_t e'_t. \tag{7.6}$$

By using this definition of the gradient descent update, all instances of e_t in the α bounding derivation can be replaced by e'_t which results in the α update rule for the scaled case:

$$\begin{aligned} \alpha_t &= \min[\alpha_{t-1}, [(\theta \cdot e_t)^T(\gamma\phi_{t-1} - \phi_t)]^{-1}] \\ \alpha_0 &= 1.0, \end{aligned} \tag{7.7}$$

only when $(\theta \cdot e_t)^T(\gamma\phi_{t-1} - \phi_t) > 0$, otherwise $\alpha_t = \alpha_{t-1}$. Likewise the gradient descent update is only done after the α update.

References

[Benoudjit and Verleysen 2003] N. Benoudjit and M. Verleysen. On the kernel widths in radial-basis function networks. *Neural Processing Letters*, 18:139–154, 2003.

[Bradtke and Barto 1996] S. Bradtke and A. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1):33–57, 1996.

[Dabney and Barto 2012] W. Dabney and A. Barto. Adaptive step-size for online temporal difference learning. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 872–878, 2012.

[Ernst *et al.* 2005] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. In *Journal of Machine Learning Research*, volume 6, pages 503–556, 2005.

[Geist *et al.* 2012] M. Geist, B. Scherrer, A. Lazaric, and M. Ghavamzadeh. A Dantzig selector approach to temporal difference learning. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1399–1406, 2012.

[Geramifard *et al.* 2011] A. Geramifard, F. Doshi, J. Redding, N. Roy, and J. How. Online discovery of feature dependencies. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 881–888, 2011.

[Geramifard 2013] A. Geramifard. *RC-Car domain*. `http://acl.mit.edu/RLPy/api/domains\_misc.html\#rccar`, March 2013.

[Ghavamzadeh *et al.* 2010] M. Ghavamzadeh, A. Lazaric, O. Maillard, and R. Munos. LSTD with random projections. In *Advances in Neural Information Processing Systems 23*, pages 721–729, 2010.

[Ghavamzadeh *et al.* 2011] M. Ghavamzadeh, A. Lazaric, R. Munos, and M. Hoffman. Finite-sample analysis of Lasso-TD. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1177–1184, 2011.

[Johns *et al.* 2010] J. Johns, C. Painter-Wakefield, and R. Parr. Linear complementarity for regularized policy evaluation and improvement. In *Advances in Neural Information Processing Systems 23*, pages 1009–1017, 2010.

[Jonschkowski and Brock 2013] R. Jonschkowski and O. Brock. Learning task-specific state representations by maximizing slowness and predictability. *6th International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS)*, 2013.

[Kolter and Ng 2009] J. Kolter and A. Ng. Regularization and feature selection in least-squares temporal difference learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 521–528, 2009.

[Konidaris and Barto 2009] G. Konidaris and A. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems 22*, pages 1015–1023, 2009.

[Konidaris *et al.* 2011] G. Konidaris, S. Osentoski, and P. Thomas. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*, pages 380–385, 2011.

[Lagoudakis and Parr 2003] M. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[MacKay 1992] D. J. C. MacKay. *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1992.

[Mahadevan and Liu 2010] S. Mahadevan and B. Liu. Basis construction from power series expansions of value functions. In *Advances in Neural Information Processing Systems 23*, pages 1540–1548, 2010.

[Mahadevan and Maggioni 2007] S. Mahadevan and M. Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8:2169–2231, 2007.

[Mahadevan 2005] S. Mahadevan. Proto-value functions: Developmental reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 553–560, 2005.

[Mallat and Zhang 1993] S. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41:3397–3415, 1993.

[Needell and Vershynin 2009] D. Needell and R. Vershynin. Uniform uncertainty principle and signal recovery via regularized orthogonal matching pursuit. *Foundations of computational mathematics*, 9(3):317–334, 2009.

[Painter-Wakefield and Parr 2012] C. Painter-Wakefield and R. Parr. Greedy algorithms for sparse reinforcement learning. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1391–1398, 2012.

[Parr *et al.* 2007] R. Parr, C. Painter-Wakefield, L. Li, and M. Littman. Analyzing feature generation for value-function approximation. In *Proceedings of the 24th international conference on Machine learning*, pages 737–744, 2007.

[Parr *et al.* 2008] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 752–759, 2008.

[Petrik *et al.* 2010] M. Petrik, G. Taylor, R. Parr, and S. Zilberstein. Feature selection using regularization in approximate linear programs for Markov decision processes. In *Proceedings of the 27th International Conference on Machine Learning*, pages 871–878, 2010.

[Sprague 2009] N. Sprague. Predictive projections. In *Proceedings of the 21st international joint conference on Artifical intelligence*, pages 1223–1229, 2009.

[Stone *et al.* 2006] P. Stone, G. Kuhlmann, M. Taylor, and Y. Liu. Keepaway soccer: From machine learning testbed to benchmark. In *RoboCup 2005: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Computer Science*, pages 93–105. 2006.

[Sun *et al.* 2011] Y. Sun, F. J. Gomez, M. B. Ring, and J. Schmidhuber. Incremental basis construction from temporal difference error. In *Proceedings of the 28th International Conference on Machine Learning*, pages 481–488, 2011.

[Sutton and Barto 1998] R. Sutton and A. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1998.

[Taylor *et al.* 2008] M.E. Taylor, G. Kuhlmann, and P. Stone. Autonomous transfer for reinforcement learning. In *The Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 283–290, 2008.

[Tikhonov 1963] A. Tikhonov. Solution of incorrectly formulated problems and the regularization method. In *Soviet Math. Dokl.*, volume 5, page 1035, 1963.

[Van Roy 1998] B. Van Roy. *Learning and value function approximation in complex decision processes*. PhD thesis, Massachusetts Institute of Technology, 1998.

[Vincent and Bengio 2002] P. Vincent and Y. Bengio. Kernel matching pursuit. *Machine Learning*, 48:165–187, 2002.

[Wahba 1990] G. Wahba. *Spline models for observational data*, volume 59. Society for industrial and applied mathematics, 1990.