Sparse Approximate Policy Evaluation using Graph-based Basis Functions

Jeff Johns and Sridhar Mahadevan
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
{johns,mahadeva}@cs.umass.edu
Technical Report No. UM-CS-2009-041

Abstract

Proto-value functions and diffusion wavelets are graph-based basis functions that capture topological structure of the MDP state space. A subset of these basis functions must be selected when approximating value functions in order to maintain computational efficiency and prevent overfitting. We evaluated four basis selection algorithms for performing this task. This is an enhancement over the previously used heuristic of always selecting the most global, or smoothest, subset of basis functions regardless of the policy being evaluated. We analyzed two schemes, one direct and one indirect, for combining basis selection and approximate policy evaluation. The indirect scheme requires more computation than the direct scheme, but gains flexibility in the manner in which basis functions are selected. The coefficients applied to the basis functions were set using least-squares methods. We also described how least-squares methods can be altered to include regularization. Laplacian-based regularization provides a bias toward smoother approximate value functions which can prevent overfitting and can be useful in stochastic domains. A thorough set of experiments was conducted on a simple chain MDP to understand how basis selection and the different least-squares policy evaluation algorithms impact one another. Although the experiments used graph-based basis functions, the algorithms described in this paper can be applied to any set of basis functions.

1 Introduction

Value function based reinforcement learning (RL) algorithms estimate the long-term expected value of each state or state-action pair. To scale RL algorithms to large and continuous domains, the value function must be approximated with a *compact* representation. The precise form of this representation is crucial in determining the success or failure of any learning algorithm. Constructing a useful representation has typically been an iterative, domain-dependent process. Recent work, however, has sought to automate this process in a dynamic, domain-independent fashion

[1, 2, 3, 4, 5]. Basis construction, feature generation, and representation discovery are all different names referring to this same task of generating a representation used to approximate a value function.

To date, RL feature construction algorithms can be categorized into two types[1]: one that iteratively generates basis functions based upon the current function approximation error [1, 4, 5] and the other that generates a dictionary of basis functions based upon a state space analysis [2, 3]. Note the latter type requires a *selection* strategy to determine which elements from the dictionary to utilize. We focus on the dictionary approach to basis construction, and the diffusion wavelet construction [3, 6] in particular, for three reasons. First, a dictionary offers the flexibility of approximating value functions associated with many different policies. The other basis construction type iteratively generates basis functions for fitting just a single policy and must start over whenever the policy is changed. Second, there is significant interest in the machine learning community on methods for generating *data-dependent dictionaries* [7, 8, 6, 9, 10]. By creating algorithms that operate on such dictionaries, we can naturally leverage future advances. Third, of the two basis construction methods based upon a state space analysis, diffusion wavelets appear to have been overlooked in the literature despite their excellent representational capabilities. We hope that our evaluation of diffusion wavelets in this paper encourages further investigation by other researchers.

Two techniques have been explored for generating basis functions based upon a state space analysis of a Markov decision process (MDP): proto-value functions (PVFs) [2] and, as just mentioned above, diffusion wavelets [3]. We refer collectively to both PVFs and diffusion wavelets as *graph-based basis functions* since both are generated from a graph. The graph, which is the tangible output of the state space analysis, encodes state similarity. Proto-value functions are generated from a Fourier-style decomposition of the graph Laplacian [11]. As such, PVFs are *global* basis functions that span the space of all possible square-integrable functions on the graph. Diffusion wavelets are a generalization of classical wavelets to manifolds and graphs. They are associated with a *multiscale* diffusion process. At small scales, diffusion wavelet bases capture local features over the graph, while the features are more global at larger scales. The entire set of diffusion wavelet bases is an *overcomplete* representation for functions on the graph or manifold. Thus, a basis selection mechanism is required for determining a subset of basis functions to use for function approximation. This promotes sparsity and computational efficiency while also preventing overfitting. All previous applications of graph-based basis functions have used the same basis selection heuristic of always using the most global, or smoothest, basis functions. This heuristic is independent of the policy being evaluated, meaning that all value functions are represented with the same set of basis functions. Using just the smoothest basis functions has the advantages of being computationally simple and robust to overfitting (although too much regularization can be just as problematic as too little regularization), but it does not exploit the full power of the basis function dictionary. One goal of this paper is to explore different selection mechanisms that better utilize the dictionary.

We evaluated four sparse basis selection algorithms: orthogonal matching pursuit (OMP) [12], order recursive matching pursuit (ORMP) [13], the LASSO [14], and least angle regression (LARS) [15]. We tested the selection algorithms using the graph-based basis functions as a dictionary, but the algorithms can be used with *any* set of basis functions. Each algorithm returns a subset of basis functions from the dictionary and a scalar coefficient associated with each selected basis function.

[1]We focus here on techniques for *explicitly* constructing features.

The selected basis functions and coefficients are linearly combined to produce an approximate value function. We tested two different schemes for combining approximate policy evaluation and basis selection. The factor distinguishing these two schemes is whether the basis selection algorithm *directly* or *indirectly* considers the Bellman equation. This distinction will be made more clear in Section 4 of this paper; however, to provide some motivation now, we note that these two schemes differ in terms of sparsity (how many basis functions are used in the approximate value function) and computational efficiency. To assess the combination of basis selection and approximate policy evaluation, experiments were conducted on a simple MDP using least-squares policy evaluation method. An additional contribution this paper makes is to employ Laplacian-based regularization to the least-squares methods.

2 MDPs and Policy Iteration

A discrete Markov decision process (MDP) $M = \langle S, A, P, R \rangle$ is a four-tuple consisting of a set of states S, a set of actions A, a state transition function $P(s, a, s')$ yielding the probability of moving from state $s \in S$ to state $s' \in S$ given action $a \in A$, and a reward function $R(s, a, s')$ specifying a scalar reward upon the transition. We also consider MDPs with continuous state spaces, but for ease of exposition the remainder of this section holds for the discrete case. Given a policy π mapping states to actions, let P^π be a matrix with $P^\pi(i, j) = P(i, \pi(i), j)$ and let R^π be a vector with $R^\pi(i) = \sum_j P^\pi(i, j) R(i, \pi(i), j)$. The value function V^π is a vector that solves the Bellman equation $V^\pi = R^\pi + \gamma P^\pi V^\pi := T^\pi(V^\pi)$ where $T^\pi : \mathbb{R}^{|S|} \rightarrow \mathbb{R}^{|S|}$ is the Bellman operator. An *optimal* policy π^* induces a value function V^* with the property $V^*(s) \geq V^{\pi'}(s)$ for all states $s \in S$ and any policy π'. V^* is the optimal value function, which is unique for the MDP.

Policy iteration [16] is one of the principal algorithms used to determine the optimal value function and an optimal policy. One round of policy iteration consists of two phases: policy evaluation and policy improvement. Policy evaluation involves computing V^π for some policy π. Policy improvement then computes a new policy π_g that is greedy with respect to V^π (i.e. $\pi_g(s) = \operatorname{argmax}_a T^a(V^\pi(s)) \; \forall s$). The new policy π_g is then used at the beginning of the next round of policy iteration. This process is guaranteed to converge in a finite number of rounds.

When an exact representation of V^π is infeasible, the value function must be approximated. We consider linear function approximation where a value function $\hat{V}$ is expressed as a linear combination of basis functions. This is written $\hat{V} = \Phi w$ where $w \in \mathbb{R}^K$ is an adjustable parameter vector and $\Phi = [\Phi_1 | \ldots | \Phi_K] \in \mathbb{R}^{|S| \times K}$ with each column $\Phi(:, j) = \Phi_j$ being a basis function and each row $\Phi(i, :) = \phi(s_i)^T$ being a state feature vector. It is typical for the number of free parameters to be much smaller than the number of states, $K \ll |S|$.

Approximate policy evaluation algorithms take as input a set of basis functions Φ and produce a weight vector w for the approximation $\hat{V} = \Phi w$. We consider least-squares algorithms due to their sample efficiency. Two common least-squares algorithms are the fixed-point (FP) algorithm (also referred to as the least-squares temporal difference (LSTD) algorithm [17]) and the Bellman residual (BR) minimization algorithm. The BR algorithm computes a value function that approximates V^π by minimizing the Bellman residual $\|T^\pi(\hat{V}) - \hat{V}\|_\rho^2$. The notation $\| \cdot \|_\rho^2$ indicates the squared L_2 norm with distribution ρ and $\| \cdot \|^2$ indicates a squared L_2 norm that is unweighted. The FP algorithm produces an approximate value function by minimizing the *projected* Bellman

residual $\|\Pi_\rho T^\pi(\hat{V}) - \hat{V}\|_\rho^2$. Π_ρ is a projection operator which, for linear subspaces Φ, is defined as $\Pi_\rho = \Phi(\Phi^T D_\rho \Phi)^{-1}\Phi^T D_\rho$ where D_ρ is a diagonal matrix with elements ρ. The solutions produced by the two algorithms have the form $\hat{V}_{FP} = \Phi A_{FP}^{-1} b_{FP}$ and $\hat{V}_{BR} = \Phi A_{BR}^{-1} b_{BR}$ where:

$$\begin{aligned} A_{FP} &= \Phi^T D_\rho(\Phi - \gamma P^\pi \Phi) \\ b_{FP} &= \Phi^T D_\rho R^\pi \\ A_{BR} &= (\Phi - \gamma P^\pi \Phi)^T D_\rho(\Phi - \gamma P^\pi \Phi) \\ b_{BR} &= (\Phi - \gamma P^\pi \Phi)^T D_\rho R^\pi. \end{aligned} \tag{1}$$

Johns et al. [18] showed that an entire spectrum of least-squares algorithms exists with the FP solution at one end and the BR solution at the other. *Hybrid* solutions can be found between the two extremes by considering a convex combination of the FP and BR objectives. Johns et al. described two hybrid algorithms H_1 and H_2 which also have the form $\hat{V}_{H_1} = \Phi A_{H_1}^{-1} b_{H_1}$ and $\hat{V}_{H_2} = \Phi A_{H_2}^{-1} b_{H_2}$ where:

$$\begin{aligned} A_{H_1} &= (\Phi - \gamma P^\pi \Phi)^T D_\rho \left(\xi I + (1-\xi)\Pi_\rho\right)(\Phi - \gamma P^\pi \Phi) \\ b_{H_1} &= (\Phi - \gamma P^\pi \Phi)^T D_\rho \left(\xi I + (1-\xi)\Pi_\rho\right) R^\pi \\ A_{H_2} &= (\Phi - \xi\gamma P^\pi \Phi)^T D_\rho(\Phi - \gamma P^\pi \Phi) \\ b_{H_2} &= (\Phi - \xi\gamma P^\pi \Phi)^T D_\rho R^\pi \end{aligned} \tag{2}$$

given the constant $\xi \in [0, 1]$. Algorithms H_1 and H_2 produce the FP solution when $\xi = 0$ and the BR solution when $\xi = 1$.

The square matrices A_{FP}, A_{BR}, A_{H_1}, and A_{H_2} have K rows and columns. When K is large, it may be infeasible to form sample-based estimates of the matrices. This issue is one of the motivating factors for sparse basis selection. If only a small subset of basis functions are necessary for approximating a value function, then these matrices can be much smaller and performing matrix inversion can become feasible.

Although we describe the basis selection algorithms in this paper using value functions, they are easily extended to approximating action-value functions. The action-value function $Q^\pi \in \mathbb{R}^{|S||A|}$ is defined as the expected value of being in state s, taking action a, and and then following policy π thereafter. Note that approximate action-value functions ($\hat{Q} = \Phi w$) have basis functions that are defined over state-action space. Graph-based basis functions, which are defined over the state space, can be used to approximate action-value functions by handling each discrete action separately. As an example, consider an action-value function with two discrete actions a_1 and a_2. The approximate action-value function can be represented as $\hat{Q}(\cdot, a_1) = \Phi w_{a_1}$ and $\hat{Q}(\cdot, a_2) = \Phi w_{a_2}$ where the same basis functions Φ are used for both actions. An alternative to this scheme is to define the basis functions directly over state-action space [19]. Given a set of samples from a MDP, the least-squares policy iteration (LSPI) [20] algorithm is used to produce an approximation of the optimal action-value function. LSPI alternates between approximate policy evaluation (i.e. estimating Q^π with $\hat{Q}$) and policy improvement. We assume policy improvement simply selects the greedy policy implicit in $\hat{Q}$ (i.e. greedy policy $\pi_g(s) = \operatorname{argmax}_a \hat{Q}(s, a) \forall s$). Farahmand et al. [21] have analyzed a *regularized* version of the LSPI algorithm.

3 Previous Work

3.1 Basis Construction from Graphs

Mahadevan and Maggioni [2, 3] proposed two techniques for generating a dictionary of basis functions based upon a state space analysis. The analysis is performed on a graph data structure where vertices correspond to states and edges connect neighboring states. Note the graph can represent either discrete or continuous state MDPs. Given this graph, a dictionary is created by either computing diffusion wavelets [3, 6] or proto-value functions [2].

We begin with a brief description of the PVF construction before addressing diffusion wavelets. Let W be a symmetric, positive weight matrix representing the state graph. $W(i,j) = 0$ indicates there is no edge between states s_i and s_j while $W(i,j) > 0$ indicates the strength of the connection between s_i and s_j. The valency matrix D is a diagonal matrix whose values are the row sums of the weight matrix ($D(i,i) = \sum_j W(i,j)$). The combinatorial graph Laplacian is defined as $L = D - W$ and the normalized graph Laplacian is $\mathscr{L} = I - D^{-0.5} W D^{-0.5}$ [11]. Since W is symmetric, the eigenvectors of both Laplacians form a complete orthonormal basis. The eigendecomposition is $L = \Phi \Lambda \Phi^T$ where the columns of Φ are the eigenvectors and Λ is a diagonal matrix containing the eigenvalues. Note that in Section 2 the symbol Φ referred to a set of basis functions; we purposefully reuse Φ here because the Laplacian eigenvectors are the proto-value functions. Given the complexity of the eigenvector computation, it is typical for large graphs to only compute the eigenvectors associated with the smallest eigenvalues as those are the *smoothest* eigenvectors over the graph[2]. If a subset of eigenvectors is computed, then the dictionary is incomplete.

Like Laplacian eigenvectors, diffusion wavelets are constructed from a neighborhood graph over the state space. Diffusion wavelets are a *multiscale*, *overcomplete* representation. They efficiently represent powers of a diffusion operator on the graph. The diffusion operator is defined as $T = (I - \mathscr{L})$ with powers T^t, $t > 0$. To make the diffusion aspect more obvious, this can be rewritten $T = D^{-0.5} W D^{-0.5} = D^{0.5} P D^{-0.5}$ where $P = D^{-1} W$ is a stochastic matrix representing a random walk (diffusion process) on the graph. Note that P is *conjugate* along with its powers to T; thus, studying T and P are equivalent in terms of spectral properties. It is computationally easier to deal with T since it is symmetric. Small powers of T^t correspond to short-term behavior in the diffusion process and large powers correspond to long-term behavior. Diffusion wavelets are naturally multiscale basis functions because they account for increasing powers of T^t. We give a brief sketch of the diffusion wavelet algorithm; a more thorough description can be found in the original paper [6]. Aside from matrix T, the other inputs to the algorithm are J, the maximum number of levels to compute, ϵ, a precision parameter, and $\mathsf{SpQR}(A, \epsilon)$, a sparse QR algorithm that outputs (sparse) matrices Q and R such that $A =_\epsilon QR$ (i.e. the columns of Q ϵ-span the columns of A). The outputs of the algorithm are a set of scaling functions $\{\phi_j\}$ and wavelet functions $\{\psi_j\}$ at different levels/scales. As the level j gets larger, the number of scaling and wavelet functions gets smaller because the diffusion process spreads out and becomes more compressible. Algorithm 1 shows the details of the construction and uses the following notation: $[T]_{\phi_a}^{\phi_b}$ is a matrix representing T with respect to the basis ϕ_a in the domain and ϕ_b in the range ($n_b \times n_a$ matrix) and

[2]Smoothness of a function f on the graph can be measured by the Dirichlet sum: $\langle f, Lf \rangle = \sum_{u \sim v} W(u,v) \left(f(u) - f(v)\right)^2$ where $u \sim v$ is an edge in the graph. For a Laplacian eigenvector ϕ, the Dirichlet sum is $\langle \phi, L\phi \rangle = \lambda$. Thus, smaller eigenvalues correspond to smoother eigenvectors.

Algorithm 1: `DiffusionWaveletTree`

Input: $[T]_{\phi_0}^{\phi_0}, \phi_0, J$, SpQR, ϵ
Output: $\{\phi_j\}_{j=0}^{J}, \{\psi_j\}_{j=0}^{J-1}$

for $j = 0$ to $(J-1)$ **do**
  $[\phi_{j+1}]_{\phi_j}, [T^{2^j}]_{\phi_j}^{\phi_{j+1}} \leftarrow \mathsf{SpQR}([T^{2^j}]_{\phi_j}^{\phi_j}, \epsilon)$
  $[T^{2^{j+1}}]_{\phi_{j+1}}^{\phi_{j+1}} \leftarrow [T^{2^j}]_{\phi_j}^{\phi_{j+1}}([T^{2^j}]_{\phi_j}^{\phi_{j+1}})^*$
  $[\psi_j]_{\phi_j} \leftarrow \mathsf{SpQR}(I_{\langle\phi_j\rangle} - [\phi_{j+1}]_{\phi_j}([\phi_{j+1}]_{\phi_j})^*, \epsilon)$
end for

$[\phi_b]_{\phi_a}$ is a set of functions ϕ_b represented on the basis ϕ_a ($n_a \times n_b$ matrix). Note that the scaling functions $[\phi_j]_{\phi_{j-1}}$ provide a mapping from level $j-1$ to level j. In order to view the functions in the original basis ϕ_0 (which is usually assumed to be the unit basis), the mapping is unrolled to give $[\phi_j]_{\phi_0} = [\phi_j]_{\phi_{j-1}}[\phi_{j-1}]_{\phi_{j-2}} \ldots [\phi_1]_{\phi_0}[\phi_0]_{\phi_0}$.

There are at least two ways to form a dictionary of basis functions given the diffusion wavelet tree. One approach is to include all the scaling functions $\{\phi_j\}_{j \geq j_{min}}^{J}$ and the wavelet functions $\{\psi_j\}_{j=j_{min}-1}^{J-1}$ above a minimum level $j_{min} > 0$. Specifying a minimum level can be useful for both computational reasons (to control the size of the dictionary) and because some of the scaling functions at lower tree levels can have a large gradient, thus providing challenges for robust function approximation. The second approach is to select an orthonormal dictionary from the set of possible orthonormal dictionaries. In this approach, a dictionary consists of the scaling functions at level $j^* \in [1, 2, \ldots, J]$, ϕ_{j^*}, and all the wavelet functions from level j_{min} up to level (j^*-1), $\{\psi_j\}_{j=j_{min}}^{j^*-1}$. There are techniques for finding the best such dictionary given a function to be approximated. We evaluate both approaches to constructing a diffusion wavelet dictionary.

3.2 Basis Selection for Regression

We provide a brief introduction to the basis selection problem and a few of the major algorithms since the literature is vast. The basic formulation is that there is a signal $y \in \mathbb{R}^N$ to be represented with elements from an overcomplete dictionary $\Phi \in \mathbb{R}^{N \times K}$. Each basis function $\Phi_j \in \mathbb{R}^N$ has unit norm. The problem is to find a vector w such that $\Phi w = y$.[3] The decomposition of y is not unique; therefore, additional constraints are added to prefer solutions with certain qualities (e.g. sparseness, independence).

Two popular approaches to the sparse regression problem are matching pursuit and basis pursuit. Matching pursuit is an iterative, greedy algorithm whereas basis pursuit is an optimization principle (that can be solved using any appropriate algorithm). Therefore, matching pursuit and basis pursuit are not mutually exclusive approaches to sparse regression.

Matching pursuit (MP) [22] is a greedy algorithm that selects elements sequentially to best capture the signal. The algorithm begins with a coefficient vector w equal to all zeros and a residual vector y_{res} equal to the signal y. The first element is selected by scanning the dictionary and finding the largest correlation with the residual: $j^* \leftarrow \text{argmax}_j |\Phi_j^T y_{res}|, j \in [1, K]$. The

[3]The model could also include a noise term, $\Phi w + e = y$.

coefficient for the selected basis function is adjusted: $w_{j^*} \leftarrow w_{j^*} + \Phi_{j^*}^T y_{res}$. Then the residual signal is computed $y_{res} \leftarrow y_{res} - (\Phi_{j^*}^T y_{res})\Phi_{j^*}$ and the process iterates. With MP, a basis function can be selected many times. There are other variants of MP, two of which are orthogonal matching pursuit (OMP) [12] and order recursive matching pursuit (ORMP) [13]. OMP differs from MP in the way the residual signal is computed. OMP makes the residual orthogonal to the selected dictionary elements, which means OMP will never select the same dictionary element more than once whereas MP can. ORMP goes even further than OMP and adds the orthogonalization step into the selection process. Moghaddam et al. [23] proposed an efficient implementation of ORMP (using partitioned matrix inverse techniques [24]) and showed that sparse least-squares regression is equivalent to a generalized eigenvalue problem.

Algorithm 2 is a side-by-side comparison of the pseudocode for MP, OMP, and ORMP. We use the symbol $\mathcal{I}$ to refer to a set of indices in $[1, K]$ that indicate the elements of the dictionary Φ that are selected by the algorithm. Similarly, $w_{\mathcal{I}}$ refers to the scalar coefficients applied to the selected basis functions. Basis functions that are not selected have a scalar coefficient of 0. Thus, the signal y is approximated as $\Phi(:, \mathcal{I})w(\mathcal{I}) = \Phi_{\mathcal{I}} w_{\mathcal{I}}$.

Algorithm 2: `MP, OMP, and ORMP`

Input: Φ, y

Output: $\mathcal{I}$, $w_{\mathcal{I}}$ such that $\hat{y} \leftarrow \Phi_{\mathcal{I}} w_{\mathcal{I}}$

$\mathcal{I} \leftarrow \emptyset$, , $w \leftarrow \mathbf{0}$, $y_{res} \leftarrow y$

while (not done) **do**

For MP: $j^* \leftarrow \text{argmax}_j |\Phi_j^T y_{res}|$
$w_{j^*} \leftarrow w_{j^*} + \Phi_{j^*}^T y_{res}$
If $(w_{j^*} \neq 0)$, $\mathcal{I} \leftarrow \mathcal{I} \cup \{j^*\}$. **Else**, $\mathcal{I} \leftarrow \mathcal{I} - \{j^*\}$
$y_{res} \leftarrow y_{res} - (\Phi_{j^*}^T y_{res})\Phi_{j^*}$

For OMP: $j^* \leftarrow \text{argmax}_{j \notin \mathcal{I}} |\Phi_j^T y_{res}|$
$\mathcal{I} \leftarrow \mathcal{I} \cup \{j^*\}$
$w_{\mathcal{I}} \leftarrow (\Phi_{\mathcal{I}}^T \Phi_{\mathcal{I}})^{-1} \Phi_{\mathcal{I}}^T y$
$y_{res} \leftarrow y - \Phi_{\mathcal{I}} w_{\mathcal{I}}$

For ORMP: $j^* \leftarrow \text{argmin}_{j \notin \mathcal{I}} \|\Phi_{\mathcal{I}_{+j}} (\Phi_{\mathcal{I}_{+j}}^T \Phi_{\mathcal{I}_{+j}})^{-1} \Phi_{\mathcal{I}_{+j}}^T y - y\|^2$ where: $\mathcal{I}_{+j} \leftarrow \mathcal{I} \cup \{j\}$
$\mathcal{I} \leftarrow \mathcal{I} \cup \{j^*\}$
$w_{\mathcal{I}} \leftarrow (\Phi_{\mathcal{I}}^T \Phi_{\mathcal{I}})^{-1} \Phi_{\mathcal{I}}^T y$

end while

Matching pursuit finds a sparse representation by greedily selecting the most promising elements. In contrast, basis pursuit (BP) [25] achieves sparsity by finding solutions to the following optimization problem: $\min \|w\|_1$ such that $\Phi w = y$. Sparsity of the solution comes from the use of L_1 norm. The BP problem can be solved using linear programming. Note the hard constraint $\Phi w = y$ is appropriate when the signal is noiseless. When the signal is noisy, it is appropriate to require $\|\Phi w - y\|^2$ to be small. The LASSO (least absolute shrinkage and selection operator) [14] implements this noisy version of basis pursuit in the following optimization problem: $\min \|y - \Phi w\|^2$ subject to $\|w\|_1 \leq k$. The LASSO can be solved using quadratic programming; however, a more efficient solution is to use the recently introduced least angle regression (LARS) algorithm [15] with a minor modification. LARS selects elements from the dictionary one at a

time (which is the same way the matching pursuit algorithms work). The first element selected is the one that is most correlated with the signal y. Then LARS proceeds until another element has as much correlation with the current residual. At that point, LARS includes this second element and then proceeds in a direction *equiangular* between the first two elements. This strategy is less greedy than other algorithms that sequentially add dictionary elements. Interestingly, a small modification to the LARS algorithm produces the LASSO solution. While LARS by itself only adds basis functions at each step, this modification for LASSO gives the algorithm the ability to remove basis functions from the selected subset as well.

We evaluated the OMP, ORMP, LASSO, and LARS algorithms. It is easy to control the sparsity of each of these algorithms by limiting the number of basis functions that can be selected.

4 Basis Selection For Approximate Policy Evaluation

The basis selection problem involves choosing elements from a dictionary to efficiently represent a target signal. The approximate policy evaluation problem is to represent the true value function V^π with an accurate approximation $\hat{V}$. If V^π were known, then basis selection could simply be performed with the target signal being V^π. However, V^π only becomes known through the Bellman equation: $V^\pi = R^\pi + \gamma P^\pi V^\pi = T^\pi(V^\pi)$. Thus, some framework is needed that effectively combines approximate policy evaluation (i.e. finding an accurate approximation $\hat{V}$) and basis selection (i.e. efficiently representing $\hat{V}$). There are at least two ways to achieve this combination. The two schemes differ in how they use the Bellman equation. The first scheme uses the Bellman equation within the basis selection algorithm. This means that when the basis selection algorithm adjusts the weight vector w, this not only changes the approximation Φw ***but also*** changes the target signal based on a function of the Bellman equation. We call this the direct scheme because the selection algorithm directly encodes the Bellman equation. The second, or indirect, scheme does *not* use the Bellman equation within the basis selection algorithm. Rather, there is an iterative process that alternates between (1) setting the target signal using the Bellman equation, and (2) representing the target signal using the basis selection algorithm. These two schemes are described below in a very general form where $f\left(T^\pi(\Phi w') - \Phi w'\right)$ is some function f of the Bellman residual (the least-squares algorithms FP, BR, H_1, and H_2 use different functions f).

Direct Scheme

$[\mathcal{I},\, w_\mathcal{I}] \leftarrow \texttt{BasisSelection}_{w'}\left(f\left(T^\pi(\Phi w') - \Phi w'\right)\right)$
OPTIONAL: $\quad w_\mathcal{I} \leftarrow \texttt{SetWeights}_{w'}\left(f\left(T^\pi(\Phi_\mathcal{I} w') - \Phi_\mathcal{I} w'\right)\right)$
$\hat{V} \leftarrow \Phi_\mathcal{I} w_\mathcal{I}$

Indirect Scheme

$\mathcal{I} \leftarrow \emptyset, \quad w_\mathcal{I} \leftarrow \emptyset$
while (not converged)
    target $y \leftarrow T^\pi(\Phi_\mathcal{I} w_\mathcal{I})$
    $[\mathcal{I},\, w_\mathcal{I}] \leftarrow \texttt{BasisSelection}_{w'}\left(y - \Phi w'\right)$
    OPTIONAL: $\quad w_\mathcal{I} \leftarrow \texttt{SetWeights}_{w'}\left(f\left(T^\pi(\Phi_\mathcal{I} w') - \Phi_\mathcal{I} w'\right)\right)$
$\hat{V} \leftarrow \Phi_\mathcal{I} w_\mathcal{I}$

The direct and indirect schemes differ in their computational complexity and degree of sparsity. The computational complexity of the indirect scheme has the potential to be greater than the direct scheme because it iteratively calls the basis selection algorithm. This could be wasteful when the target signal given to the basis selection algorithm does not change significantly between iterations. On the other hand, the direct scheme, by using the Bellman residual as the target function for the basis selection algorithm, forces the regression algorithm to follow a specific path. To see this, consider the beginning of the basis selection algorithm when no basis functions have yet been selected. The Bellman residual is equal to the immediate reward function R^π. This means the first basis function selected is attempting to fit the immediate reward. For the sake of argument, assume the first basis function exactly fits the immediate reward. Now the Bellman residual is equal to the Bellman backup of the immediate reward, $(T^\pi(R^\pi) - R^\pi) = \gamma P^\pi R^\pi$. This same logic can be used inductively to show the basis selection process proceeds in order of the elements in the Neumann series, $\sum_{i=0}^{\infty}(\gamma P^\pi)^i R^\pi$.[4] Attempting to fit the elements in the Neumann series can lead to inefficient use of the basis functions. This occurs when there is structure in V^π that does not exist in the Neumann series; hence, the basis selection algorithm is unable to exploit the structure. Since the indirect scheme is not confined to this path, it has the potential to use fewer basis functions when representing the eventual approximate value function $\hat{V}$.

As an example of the potential inefficiency of the direct scheme, consider an undiscounted, deterministic chain MDP with an absorbing state at one end of the chain. Assume the reward function is 0 everywhere except $+1$ at the absorbing state. The optimal value function is a constant function equaling 1 in each state, but the Neumann series is a sequence of delta functions from one end of the chain to the other. Given a dictionary consisting of all the delta functions and a constant function, a basis selection algorithm implementing the direct scheme will select all the delta functions rather than the constant function. This may be an extreme example, but it is not uncommon for a MDP to have a spiky reward function that would cause similar behavior. Note this behavior can be particularly problematic for the multiscale diffusion wavelet dictionary where very localized basis functions (that are not necessary for representing V^π) can get selected before larger scale basis functions.

Before outlining the basis selection algorithms within the direct and indirect schemes, we first describe how we augmented the least-squares algorithms to include Laplacian-based regularization.

4.1 Least-Squares Methods with Laplacian-based Regularization

In Section 2, four least-squares algorithms were described for approximate policy evaluation. Those four algorithms compute approximate value functions that minimize a loss function associated with the Bellman residual. Here, we include an additional loss function that *regularizes* the solution. We specifically use a *data-dependent* form of regularization that uses the graph Laplacian [26]. This is in fact the same graph Laplacian used to produce PVFs and diffusion wavelets. Laplacian-based regularization has been applied with great success to semi-supervised learning problems where the geometric structure of unlabeled data points can be exploited. To understand how the graph Laplacian provides regularization, consider again the Dirichlet sum which

[4] For a bounded operator T, the Neumann series is defined as $\sum_{i=0}^{\infty} T^i$. One can show $\sum_{i=0}^{\infty} T^i = (I-T)^{-1}$. The value function V^π can be defined using the Neumann series as $V^\pi = (I - \gamma P^\pi)^{-1} R^\pi = \sum_{i=0}^{\infty}(\gamma P^\pi)^i R^\pi$.

was described in a footnote in Section 3.1. Given function f, the Dirichlet sum is $\langle f, Lf \rangle = \sum_{u \sim v} W(u,v) \left(f(u) - f(v)\right)^2$. The Dirichlet sum is large when f is not smooth according to the structure of the graph. For functions that are smooth, the Dirichlet sum is small. Thus, the Laplacian can be used to penalize (regularize) functions that are not smooth according to the structure of the MDP state space that is encoded in the graph.

As a concrete example, consider the fixed-point least-squares algorithm. FP's loss function is based on the projected Bellman residual. We augment that loss function with a Laplacian-based regularization (LR) term as follows:

$$w_{FP,LR} = \underset{w' \in \mathbb{R}^K}{\operatorname{argmin}} \frac{1}{2} \|\Pi_\rho T^\pi(\Phi w') - \Phi w'\|_\rho^2 + \frac{\beta_m}{2} \|L \Phi w'\|_\rho^2 \tag{3}$$

where $\beta_m \in \mathbb{R}^+$ is a parameter controlling the influence of the regularization term. It is easy to show that $w_{FP,LR} = A_{FP,LR}^{-1} b_{FP,LR}$ where:

$$\begin{aligned} A_{FP,LR} &= \Phi^T D_\rho (\Phi - \gamma P^\pi \Phi) + \beta_m \Phi^T L D_\rho L \Phi \\ b_{FP,LR} &= \Phi^T D_\rho R^\pi. \end{aligned} \tag{4}$$

Notice that $b_{FP,LR} = b_{FP}$ and $A_{FP,LR} = A_{FP} + \beta_m \Phi^T L D_\rho L \Phi$. Laplacian-based regularization has this same effect on the three other least-squares algorithms (BR, H_1, and H_2). Given a sample $\langle s, \pi(s), r, s' \rangle$ from the MDP, estimates of the matrix $A_{FP,LR}$ and vector $b_{FP,LR}$ can be formed using the following updates:

$$\begin{aligned} \hat{b}_{FP,LR} &\leftarrow \hat{b}_{FP,LR} + \rho(s) \phi(s) r \\ \hat{A}_{FP,LR} &\leftarrow \hat{A}_{FP,LR} + \rho(s) \left[\phi(s) (\phi(s) - \gamma \phi(s'))^T + \beta_m g(s) g(s)^T \right]. \end{aligned}$$

The term $g(s)$ in the updates is computed as:

$$\begin{aligned} g(s) &\leftarrow L(s,s) \phi(s) \\ g(s) &\leftarrow g(s) + L(s, s_{nbr}) \phi(s_{nbr}) \quad \forall \{ s_{nbr} | s_{nbr} \neq s \wedge s \sim s_{nbr} \text{ in graph} \}. \end{aligned}$$

A common assumption is that MDP state space graphs are sparsely connected. This means that any state s has at most a few neighboring states s_{nbr} in the graph. In this case, the time to compute $g(s)$ is negligible. Of course, if the basis functions $\phi(s)$ are the PVFs, then the eigendecomposition $L\Phi = \Phi\Lambda$ can be exploited to simplify the computation as $g(s) \leftarrow \Lambda \phi(s)$.

4.2 Direct Schemes

The next three sections outline the OMP-FP algorithm (i.e. OMP for basis selection and FP for setting the coefficients), the ORMP-FP algorithm, and the LASSO-FP and LARS-FP algorithms. Laplacian-based regularization is used in each algorithm. The LASSO-FP and LARS-FP algorithms are nearly identical, so we describe them simultaneously. It is important to point out that all of these algorithms can easily be adapted to use the BR, H_1, or H_2 least-squares methods rather than FP. We simply chose to illustrate the algorithms using the FP least-squares method because that is the most common technique used in the literature.

Each algorithm takes as input a set of MDP samples $\{s_i, r_i, s'_i\}_{i=1}^n$, the discount factor γ, the dictionary Φ of basis functions, the graph Laplacian L along with the regularization parameter β_m, a distribution ρ over the states for weighting the least-squares problem, and a maximum allowable number of basis functions k' that the algorithm can select. Each algorithm returns a set of indices $\mathcal{I}$ into the columns of Φ and scalar coefficients $w_\mathcal{I}$ such that the approximate value function $\hat{V} = \Phi_\mathcal{I} w_\mathcal{I}$. The sparsity of the solution is directly controlled by limiting the basis selection algorithm to at most $|\mathcal{I}| \leq k'$ basis functions. The parameter k' also limits the basis selection algorithm's computation and memory usage. Since the selection algorithm builds up sample-based estimates of the least-squares data structures (e.g. $\hat{A}^{-1}_{FP,LR}$ and $\hat{b}_{FP,LR}$), the size of the data structures cannot be larger than k'. This can be very important when the number of basis functions in the dictionary is large. To further speed up OMP-FP, ORMP-FP, LASSO-FP, and LARS-FP, we take advantage of the fact that the algorithms insert or remove one basis function at a time to the active set $\mathcal{I}$. The matrix $\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}$ can be incrementally formed. However, to keep the pseudocode simple, the algorithms are not shown with this improvement. The appendix describes how the algorithms can incrementally update $\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}$.

The OMP-FP and ORMP-FP algorithms terminate when either k' basis functions have been selected or when the change in the norm of the Bellman residual goes beneath a threshold.[5] The LASSO-FP and LARS-FP algorithms use both of those termination conditions as well as one other condition (related to the parameter k') that we discuss in that section.

4.2.1 OMP

Algorithm 3 shows the direct approach for combining orthogonal matching pursuit and the fixed-point least-squares algorithm with Laplacian-based regularization. The algorithm maintains a sample-based estimate of the vector c where

$$\begin{aligned} c_j &= \left[\Phi^T D_\rho R^\pi - \Phi^T D_\rho (\Phi - \gamma P^\pi \Phi) w - \beta_m \Phi^T L D_\rho L \Phi w\right]_j \\ &= \left[\Phi^T D_\rho R^\pi - \Phi^T D_\rho (\Phi_\mathcal{I} - \gamma P^\pi \Phi_\mathcal{I}) w_\mathcal{I} - \beta_m \Phi^T L D_\rho L \Phi_\mathcal{I} w_\mathcal{I}\right]_j . \end{aligned} \tag{5}$$

This equation for c_j is based on the FP least-squares method. If a different algorithm is used (BR, H_1, H_2), then c_j will have a different form. These changes are discussed in Section 4.2.4.

Each iteration of OMP-FP selects a new basis function to add to the active set by finding $j \notin \mathcal{I}$ that maximizes $|c_j|$. Then the weights $w_\mathcal{I}$ are adjusted to make the residual orthogonal to $\Phi_\mathcal{I}$.

4.2.2 ORMP

Algorithm 4 shows the direct approach for combining ORMP and the FP least-squares algorithm with Laplacian-based regularization. We present Algorithm 4 using FP to be consistent with our presentations of OMP-FP, LASSO-FP, and LARS-FP. This helps make the pseudocode more readable since the FP least-squares data structures are identical from one algorithm to the next. However, as we show later in Section 4.2.4, it is only valid to combine ORMP and the BR least-squares method. Section 4.2.4 also describes the changes that must occur to switch from FP to BR.

[5] Using the terminology described in the algorithm boxes, the squared norm of the Bellman residual is written $\sum_{i=1}^n \rho(s_i) \left[r_i - (\phi_\mathcal{I}(s_i) + \beta_m\, g_\mathcal{I}(s_i) - \gamma \phi_\mathcal{I}(s'_i))^T w_\mathcal{I}\right]^2$. The change in the norm of the Bellman residual can easily be computed when inserting or removing a new basis function from the active set $\mathcal{I}$.

Algorithm 3: `OMP-FP with Laplacian-based Regularization`

Input: $\{s_i, r_i, s'_i\}_{i=1}^n$, samples generated using policy π
$\phi : S \to \mathbb{R}^K$, basis function
$\rho : S \to \mathbb{R}^+$, weighting over the states
L, graph Laplacian defined over states $\{s_i\}_{i=1}^n$ (graph edges denoted with $\sim$)
$\gamma \in [0, 1]$, discount factor
$\beta_m \in \mathbb{R}^+$, Laplacian-based regularization parameter
$k' \leq K$, maximum allowable number of basis functions

Output: $\mathcal{I}$, set of selected basis functions (indices into ϕ)
$w_{\mathcal{I}}$, weight vector such that $\hat{V}(s) = \phi_{\mathcal{I}}(s)^T w_{\mathcal{I}}$

$c \leftarrow \sum_{i=1}^n \rho(s_i)\phi(s_i)r_i$
Initialize active set $\mathcal{I} \leftarrow \emptyset$

while $(|\mathcal{I}| < k')$ and (Bellman residual not converged) **do**

1. Find most correlated inactive element:
   $j^* \leftarrow \text{argmax}_{j \notin \mathcal{I}}(|c_j|)$
2. Adjust active set:
   $\mathcal{I} \leftarrow \mathcal{I} \cup \{j^*\}$
3. Compute $\hat{A}_{\mathcal{I},\mathcal{I}}$ and $\hat{b}_{\mathcal{I}}$:
   $\hat{A}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^n \rho(s_i) \left[\phi_{\mathcal{I}}(s_i)(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s'_i))^T + \beta_m g_{\mathcal{I}}(s_i)g_{\mathcal{I}}(s_i)^T\right]$
   $\hat{b}_{\mathcal{I}} \leftarrow \sum_{i=1}^n \rho(s_i)\phi_{\mathcal{I}}(s_i)r_i$
   where: $g(s_i) \leftarrow L(s_i, s_i)\,\phi(s_i)$
   $g(s_i) \leftarrow g(s_i) + L(s_i, s_{nbr})\,\phi(s_{nbr}) \qquad \forall\{s_{nbr} | s_{nbr} \neq s \ \wedge \ s \sim s_{nbr}\}$
4. Compute least-squares weights:
   $w_{\mathcal{I}} \leftarrow \hat{A}_{\mathcal{I},\mathcal{I}}^{-1}\hat{b}_{\mathcal{I}}$
5. Compute updated correlations:
   $c \leftarrow \sum_{i=1}^n \rho(s_i) \left[\phi(s_i)\left(r_i - (\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s'_i))^T w_{\mathcal{I}}\right) - \beta_m\, g(s_i)g_{\mathcal{I}}(s_i)^T w_{\mathcal{I}}\right]$

end while

The ORMP algorithm works by considering the impact each inactive basis function has on the least-squares problem. We use the terminology $\mathcal{I}_{+j}$ to indicate the inclusion of basis function j in the active set (i.e. $\mathcal{I}_{+j} \leftarrow \mathcal{I} \cup \{j\}$). The first step of Algorithm 4 determines the best inactive basis function $j \notin \mathcal{I}$ that maximizes $\left(\hat{b}_{\mathcal{I}_{+j}}^T \hat{A}_{\mathcal{I}_{+j},\mathcal{I}_{+j}}^{-1} \hat{b}_{\mathcal{I}_{+j}}\right)$.

However, it was pointed out by Moghaddam et al. [23] that it is actually faster to find the inactive basis function that maximizes $\left(\hat{b}_{\mathcal{I}_{+j}}^T \hat{A}_{\mathcal{I}_{+j},\mathcal{I}_{+j}}^{-1} \hat{b}_{\mathcal{I}_{+j}} - \hat{b}_{\mathcal{I}}^T \hat{A}_{\mathcal{I},\mathcal{I}}^{-1} \hat{b}_{\mathcal{I}}\right)$ because some of the intermediate computation cancels out. The intermediate terms cancel due to properties of the partitioned matrix inverse. Note that since the extra term $\left(\hat{b}_{\mathcal{I}}^T \hat{A}_{\mathcal{I},\mathcal{I}}^{-1} \hat{b}_{\mathcal{I}}\right)$ is independent of all inactive basis functions, it does not alter the result of the maximization problem. ORMP-FP then inserts the best basis function into the active set, updates $\hat{A}_{\mathcal{I},\mathcal{I}}^{-1}$ and $\hat{b}_{\mathcal{I}}$, and iterates.

Algorithm 4: `ORMP-FP with Laplacian-based Regularization`

Input: $\{s_i, r_i, s'_i\}_{i=1}^n$, samples generated using policy π
$\phi : S \rightarrow \mathbb{R}^K$, basis function
$\rho : S \rightarrow \mathbb{R}^+$, weighting over the states
L, graph Laplacian defined over states $\{s_i\}_{i=1}^n$ (graph edges denoted with $\sim$)
$\gamma \in [0, 1]$, discount factor
$\beta_m \in \mathbb{R}^+$, Laplacian-based regularization parameter
$k' \leq K$, maximum allowable number of basis functions

Output: $\mathcal{I}$, set of selected basis functions (indices into ϕ)
$w_{\mathcal{I}}$, weight vector such that $\hat{V}(s) = \phi_{\mathcal{I}}(s)^T w_{\mathcal{I}}$

Initialize active set $\mathcal{I} \leftarrow \emptyset$

while $(|\mathcal{I}| < k')$ and (Bellman residual not converged) **do**

1. Find best inactive element:

$$j^* \leftarrow \text{argmax}_{j \notin \mathcal{I}} \left(\hat{b}_{\mathcal{I}_{+j}}^T \hat{A}_{\mathcal{I}_{+j}, \mathcal{I}_{+j}}^{-1} \hat{b}_{\mathcal{I}_{+j}} \right)$$

where: $\mathcal{I}_{+j} \leftarrow \mathcal{I} \cup \{j\}$

$$\hat{b}_{\mathcal{I}_{+j}} \leftarrow \sum_{i=1}^n \rho(s_i) \phi_{\mathcal{I}_{+j}}(s_i) r_i$$

$$\hat{A}_{\mathcal{I}_{+j}, \mathcal{I}_{+j}} \leftarrow \sum_{i=1}^n \rho(s_i) [\phi_{\mathcal{I}_{+j}}(s_i)(\phi_{\mathcal{I}_{+j}}(s_i) - \gamma \phi_{\mathcal{I}_{+j}}(s'_i))^T + \ldots \beta_m g_{\mathcal{I}_{+j}}(s_i) g_{\mathcal{I}_{+j}}(s_i)^T]$$

where: $g(s_i) \leftarrow L(s_i, s_i)\, \phi(s_i)$

$$g(s_i) \leftarrow g(s_i) + L(s_i, s_{nbr})\, \phi(s_{nbr}) \qquad \forall \{s_{nbr} | s_{nbr} \neq s \ \wedge \ s \sim s_{nbr}\}$$

2. Adjust active set:

$$\mathcal{I} \leftarrow \mathcal{I} \cup \{j^*\}$$

3. Compute $\hat{A}_{\mathcal{I},\mathcal{I}}$ and $\hat{b}_{\mathcal{I}}$:

$$\hat{A}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^n \rho(s_i) \left[\phi_{\mathcal{I}}(s_i)(\phi_{\mathcal{I}}(s_i) - \gamma \phi_{\mathcal{I}}(s'_i))^T + \beta_m g_{\mathcal{I}}(s_i) g_{\mathcal{I}}(s_i)^T \right]$$

$$\hat{b}_{\mathcal{I}} \leftarrow \sum_{i=1}^n \rho(s_i) \phi_{\mathcal{I}}(s_i) r_i$$

4. Compute least-squares weights:

$$w_{\mathcal{I}} \leftarrow \hat{A}_{\mathcal{I},\mathcal{I}}^{-1} \hat{b}_{\mathcal{I}}$$

end while

4.2.3 LASSO and LARS

To achieve sparsity, the LASSO algorithm takes the loss function from Equation 3 and includes a L_1 constraint on the coefficient vector. This takes the form:

$$w_{FP,LR} = \underset{w' \in \mathbb{R}^K}{\text{argmin}} \frac{1}{2} \|\Pi_\rho T^\pi(\Phi w') - \Phi w'\|_\rho^2 + \frac{\beta_m}{2} \|L \Phi w'\|_\rho^2 + \beta_s \|w'\|_1 \tag{6}$$

where $\beta_s \in \mathbf{R}^+$ is a regularization parameter that dictates the sparsity of the solution. Larger values of β_s result in a coefficient vector w with more zero entries. In fact, there exists a value of β_s for which the resulting vector w has all zero entries.

Kolter and Ng [27] recently proposed using the LASSO algorithm with the FP least-squares method. Our description of the algorithm and its derivation follows along the same lines as their paper. The only exception is that we consider Laplacian-based regularization and they did not.

Therefore, our LASSO-FP algorithm with $\beta_m = 0$ exactly coincides with their algorithm.[6]

The minimization problem in Equation 6 can be converted into the following set of optimality conditions:

$$\begin{aligned} -\beta_s \leq c_j \leq \beta_s \quad \forall j \\ c_j = \beta_s \Rightarrow w_j \geq 0 \\ c_j = -\beta_s \Rightarrow w_j \leq 0 \\ -\beta_s < c_j < \beta_s \Rightarrow w_j = 0 \end{aligned} \tag{7}$$

where variable c_j is defined according to Equation 5. The LASSO-FP algorithm continually adjusts the weight vector (by adding or subtracting basis functions from the active set) while satisfying the optimality conditions. The algorithm is initialized with $\mathcal{I} \leftarrow \emptyset$ and $w \leftarrow \mathbf{0}$. The optimality conditions can be satisfied with this initialization for some $\bar{\beta}_s > \beta_s$. The algorithm proceeds to reduce $\bar{\beta}_s$ while satisfying the optimality conditions until $\bar{\beta}_s = \beta_s$ or some other termination criteria is triggered. The other termination criteria we used were a maximum number of basis functions (k') and a threshold on the change in the norm of the Bellman residual. Note that k' and β_s are related.

The optimality conditions ensure that $|c_\mathcal{I}| = \bar{\beta}_s$ for all basis functions in the active set. This property is maintained by changing the weight vector according to

$$\Delta w_\mathcal{I} = \left(\Phi_\mathcal{I}^T D_\rho (\Phi_\mathcal{I} - \gamma P^\pi \Phi_\mathcal{I}) + \beta_m \Phi_\mathcal{I}^T L D_\rho L \Phi_\mathcal{I}\right)^{-1} \text{sign}(c_\mathcal{I})$$

where $\text{sign}(c_\mathcal{I})$ replaces the entries in $c_\mathcal{I}$ with values ± 1 depending on the sign. The change in the weight vector $\Delta w_\mathcal{I}$ dictates how the vector c changes:

$$\Delta c = \left(\Phi^T D_\rho (\Phi_\mathcal{I} - \gamma P^\pi \Phi_\mathcal{I}) + \beta_m \Phi^T L D_\rho L \Phi_\mathcal{I}\right) \Delta w_\mathcal{I}.$$

The vector Δc allows one to compute if and when an inactive basis function $j \notin \mathcal{I}$ will have a value c_j that reaches the same value as those in the active set. The first inactive basis function that reaches this point is computed as:

$$[\alpha^*, j^*] = [\min{}^+, \text{argmin}]_{j \notin \mathcal{I}} \left(\frac{c_j - \bar{\beta}_s}{\Delta c_j - 1}, \frac{c_j + \bar{\beta}_s}{\Delta c_j + 1} \right)$$

where $\min^+$ indicates the minimization is only over positive values, α^* is the minimizing value, and j^* is the minimizing argument.

Before adding basis function j^* to the active set, the LASSO-FP algorithm must check to see whether an element in the active set $j \in \mathcal{I}$ has a coefficient w_j differing in sign with c_j as such an event would violate the optimality conditions.[7] The first active basis function that reaches this point is computed as:

$$[\alpha^\#, j^\#] = [\min{}^+, \text{argmin}]_{j \in \mathcal{I}} \left(-\frac{w_j}{\Delta w_j} \right).$$

[6] Our terminology is slightly different from that used by Kolter and Ng [27]. Their LARS-TD algorithm is the same as our LASSO-FP algorithm with $\beta_m = 0$. The distinction we draw between LARS and LASSO is whether the algorithm only adds basis functions to the active set (LARS) or both adds and removes basis functions (LASSO).

[7] Note this is the only difference between LASSO-FP and LARS-FP. LARS-FP is not required to ensure w_j and c_j have the same sign. Therefore, LARS-FP does not remove basis functions from the active set.

If all elements in the minimization are negative, then $\alpha^{\#}$ is set to ∞. If the step size $\alpha^* < \alpha^{\#}$, then basis function j^* is added to the active set. If the reverse is true, then basis function $j^{\#}$ is removed from the active set. Pseudocode for LARS-FP and LASSO-FP is given in Algorithm 5.

The LARS-FP and LASSO-FP algorithms adjust the coefficient vector $w_{\mathcal{I}}$ in an equiangular direction. This means that the residual is never made completely orthogonal with the selected basis functions $\Phi_{\mathcal{I}}$. A common "fix" to this issue is to enforce orthogonality once LARS-FP and LASSO-FP terminate. We list this as an optional step at the end of the Algorithm 5.

Algorithm 5: `LARS-FP and LASSO-FP with Laplacian-based Regularization`

Input: $\{s_i, r_i, s'_i\}_{i=1}^n$, samples generated using policy π
$\phi : S \to \mathbb{R}^K$, basis function
$\rho : S \to \mathbb{R}^+$, weighting over the states
L, graph Laplacian defined over states $\{s_i\}_{i=1}^n$ (graph edges denoted with $\sim$)
$\gamma \in [0, 1]$, discount factor
$\beta_s \in \mathbb{R}^+$, L_1 regularization parameter
$\beta_m \in \mathbb{R}^+$, Laplacian-based regularization parameter
$k' \leq K$, maximum allowable number of basis functions

Output: $\mathcal{I}$, set of selected basis functions (indices into ϕ)
$w_{\mathcal{I}}$, weight vector such that $\hat{V}(s) = \phi_{\mathcal{I}}(s)^T w_{\mathcal{I}}$

$c \leftarrow \sum_{i=1}^n \rho(s_i)\phi(s_i)r_i$
$\left[\bar{\beta}_s,\, j^*\right] \leftarrow \left[\max, \operatorname{argmax}\right]_j (|c_j|)$
Initialize active set $\mathcal{I} \leftarrow \{j^*\}$, $w \leftarrow \mathbf{0}$

while $(\bar{\beta}_s > \beta_s)$ and $(|\mathcal{I}| \leq k')$ and (Bellman residual not converged) **do**

1. Compute weight update direction $\Delta w_{\mathcal{I}}$:
   $\Delta w_{\mathcal{I}} \leftarrow \hat{A}_{\mathcal{I},\mathcal{I}}^{-1} \operatorname{sign}(c_{\mathcal{I}})$
   where: $\hat{A}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^n \rho(s_i) \left[\phi_{\mathcal{I}}(s_i)(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s'_i))^T + \beta_m g_{\mathcal{I}}(s_i) g_{\mathcal{I}}(s_i)^T\right]$
   $g(s_i) \leftarrow L(s_i, s_i)\, \phi(s_i)$
   $g(s_i) \leftarrow g(s_i) + L(s_i, s_{nbr})\, \phi(s_{nbr}) \qquad \forall \{s_{nbr} | s_{nbr} \neq s \wedge s \sim s_{nbr}\}$
2. Compute correlation update direction Δc:
   $\Delta c \leftarrow \sum_{i=1}^n \rho(s_i) \left[\phi(s_i)\, (\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s'_i))^T \Delta w_{\mathcal{I}} + \beta_m\, g(s_i) g_{\mathcal{I}}\, (s_i)^T \Delta w_{\mathcal{I}}\right]$
3. Find step size to add element to active set:
   $[\alpha^*,\, j^*] \leftarrow \left[\min^+, \operatorname{argmin}\right]_{j \notin \mathcal{I}} \left(\frac{c_j - \bar{\beta}_s}{\Delta c_j - 1},\, \frac{c_j + \bar{\beta}_s}{\Delta c_j + 1}\right)$
4. Find step size to remove element from active set:
   If (using LARS-FP), $\alpha^{\#} \leftarrow \infty$
   Else, $\left[\alpha^{\#},\, j^{\#}\right] \leftarrow \left[\min^+, \operatorname{argmin}\right]_{j \in \mathcal{I}} \left(-\frac{w_j}{\Delta w_j}\right)$
5. Update $\bar{\beta}_s, w_{\mathcal{I}}, c$:
   $\alpha \leftarrow \min(\alpha^*, \alpha^{\#}, \bar{\beta}_s - \beta_s), \quad \bar{\beta}_s \leftarrow \bar{\beta}_s - \alpha, \quad w_{\mathcal{I}} \leftarrow w_{\mathcal{I}} + \alpha \Delta w_{\mathcal{I}}, \quad c \leftarrow c - \alpha \Delta c$
6. Adjust active set:
   If $(\alpha^* < \alpha^{\#})$, $\mathcal{I} \leftarrow \mathcal{I} \cup \{j^*\}$
   Else, $\mathcal{I} \leftarrow \mathcal{I} - \{j^{\#}\}$

end while

OPTIONAL: $w_{\mathcal{I}} \leftarrow \hat{A}_{\mathcal{I},\mathcal{I}}^{-1} \hat{b}_{\mathcal{I}}$ where: $\hat{b}_{\mathcal{I}} \leftarrow \sum_{i=1}^n \rho(s_i)\phi_{\mathcal{I}}(s_i) r_i$

4.2.4 Other Least-Squares Algorithms

The previous three sections described the OMP-FP, ORMP-FP, LASSO-FP, and LARS-FP algorithms. Each algorithm can be changed to instead optimize the BR, H_1, and H_2 least-squares criteria. We describe here how the algorithms would change to meet these different objectives. We do this in detail for OMP and then simply highlight where the (similar) changes need to be made in ORMP, LASSO, and LARS.

The memory and computation requirements are identical whether using the FP, BR, or H_2 least-squares criteria. The hybrid algorithm H_1 however requires more memory and computation time. As shown in the equations below, H_1 requires forming two matrices of size $K \times K$ where K is the number of basis functions in the dictionary. This can be prohibitively large depending on the size of the dictionary. Note that all basis selection algorithms when using FP, BR, and H_1 do not form matrices larger than $k' \times k'$ where $k' \leq K$ is specified by the user to be the maximum number of basis functions that the algorithm can select.

The following four lines of Algorithm 3 (OMP-FP) would need to change to accommodate the objective functions of BR, H_1, and H_2.

(1) The first time c is initialized.

BR	$c \leftarrow \sum_{i=1}^{n} \rho(s_i)(\phi(s_i) - \gamma\phi(s_i'))r_i$
H_2	$c \leftarrow \sum_{i=1}^{n} \rho(s_i)(\phi(s_i) - \xi\gamma\phi(s_i'))r_i$
H_1	$c \leftarrow \xi\hat{b}_{BR} + (1-\xi)(\hat{A}^{FP})^T\hat{C}^{-1}\hat{b}_{FP}$
	$\hat{b}_{BR} \leftarrow \sum_{i=1}^{n} \rho(s_i)(\phi(s_i) - \gamma\phi(s_i'))r_i$
	$\hat{b}_{FP} \leftarrow \sum_{i=1}^{n} \rho(s_i)\phi(s_i)r_i$
	$\hat{A}^{FP} \leftarrow \sum_{i=1}^{n} \rho(s_i)\left[\phi(s_i)(\phi(s_i) - \gamma\phi(s_i'))^T + \beta_m g(s_i)g(s_i)^T\right]$
	$\hat{C} \leftarrow \sum_{i=1}^{n} \rho(s_i)\phi(s_i)\phi(s_i)^T$

(2) Computing $\hat{A}_{\mathcal{I},\mathcal{I}}$ in Step 3.

BR	$\hat{A}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^{n} \rho(s_i)\left[(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))^T + \beta_m g_{\mathcal{I}}(s_i)g_{\mathcal{I}}(s_i)^T\right]$
H_2	$\hat{A}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^{n} \rho(s_i)\left[(\phi_{\mathcal{I}}(s_i) - \xi\gamma\phi_{\mathcal{I}}(s_i'))(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))^T + \beta_m g_{\mathcal{I}}(s_i)g_{\mathcal{I}}(s_i)^T\right]$
H_1	$\hat{A}_{\mathcal{I},\mathcal{I}} \leftarrow \xi\hat{A}^{BR}_{\mathcal{I},\mathcal{I}} + (1-\xi)(\hat{A}^{FP}_{\mathcal{I},\mathcal{I}})^T\hat{C}^{-1}_{\mathcal{I},\mathcal{I}}\hat{A}^{FP}_{\mathcal{I},\mathcal{I}}$
	$\hat{A}^{BR}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^{n} \rho(s_i)\left[(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))^T + \beta_m g_{\mathcal{I}}(s_i)g_{\mathcal{I}}(s_i)^T\right]$
	$\hat{A}^{FP}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^{n} \rho(s_i)\left[\phi_{\mathcal{I}}(s_i)(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))^T + \beta_m g_{\mathcal{I}}(s_i)g_{\mathcal{I}}(s_i)^T\right]$
	$\hat{C}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^{n} \rho(s_i)\phi_{\mathcal{I}}(s_i)\phi_{\mathcal{I}}(s_i)^T$

(3) Computing $\hat{b}_{\mathcal{I}}$ in Step 3.

BR	$\hat{b}_{\mathcal{I}} \leftarrow \sum_{i=1}^{n} \rho(s_i)(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))r_i$
H_2	$\hat{b}_{\mathcal{I}} \leftarrow \sum_{i=1}^{n} \rho(s_i)(\phi_{\mathcal{I}}(s_i) - \xi\gamma\phi_{\mathcal{I}}(s_i'))r_i$
H_1	$\hat{b}_{\mathcal{I}} \leftarrow \xi\hat{b}_{\mathcal{I}}^{BR} + (1-\xi)(\hat{A}_{\mathcal{I},\mathcal{I}}^{FP})^T\hat{C}_{\mathcal{I},\mathcal{I}}^{-1}\hat{b}_{\mathcal{I}}^{FP}$
	$\hat{b}_{\mathcal{I}}^{BR} \leftarrow \sum_{i=1}^{n} \rho(s_i)(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))r_i$
	$\hat{b}_{\mathcal{I}}^{FP} \leftarrow \sum_{i=1}^{n} \rho(s_i)\phi_{\mathcal{I}}(s_i)r_i$
	$\hat{A}_{\mathcal{I},\mathcal{I}}^{FP} \leftarrow \sum_{i=1}^{n} \rho(s_i)\left[\phi_{\mathcal{I}}(s_i)(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))^T + \beta_m g_{\mathcal{I}}(s_i)g_{\mathcal{I}}(s_i)^T\right]$
	$\hat{C}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^{n} \rho(s_i)\phi_{\mathcal{I}}(s_i)\phi_{\mathcal{I}}(s_i)^T$

(4) Updating c in Step 5.

BR	$c \leftarrow \sum_{i=1}^{n} \rho(s_i)\left[(\phi(s_i) - \gamma\phi(s_i'))(r_i - (\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))^T w_{\mathcal{I}}) - \beta_m g(s_i)g_{\mathcal{I}}(s_i)^T w_{\mathcal{I}}\right]$
H_2	$c \leftarrow \sum_{i=1}^{n} \rho(s_i)\left[(\phi(s_i) - \xi\gamma\phi(s_i'))(r_i - (\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))^T w_{\mathcal{I}}) - \beta_m g(s_i)g_{\mathcal{I}}(s_i)^T w_{\mathcal{I}}\right]$
H_1	$c \leftarrow \xi c_{BR} + (1-\xi)(\hat{A}^{FP})^T\hat{C}^{-1}c_{FP}$
	$c_{BR} \leftarrow \sum_{i=1}^{n} \rho(s_i)\left[(\phi(s_i) - \gamma\phi(s_i'))(r_i - (\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))^T w_{\mathcal{I}}) - \beta_m g(s_i)g_{\mathcal{I}}(s_i)^T w_{\mathcal{I}}\right]$
	$c_{FP} \leftarrow \sum_{i=1}^{n} \rho(s_i)\left[\phi(s_i)(r_i - (\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s_i'))^T w_{\mathcal{I}}) - \beta_m g(s_i)g_{\mathcal{I}}(s_i)^T w_{\mathcal{I}}\right]$
	$\hat{A}^{FP} \leftarrow \sum_{i=1}^{n} \rho(s_i)\left[\phi(s_i)(\phi(s_i) - \gamma\phi(s_i'))^T + \beta_m g(s_i)g(s_i)^T\right]$
	$\hat{C} \leftarrow \sum_{i=1}^{n} \rho(s_i)\phi(s_i)\phi(s_i)^T$

The changes to ORMP, LARS, and LASSO are very similar to the changes made for OMP; therefore, we just point out the lines that need to be edited. For ORMP, four lines would need to change: computing $\hat{b}_{\mathcal{I}_{+j}}$ in Step 1, computing $\hat{A}_{\mathcal{I}_{+j},\mathcal{I}_{+j}}$ in Step 1, computing $\hat{A}_{\mathcal{I},\mathcal{I}}$ in Step 3, and computing $\hat{b}_{\mathcal{I}}$ in Step 3. For LARS and LASSO, four lines would need to change: the first time c is initialized, computing $\hat{A}_{\mathcal{I},\mathcal{I}}$ in Step 1, computing Δc in Step 2, and computing $\hat{b}_{\mathcal{I}}$ at the final optional step of the algorithm.

The ORMP algorithm merits further attention. This algorithm is particularly interesting because it uses the least-squares method to determine which basis function to include in the active set. The best basis function is determined by: $\operatorname{argmax}_{j\notin\mathcal{I}} \left(b_{\mathcal{I}_{+j}}^T A_{\mathcal{I}_{+j},\mathcal{I}_{+j}}^{-1} b_{\mathcal{I}_{+j}}\right)$. In other words, ORMP considers the impact of each inactive basis function on the least-squares problem. When the BR least-squares algorithm is used, the best basis function is:

$$
\begin{aligned}
j^* &\leftarrow \operatorname*{argmax}_{j\notin\mathcal{I}} \left((b_{\mathcal{I}_{+j}}^{BR})^T (A_{\mathcal{I}_{+j},\mathcal{I}_{+j}}^{BR})^{-1} b_{\mathcal{I}_{+j}}^{BR}\right) \\
&\leftarrow \operatorname*{argmax}_{j\notin\mathcal{I}} \left((b_{\mathcal{I}_{+j}}^{BR})^T w_{\mathcal{I}_{+j}}^{BR}\right) \\
&\leftarrow \operatorname*{argmax}_{j\notin\mathcal{I}} \left((R^\pi)^T D_\rho(\Phi_{\mathcal{I}_{+j}} - \gamma P^\pi \Phi_{\mathcal{I}_{+j}}) w_{\mathcal{I}_{+j}}^{BR}\right) \\
&\leftarrow \operatorname*{argmax}_{j\notin\mathcal{I}} \langle R^\pi,\, \hat{V}_{\mathcal{I}_{+j}}^{BR} - \gamma P^\pi \hat{V}_{\mathcal{I}_{+j}}^{BR}\rangle_\rho
\end{aligned}
$$

where $\langle\cdot,\cdot\rangle_\rho$ denotes the ρ-weighted inner product. This makes intuitive sense since the BR least-squares problem is fitting a function $\hat{V}^{BR}$ that minimizes $\|R^\pi + \gamma P^\pi \hat{V}^{BR} - \hat{V}^{BR}\|_\rho^2$. Now consider the direct scheme for combining ORMP and the FP least-squares algorithm. One can show the

best inactive basis function for ORMP-FP is: $\text{argmax}_{j \notin \mathcal{I}} \langle R^\pi, \hat{V}^{FP}_{\mathcal{I}_{+j}} \rangle_\rho$. This maximization does not make sense since selecting basis functions using this criteria leads to a value function that approximates the reward R^π. A simple idea to try to rescue ORMP-FP is to change the maximization to: $\text{argmax}_{j \notin \mathcal{I}} \left((b^{BR}_{\mathcal{I}_{+j}})^T (A^{FP}_{\mathcal{I}_{+j}, \mathcal{I}_{+j}})^{-1} b^{FP}_{\mathcal{I}_{+j}} \right)$. Notice the use of the two different vectors $b^{BR}_{\mathcal{I}_{+j}}$ and $b^{FP}_{\mathcal{I}_{+j}}$. This leads to selecting basis functions according to: $\text{argmax}_{j \notin \mathcal{I}} \langle R^\pi, \hat{V}^{FP}_{\mathcal{I}_{+j}} - \gamma P^\pi \hat{V}^{FP}_{\mathcal{I}_{+j}} \rangle_\rho$. Although this is seemingly more valid than the original formulation, it is still problematic. The underlying problem is that the FP least-squares formulation does not correspond to any optimization problem (i.e. the FP objective function $\|\Pi_\rho(R^\pi + \gamma P^\pi \hat{V}^{FP}) - \hat{V}^{FP}\|^2_\rho$ can always be set to 0 for any set of basis functions).

One must be careful when directly combining least-squares policy evaluation algorithms and basis selection algorithms. The result of this analysis is that ORMP-FP is **not** valid but ORMP-BR is valid. However, ORMP can be used with both FP and BR in the *indirect* schemes that we describe next.

4.3 Indirect Schemes

The indirect scheme uses an iterative approach to sparse approximate policy evaluation. The iterative approach alternates between (1) setting the target function using the Bellman backup operator, and (2) representing the the target function using the basis selection algorithm. This potentially makes the indirect scheme more computationally intensive than the the direct scheme, but it frees up the basis selection algorithm to choose the best basis functions for fitting the approximate value function (instead of fitting the ordered elements in the Neumann series). We describe the iterative, indirect scheme in Algorithm 6. This is a general framework which can utilize any sparse basis selection (regression) algorithm. The sparse basis selection algorithm is denoted as input ***BSel***(y) where y is the target function that ***BSel*** fits using dictionary Φ. For ***BSel***, we evaluated the pure regression versions of OMP, ORMP, LASSO, and LARS with the only exception being they were augmented to include Laplacian-based regularization. The pure regression versions of OMP and ORMP without regularization are described in Algorithm 2.

4.4 Approximating the Action-Value Function

The previous two sections described the direct and indirect schemes for approximating the value function. The same algorithms can also be used to approximate the action-value function. The graph-based basis functions, which are defined just over states, can be also used to approximate the action-value function. This is accomplished by using the basis functions for each discrete action. For example, consider a MDP with two actions, a_1 and a_2. The approximate action-value function $\hat{Q}$ can take the form:

$$\hat{Q} = \begin{bmatrix} \hat{Q}(\cdot, a_1) \\ \hat{Q}(\cdot, a_2) \end{bmatrix} = \begin{bmatrix} \Phi_{\mathcal{I}_{a_1}} & \mathbf{0} \\ \mathbf{0} & \Phi_{\mathcal{I}_{a_2}} \end{bmatrix} \begin{bmatrix} w_{\mathcal{I}_{a_1}} \\ w_{\mathcal{I}_{a_2}} \end{bmatrix} = \Phi_{\mathcal{I}} \, w_{\mathcal{I}}.$$

Notice the approximate action-value function can use a different set of basis functions for each action: $\hat{Q}(\cdot, a_1)$ uses the basis functions indexed by $\mathcal{I}_{a_1}$ and $\hat{Q}(\cdot, a_2)$ uses basis functions indexed by $\mathcal{I}_{a_2}$.

Algorithm 6: `Indirect Scheme for Sparse Approximate Policy Evaluation`

Input: $\{s_i, r_i, s'_i\}_{i=1}^n$, samples generated using policy π
$\phi : S \rightarrow \mathbb{R}^K$, basis function
$\rho : S \rightarrow \mathbb{R}^+$, weighting over the states
L, graph Laplacian defined over states $\{s_i\}_{i=1}^n$ (graph edges denoted with $\sim$)
$\gamma \in [0, 1]$, discount factor
$\beta_m \in \mathbb{R}^+$, Laplacian-based regularization parameter
$maxIter \in \mathbb{N}$, maximum number of iterations
BSel(y), sparse basis selection algorithm that approximates a target function y using the dictionary ϕ. The termination criteria for ***BSel*** includes:
$k' \leq K$, maximum allowable number of basis functions
a threshold on the residual $\|y - \Phi w\|_\rho^2$
any other algorithm specific parameters (e.g. β_s for LARS/LASSO)

Output: $\mathcal{I}$, set of selected basis functions (indices into ϕ)
$w_\mathcal{I}$, weight vector such that $\hat{V}(s) = \phi_\mathcal{I}(s)^T w_\mathcal{I}$

Initialize active set $\mathcal{I} \leftarrow \emptyset$, $\hat{w}_\mathcal{I} \leftarrow \emptyset$, $iter \leftarrow 0$

while $(iter < maxIter)$ and (Bellman residual not converged) **do**

1. Form target vector y using the Bellman backup:
$y_i \leftarrow r_i + \gamma \phi_\mathcal{I}(s'_i)^T w_\mathcal{I} \qquad \forall i$
2. Run the sparse basis selection (regression) algorithm to fit y:
$[\mathcal{I}, w_\mathcal{I}] \leftarrow$ ***BSel***(y)
3. OPTIONAL: Adjust $w_\mathcal{I}$ using one of the least-squares methods:
$w_\mathcal{I} \leftarrow \hat{A}_{\mathcal{I},\mathcal{I}}^{-1} \hat{b}_\mathcal{I}$
For example, if using FP least-squares method, then:
$\hat{A}_{\mathcal{I},\mathcal{I}} \leftarrow \sum_{i=1}^n \rho(s_i) \left[\phi_\mathcal{I}(s_i)(\phi_\mathcal{I}(s_i) - \gamma \phi_\mathcal{I}(s'_i))^T + \beta_m g_\mathcal{I}(s_i) g_\mathcal{I}(s_i)^T\right]$
$\hat{b}_\mathcal{I} \leftarrow \sum_{i=1}^n \rho(s_i) \phi_\mathcal{I}(s_i) r_i$
4. Increment the iteration count:
$iter \leftarrow iter + 1$

end while

Algorithms 3, 4, 5, and 6 can be used with this definition without changing any steps. However, if these algorithms are used without changes, the number of selected basis functions per action may not be equal. For the MDP with two actions a_1 and a_2, this means $|\mathcal{I}_{a_1}|$ will not necessarily be equal to $|\mathcal{I}_{a_2}|$. It may be desirable to require the number of basis functions per action to be equal (or approximately equal). This constraint can easily be added to the indirect scheme (Algorithm 6) and to the direct schemes involving OMP and ORMP (Algorithms 3 and 4). It does not seem possible to add this constraint to the direct scheme involving LASSO and LARS (Algorithm 5) without making other significant algorithmic changes.

Algorithms 3, 4, 5, and 6 can produce approximate action-value functions for a specific policy. These algorithms can also be used within least-squares policy iteration (LSPI) [20] to compute an approximation of Q^*. One LSPI iteration takes a batch of MDP samples $\{s_i, a_i, r_i, s'_i\}_{i=1}^n$ and a policy π and produces $\hat{Q}$, an approximation of Q^π. The greedy policy implicitly defined by $\hat{Q}$ is then used in the next iteration of LSPI.

5 Policy Evaluation Experiments

The following components were varied in the experiments:

- least-squares method (FP, BR, and H_2).
- basis selection method (OMP, ORMP, LASSO, and LARS).
- scheme for sparse approximate policy evaluation (direct and indirect).
- amount of Laplacian-based regularization (β_m).
- dictionary (PVF and diffusion wavelet basis functions).

To get a solid understanding of how each component influences the policy evaluation problem, we chose the 50 state chain MDP [20]. This domain is easily visualized. The problem consists of 50 states ($s_i, i \in [1, 50]$) and two actions moving the agent left ($s_i \rightsquigarrow s_{i-1}$) or right ($s_i \rightsquigarrow s_{i+1}$). Actions succeed with probability 0.8; failed actions result in staying in the same state (probability 0.1) or moving to the adjacent state in the opposite direction (probability 0.1). The reward function is defined as $+1$ in states s_{10} and s_{41} and zero everywhere else. The discount factor is $\gamma = 0.9$.

We consider the task of evaluating the optimal policy π^*. Rather than sampling from π^* to generate a data set, we used the true model P^{π^*} and R^{π^*} in the following experiments. This choice was made to remove any influence that sampling may have on each component that we vary so that we can adequately compare and contrast performance.

The graph used to form the PVFs and diffusion wavelets consists of 50 vertices with self-edges and edges between "adjacent" vertices. The PVF dictionary, which was constructed using the combinatorial Laplacian, consists of 50 global basis functions. The diffusion wavelet tree was constructed using the parameter $\epsilon = 10^{-4}$. The number of scaling and wavelet functions is shown in Table 5. We evaluated three dictionaries constructed from this tree. The first dictionary con-

Tree Level j	$\lvert\psi_{j-1}\rvert$	$\lvert\phi_j\rvert$
1	0	50
2	9	41
3	13	28
4	7	21
5	5	16
6	5	11
7	3	8
8	2	6
9	2	4
10	1	3

Table 1: Number of wavelet and scaling functions at each tree level for the 50 state chain MDP.

sisted of all 235 functions in the tree (47 wavelet and 188 scaling functions). The second dictionary consisted of the 135 functions at tree level 3 or greater (38 wavelet and 97 scaling functions). The 100 extra functions in the first dictionary consist of very localized basis functions as well as some

oscillatory functions. Note that both the first and second dictionaries are overcomplete, so selecting elements from these dictionaries can lead to linear dependence in the basis functions. The third dictionary consisted of all 47 wavelet functions and the 3 scaling functions at tree level 10. This third dictionary is orthonormal whereas the first two dictionaries are overcomplete. A further optimization that we did not pursue would be to select the "best" such orthonormal dictionary (amongst the 10 possible orthonormal dictionaries) instead of just using the dictionary that reaches to tree level 10.

To help navigate the figures at the back of this paper, Table 5 provides a list of the figures indexed by scheme, algorithm, and dictionary. The next four sections describe the results in detail. The reader may find it easier to read the summary of the results in Section 6 before going into the finer details.

Figure	Scheme	Algorithm	Dictionary
1	Direct	OMP-FP, LASSO-FP	PVFs
2	Direct	OMP-BR, ORMP-BR, LASSO-BR	PVFs
3	Direct	OMP-H_2	PVFs
4	Indirect FP & BR	OMP	PVFs
5	Indirect FP & BR	ORMP	PVFs
6	Indirect FP & BR	LASSO	PVFs
7	Direct	OMP-FP, LASSO-FP, LARS-FP	235 Diffusion Wavelets
8	Direct	OMP-FP, LASSO-FP, LARS-FP	135 Diffusion Wavelets
9	Direct	OMP-FP, LASSO-FP	50 Orthog. Diffusion Wavelets
10	Direct	ORMP-BR	235 Diffusion Wavelets
11	Direct	OMP-BR, ORMP-BR, LASSO-BR	135 Diffusion Wavelets
12	Direct	OMP-BR, ORMP-BR, LASSO-BR	50 Orthog. Diffusion Wavelets
13	Indirect FP	OMP, ORMP, LASSO, LARS	235 Diffusion Wavelets
14	Indirect FP	OMP, ORMP, LASSO, LARS	135 Diffusion Wavelets
15	Indirect FP	OMP, ORMP, LASSO	50 Orthog. Diffusion Wavelets
16	Indirect BR	OMP, ORMP, LASSO, LARS	235 Diffusion Wavelets
17	Indirect BR	OMP, ORMP, LASSO, LARS	135 Diffusion Wavelets
18	Indirect BR	OMP, ORMP, LASSO	50 Orthog. Diffusion Wavelets

Table 2: List of figures.

5.1 Direct Scheme with PVF Dictionary

Figure 1 shows results using OMP-FP and LASSO-FP. We varied the amount of regularization ($\beta_m = 0$ and $\beta_m = 0.1$) and the number of basis functions (4, 8, and 12) that the algorithms could select. For LASSO-FP, we also present results where the optional orthogonalization step at the end of Algorithm 5 is used (Orthog.) and not used (Not Orthog.). Since LASSO-FP did not remove any basis functions from the active set in this experiment, it produces identical results to LARS-FP. In all of the figures in this paper, if there is no plot involving LARS, that is because LARS and LASSO produced identical results.

The two figures in the top row of Figure 1 are for OMP-FP with and without Laplacian-based regularization. The results with regularization clearly demonstrate the potential benefits the smoothness assumption can have on basis selection under the direct scheme. The two figures in the second row of Figure 1 show the LASSO-FP results without the optional orthogonalization step. Notice how the approximate value functions have small magnitude. This is due to the conservative nature of the LASSO algorithm's equiangular approach. The two figures in the third row show the same approximate value functions as those in the second row, but with the y-axis limits adjusted to show finer detail. The two figures in the last row of Figure 1 show the LASSO-FP results with the optional orthogonalization step. Qualitatively, OMP-FP seems to perform slightly better than LASSO-FP on this problem.

Figure 2 shows results using OMP-BR, ORMP-BR, and LASSO-BR. The regularization parameter was set to $\beta_m = 0$ and $\beta_m = 0.1$. Notice that OMP-BR and LASSO-BR did not perform well. We show results using 20 basis functions, which is more than enough for an excellent approximation of the value function. On the other hand, ORMP-BR produced good approximations. The true value function is almost exactly fit using 12 basis functions and no regularization. When using 4 or 8 basis functions, Laplacian-based regularization smooths out some of the jaggedness of the approximate value functions.

Recall the hybrid algorithm H_2 uses a parameter ξ that causes the algorithm to produce the same results as FP when $\xi = 0$ and BR when $\xi = 1$. Figure 3 shows results using OMP-H_2 and LASSO-H_2. Intermediate values of ξ between 0 and 1 tend to produce approximate value functions between the extremes produced by the FP and BR algorithms.

5.2 Indirect Scheme with PVF Dictionary

The experiments in this section were all conducted using Algorithm 6 under three conditions. First, the while loop in Algorithm 6 was executed for 10 iterations. Second, we used a single termination criterion for the basis selection algorithm. The algorithm stopped when it had selected a specified number of basis functions. Third, we always used the optional third step in Algorithm 6 which is to set the weights on the selected features using a least-squares method. We used the BR and FP least-squares methods. Since BR and FP produced similar results, we do not report results using the hybrid method H_2.

Figure 4 shows results using the OMP algorithm with both the BR and FP least-squares methods. The regularization was again varied by setting $\beta_m = 0$ and $\beta_m = 0.1$. Likewise, the number of basis functions was varied between 4, 8, and 12. The plots on the left of Figure 4 show the final approximate value function and the plots on the right show the norm of the Bellman error $\|T^\pi(\Phi_\mathcal{I} w_\mathcal{I}) - \Phi_\mathcal{I} w_\mathcal{I}\|^2$ after each iteration of Algorithm 6. We just show the Bellman error plots for the OMP algorithm to point out that the Bellman error is not monotonically decreasing. The Bellman error plots for ORMP and LASSO were very similar to those for OMP.

Figures 5 and 6 show results using the ORMP and LASSO algorithms respectively. LARS is now shown because it was identical to LASSO. ORMP, LASSO, and OMP all achieved very similar performance for both the FP and BR least-squares methods. The only noticeable difference was when just 4 basis functions were used. In this case, the approximate value function using the FP least-squares method proved to be smoother than the value function learned using BR.

The indirect scheme for sparse approximate policy evaluation performed well for all basis

selection algorithms and both the FP and BR least-squares methods. The same cannot be said for the direct scheme. The direct scheme worked well for OMP-FP and LASSO-FP (although not as well as their indirect versions) and for ORMP-BR. As we show in the next two sections, this split (OMP and LASSO performing well with FP under the direct scheme with ORMP performing well with BR) also held up when using the diffusion wavelet dictionary.

5.3 Direct Scheme with Diffusion Wavelet Dictionary

Figures 7, 8, and 9 show the results using OMP-FP, LASSO-FP, and LARS-FP with the three different diffusion wavelet dictionaries. The dictionary containing all 235 scaling and wavelet functions is used for Figure 7, the dictionary containing the 135 scaling and wavelet functions beyond tree level 3 is used for Figure 8, and the orthonormal dictionary is used for Figure 9. We varied the amount of regularization ($\beta_m = 0$ and $\beta_m = 0.1$) and the number of basis functions (4, 8, and 12) that the algorithms could select.

The two figures in the top row of Figure 7 are for OMP-FP with and without Laplacian-based regularization. Notice the approximate value function estimated using 12 basis functions (with $\beta_m = 0$) is unstable. This occurs because the matrix $\hat{A}_{\mathcal{I},\mathcal{I}}^{-1}$ became nearly singular. The resulting value functions were slightly more stable when $\beta_m = 0.1$. The results using LASSO-FP and LARS-FP were nearly identical. When the optional orthogonalization step was not used, the approximate value functions tracked the shape of the true value function well. LASSO-FP produced more accurate approximations using 8 and 12 basis functions than LARS-FP when the orthogonalization step was used.

The approximate value functions shown in Figure 8 are formed from a less expressive, but still overcomplete, dictionary than those in Figure 7. Interestingly, the results using OMP-FP were better while those using LASSO-FP and LARS-FP were worse. This trend continued with the approximate value functions shown in Figure 9. In Figure 9, OMP-FP produced even more accurate approximate value functions than those shown in Figure 8 while LASSO-FP produced worse approximations. This trend is best explained by highlighting the difference between OMP-FP's aggressive behavior and LASSO-FP's conservative behavior. OMP-FP fully utilizes the selected basis functions, $\Phi_{\mathcal{I}}$, by forcing $\Phi_{\mathcal{I}}$ to be orthogonal with the Bellman residual. This lead to unstable behavior given the most expressive diffusion wavelet dictionary. The conservative LASSO-FP updates help to maintain stability.

Figures 10, 11, and 12 show the results using OMP-BR, ORMP-BR, and LASSO-BR with the three diffusion wavelet dictionaries. Just like with the PVF dictionary, the only algorithm that performed well was ORMP-BR. The approximate value functions formed using 8 and 12 basis functions were similar across all three dictionaries. The only noticeable difference was when 4 basis functions were used. In that case, the approximate value functions learned using the overcomplete dictionaries were much more accurate than the approximate value function learned using the orthonormal dictionary.

5.4 Indirect Scheme with Diffusion Wavelet Dictionary

Algorithm 6 was used for all the experiments in this section under the same conditions described in Section 5.2 when using the PVF dictionary. To save space, we just show the approximate

value functions estimated without regularization. The impact of Laplacian-based regularization ($\beta_m = 0.1$) was similar to that shown in Figures 4, 5, and 6 when using the PVF dictionary.

Figures 13, 14, and 15 show the results using the OMP, ORMP, LASSO, and LARS algorithms with the FP least-squares method. For OMP and ORMP, the approximate value functions formed from the orthonormal diffusion wavelet dictionary are smoother and more accurate than those formed from the overcomplete dictionaries. LASSO and LARS, on the other hand, produce accurate approximate value functions using 8 and 12 basis functions from all three diffusion wavelet dictionaries. When only 4 basis functions are used, LASSO and LARS produce much better approximate value functions when using the orthonormal dictionary (compared to the overcomplete dictionaries).

Figures 16, 17, and 18 show the results using OMP, ORMP, LASSO, and LARS with the BR least-squares method. All four selection algorithms produce accurate approximate value functions when using 12 basis functions. The only exception was OMP when using the dictionary of 135 scaling and wavelet functions beyond tree level three. That particular approximate value function had the correct shape but smaller magnitude. When using 8 basis functions, LASSO and LARS produced accurate value functions for all three dictionaries while OMP and ORMP only performed well using the orthonormal dictionary and the 135 element dictionary. The only time 4 basis functions resulted in an accurate approximate value function was when the orthonormal dictionary was used. This held true for all four selection algorithms.

6 Summary and Future Work

Proto-value functions and diffusion wavelets are graph-based basis functions that capture topological structure of the MDP state space. The basis functions are independent of any policy and therefore can be used to approximate any policy's value function. A mechanism is required though to select a subset of the basis functions for approximating a value function. The previous approach to using PVFs and diffusion wavelets used the following basis selection heuristic: the most global functions were selected regardless of the policy being evaluated. This heuristic is simple and leads to smooth approximations, but it does not fully utilize the graph-based dictionaries. To make better use of the dictionaries, a sparse basis selection algorithm must be combined with approximate policy evaluation. In this paper, we evaluated a scheme that directly combines basis selection and policy evaluation and a scheme that indirectly combines them via an iterative process. Both schemes are general and can be used with any set of basis functions. We considered least-squares policy evaluation algorithms based on the fixed-point method, the Bellman residual minimization method, and a hybrid method [18]. We augmented the least-squares algorithms with a regularization term that penalizes non-smoothness[26]. This form of regularization is provided by the graph Laplacian, which is also used in the construction of PVFs and diffusion wavelets. For the basis selection algorithm, we implemented orthogonal matching pursuit (OMP), order recursive matching pursuit (ORMP), and LASSO and LARS. A systematic study was conducted on a simple chain MDP to determine the most promising way(s) of combining these various components. From these experiments, we summarize with the following six findings.

1. We showed that the direct scheme for sparse approximate policy evaluation, when combined with the fixed-point least-squares method, constrains the order in which a basis selection

algorithm selects elements from a dictionary. The order is dictated by the elements in the Neumann series, $\sum_{i=0}^{\infty}(\gamma P^{\pi})^{i}R^{\pi}$. This can lead to the selection of basis functions that fit some of the early terms in the series, but are in fact not useful for representing the underlying value function. Of course, an algorithm like LASSO that can prune basis functions has the possibility of removing basis functions that become useless. The indirect scheme for sparse approximate policy evaluation sidesteps this issue by separating the Bellman equation from the basis selection algorithm. This adds computational complexity, but frees up the basis selection algorithm to represent the value function in the order it sees fit.

2. The graph Laplacian, which is used in constructing PVFs and diffusion wavelets, can also be used to provide regularization. This was accomplished by adding a term to the least-squares policy evaluation objective functions. Laplacian-based regularization can help smooth out the approximate value function. It also provides a bias toward smoother basis functions in the dictionary. This bias can be helpful when using the direct scheme for sparse approximate policy evaluation. We speculate that in an online setting, it may be beneficial to adjust the amount of regularization over time as more samples are seen.

3. *For direct sparse approximate policy evaluation with an orthonormal dictionary:*
OMP-FP produce accurate approximations and outperformed LASSO-FP. The approximate value functions had small magnitude for LASSO-FP without the orthogonalization step at the end of Algorithm 5. The only algorithm that worked using the Bellman residual least-squares method was ORMP-BR. This was an interesting result that shows one must be careful when combining basis selection and approximate policy evaluation algorithms.

4. *For direct sparse approximate policy evaluation with an overcomplete dictionary:*
OMP-FP became unstable in some of the experiments. This was because the least-squares matrix $\hat{A}_{\mathcal{I},\mathcal{I}}^{-1}$ became nearly singular. The algorithm could be made more robust by checking the condition number of the matrix before including a new basis function. The more conservative nature of LASSO-FP and LARS-FP produced accurate approximate value functions (without the orthogonalization step at the end of Algorithm 5). The magnitude of the value functions was much greater compared to those using an orthonormal dictionary. ORMP-BR remained the only algorithm that worked when using the Bellman residual least-squares method.

5. *For indirect sparse approximate policy evaluation with an orthonormal dictionary:*
OMP, ORMP, and LASSO all produced accurate approximate value functions while using both the fixed-point and Bellman residual least-squares methods. The results were particularly impressive when using a very small number of basis functions. Overall, the results were noticeably better than using an orthonormal dictionary with the direct scheme for sparse approximate policy evaluation. This confirms the hypothesis that the indirect scheme can select a more efficient set of basis functions than the direct scheme.

6. *For indirect sparse approximate policy evaluation with an overcomplete dictionary:*
OMP, ORMP, and LASSO all produced accurate approximate value functions while using both the FP and BR least-squares methods when allowed enough basis functions. The result-

ing value functions were much worse than those produced using an orthonormal dictionary when using a small number of basis functions.

The experiments in this paper partially demonstrate the expressiveness and flexibility of the diffusion wavelet dictionary. However, we believe the true value of diffusion wavelets will be evident on more challenging value functions with discontinuities and different degrees of smoothness. For future work, it would be worthwhile further decomposing the diffusion wavelet tree using diffusion wavelet packets [28]. This increases the size of the dictionary and provides even more flexibility for function approximation. Another area for future work is to design a faster sparse QR algorithm that would help scale diffusion wavelets to larger problems.

References

[1] P. Keller, S. Mannor, and D. Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 449–456, 2006.

[2] 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.

[3] M. Maggioni and S. Mahadevan. A multiscale framework for Markov decision processes using diffusion wavelets. Technical Report UM-CS-2006-036, University of Massachusetts Amherst, 2006.

[4] M. Petrik. An analysis of Laplacian methods for value function approximation in MDPs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2574–2579, 2007.

[5] R. Parr, C. Painter-Wakefield, L. Li, and M. Littman. Analying feature generation for value function approximation. In *Proceedings of the 24th International Conference on Machine Learning*, pages 737–744, 2007.

[6] R. Coifman and M. Maggioni. Diffusion wavelets. *Applied and Computational Harmonic Analysis*, 21(1):53–94, 2006.

[7] B. Olshausen and D. Field. Emergence of simple-cell receptive fields by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[8] M. Lewicki and T. Sejnowski. Learning overcomplete representations. *Neural Computation*, 12(2):337–365, 2000.

[9] H. Lee, A. Battle, R. Raina, and A. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems*, volume 19, pages 801–808, 2007.

[10] G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[11] F. Chung. *Spectral Graph Theory*. Number 92 in CBMS Regional Conference Series in Mathematics. American Mathematical Society, Providence, RI, 1997.

[12] Y. Pati, R. Rezaiifar, and P. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of the 27th Annual Asilomar Conference on Signals, Systems, and Computers*, pages 40–44, 1993.

[13] B. Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24(2):227–234, 1995.

[14] R. Tibshirani. Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1996.

[15] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of Statistics*, 32:407–451, 2004.

[16] R. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[17] S. Bradtke and A. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1-3):33–57, 1996.

[18] J. Johns, M. Petrik, and S. Mahadevan. Hybrid least-squares algorithms for approximate policy evaluation. *Machine Learning*, 76(2):243–256, 2009.

[19] S. Osentoski and S. Mahadevan. Learning state-action basis functions for hierarchical MDPs. In *Proceedings of the 24th International Conference on Machine Learning*, pages 705–712, 2007.

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

[21] A. Farahmand, M. Ghavamzadeh, C. Szepesvari, and S. Mannor. Regularized policy iteration. In *Advances in Neural Information Processing Systems*, volume 21, 2009.

[22] S. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993.

[23] B. Moghaddam, A. Gruber, Y. Weiss, and S. Avidan. Sparse regression as a sparse eigenvalue problem. In *Information Theory and Applications Workshop*, pages 121–127, 2008.

[24] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.

[25] S. Chen, D. Donoho, and M. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal of Scientific Computing*, 20(1):33–61, 1998.

[26] M. Belkin, P. Niyogi, V. Sindhwani, and P. Bartlett. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006.

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

[28] J. Bremer, R. Coifman, M. Maggioni, and A. Szlam. Diffusion wavelet packets. *Applied and Computational Harmonic Analysis*, 21(1):95–112, 2006.

APPENDIX

We described the OMP-FP, ORMP-FP, LASSO-FP, and LARS-FP algorithms as forming the matrix $\hat{A}_{\mathcal{I},\mathcal{I}}$ and the vector $\hat{b}_{\mathcal{I}}$. Then each of the algorithms invert the matrix $\hat{A}_{\mathcal{I},\mathcal{I}}$. This is very wasteful when the active set $\mathcal{I}$ only changes by one element at a time. To take advantage of the single element insertion and removal, $\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}$ can be incrementally formed using the following partitioned matrix inverse property. Consider a square matrix A' partitioned as follows:

$$A' = \begin{bmatrix} A & b \\ c^T & d \end{bmatrix}$$

where matrix A is square, b and c are vectors, and d is a scalar. Then the inverse of A' can be computed from the inverse of A as:

$$A'^{-1} = e \begin{bmatrix} (e^{-1}A^{-1} + A^{-1}bc^TA^{-1}) & -A^{-1}b \\ -c^TA^{-1} & 1 \end{bmatrix}$$

where $e = (d - c^TA^{-1}b)^{-1}$. Computing A'^{-1} in this manner has quadratic complexity instead of cubic. OMP-FP, ORMP-FP, LASSO-FP, and LARS-FP can exploit this property by maintaining the matrix $\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}$. When inserting a new element j^* into $\mathcal{I}$, the update is as follows:

$$\begin{aligned}
\mathcal{I} &\leftarrow \mathcal{I} \cup \{j^*\} \\
\hat{A}^{-1}_{\mathcal{I},\mathcal{I}} &\leftarrow \begin{bmatrix} (\hat{A}^{-1}_{\mathcal{I},\mathcal{I}} + u_{j^*}\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}\hat{A}_{\mathcal{I},j^*}\hat{A}_{j^*,\mathcal{I}}\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}) & -u_{j^*}\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}\hat{A}_{\mathcal{I},j^*} \\ -u_{j^*}\hat{A}_{j^*,\mathcal{I}}\hat{A}^{-1}_{\mathcal{I},\mathcal{I}} & u_{j^*} \end{bmatrix} \\
\hat{b}_{\mathcal{I}} &\leftarrow \begin{bmatrix} \hat{b}_{\mathcal{I}} \\ \hat{b}_{j^*} \end{bmatrix}
\end{aligned}$$

where

$$\begin{aligned}
u_{j^*} &\leftarrow (\hat{A}_{j^*,j^*} - \hat{A}_{j^*,\mathcal{I}}\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}\hat{A}_{\mathcal{I},j^*})^{-1} \\
\hat{A}_{j^*,j^*} &\leftarrow \sum_{i=1}^{n} \rho(s_i) \left[\phi_{j^*}(s_i)(\phi_{j^*}(s_i) - \gamma\phi_{j^*}(s'_i)) + \beta_m g_{j^*}(s_i)g_{j^*}(s_i)\right] \\
\hat{A}_{\mathcal{I},j^*} &\leftarrow \sum_{i=1}^{n} \rho(s_i) \left[\phi_{\mathcal{I}}(s_i)(\phi_{j^*}(s_i) - \gamma\phi_{j^*}(s'_i)) + \beta_m g_{\mathcal{I}}(s_i)g_{j^*}(s_i)\right] \\
\hat{A}_{j^*,\mathcal{I}} &\leftarrow \sum_{i=1}^{n} \rho(s_i) \left[\phi_{j^*}(s_i)(\phi_{\mathcal{I}}(s_i) - \gamma\phi_{\mathcal{I}}(s'_i))^T + \beta_m g_{j^*}(s_i)g_{\mathcal{I}}(s_i)^T\right] \\
\hat{b}_{j^*} &\leftarrow \sum_{i=1}^{n} \rho(s_i)\phi_{j^*}(s_i)r_i.
\end{aligned}$$

Similarly, when LASSO-FP removes an element $j^\#$ from $\mathcal{I}$, the matrix $\hat{A}^{-1}_{\mathcal{I},\mathcal{I}}$ can be shrunk with the following update:

$$\begin{gathered}
\mathcal{I} \leftarrow \mathcal{I} - \{j^\#\} \\
\text{Partition the current } \hat{A}^{-1}_{\mathcal{I},\mathcal{I}} \leftarrow \begin{bmatrix} U & x_{j^\#} \\ y^T_{j^\#} & z_{j^\#} \end{bmatrix} \text{ to isolate the influence of } j^\# \\
\hat{A}^{-1}_{\mathcal{I},\mathcal{I}} \leftarrow U - x_{j^\#}y^T_{j^\#}/z_{j^\#}.
\end{gathered}$$

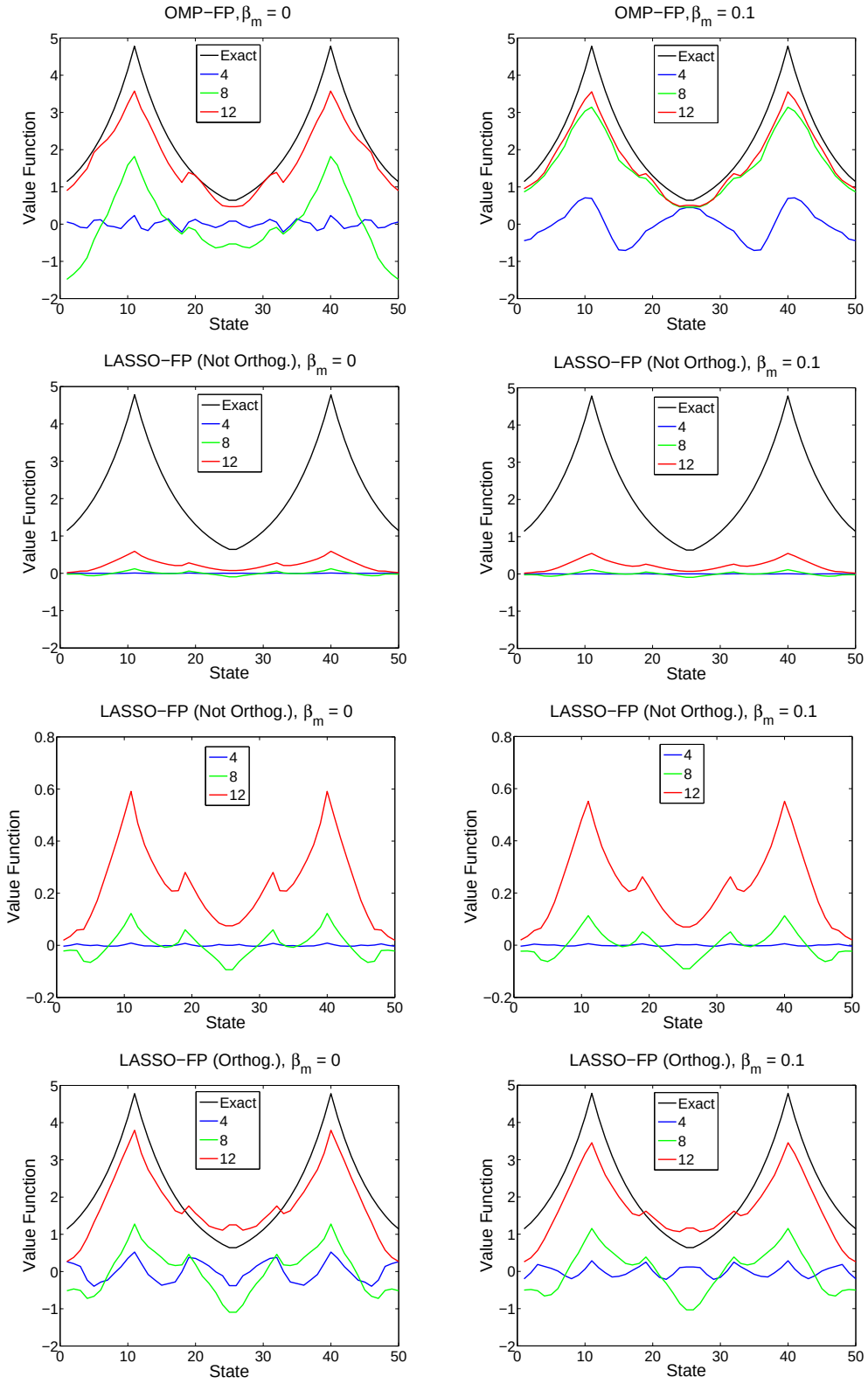

Figure 1: Results of OMP-FP and LASSO-FP with the PVF dictionary.

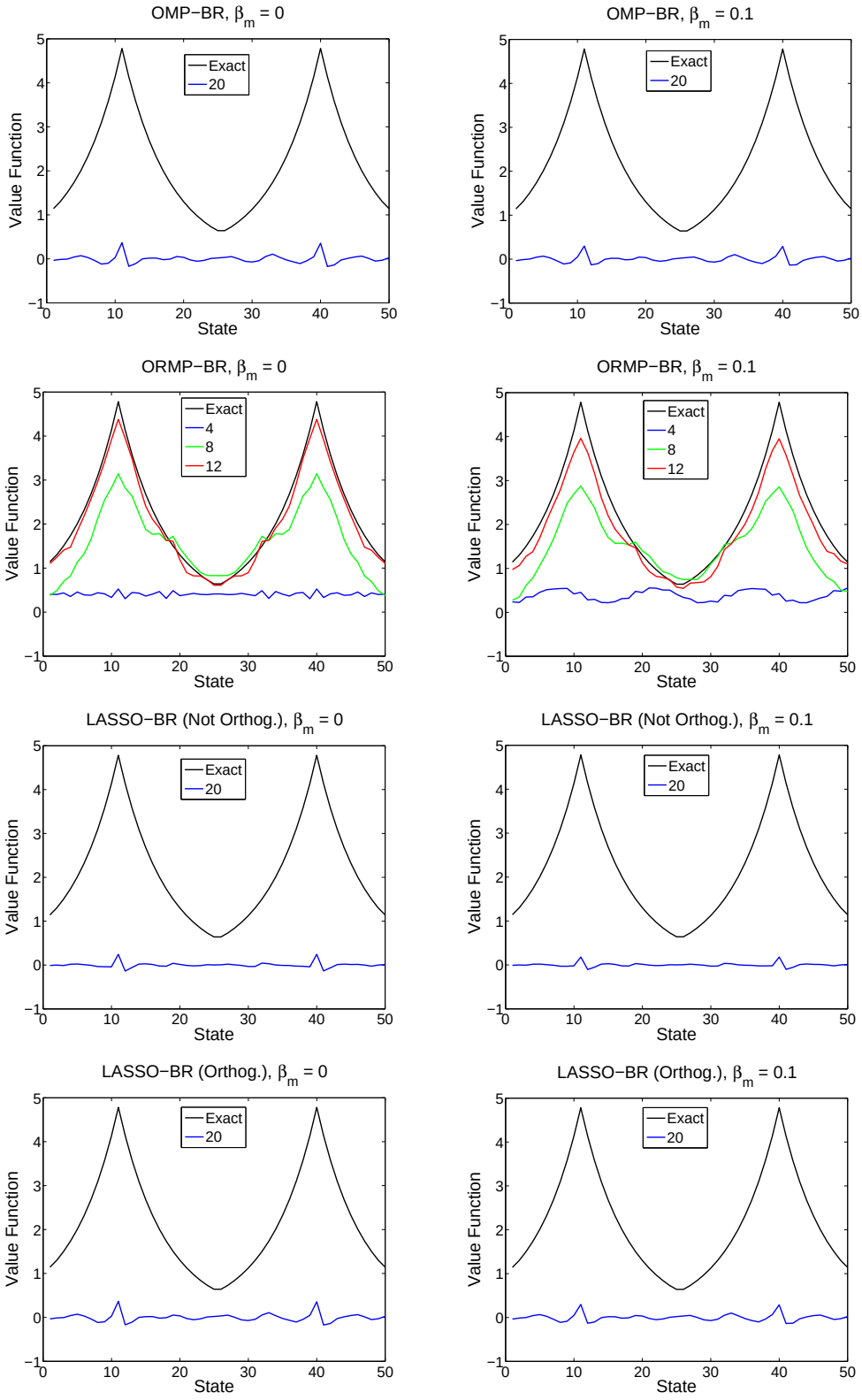

Figure 2: Results of OMP-BR, ORMP-BR, and LASSO-BR with the PVF dictionary.

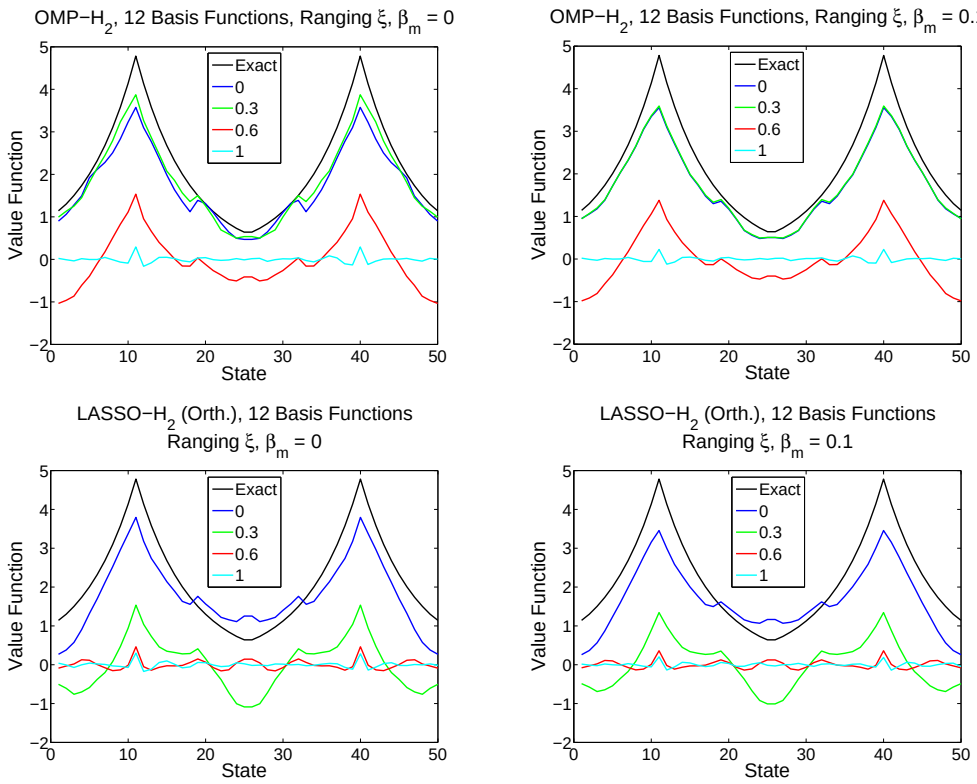

Figure 3: Results of OMP-H_2 and LASSO-H_2 with the PVF dictionary using 12 basis functions while varying ξ ($\xi = 0$ is equivalent to FP and $\xi = 1$ is equivalent to BR).

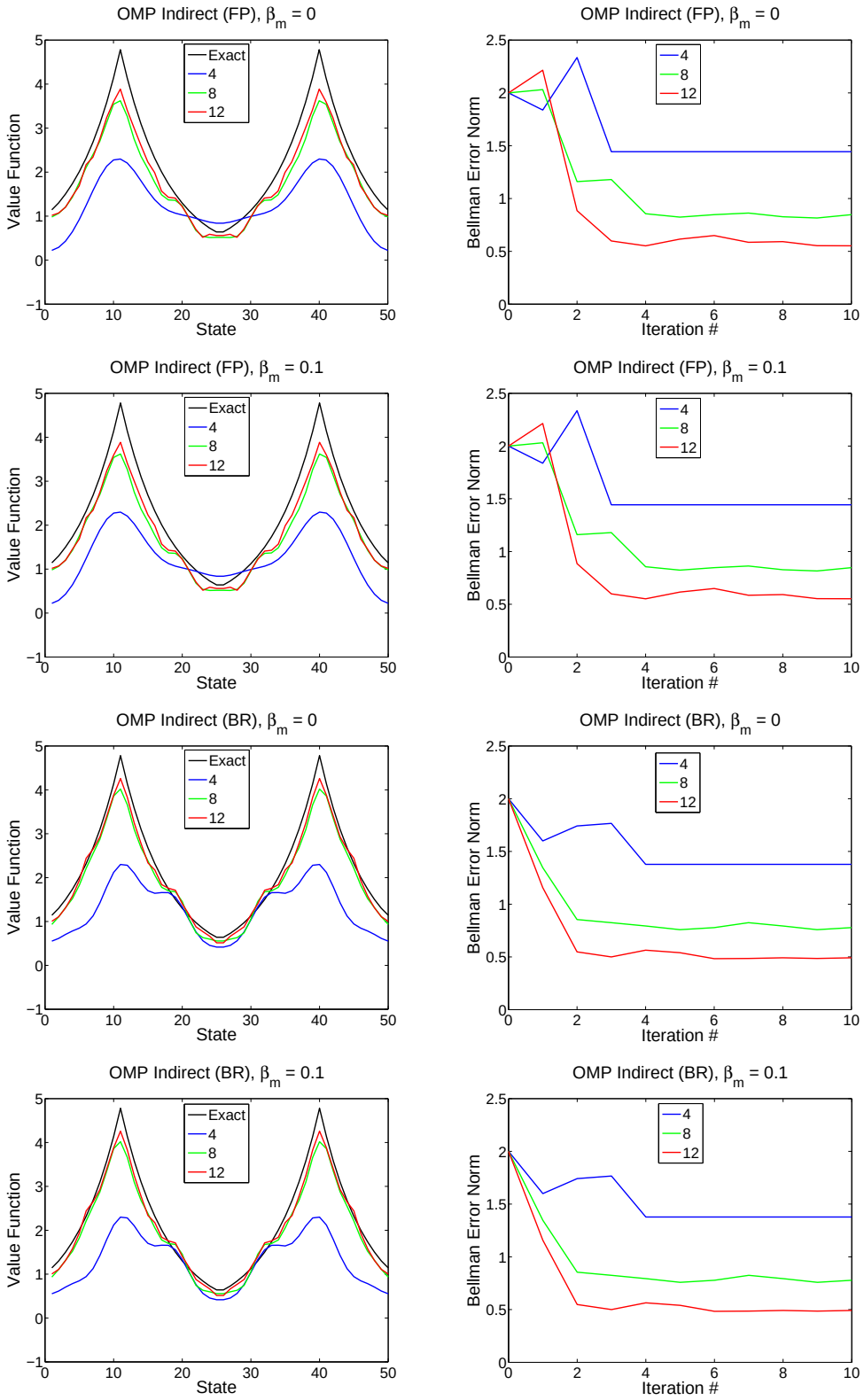

Figure 4: Results using the indirect scheme with OMP for basis selection and FP and BR for setting the coefficients. The PVF dictionary was used.

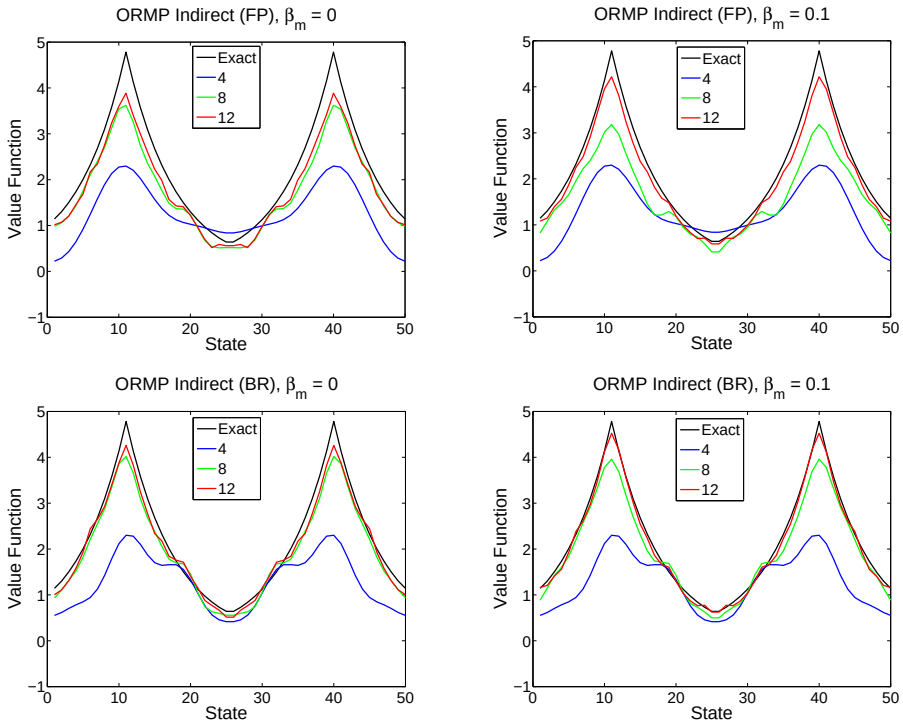

Figure 5: Results using the indirect scheme with ORMP for basis selection and FP and BR for setting the coefficients. The PVF dictionary was used.

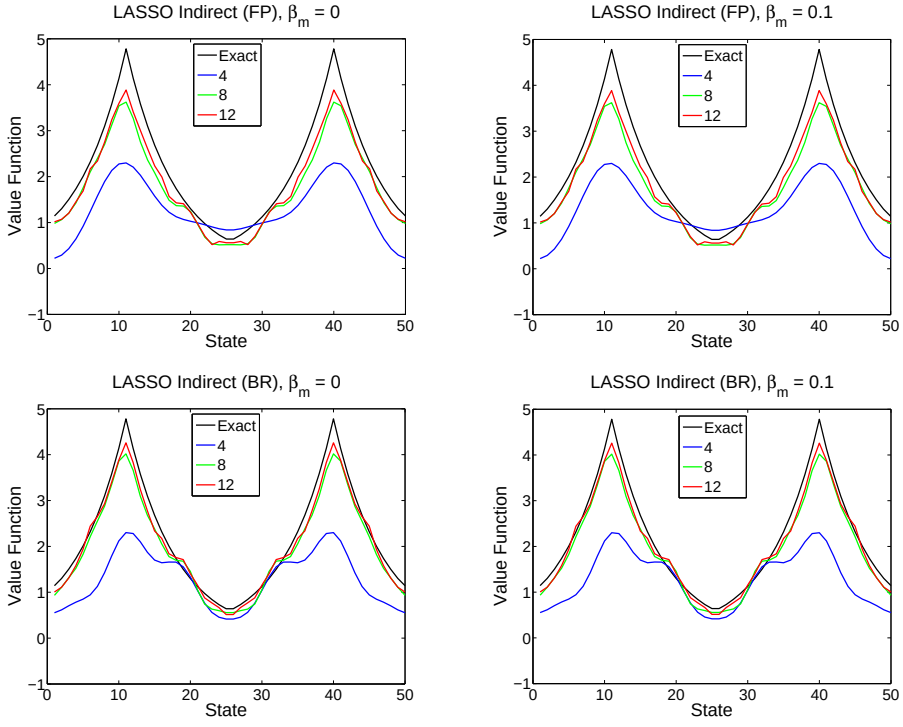

Figure 6: Results using the indirect scheme with LASSO for basis selection and FP and BR for setting the coefficients. The PVF dictionary was used.

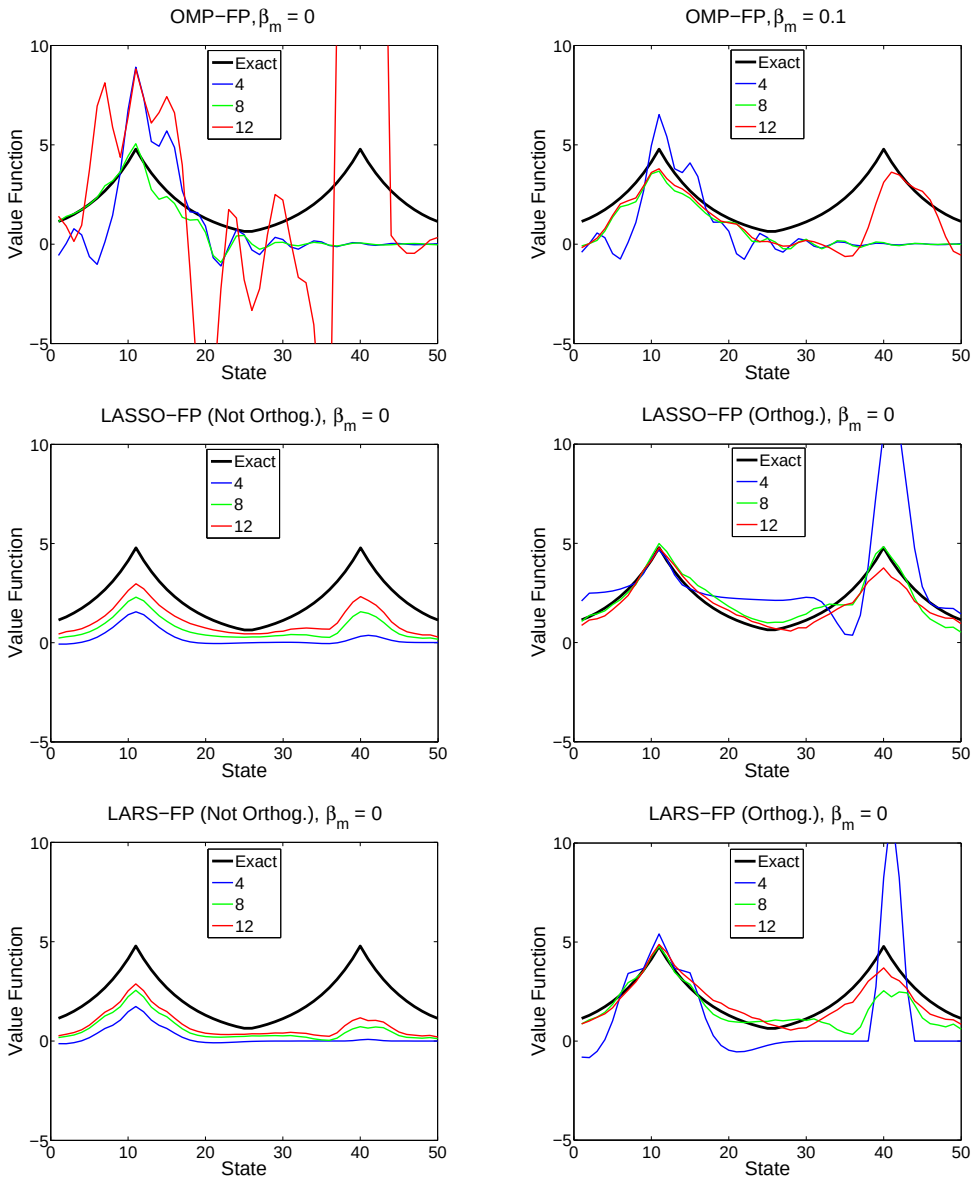

Figure 7: Results of OMP-FP, LASSO-FP, and LARS-FP using the diffusion wavelet dictionary with all 235 scaling and wavelet functions.

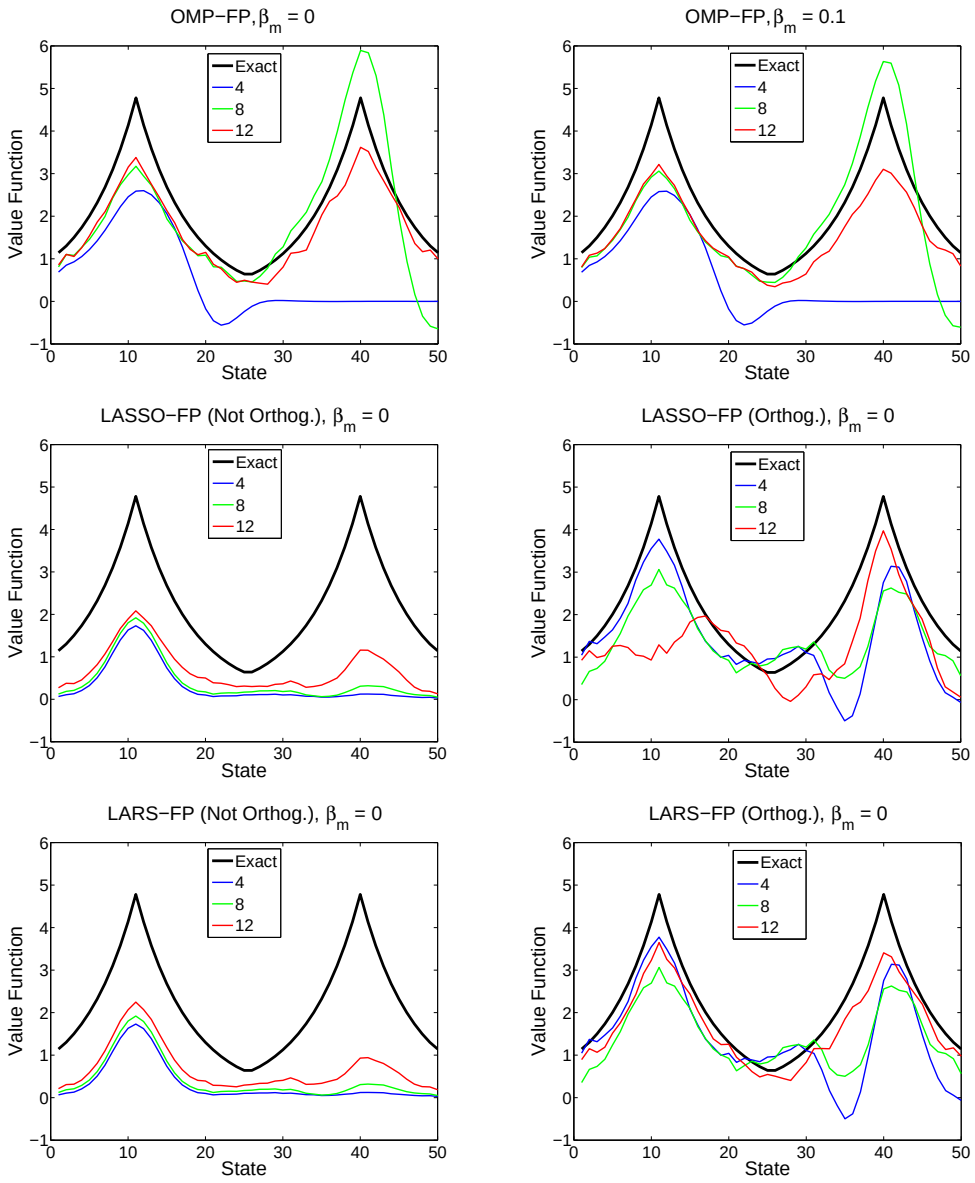

Figure 8: Results of OMP-FP, LASSO-FP, and LARS-FP using the diffusion wavelet dictionary with the 135 scaling and wavelet functions beyond tree level 3.

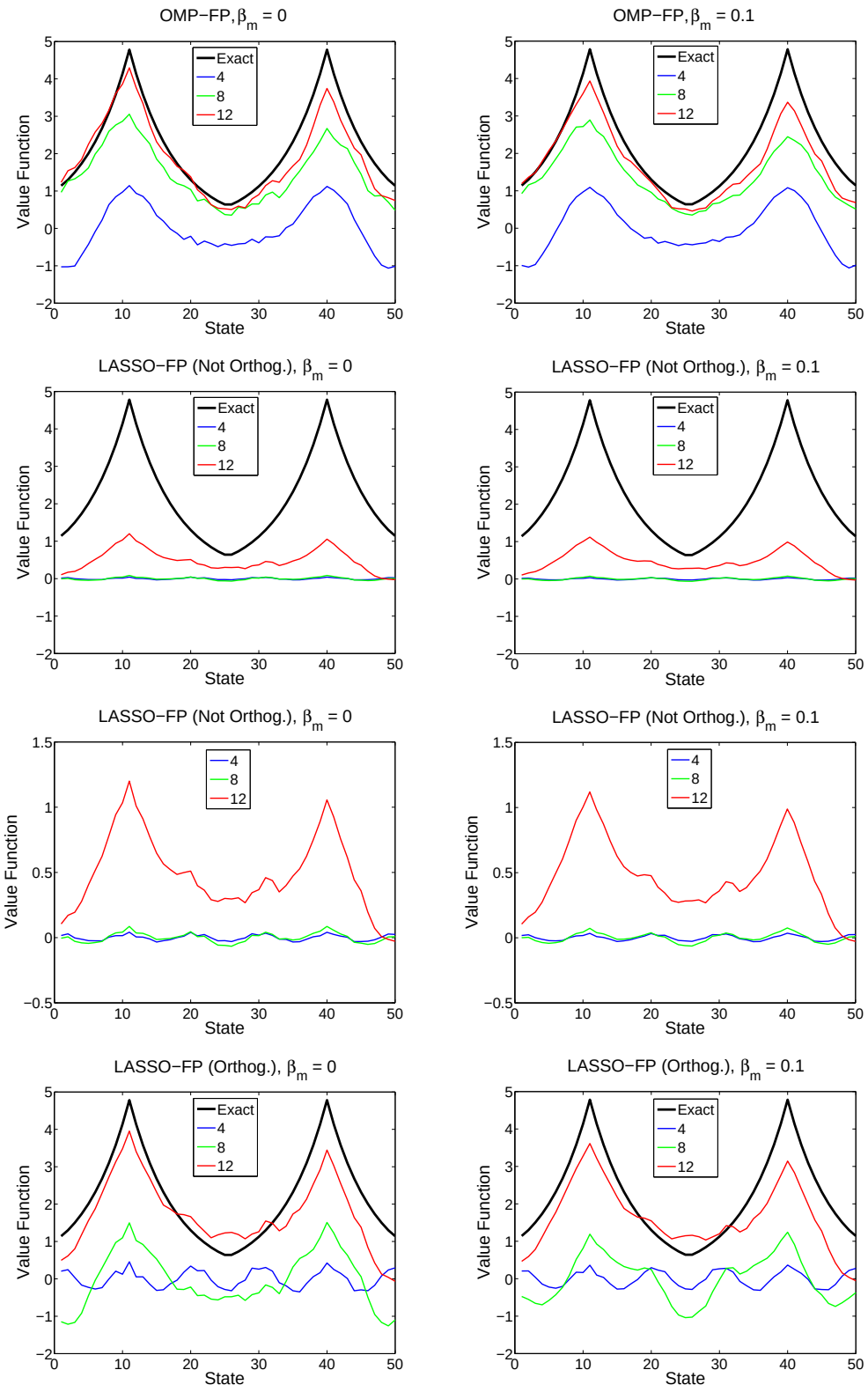

Figure 9: Results of OMP-FP and LASSO-FP using the diffusion wavelet dictionary with all the wavelet functions and just the scaling functions at tree level 10.

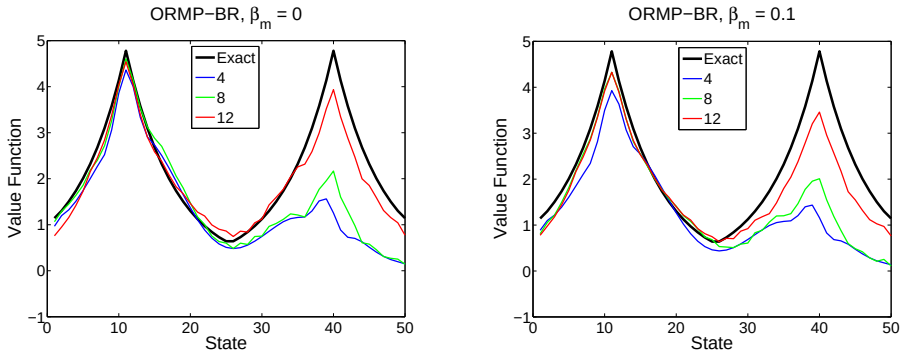

Figure 10: Results of ORMP-BR using the diffusion wavelet dictionary with all 235 scaling and wavelet functions. Results of OMP-BR and LASSO-BR were omitted because they produced approximations similar to those in Fig. 11.

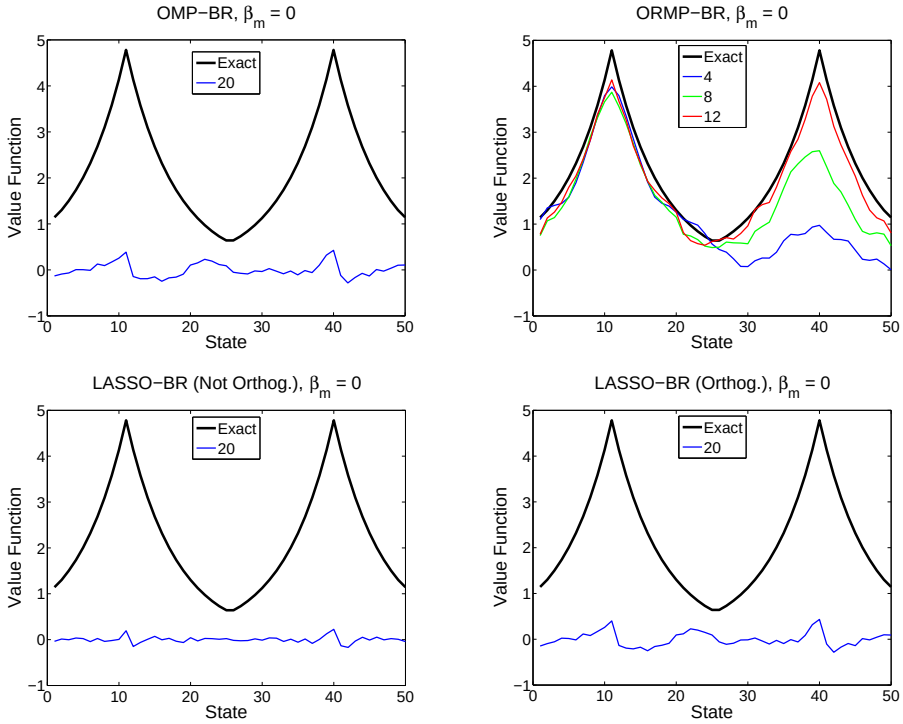

Figure 11: Results of OMP-BR, ORMP-BR, and LASSO-BR using the diffusion wavelet dictionary with the 135 scaling and wavelet functions beyond tree level 3.

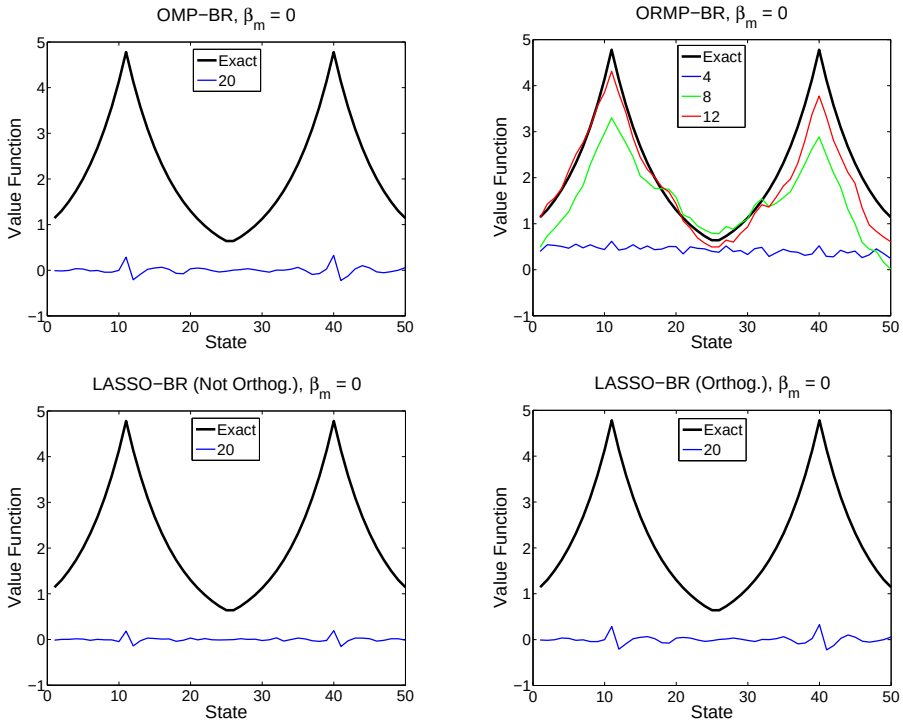

Figure 12: Results of OMP-BR, ORMP-BR, and LASSO-BR using the diffusion wavelet dictionary with all wavelet functions and just the scaling functions at tree level 10.

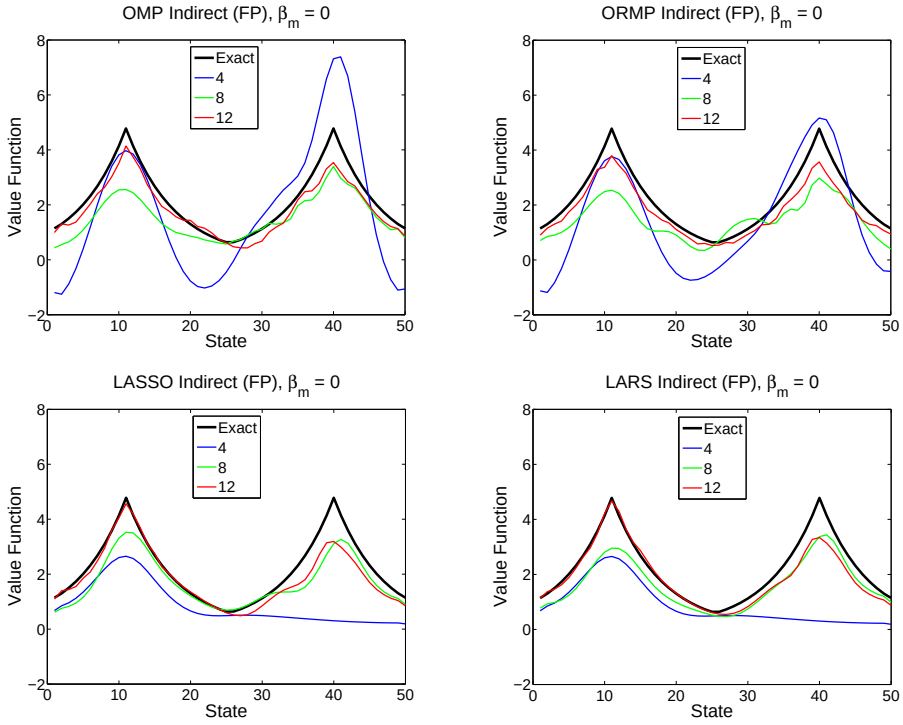

Figure 13: Results using the indirect scheme with FP. The diffusion wavelet dictionary included all 235 scaling and wavelet functions.

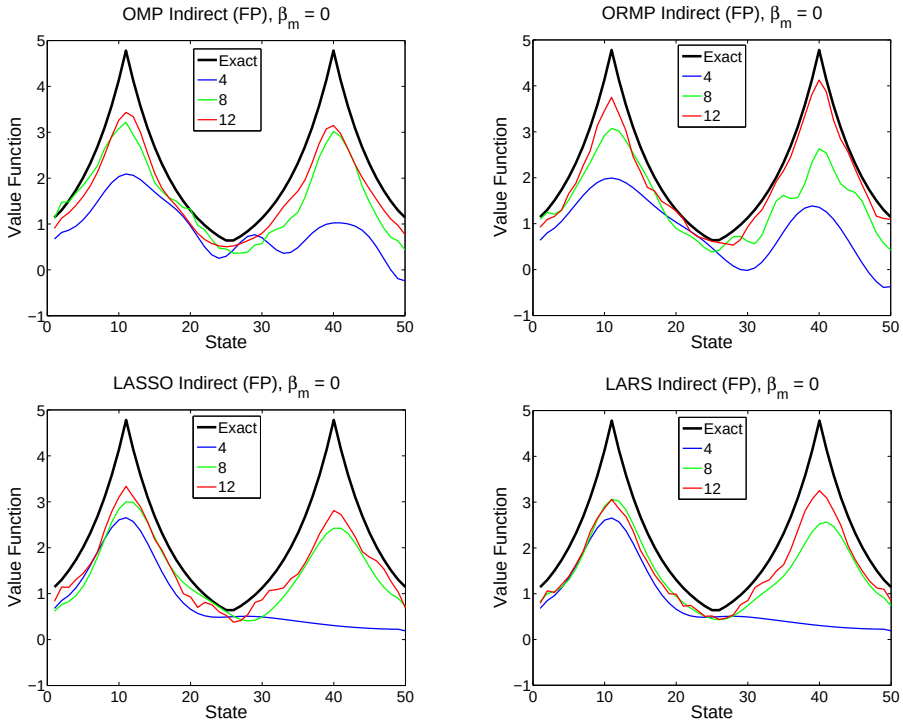

Figure 14: Results using the indirect scheme with FP. The diffusion wavelet dictionary included 135 scaling and wavelet functions beyond tree level 3.

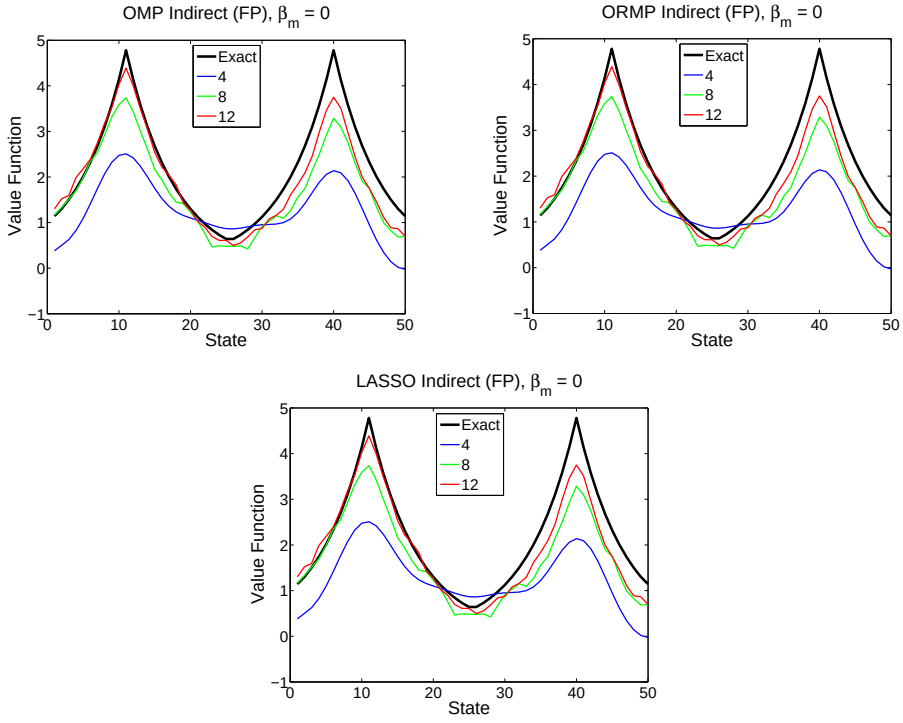

Figure 15: Results using the indirect scheme with FP. The diffusion wavelet dictionary included all wavelet functions and just the scaling functions at tree level 10.

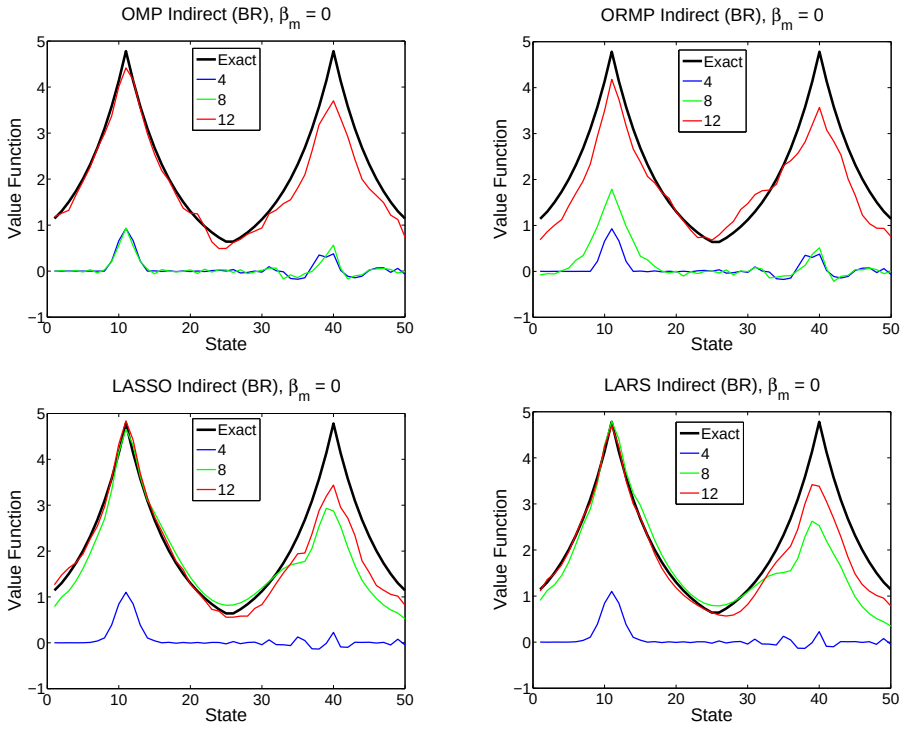

Figure 16: Results using the indirect scheme with BR. The diffusion wavelet dictionary included all 235 scaling and wavelet functions.

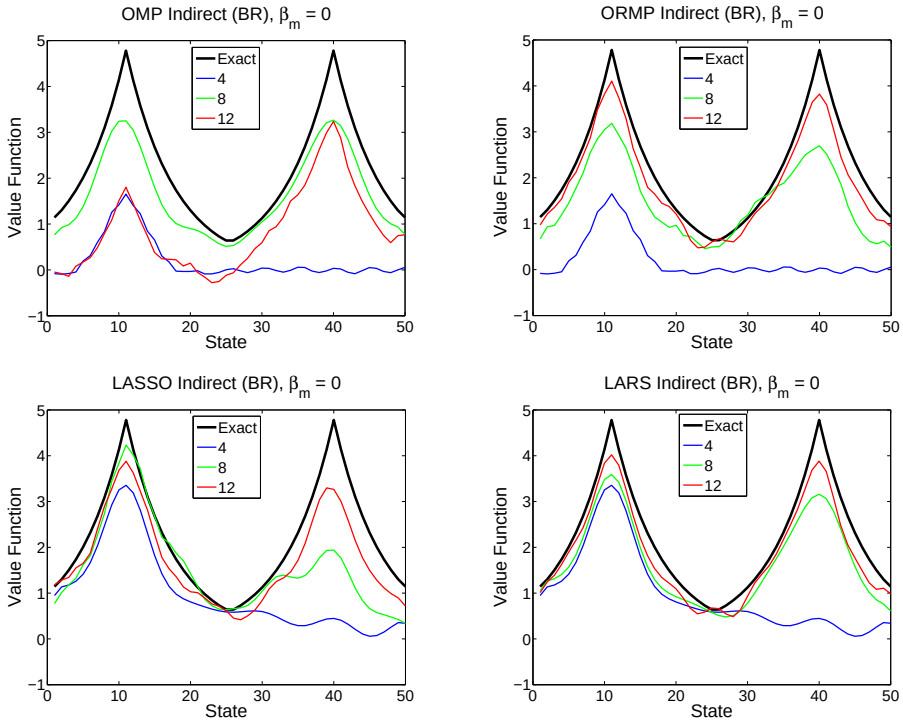

Figure 17: Results using the indirect scheme with BR. The diffusion wavelet dictionary included 135 scaling and wavelet functions beyond tree level 3.

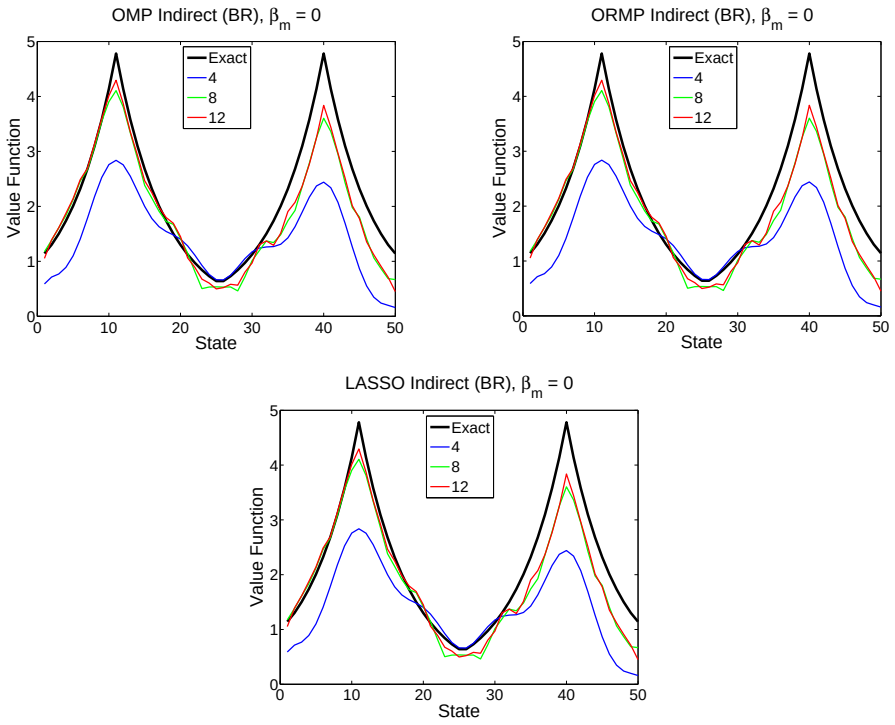

Figure 18: Results using the indirect scheme with BR. The diffusion wavelet dictionary included all wavelet functions and just the scaling functions at tree level 10.