Introduction to Markov Decision Processes

Martin L. Puterman and Timothy C. Y. Chan

April 14, 2023

Chapter 1

Approximate Dynamic Programming

> *An approximate answer to the right problem is worth a good deal more than an exact answer to an approximate problem.*
>
> John Tukey, American statistician (1915-2000)

and

> *With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.*
>
> John van Neumann, Hungarian-American mathematician, physicist, computer scientist, game theorist and more (1903-1957)

We welcome all feedback and suggestions at:
martin.puterman@sauder.ubc.ca and tcychan@mie.utoronto.ca

1.1 Introduction

In models with large state spaces or action sets or both, direct evaluation and storage of value functions or state-action value functions may be prohibitive. To overcome this

challenge we may approximate these quantities by lower dimensional functions such as polynomials, splines or neural networks. Doing so presents several challenges which we discuss in this chapter.

As an example, consider the advanced appointment scheduling model of Section **??** which we analyze in Section 1.6 below. In it, the vector-valued state represents the number of booked appointments each day over a multi-day booking horizon and the number of requests for appointments of each type[1] arriving on particular day. Actions represent the number of incoming appointment requests of each type to book in an available appointment slot on a given day. The objective in this application is to find a good policy for assigning appointment slots to incoming appointment requests.

In a realistic application, the booking horizon N may be 30, the number of appointment types K may be 3, the daily capacity C may equal 20 and the maximum number of arrivals of request of each type M may be 10. In such a case the model will have $20^{30} \times 10^3$ states and as well, a very large number of actions. Clearly in such a model, calculation and storage of exact value functions or state-action value functions in tabular form would be impossible. This chapter offers an alternative approach.

To motivate this chapter, suppose in a discounted infinite horizon model we have an approximation $\tilde{v}^*_\lambda(s)$ to the optimal value function $v^*_\lambda(s)$ expressed as a low-dimensional *parametric* function represented in terms of model characteristics referred to as *basis functions*. Instead of storing $v^*_\lambda(s)$ in tabular form, we store only the parameter estimates. To obtain a "good" action in state s, we need only evaluate $v^*_\lambda(j)$ in states j that can be reached from s' in one transition and then choose the "good" action $a^*_{s'}$ greedily based on the one-step Bellman update

$$a^*_{s'} \in \arg\max_{a \in A_{s'}} \left\{ r(s', a) + \lambda \sum_{j \in S} p(j|s', a) \hat{v}^*_\lambda(j) \right\}. \tag{1.1}$$

Since in most applications, there are few successor states to s', this calculation is very efficient. Recall that this is equivalent to solving a one-period problem with terminal reward $\hat{v}^*_\lambda(s)$ [2].

Instead of approximating value functions the approximations can be applied to state-action value functions to obtain $\hat{q}^*(s, a)$. Then $a^*_{s'}$ can be determined from

$$a^*_{s'} \in \arg\max_{a \in A_{s'}} \{\hat{q}^*(s', a)\}.$$

> We emphasize that a policy based on greedy action choice using an approximation to the value function need not be optimal.

To implement this approach in the appointment scheduling problem described above, each day:

[1] In the motivating application, type corresponds to one of three urgency levels. Urgency is determined by a medical professional. Each urgency type has a different target wait time.

[2] Some authors refer to this procedure as *rollout*.

1. The scheduler observes the system state (s'),
2. Evaluates the expression in brackets in (1.1) for each $a \in A_{s'}$, and
3. Assigns patients to future appointment dates according to $a^*_{s'}$.

In this case as in most implementations of Markov decision process, the optimal action in all states is not necessary, it can be determined on an as needed basis from either $v^*_\lambda(\cdot)$, $q^*(s,a)$ or an approximation.

Questions arising are:

1. How do we find good value function approximations?
2. How close to optimal is the stationary policy based on decision rule $\tilde{d}^*(s) = a^*_s$?

In most of the literature, approximation and simulation are discussed together. To identify issues particular to each, we separate these discussions. This chapter focuses on approximation. Chapter **??** discusses simulation methods. Then, Chapter **??** combines simulation and approximation, which is the cornerstone of *reinforcement learning*. Moreover, our focus in this chapter will be models in which rewards are discounted. Extensions to average and total reward models is left to the reader.

1.2 Value Function Approximation

We start with approximating value functions and subsequently generalize to state-action value functions. Moreover, we will initially concern ourselves with approximating the reward of a fixed stationary deterministic policy. We motivate this discussion by considering the discounted discrete time queuing service rate control model formulated in Section **??**.

Example 1.1. Recall that in the queuing service rate control model (Section **??**) the state represents the number of jobs in the system and the action represents the chosen service probability. In each period a job arrives with probability $b = 0.2$ and the controller can choose among three service probabilities $a_1 = 0.2$, $a_2 = 0.4$ and $a_3 = 0.6$. In any period, there is either an arrival, a service or neither. We assume a delay cost of $f(s) = s^2$ and the cost per period of serving at rate a_k is $m(a_k) = 5k^3$.

We truncate the unbounded state space to $S = \{0, 1, \ldots, 50\}$ and set the discount rate $\lambda = 0.98$. We show approximations to the value function of the stationary policy $\pi = (d, d, \ldots)$ that uses the deterministic decision rule

$$d(s) = \begin{cases} a_1 & \text{for } s < 20 \\ a_3 & \text{for } s \geq 20. \end{cases}$$

Since the state space is ordered and interpretable, we can use functions of the state as value function approximators. To illustrate this approach we evaluate the exact value function and then approximate it by linear and quadratic polynomials and an exponential function. We obtain parameter estimates for the polynomials using least squares linear regression and for the exponential function using non-linear least squares.

Figure 1.1a shows the true value function corresponding to the above policy and linear and quadratic approximations to it. Clearly the linear fit is poor and the quadratic fit is quite good. We leave it as an exercise to compare the fits of a cubic polynomial or a quadratic spline with a knot at 20.

The equations of the two fitted models are

$$\hat{v}_\lambda^\pi(s) = -7603.3 + 1320.9s$$

and

$$\hat{v}_\lambda^\pi(s) = 2096.3 + 133.2s + 23.78s^2.$$

In addition we consider approximating the value function by the nonlinear function $\beta_0 + \beta_1 e^{\beta_2 s}$. This requires using software that implements non-linear least squares. We use the package *nls* in R [2021], which implements Gauss-Newton iteration to obtain parameter estimates. Note that such methods are highly sensitive to starting values. After some experimentation with starting values we obtain the fitted equation:

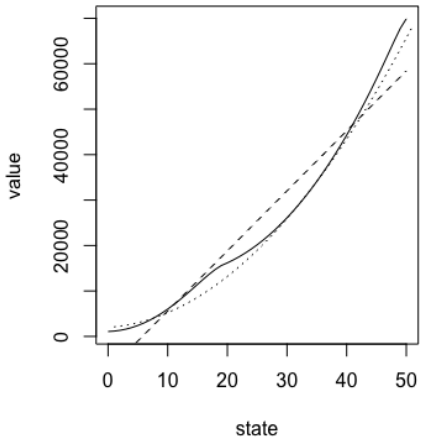
$$\hat{v}_\lambda^\pi(s) = -11403.2 + 12222.5e^{0.038s}$$

Figure 1.1b shows the actual value (solid line) and fitted value (dashed line). It appears that the exponential fit is superior to the quadratic fit.

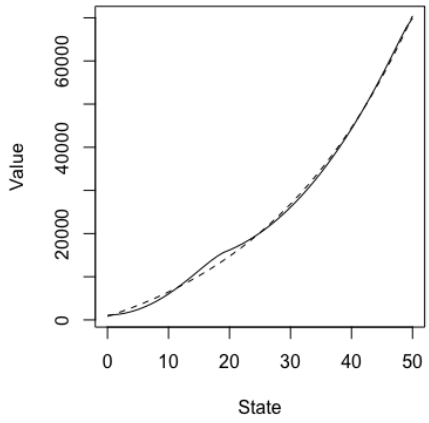
Note that value function approximation is not necessary in such a small model and we only do so as motivation for the material in this chapter. In practice we would not know (nor be able to compute) $v_\lambda^\pi(s)$ for all $s \in S$, so approximation would be necessary. As an aside, if we were studying the countable state version of this model, we could derive the approximation on $s \leq 50$ and approximate the value function by the fitted exponential value function for $s > 50$.

1.2.1 Features

It is customary to use features of the state space as independent variables when approximating value functions. Feature generation is similar to the choice of independent variables in regression models. They are usually low dimension summaries of key aspects of the state space that one believes explain the variability in the value function. Formally, a *feature* is a real valued function defined on S. Often, features are referred to as *basis functions*.



(a) The value function for the specified policy in the queuing service rate control model (solid line) with linear (dashed line) and quadratic (dotted line) approximations.



(b) The value function for the specified policy the queuing service rate control model (solid line) with the exponential fitted model (dashed line).

We denote the set of features evaluated at s by $b_0(s), b_1(s), \ldots, b_K(s)$ or by a vector $\mathbf{b}(s)$[3] We will often $b_0(s) = 1$ to correspond to a constant in a regression model.

For $S = \{s_1, s_2, \ldots, s_N\}$ define the $N \times (K+1)$ matrix $\mathbf{B}$ of features by

$$\mathbf{B} := \begin{bmatrix} b_0(s_1) & \ldots & b_K(s_1) \\ \cdot & \ldots & \cdot \\ \cdot & \ldots & \cdot \\ \cdot & \ldots & \cdot \\ b_0(s_N) & \ldots & b_K(s_N) \end{bmatrix} \tag{1.2}$$

We will assume that the features are constructed so that the columns of $\mathbf{B}$ are linearly independent (in the linear algebra sense[4])so that the matrix $\mathbf{B}^\mathsf{T}\mathbf{B}$ is invertible.

Feature choice depends on the setting:

- If the state space is a set of **real numbers or ordered integers** possible feature choices include powers of s, b-splines, trigonometric functions or exponentials. In Example 1.1 when using a quadratic value function approximation, $K = 2$ and $b_0(s) = 1$, $b_1(s) = s$ and $b_2(s) = s^2$ for all $s \in S$.
- If the state space is **vector-valued**, possible feature choices include component values, powers of component values, interactions between components or

[3]In chapter **??** we use the quantity $\mathbf{b}(s)$ to denote the belief state. We trust that using it here to represent the vector of basis function evaluated at state s will not cause confusion.

[4]This means that the only solution of $\mathbf{Bx} = \mathbf{0}$ is $\mathbf{x} = \mathbf{0}$. Equivalently the rank of $\mathbf{B}$ is $K + 1$.

non-linear functions of the component values. For example in the appointment scheduling model described in the introduction to this chapter when there is a five-day booking window and two appointment types the state space is

$$S = \{(x_1, x_2, \ldots, x_5, y_1, y_2) \mid 0 \leq x_i \leq C, i = 1, \ldots, 5;\ 0 \leq y_k \leq D\,, k = 1, 2\},$$

where C denotes the maximum daily capacity and D denotes the maximum number of appointments that can arrive on a given day.

A possible set of features is:

$$b_0((x_1, x_2, \ldots, x_5, y_1, y_2) = 1; \quad b_i((x_1, x_2, \ldots, x_5, y_1, y_2)) = x_i \text{ for } i = 1, \ldots, 5;$$
$$b_{i,j}((x_1, x_2, \ldots, x_5, y_1, y_2)) = x_i x_j \text{ for } i = 1, \ldots, 5,\ j = 1, \ldots, 5 \quad \text{and}$$
$$b'_k((x_1, x_2, \ldots, x_5, y_1, y_2)) = y_k \text{ for } k = 1, 2.$$

These features correspond to a quadratic model in the number of booked appointments and a linear function of the number of appointment requests.

- In **more general settings**, the application and subject matter knowledge may dictate feature choice. For example, in the game of checkers[5] (Figure 1.2) there are 32 squares on the board that can be occupied (numbered in some way) and red and black each start with 12 pieces so that

$$S = \Bigg\{(x_1, \ldots, x_{32}) \mid x_i \in \{E, B_1, R_1, B_2, R_2\} \text{ for } i = 1, \ldots, 32 \ \text{ and } \sum_{i=1}^{32} 1_{\{x_i \in \{B_1, B_2\}\}} \leq 12,\ \sum_{i=1}^{32} 1_{\{x_i \in \{R_1, R_2\}\}} \leq 12 \Bigg\}$$

where E indicates that the square is empty, B_1 denotes its is occupied by a black checker, R_1 denotes it is occupied by a red checker, B_2 denotes it is occupied by a black king and R_2 denotes it occupied by a red king. The summations indicate that there are at most 12 pieces of each color on the board.

Possible features for analyzing this game include

$$b_0((x_1, \ldots, x_{32})) = 1;$$
$$b_1((x_1, \ldots, x_{32})) = \sum_{i=1}^{32} 1_{\{x_i = B_1\}}; \quad b_2((x_1, \ldots, x_{32})) = \sum_{i=1}^{32} 1_{\{x_i = B_2\}}$$
$$b_3((x_1, \ldots, x_{32}) = \sum_{i=1}^{32} 1_{\{x_i = R_1\}}; \quad b_4((x_1, \ldots, x_{32})) = \sum_{i=1}^{32} 1_{\{x_i = R_2\}}$$

as well as the number of pieces of each color that are within κ rows of becoming kings for specific choices of κ. We interpret the above features as follows: $b_1(\cdot)$ and $b_3(\cdot)$ denote the number of (single) checkers and $b_2(\cdot)$ and $b_4(\cdot)$ denote the number of kings each player has on the board.

Figure 1.2: Checkerboard showing positions of pieces at the start of a game.

In summary, feature types include univariate or multivariate polynomials, interpolation functions or indicator functions of subsets. Choosing the set of features equal to the indicator of each state, that is,

$$b_i(s) = \begin{cases} 1 & \text{if } s = s_i, \\ 0 & \text{otherwise} \end{cases}$$

for $i = 1, \ldots, N$, is equivalent to a *tabular representation* of the model.

1.2.2 Feature-based value function approximators

We consider linear and nonlinear approximations. Neural networks represent an important class of nonlinear approximations that we discuss separately. Some authors refer to the functional form used as an *architecture*.

Linear architectures

By a linear[6] approximation, we mean a function of the form

$$v(s) \approx \beta_0 b_0(s) + \beta_1 b_1(s) + \ldots + \beta_K b_K(s), \tag{1.3}$$

where as above we assume $b_0(s) = 1$ for all $s \in S$ so that β_0 denotes the model constant. With judicious choice of features, such models can represent a wide range of functional forms. Advantages of such an approximation include having parameters that can be estimated by least squares regression (i.e., where estimates are available in closed form;

[5]Checkers is a two-person game but it can be analyzed by *self play*, that is, playing against yourself.

[6]This may also be referred to as an *affine* approximation. Precisely speaking, affine is more accurate when the approximation includes a constant term.

see the chapter appendix) and having a function form that can be directly used in a linear programming model.

We can represent approximations of this form using matrix notation. We can write

$$\mathbf{v} \approx \mathbf{B}\boldsymbol{\beta} \tag{1.4}$$

where

$$\mathbf{B} = \begin{bmatrix} b_0(s_1) & \dots & b_K(s_1) \\ \cdot & \dots & \cdot \\ \cdot & \dots & \cdot \\ \cdot & \dots & \cdot \\ b_0(s_N) & \dots & b_K(s_N) \end{bmatrix} \quad \text{and} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \cdot \\ \cdot \\ \cdot \\ \beta_K \end{bmatrix} \tag{1.5}$$

where $S = \{s_1, \dots, s_N\}$. In statistical literature, the matrix $\mathbf{B}$ is often referred to as the *design matrix*[7].

Note that we do include an error term ϵ in our representation for an approximation because in this chapter we assume that there is no *sampling error* when determining the value function. The only source of imprecision is the inaccuracy of the approximation.

Nonlinear architectures

Although in many applications, a linear approximation will suffice, we might wish to use a nonlinear function f to represent value functions. In general, nonlinear architectures can be represented by

$$v(s) \approx f(b_0(s), b_1(s), \dots, b_K(s); \boldsymbol{\beta}) \tag{1.6}$$

where the unknown vector of parameters in state s are represented by $\boldsymbol{\beta}$. In nonlinear models, the number of parameters need not equal the number of features used. That is, the dimension of $\boldsymbol{\beta}$ need not be $K+1$. For example, in a representation with features $b_0(s) = 1$ and $b_1(s)$ we might use a function of the form

$$f(b_0(s), b_1(s); \boldsymbol{\beta}) = \beta_0 b_0(s) + \beta_1 e^{\beta_2 b_1(s)}$$

where $\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2)^T$. In this case there are two features and three parameters. In the special case where In the special case when $b_0(s) = 1$ and $b_1(s) = s$ for all $s \in S$, the above approximation becomes

$$f(s; \boldsymbol{\beta}) = \beta_0 + \beta_1 e^{\beta_2 s},$$

which is the same as the exponential function from Example 1.1.

[7]The expression *design matrix* originates in the experimental design literature where the analyst chooses the allocation of treatments to experimental units. This allocation can be summarized in terms of a matrix.

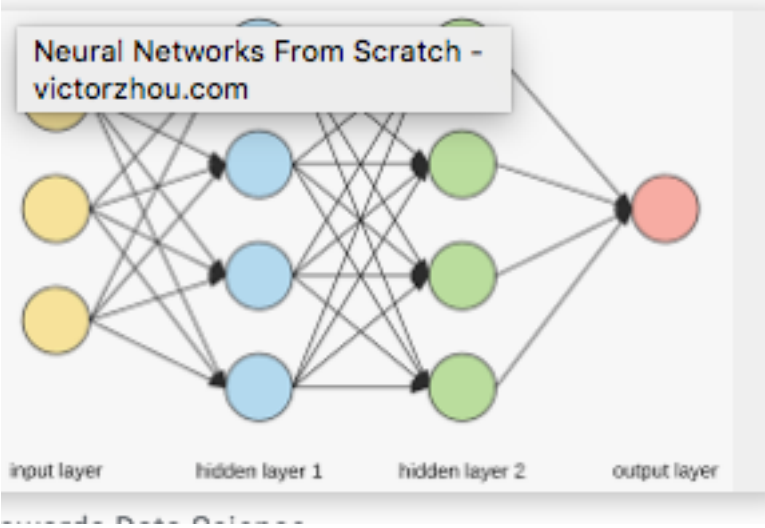


Figure 1.3: Generic Neural Network **(needs to be redrawn)**

Neural network approximations

Artificial neural networks or simply neural networks (NNs)[8] have been widely used in large modern applications of Markov decision processes. In the Markov decision process context they transform states or features into approximate value functions or state-action value functions. However they also have been used in numerous significant machine learning implementations including image recognition, recommendation systems, forecasting, and natural language processing.

A neural network consists of *nodes* organized in *layers*, uni-directional *arcs* connecting nodes from "left" to "right"[9] and *activation functions* corresponding to nodes in interior *hidden* layers. Values in the input layer are weighted by parameters corresponding to each arc and summed to generate inputs to the nodes at next layer where they are transformed non-linearly by the activation function.

Activation functions mimic the behavior of neurons that "fire" when the incoming signal reaches a threshold. They can be modelled by a step function that equals 0 for inputs less than a threshold and 1 for inputs greater than the threshold. Alternatively they can be represented by a non-decreasing continuous function $f : \Re \rightarrow [0, 1]$.

Figure 1.3 depicts a generic neural network with an input layer consisting of two nodes, two hidden layers containing 4 nodes each and an output layer consisting of a single node. Corresponding to each arc is a parameter to be estimated. Each node in a hidden layers corresponds to an activation function.

When using NNs, you must specify the number of layers, the number of nodes at each layer and the form of the activation function. From the perspective of this chapter,

[8]Some authors refer to NNs as *multi-layer perceptrons*.

[9]This is referred to as a *feed forward* neural network. Alternatively when there are loops in the network, it is referred to as a *recurrent* neural network.

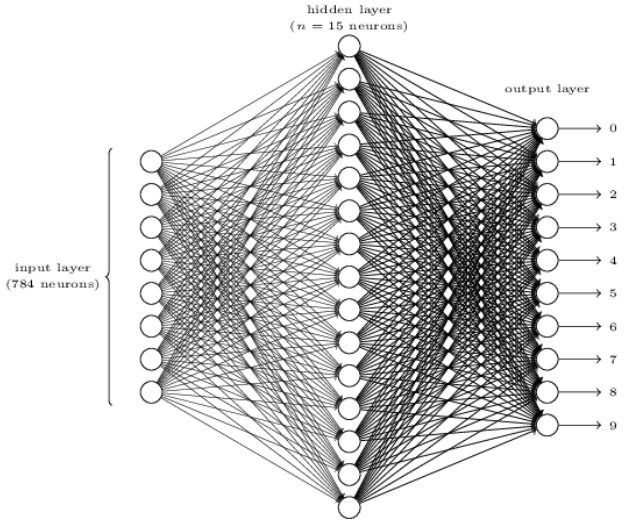


Figure 1.4: Symbolic representation of a neural network to interpret scanned handwritten digits. **(taken from Nielsen as placeholder -need to redraw)**

neural networks may be regraded as "black box" nonlinear function approximators.

To motivate our discussion and as well to describe a practical application that is not a Markov decision process, suppose we seek to develop a system to interpret a scanned handwritten digit from 0 to 9. The input to the NN (Figure 1.4) is an $N \times N$ array of numbers between 0 and 1[10] and the output is a probability distribution on the digits 0 to 9. The predicted digit would be that with maximum probability.

We illustrate the NN formulation in the following continuation of the above example.

Example 1.2. Suppose we want to approximate the value function for the queuing service rate control model in Example 1.1. We will use the neural network in Figure 1.5. This neural network has two nodes in the input layer, a hidden layer consisting of two nodes and a single node in the output layer. The objective is to use the neural network to approximate the value function.

In this example we choose the inputs to be a constant[a] equal to 1 and the state s (i.e., $b_0(s) = 1$ and $b_1(s) = s$) and choose the activation function

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{1.7}$$

[10]In a standard data set (MNIST), the images are represented by 28×28 grid of 764 pixels where each pixel assumes a grey scale value between 0 (white) and 1 (black).

An alternative choice for the activation function could be

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{1.8}$$

The output corresponds to the approximate value function $\tilde{v}_\lambda^{d^\infty}(s)$. The quantities $w_{i,j}^k$ represent parameters where i denote the origin of the arc, j denotes the destination of the arc and k denotes the layer of the destination.

We expand the neural network in functional form in three steps. The input to the node 1 in the hidden layer is the linear function of the inputs 1 and s

$$w_{11}^1 + w_{21}^1 s$$

and the input to node 2 in the hidden layer is

$$w_{12}^1 + w_{22}^1 s.$$

The activation function transform these inputs to

$$\frac{1}{1 + e^{-(w_{11}^1 + w_{21}^1 s)}} \quad \text{and} \quad \frac{1}{1 + e^{-(w_{12}^1 + w_{22}^1 s)}}.$$

Finally the input to the output layer is

$$w_{11}^2 \frac{1}{1 + e^{-(w_{11}^1 + w_{21}^1 s)}} + w_{21}^2 \frac{1}{1 + e^{-(w_{12}^1 + w_{22}^1 s)}}. \tag{1.9}$$

Assuming the activation function in the output layer is $f(x) = x$, the NN approximates the value function by the function of 6 parameters represented in (1.9). In light of the von Neumann quote at the beginning of this chapter, we would expect an accurate approximation of the value function using this architecture.

Parameters can be estimated by any a nonlinear regression package, but specialized gradient-based methods have been developed for parameter estimation in neural networks.

[a] Some descriptions represent the constant by a separate expression called the *bias*. The figure above only includes a bias at the input level, many implementations (including the one we use below), allow a bias to enter at each level. We use the input 1 to correspond to the approach we use when including a constant in a multiple regression model.

1.2.3 State-action value function approximation

The above sections describe how to approximate value functions using basis functions and linear or nonlinear architectures. Alternatively we could use these constructs to approximate state-action value functions $q(s, a)$. Approximations may apply direct to $q(s, a)$ or individually to the state and action components. In models with a few actions,

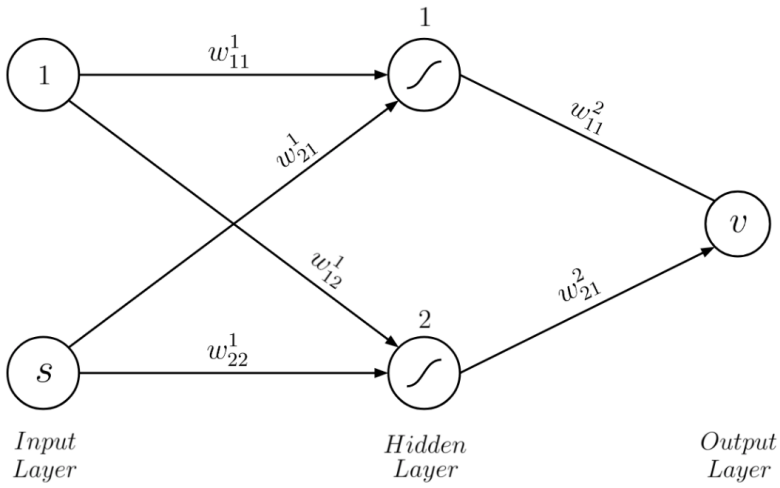


Figure 1.5: Neural network described in Example 1.2

such as the queuing service rate control model described above, we may develop an approximation for each action separately, that is, we regard $q(s, a_k)$ as a value function in which the decision rule $d(s) = a_k$ for all $s \in S$.

1.3 Parameter estimation for a fixed policy: linear architectures

This section focuses on approximating the value function of a *fixed* stationary policy, d^∞, based on linear combinations of basis variables. The next section considers nonlinear approximations.

We use approximations to reduce the problem of solving a system of equations with $N = |S|$ unknowns to one with $K+1$ unknowns where $K+1$ is the number of parameters used to estimate the value function. Since N is usually much greater than $K + 1$, the problem becomes that of finding a $\hat{\boldsymbol{\beta}}$ such that $\mathbf{B}\hat{\boldsymbol{\beta}}$ *best* approximates $\mathbf{v}_\lambda^{d^\infty}$ in some sense. We focus primarily on least squares and linear programming approximations. We strongly recommend reviewing the Appendix of this chapter, which reviews least squares regression emphasizing matrix methods.

Motivation

Let $d \in D^{\text{MR}}$ correspond to the randomized stationary policy d^∞. As noted above, our goal is to find the best approximation of $\mathbf{v}_\lambda^{d^\infty}$ using a lower-dimensional representation. Without direct computation, which we are trying to avoid, we use the result, established

in Chapter **??**, that when $\lambda < 1$, $\mathbf{v}_\lambda^{d\infty}$ is the unique solution of the equation

$$\mathbf{v} = \mathbf{r}_d + \lambda \mathbf{P}_d \mathbf{v} = L_d \mathbf{v}. \tag{1.10}$$

We propose two approximation methods based on different ways of measuring the impact of substituting approximations into the above expression.

We represent the general form of a value function estimate based on a linear combination of basis functions by $\mathbf{v} = \mathbf{B}\boldsymbol{\beta}$ and denote a particular estimate based on an estimator[11] $\hat{\boldsymbol{\beta}}$ of $\boldsymbol{\beta}$ by $\hat{\mathbf{v}} = \mathbf{B}\hat{\boldsymbol{\beta}}$. To simplify awkward grammatical expressions we write col($\mathbf{B}$) to represent the subspace of $\Re^N$ spanned by the columns of $\mathbf{B}$. That is, it the span[12] of the basis functions $\{b_0(s), \ldots, b_K(s)\}$.

Two related approaches can be used to obtain an estimator of $\boldsymbol{\beta}$ from (1.10).

1. **Bellman residual minimization.** Choose $\hat{\boldsymbol{\beta}}_{BR}$ such that

$$\hat{\boldsymbol{\beta}}_{BR} \in \arg\min_{\boldsymbol{\beta} \in \Re^K} \|\mathbf{B}\boldsymbol{\beta} - (\mathbf{r}_d + \lambda \mathbf{P}_d \mathbf{B}\boldsymbol{\beta})\|_2 \tag{1.11}$$

   Given the approximation $\tilde{\mathbf{v}} = \mathbf{B}\tilde{\boldsymbol{\beta}}$, $\hat{\boldsymbol{\beta}}_{BR}$ aims to minimize the Euclidean difference between $\tilde{\mathbf{v}}$ and $\mathbf{r}_d + \lambda \mathbf{P}_d \tilde{\mathbf{v}}$.

2. **Least-squares approximation.** Choose $\hat{\boldsymbol{\beta}}_{LSPE}$ such that

$$\hat{\boldsymbol{\beta}}_{LSPE} \in \arg\min_{\boldsymbol{\beta} \in \Re^K} \|\mathbf{B}\boldsymbol{\beta} - \boldsymbol{\Pi}(\mathbf{r}_d + \lambda \mathbf{P}_d \mathbf{B}\boldsymbol{\beta})\|_2, \tag{1.12}$$

   where $\boldsymbol{\Pi}\mathbf{u}$ denotes the projection of $\mathbf{u} \in \Re^N$ onto col($\mathbf{B}$). That is

$$\boldsymbol{\Pi}\mathbf{u} \in \arg\min_{\mathbf{u}' \in \text{col}(\mathbf{B})} \|\mathbf{u}' - \mathbf{u}\|_2. \tag{1.13}$$

   Given the approximation $\tilde{\mathbf{v}} = \mathbf{B}\tilde{\boldsymbol{\beta}}$, $\hat{\boldsymbol{\beta}}_{LSPE}$ aims to minimize the difference between $\tilde{\mathbf{v}}$ and the projection of $\mathbf{r}_d + \lambda \mathbf{P}_d \tilde{\mathbf{v}}$ onto col($\mathbf{B}$).

   Note that $\boldsymbol{\Pi}$ (and the closely related matrix $\boldsymbol{\Gamma}$) can be expressed as closed form expressions of $\mathbf{B}$ as follows:

$$\boldsymbol{\Pi} := \mathbf{B}(\mathbf{B}^\mathsf{T}\mathbf{B})^{-1}\mathbf{B}^\mathsf{T} \quad \text{and} \quad \boldsymbol{\Gamma} := (\mathbf{B}^\mathsf{T}\mathbf{B})^{-1}\mathbf{B}^\mathsf{T}. \tag{1.14}$$

   The matrix $\boldsymbol{\Pi} = \mathbf{B}\boldsymbol{\Gamma}$ is a projection of $\Re^N$ onto the K-dimensional subspace col($\mathbf{B}$). We emphasize that $\boldsymbol{\Gamma}\mathbf{v}$ is a vector of parameters and $\boldsymbol{\Pi}\mathbf{v}$ is a vector of values.

[11] We use the statistical expression *estimator*, even though there is no randomness in the model.

[12] We are using the expression *span* in the linear algebra sense.

Geometrically these two approaches minimize different quantities (see Figure 1.6). The Bellman residual minimization approach finds a value for $\boldsymbol{\beta}$ that minimizes the distance between $L_d\mathbf{B}\boldsymbol{\beta}$ and $\mathbf{B}\boldsymbol{\beta}$. In this case $L_d\mathbf{B}\boldsymbol{\beta}$ need **not** be in col($\mathbf{B}$).

The least-squares approach seeks a $\boldsymbol{\beta}$ that minimizes the distance between $L_d\mathbf{B}\boldsymbol{\beta}$ and its projection on col($\mathbf{B}$). Another way to view this suggests the following recursive scheme. Begin with $\mathbf{v}$ and project it onto col($\mathbf{B}$) to obtain $\mathbf{B}\boldsymbol{\beta}$. Apply the "Bellman" operator to obtain $L_d\mathbf{B}\boldsymbol{\beta}$ (which most likely lies outside col($\mathbf{B}$)). Set this quantity equal to $\mathbf{v}'$ and project it onto col($\mathbf{B}$) to obtain $\mathbf{B}\boldsymbol{\beta}'$. The goal is to repeat this scheme until $\mathbf{B}\boldsymbol{\beta}$ and $\mathbf{B}\boldsymbol{\beta}'$ are very close. Thus, instead of seeking a solution of $\mathbf{v} = L_d\mathbf{v}$ in $\Re^N$, this approach seeks a solution of this equation in col($\mathbf{B}$).

In principle, to find these two estimates, we vary $\boldsymbol{\beta}$ to make each of these distances as small as possible. In the one dimensional example shown in Figure 1.6, this would correspond to sliding the scalar value β along the horizontal axis and noting how each of the above differences vary.

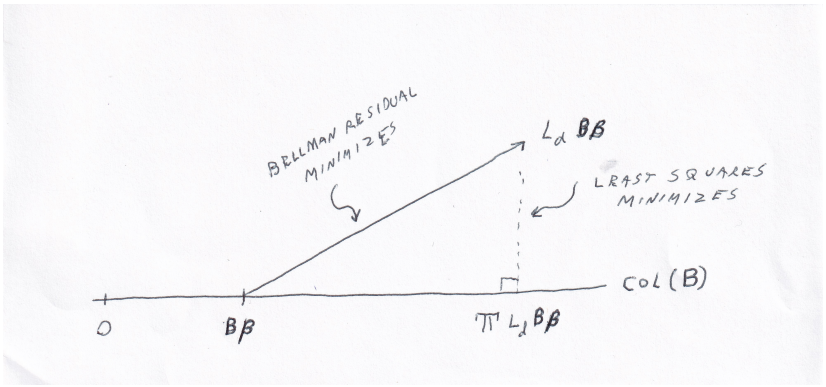


Figure 1.6: Graphical comparison of two approaches for estimating $\boldsymbol{\beta}$. This figure assumes that $\mathbf{B}$ contains a single column so that col($\mathbf{B}$) (corresponding to a single basis function) is a one-dimensional subspace of $\Re^2$.

1.3.1 Least squares policy evaluation

We will see below that we can implement the least squares approach by an iterative algorithm we refer to as *least-squares policy evaluation (LSPE)*. We focus primarily on LSPE below because it is transparent, generalizes easily to nonlinear architectures, is the most widely used approach and is easy to extend to simulation-based methods underlying reinforcement learning. We will briefly discuss Bellman residual minimization in a subsequent starred section .

Least squares estimation – $\mathbf{v}_\lambda^{d^\infty}$ known

Suppose that $\mathbf{v}_\lambda^{d^\infty}$ were *known*[13], but we seek a $K+1$-dimensional linear approximation in col($\mathbf{B}$). The ordinary least-squares[14] (OLS) parameter and value function estimates ((1.62) and (1.63) in the chapter Appendix) become

$$\hat{\boldsymbol{\beta}}_{OLS} = (\mathbf{B}^\mathsf{T}\mathbf{B})^{-1}\mathbf{B}^\mathsf{T}\mathbf{v}_\lambda^{d^\infty} = \boldsymbol{\Gamma}\mathbf{v}_\lambda^{d^\infty} = \boldsymbol{\Gamma}(\mathbf{I} - \lambda\mathbf{P}_d)^{-1}\mathbf{r}_d \tag{1.15}$$

and

$$\hat{\mathbf{v}}_{OLS} = \mathbf{B}(\mathbf{B}^\mathsf{T}\mathbf{B})^{-1}\mathbf{B}^\mathsf{T}\mathbf{v}_\lambda^{d^\infty} = \boldsymbol{\Pi}\mathbf{v}_\lambda^{d^\infty} = \boldsymbol{\Pi}(\mathbf{I} - \lambda\mathbf{P}_d)^{-1}\mathbf{r}_d. \tag{1.16}$$

Least-squares policy evaluation (LSPE)

The following iterative algorithm (stated in matrix form) which corresponds to applying value iteration to a fixed policy provides an estimate of $\boldsymbol{\beta}$. Its beauty is that it:

1. is intuitive,
2. avoids computation of $\mathbf{v}_\lambda^{d^\infty}$,
3. can be applied on a fixed or randomly generated subset of S, and
4. easily generalizes to nonlinear architectures such as neural networks.

Algorithm 1.1. Least-squares policy evaluation (LSPE) – linear architectures

1. Specify $\boldsymbol{\beta}$, $\epsilon > 0$, and $\Delta > \epsilon$.
2. Do until $\Delta < \epsilon$:
   (a) $\tilde{\mathbf{v}} \leftarrow \mathbf{B}\boldsymbol{\beta}$.
   (b) $\mathbf{v} \leftarrow \mathbf{r}_d + \lambda\mathbf{P}_d\tilde{\mathbf{v}}$
   (c) $\boldsymbol{\beta}' = \boldsymbol{\Gamma}\mathbf{v}$[a]
   (d) $\Delta = \|\boldsymbol{\beta}' - \boldsymbol{\beta}\|_2$[b]
   (e) $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$
3. Return $\hat{\boldsymbol{\beta}}_{LSPE} = \boldsymbol{\beta}$.

[a] In practice we would use a regression function such as *lm* in R [2021] to implement this step.

[13] Of course if this quantity were known, we would not need to approximate it unless we needed to interpolate or extrapolate, as is the case when truncating a countable state model.

[14] Alternatives to OLS include weighted least squares and minimum absolute deviation regression.

[b]Recall that the the *Euclidean or L^2 norm*, $\|\mathbf{x}\|_2$ of $\mathbf{x} \in \Re^N$ is defined by

$$\|\mathbf{x}\|_2 := (x_1^2 + \ldots + x_N^2)^{\frac{1}{2}} = (\mathbf{x}^T\mathbf{x})^{\frac{1}{2}}.$$

The parameter estimate generated by LSPE can be represented in closed form as follows. Starting with $\boldsymbol{\beta}$, 2(a) generates the approximate value funciton $\tilde{\mathbf{v}} = \mathbf{B}\boldsymbol{\beta}$. Then step 2(b) performs a one-step update on the approximation to obtain

$$\mathbf{v} = \mathbf{r}_d + \lambda \mathbf{P}_d \tilde{\mathbf{v}} = \mathbf{r}_d + \lambda \mathbf{P}_d \mathbf{B}\boldsymbol{\beta}.$$

Step 2(c) represents least squares regression of $\mathbf{v}$ on the basis vectors (represented as columns of $\mathbf{B}$) to obtain

$$\boldsymbol{\beta}' = (\mathbf{B}^{\mathsf{T}}\mathbf{B})^{-1}\mathbf{B}^{\mathsf{T}}\mathbf{v} = \boldsymbol{\Gamma}\mathbf{v}$$

Combining the above two expressions we see that LSPE generates $\boldsymbol{\beta}'$ from $\boldsymbol{\beta}$ by

$$\boldsymbol{\beta}' = \boldsymbol{\Gamma}\Big(\mathbf{r}_d + \lambda \mathbf{P}_d \mathbf{B}\boldsymbol{\beta}\Big) = \boldsymbol{\Gamma}\mathbf{r}_d + \lambda \boldsymbol{\Gamma}\mathbf{P}_d \mathbf{B}\boldsymbol{\beta}. \tag{1.17}$$

Thus, the updated approximation at the subsequent application of step 2(a) becomes

$$\tilde{\mathbf{v}} = \mathbf{B}\boldsymbol{\Gamma}\Big(\mathbf{r}_d + \lambda \mathbf{P}_d \mathbf{v}\Big) = \boldsymbol{\Pi}\Big(\mathbf{r}_d + \lambda \mathbf{P}_d \mathbf{v}\Big) = \boldsymbol{\Pi} L_d \mathbf{v}. \tag{1.18}$$

This expression involving the projection matrix $\boldsymbol{\Pi}$ justifies also referring to this algorithm as "projected value iteration".

As a result of (1.17) the estimate $\hat{\boldsymbol{\beta}}_{LSPE}$ is a solution of

$$(\mathbf{I} - \lambda \boldsymbol{\Gamma}\mathbf{P}_d \mathbf{B})\boldsymbol{\beta} = \boldsymbol{\Gamma}\mathbf{r}_d. \tag{1.19}$$

where $\mathbf{I}$ is the $(K+1) \times (K+1)$ identity matrix. When $(\mathbf{I} - \lambda \boldsymbol{\Gamma}\mathbf{P}_d \mathbf{B})$ is invertible we can represent the fixed point by

$$\hat{\boldsymbol{\beta}}_{LSVI} = (\mathbf{I} - \lambda \boldsymbol{\Gamma}\mathbf{P}_d \mathbf{B})^{-1}\boldsymbol{\Gamma}\mathbf{r}_d. \tag{1.20}$$

The following example[15] shows that this inverse need not exist and as well illustrates the notation above.

Example 1.3. Set $S = \{s_1, s_2\}$, $\mathbf{r}_d$ be arbitrary, $\lambda = 5/5.4 = 0.926$ and

$$\mathbf{P}_d = \begin{bmatrix} 0.2 & 0.8 \\ 0.2 & 0.8 \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

Hence

$$\boldsymbol{\Gamma} = \left(\begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 2 \end{bmatrix} = \frac{1}{5} \begin{bmatrix} 1 & 2 \end{bmatrix}$$

[15]This example appears in deFarias and van Roy [2000].

so that

$$\lambda \mathbf{\Gamma P}_d \mathbf{B} = \frac{5}{5.4}\left(\frac{1}{5}\begin{bmatrix}1 & 2\end{bmatrix}\begin{bmatrix}0.2 & 0.8\\ 0.2 & 0.8\end{bmatrix}\begin{bmatrix}1\\ 2\end{bmatrix}\right) = 1.$$

This implies that $\mathbf{I} - \lambda \mathbf{\Gamma P}_d \mathbf{B} = 0$ so it is **not** invertible. The iterative scheme implicit in (1.17) reduces to the non-divergent recursion $\boldsymbol{\beta}' \leftarrow \mathbf{\Gamma r}_d + \boldsymbol{\beta}$. Hence, LSPE does not converge for this example.

We view this example as an *edge case* that is unlikely to occur in practice. In most settings, LSPE will converge as shown in the example below.

Example 1.4. We illustrate the calculations described above in the context of the model in Example 1.1. We approximate the value function for the previously specified stationary policy d^∞ by linear and quadratic functions of the state and compare the estimates based on LSPE (or its fixed point representation (1.19)) to the exact value function and its OLS estimate.

Let $c(s,a) := f(s) + m(a)$. We implement step 2(b) of LSPE on a componentwise basis by the expressions:

$$\begin{aligned}
v(0) &= c(0, d(0)) + \lambda((1-b)\tilde{v}(0) + b\tilde{v}(1))\\
v(s) &= c(s, d(s)) + \lambda((1-b)\tilde{v}(s-1) + (1-b-d(s))\tilde{v}(s) + d(s)\tilde{v}(s)), \text{ for } s = 1, \ldots, 49\\
v(50) &= c(50, d(50)) + \lambda(d(50)\tilde{v}(49) + (1-d(50))\tilde{v}(50))
\end{aligned}$$

where $\tilde{v}(s)$ denotes the linear approximation

$$\tilde{v}(s) = \beta_0 + \beta_1 s$$

or the quadratic approximation

$$\tilde{v}(s) = \beta_0 + \beta_1 s + \beta_2 s^2.$$

As an aside, note that there is no need to distinguish the truncation state 50 since the approximation $\tilde{v}(51)$ is available. Thus we can view this approach as evaluating the infinite state model with $S = \{0, 1, \ldots\}$ by an approximation of $\boldsymbol{\beta}$ based on the reduced set of states $S' = \{0, \ldots, 50\}$.

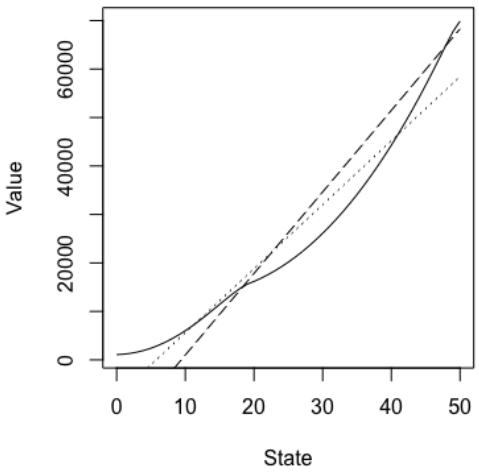
Implementing LSPE for the linear approximations yields $\hat{\beta}_0 = -15825.3$ and $\hat{\beta}_1 = 1682.6$ compared to OLS values $\hat{\beta}_0 = -7603.3$ and $\hat{\beta}_1 = 1320.9$. Graphical comparisons are provided in Figure 1.7a. The main takeaways are that these two estimates are quite different and neither approximates $v_\lambda^{d^\infty}(s)$ well.

We now compare LSPE and OLS quadratic polynomial approximations. We find that LSPE gives $\hat{\beta}_0 = 3371.2$, $\hat{\beta}_1 = -216.1$ and $\hat{\beta}_2 = -30.6$ in contrast to the OLS values $\hat{\beta}_0 = 2096.3$, $\hat{\beta}_1 = 133.2$ and $\hat{\beta}_2 = 23.8$. Graphical comparisons of fitted values appear in Figure **??**. Observe that the quadratic fit is superior to the

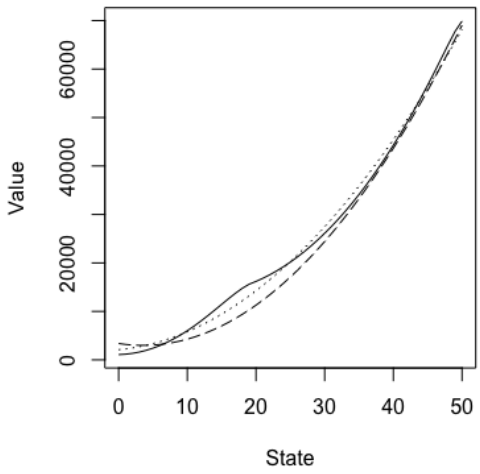
linear fit. Moreover, both approximations are close to the true value function for $s \geq 30$ but differ at lower values. Note also that the LSPE approximation is not monotone in s for low occupancy levels.

If we took into account the fact that the decision rule $d(s)$ policy has a step change at $s = 20$, then we might obtain a better fit for this policy using a spline with a knot at 20. However letting this particular policy determine the form of the approximation will have downsides when we turn to optimization.

We leave it to the reader to compare approximations that use cubic polynomials.



(a) Comparison of value function approximations using $\beta_0 + \beta_1 s$. The solid line indicates the exact value function, the dotted line indicates the OLS estimate of the exact value function and the dashed line indicates the LSPE estimate.



(b) Comparison of value functions approximations based on $\beta_0 + \beta_1 s + \beta_2 s^2$. The solid line indicates the exact value function, the dotted line indicates the OLS estimate of the exact value function and the dashed line indicates the LSPE approximation.

Figure 1.7: Linear and quadratic approximations to the exact value function for the specified policy in Example 1.4

Bellman residual approximation*

As noted earlier we can also obtain a linear approximation to the value function by substituting $\mathbf{v} = \mathbf{B}\boldsymbol{\beta}$ into $(\mathbf{I} - \lambda \mathbf{P}_d)\mathbf{v} = \mathbf{r}_d$ to obtain

$$(\mathbf{I} - \lambda \mathbf{P}_d)\mathbf{B}\boldsymbol{\beta} = (\mathbf{B} - \lambda \mathbf{P}_d \mathbf{B})\boldsymbol{\beta} \approx \mathbf{r}_d. \tag{1.21}$$

This approach regresses $\mathbf{r}_d$ on the columns of $(\mathbf{I} - \lambda \mathbf{P}_d)\mathbf{B}$ directly. Defining $\mathbf{G}_d := \mathbf{B} - \lambda \mathbf{P}_d \mathbf{B}$, the least squares approximations based on (1.21) become

$$\hat{\mathbf{v}} = \mathbf{G}_d(\mathbf{G}_d^\mathsf{T}\mathbf{G}_d)^{-1}\mathbf{G}_d^\mathsf{T}\mathbf{r}_d \tag{1.22}$$

and

$$\hat{\boldsymbol{\beta}}_{BR} = (\mathbf{G}_d^\mathsf{T}\mathbf{G}_d)^{-1}\mathbf{G}_d^\mathsf{T}\mathbf{r}_d. \tag{1.23}$$

Note that in the above the matrix $\mathbf{G}_d = \mathbf{B} - \lambda \mathbf{P}_d \mathbf{B}$ is $N \times K$ so that $\mathbf{G}_d \mathbf{G}_d^\mathsf{T}$ is $K \times K$. We do not explore this approximation further, but will ask you to evaluate it in some exercises.

To avoid matrix inversion, we can instead solve the normal equations

$$\mathbf{G}_d^\mathsf{T}\mathbf{G}_d\hat{\boldsymbol{\beta}} = \mathbf{G}_d^\mathsf{T}\mathbf{r}_d. \tag{1.24}$$

directly.

1.3.2 Linear programming approximations: fixed policies

Linear programming offers another way of finding approximations for linear architectures.

Approximate linear programming - primal model

Chapter **??** formulated the following (primal) LP to determine optimal value functions:

Primal Linear Program – exact model

$$\text{minimize} \quad \sum_{s \in S} \alpha(s) v(s) \tag{1.25a}$$

$$\text{subject to} \quad v(s) - \lambda \sum_{j \in S} p(j|s,a) v(j) \geq r(s,a), \quad a \in A_s, s \in S \tag{1.25b}$$

When there is **only one** deterministic stationary policy $d^\infty(s)$ to evaluate, this primal LP reduces to

$$\text{minimize} \quad \sum_{s \in S} \alpha(s) v(s) \tag{1.26a}$$

$$\text{subject to} \quad v(s) - \lambda \sum_{j \in S} p(j|s,d(s)) v(j) \geq r(s,d(s)), \quad s \in S. \tag{1.26b}$$

Observe that when restricted to a single stationary policy, this LP has $|S|$ variables and $|S|$ constraints. Now suppose we approximate $v(s)$ by a linear function of prespecified basis functions $(b_0(s), b_1(s), \ldots, b_K(s))$ of the form

$$v(s) = \beta_0 b_0(s) + \beta_1 b_1(s) + \ldots + \beta_K b_K(s)$$

and substitute this expression into (1.26a) and (1.26b). Then after rearranging terms we obtain the approximate primal linear program:

Approximate primal linear program (APLP) – fixed decision rule

$$\text{minimize} \quad \sum_{k=0}^{K} \beta_k \sum_{s\in S} \alpha(s) b_k(s) \tag{1.27a}$$

$$\text{subject to} \quad \sum_{k=0}^{K} \beta_k b_k(s) - \lambda \left(\sum_{k=0}^{K} \beta_k \sum_{j\in S} p(j|s, d(s)) b_k(j) \right) \geq r(s, d(s)), \quad s \in S. \tag{1.27b}$$

Observe that this LP has $K+1$ variables $\beta_0, \beta_1, \ldots, \beta_K$ and $|S|$ constraints. Since in most applications $|S| \gg K$, this system has many more constraints than variables. The following example illustrates a very simple case of this approximation and sheds some light on the geometry underlying the LP approximation.

Example 1.5. Suppose there is a single feature $b_0(s) = 1$ for all $s \in S$ so that $\mathbf{B} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Consequently the value function will be estimated[a] by a constant β_0. We choose this simple model so that we can obtain the LP solution easily and can represent results graphically. Assuming $\sum_{s\in S} \alpha(s) = 1$, the APLP becomes

$$\begin{aligned} \text{minimize} \quad & \beta_0 \\ \text{subject to} \quad & (1-\lambda)\beta_0 \geq r(s, d(s)), \quad s \in S. \end{aligned}$$

Moreover this LP has one variable and $|S|$ constraints and is easy to solve. Indeed, the optimal solution is

$$\hat{\beta}_0^{LP} = \max_{s\in S} \frac{r(s, d(s))}{1-\lambda}.$$

For a more interesting example, consider a two-state ($S = \{0, 1\}$) version of the queuing service rate control model that we have previously analyzed in this chapter. We assume an arrival rate of 0.2 and service rate of $d(0) = a_1 = 0.2$ and $d(1) = a_2 = 0.4$. To be consistent with our LP formulation, we formulate it as a reward maximization problem so that $r(0, d(0)) = -5$ and $r(1, d(1)) = -41$.

Therefore the linear program when $\lambda = 0.5$ becomes

$$\begin{aligned} \text{minimize} \quad & \beta_0 \\ \text{subject to} \quad & 0.5\beta_0 \geq -5 \\ & 0.5\beta_0 \geq -41 \end{aligned}$$

and its optimal solution is $\hat{\beta}_0^{LP} = -10$. Consequently the LP approximation to the value function is $\hat{v}_{LP}(0) = \hat{v}_{LP}(1) = -10$.

Note that if S were instead truncated at $N = 50$, there would be 51 inequalities of a similar form to those above and we would use a higher order approximation to the value function.

The exact LP for this model is

$$\begin{aligned} \text{minimize} \quad & \alpha_0 v(0) + \alpha_1 v(1) \\ \text{subject to} \quad & 0.6v(0) - 0.1v(1) \geq -5 \\ & -0.2v(0) + 0.7v(1) \geq -41 \end{aligned}$$

The solution of the exact LP is $v_\lambda^{d^\infty}(0) = -19$ and $v_\lambda^{d^\infty}(1) = -64$. Note that the least squares approximation to this solution would be the average of these two values, $\hat{\beta}_0^{OLS} = -41.5$, so that the least squares approximation is $\hat{v}_{OLS}(0) = \hat{v}_{OLS}(1) = -41.5$.

Figure 1.8 illustrates these estimates. Its caption explains what is shown in considerable detail. The takeaway is that when the subspace spanned by the basis functions intersects the feasible region near the optimal vertex, then the LP approach will provide good estimates of the optimal value function. Unfortunately, we cannot know this a priori, so care must be taken when choosing basis functions.

[a] In regression a model with only constant, the estimate of the constant is the mean of the observations.

We will expand on the approximate LP model below when we use it to find approximate optimal policies.

1.4 Parameter estimation for a fixed policy: nonlinear architectures

Nonlinear functions of features

We modify Algorithm 1.1 to use nonlinear architectures. To do so we replace computation of $\tilde{\mathbf{v}}$ in step 2(a) by

$$\tilde{v}(s) = f(s, \boldsymbol{\beta}),$$

which we write as $\tilde{\mathbf{v}} \leftarrow \mathbf{f}(\boldsymbol{\beta})$. In step 2(c), we replace the computation of $\tilde{\mathbf{v}}$ by its least squares estimate obtained using Gauss-Newton iteration or some other method. We express this step as $\boldsymbol{\beta}' \in \arg\min_{\boldsymbol{\beta}} \|\mathbf{f}(\boldsymbol{\beta}) - \mathbf{v}\|_2$ since a closed form representation for $\hat{\boldsymbol{\beta}}$ is unavailable for nonlinear architectures.

Algorithm 1.2. Least-squares value iteration - non-linear architectures

1. Specify $\boldsymbol{\beta}$, $\epsilon > 0$, and $\Delta > \epsilon$.

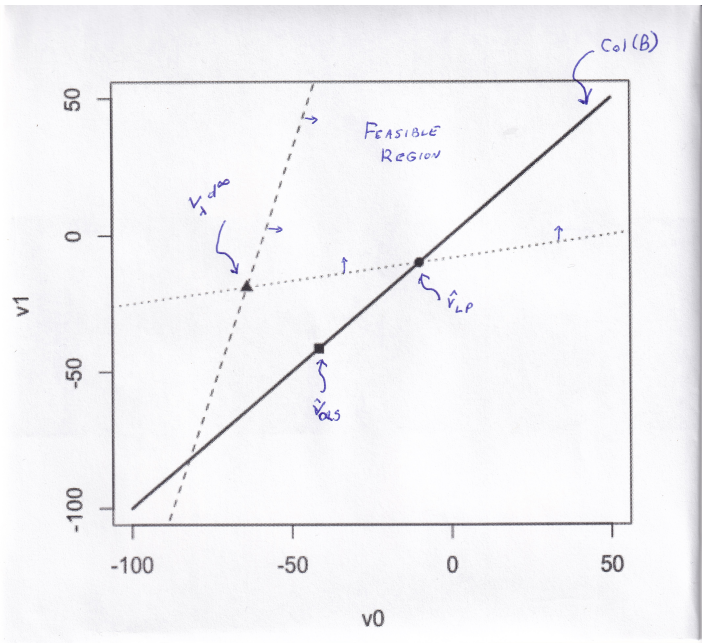


Figure 1.8: Graphical representation of the value function and two approximations for Example 1.5. It shows the feasible region for the exact LP and its solution $\mathbf{v}_\lambda^{d^\infty}$ and the subspace col($\mathbf{B}$) where $\mathbf{B} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. $\hat{\mathbf{v}}_{LP}$ and $\hat{\mathbf{v}}_{OLS}$ denote the LP and OLS estimates in that subspace. Observe that the LP estimate is the minimum value of the intersection of col($\mathbf{B}$) and the feasible region of the LP and the OLS estimate is the orthogonal projection of the exact value function on to that subspace.

2. Do until $\Delta < \epsilon$:
   (a) $\tilde{\mathbf{v}} \leftarrow \mathbf{f}(\boldsymbol{\beta})$.
   (b) $\mathbf{v} = \mathbf{r}_d + \lambda \mathbf{P}_d \tilde{\mathbf{v}}$
   (c) $\boldsymbol{\beta}' \in \arg\min_{\boldsymbol{\beta}} \|\mathbf{f}(\boldsymbol{\beta}) - \mathbf{v}\|_2$
   (d) $\Delta = \|\boldsymbol{\beta}' - \boldsymbol{\beta}\|_2$
   (e) $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$
3. Return $\boldsymbol{\beta}$.

We now apply this algorithm to queuing service rate control model.

Example 1.6. We illustrate the nonlinear LSPE algorithm above in the queuing service rate control model in Example 1.1. We approximate the value function for the specified stationary policy d^∞ by an exponential function and compare the

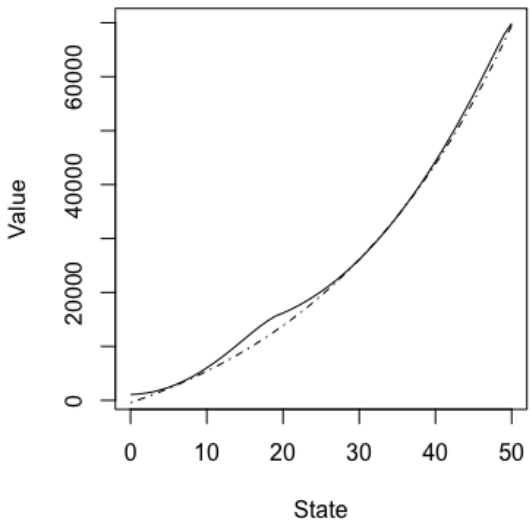


Figure 1.9: True value function (solid line) for queuing service rate control model with exponential approximation where parameters are estimated using nonlinear LSVI (dashed line).

estimates based on LSPE to the exponential fit when the the value function is known. To implement the algorithm we chose $\boldsymbol{\beta} = \mathbf{0}$ so that the corresponding initial value function $\tilde{\mathbf{v}} = \mathbf{0}$ and $\mathbf{v}$ in step 2(b) becomes $\mathbf{v} = \mathbf{r}_d$. We implemented step 2(c) using the nls function in R [2021]. This required some adjustment of starting value to ensure convergence at every iteration of step 2(c).

The algorithm terminated in 386 iterations with the approximation

$$\hat{v}_\lambda^{d^\infty}(s) = -13427.3 + 13004.1e^{0.037s}$$

which is shown in Figure 1.9. Observe that $\hat{\beta}_1$ and $\hat{\beta}_2$ differ from that in Example 1.1 but $\hat{\beta}_3$ is almost identical to that obtained when the value function was known. Moreover the approximation was excellent.

We emphasize that applying non-linear architectures requires considerable more care than in the linear case because of the sensitivity of non-linear algorithms to starting values and other algorithmic parameters.

As in the case of a linear architecture, this method can be applied to a subset of states that are pre-determined or sampled.

Neural network approximations

We now show how neural networks (NNs) can be used for curve fitting by estimating a value function in the queuing service rate control model analyzed throughout this section. In contrast with other examples in this section, we fit the model on a **sample** of states and values referred to as the *training set*. We do so as a prelude to reinforcement learning and in the spirit of how neural networks are used in other applications where the model is fit on a training set and its quality is assessed on a holdout sample.

Example 1.7. We set N=200 and use the optimal value function found by solving the primal linear program[a]. We choose a larger model than before so we can sample from the state space. In our calculations described below, we take a random sample of 85% of the observations as the training set and withhold 15% for cross-validation.

After some experimentation, we settled on the neural network shown (with estimated weights) in Figure 1.10a. The input to the NN is the state. The NN contains two hidden layers, the first having one node and the second having two nodes. Each layer is offset by a constant bias term. The value function estimate is a weighted sum of the outputs from the second layer plus a constant.

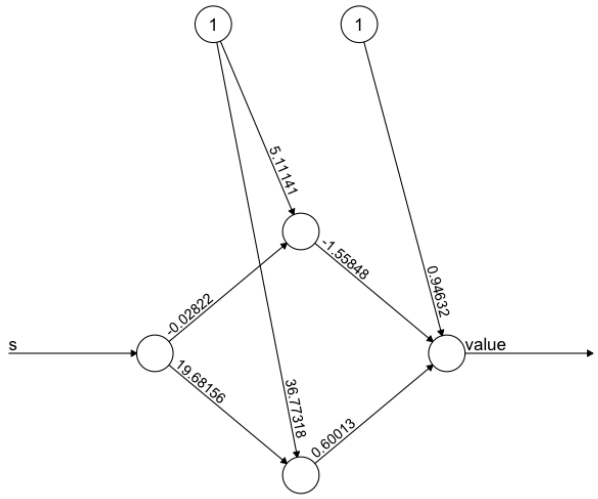
We used the module *neuralnet* in R [2021] to obtain the output in Figure 1.10. Note that we scaled the value function to lie between 0 and 1 prior to fitting. The predicted value function was then rescaled for comparison purposes. Note that results varied from sample to sample and in some instances produced constant estimate. Figure 1.10b shows that the fit from this model was excellent. Recalling the von Neumann quote at the beginning of this chapter, this is to be expected in a model with 7 parameters . Note also that many other NN configurations gave similar fits.

[a]We could have just as easily used policy iteration or an iteration algorithm with a small tolerance.

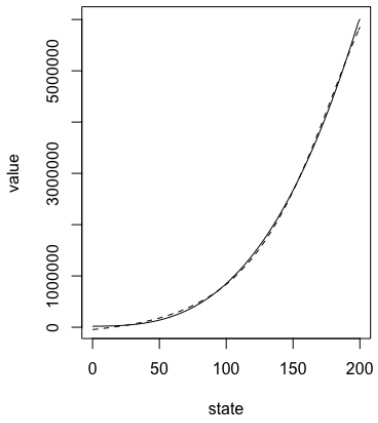
1.5 Combining approximation and optimization

We now generalize the results in the previous section to provide methods that seek good policies in a Markov decision process based on approximations. Such methods are usually referred to as approximate dynamic programming (ADP). We describe value iteration, policy iteration and modified policy iteration algorithms based on linear, nonlinear and neural network architectures and linear programming which by necessity applies only to linear architectures.

In practice these iterative methods can be applied to pre-specified or randomly selected subsets of the state space, but most of our examples implement them on the entire state space to illustrate the most optimistic outcomes.



(a) Neural network model used in Example 1.7 with estimated weights. The input is the state s and each layer has an offset denoted by 1.



(b) Comparison fitted NN model (dashed line) and actual (solid line).

Figure 1.10: NN Example

1.5.1 Iterative methods for linear architectures

We provide an optimization version of LSPE and generalize it to obtain policy iteration and modified policy iteration algorithms. We assume a pre-specified set of basic functions and a set of states at which to evaluate them so that the matrix $\mathbf{B}$ can be specified a priori.

Optimization using LSVI - Linear Architectures

We provide the following generalization of Algorithm 1.1 to incorporate optimization. The only change is in step 2(b) where the Bellman operator replaces the one-step policy update $\mathbf{v} \leftarrow \mathbf{r}_d + \lambda \mathbf{P}_d \tilde{\mathbf{v}}$.

Algorithm 1.3. Least-squares value iteration – linear architectures

1. Specify $\boldsymbol{\beta}$, $\epsilon > 0$, and $\Delta > \epsilon$.
2. Do until $\Delta < \epsilon$:
   (a) $\tilde{\mathbf{v}} \leftarrow \mathbf{B}\boldsymbol{\beta}$.
   (b) $\mathbf{v} \leftarrow \text{c-max}_{d \in D^{\text{MD}}} \{\mathbf{r}_d + \lambda \mathbf{P}_d \tilde{\mathbf{v}}\}$
   (c) $\boldsymbol{\beta}' = \boldsymbol{\Gamma} \mathbf{v}^a$
   (d) $\Delta = \|\boldsymbol{\beta}' - \boldsymbol{\beta}\|_2$

(e) $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$

3. Return $\hat{\boldsymbol{\beta}}_{LSVI} = \boldsymbol{\beta}$.

[a] In practice we often use a regression function such as *lm* in R [2021] to implement this step.

As noted in Example 1.3 above, the algorithm cannot be guaranteed to converge because the operator implicitly defined by the algorithm need not be a contraction mapping. However, empirically, this algorithm performs well as illustrated by the examples below.

Least squares policy iteration – linear architectures

We now provide a policy iteration based method (LSPI) for finding "good" policies. Instead of choosing the value in step 2(b) of Algorithm 1.3, it obtains the arg c-max, evaluates the resulting policy and then finds the best approximation. We can initiate the algorithm with either:

1. a policy,
2. a value, or
3. a parameter vector.

The order of the steps in the statement of the algorithm depends on how it is initialized. With linear architectures we found it best to begin with a parameter or value, with non-linear architectures, we prefer initialization with a value. In the latter case at the first pass through step 2., we would begin step 2. with an improvement .

Algorithm 1.4. Least-squares policy iteration – linear architectures

1. Specify $d \in D^{\text{MD}}$, $\epsilon > 0$, $\boldsymbol{\beta}$ arbitrary, and $\Delta > \epsilon$.
2. Do until $\Delta < \epsilon$:
   (a) **Parameter Evaluation:** $\boldsymbol{\beta}' \leftarrow (\mathbf{I} - \lambda \boldsymbol{\Gamma} \mathbf{P}_{d'} \mathbf{B})^{-1} \boldsymbol{\Gamma} \mathbf{r}_{d'}$.
   (b) **Value Function Approximation:** $\tilde{\mathbf{v}} \leftarrow \mathbf{B}\boldsymbol{\beta}$
   (c) **Improvement:** $d' \leftarrow \arg\,\text{c-max}_{d \in D^{\text{MD}}} \{\mathbf{r}_d + \lambda \mathbf{P}_d \tilde{\mathbf{v}}\}$.
   (d) $\Delta = \|\boldsymbol{\beta}' - \boldsymbol{\beta}\|_2$
   (e) $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$.
3. Return $d_{LSPI} = d'$ and $\boldsymbol{\beta}_{LSPI} = \boldsymbol{\beta}$.

Some comments about this algorithm follow.

1. This algorithm can be applied to a subset of S as well.

2. The stopping criterion in step 2 is stated in terms of parameter estimates. It can be replaced by either: "Stop when $d' = d$" or stop when the change in $\mathbf{v}$ is sufficiently small. We found no difference in algorithmic behavior between these criteria in examples. Note if the criterion based on decision rules is used, we need to add the specification "set $d' = d$ if possible" to avoid cycling.

3. In step 2(a), we provide a closed form representation for the parameters $\boldsymbol{\beta}$ corresponding to decision rule d.This representation is valid for linear architectures. In practice, it might be easier to implement it using LSPE. When using nonlinear architectures including neural networks, this step would be replaced by LSPE.

4. Note that the matrix $\boldsymbol{\Gamma}$ is defined in (1.14).

5. Step 2(c) is the improvement step. It identifies an "improved" decision rule.

6. We initialize the algorithm with a decision rule. Alternatively it can be initialized with a starting value for $\boldsymbol{\beta}$ in which case step 2(a) can be skipped at the first pass through the algorithm.

7. This algorithm can be modified easily to use approximations of state-action value functions instead of value functions.

We illustrate this algorithm by applying it to the queuing service rate control model.

Example 1.8. We solved the queuing service rate control model on $S = \{0, \ldots, 50\}$ using linear, quadratic and cubic polynomial approximations. The discount rate was $\lambda = 0.9$. All instances required 3 iterations for convergence.

Figure 1.11 shows the optimal policy and the policies determined by the algorithm. In all cases the approximations resulted in monotone non-decreasing decision rules. Clearly the linear approximation was inadequate. The quadratic and cubic approximations produced similar policies differing only in states 11,12, 13 and 30 and the policy identified by the cubic approximation was optimal.

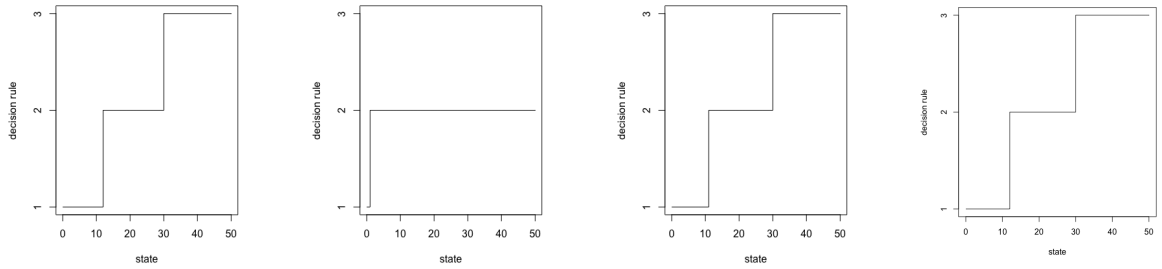
Further calculations revealed that the policies identified by the cubic approximation agreed with the optimal policy for $S = \{0, \ldots, 500\}$ and $\lambda = 0.9$.[a]

To explore the impact of the approximation further, we set $\lambda = 0.98$ and compared the policy from the cubic approximation (d^{∞}_{cub}) to the optimal policy (d^{∞}_{opt}). In this case we observed the following differences:

$$d_{cub}(s) = \begin{cases} a_1 & s \in \{0, \ldots, 20\} \\ a_2 & s \in \{21, 22, 23\} \\ a_3 & s \in \{24, \ldots, 500\} \end{cases} \quad \text{and} \quad d_{opt}(s) = \begin{cases} a_1 & s \in \{0, \ldots, 8\} \\ a_2 & s \in \{10, \ldots, 15\} \\ a_3 & s \in \{16, \ldots, 500\}. \end{cases}$$

To gain further insight to the effect of the approximation, we compared the values of these two policies. Note that LSPI does not provide the value of d^{∞}_{cub}, this requires computing $(\mathbf{I} - \lambda \mathbf{P}_{d_{cub}})^{-1}\mathbf{r}_{d_{cub}}$. These calculations show that values differ most at low occupancy levels. Bounds in the next subsection will provide further insight. However, more accurate approximations may be needed when λ is close to 1.

[a]When solving the larger instance, we centered the approximating polynomials at 250 to avoid numerical instability in step 2(a) when computing $(\mathbf{B}^{\mathsf{T}}\mathbf{B})^{-1}$. That is, the components in the $\mathbf{B}$ matrix were of the form $s - 250$, $(s - 250)^2$ and $(s - 250)^3$. Alternatively, we could have replaced this step by iterative policy evaluation as in the LSMPI algorithm below. algorithm (Algorithm **??**).



(a) Exact policy iteration.

(b) Linear approximation.

(c) Quadratic approximation.

(d) Cubic approximation.

Figure 1.11: Stationary policies obtained using exact policy iteration and LSPI with linear, quadratic and cubic approximations.

We include another example that shows that LSPI may not converge when using linear approximations.

Example 1.9. We apply LSPI to the two-state example from Chapter **??**. We use a single basis function with $b(s_1) = .05$ and $b(s_2) = 1$. Therefore $\mathbf{B} = \begin{bmatrix} .05 \\ 1 \end{bmatrix}$ and $\mathbf{B}\beta = \begin{bmatrix} .05\beta \\ \beta \end{bmatrix}$ and the Bellman update (in component notation) applied to the approximation becomes:

$$v(s_1) \leftarrow \max(3 + \lambda(0.8 \times 0.05 \times \beta + 0.2\beta),\ 5 + \lambda\beta)$$
$$v(s_2) \leftarrow \max(-5 + \lambda\beta,\ 3 + \lambda(0.4 \times 0.05 \times \beta) + 0.6\beta)$$

Set $\lambda = 0.9$ We leave it as exercise to show that for initial values $\beta = 0$ and $\beta = -50$, LSVI converges quickly and identifies an optimal policy.

However for both initial values, LSPI does not converge. Initially it chooses the

optimal decision rule $(a_{1,2}, a_{2,2})$ but then it **cycles** between the parameter estimates -51.87 and 26.92 and decision rules $(a_{1,2}, a_{2,1})$ and $(a_{1,1}, a_{2,2})$ indefinitely.

In contrast to the above observation, We find that for other basis functions and starting values of β that both LSPI and LSVI converge to either an optimal policy or a sub-optimal policy[a].

This simple example shows that it is difficult to predict the behavior of LSPI and LSVI. This phenomenon is explored further in Bertsekas [2011] and Bertseksas [2012].

[a]To be precise in the case of LSVI, we are referring to the policy derived from the greedy decision rule based on the value approximation when a convergence criterion has been satisfied.

Bounds for LSPI

In this section we apply the bounds from Section **??** to LSPI. Step 2(c) of the algorithm implicitly computes the quantity

$$L\tilde{\mathbf{v}} = \underset{d\in D^{\text{MD}}}{\text{c-max}}\{\mathbf{r}_d + \lambda \mathbf{P}_d\tilde{\mathbf{v}}\}.$$

Therefore with minimal additional computation we can apply the bounds from Theorem **??** to obtain

$$\begin{aligned}\tilde{\mathbf{v}} + \frac{1}{(1-\lambda)}\underline{(L\tilde{\mathbf{v}} - \tilde{\mathbf{v}})}\mathbf{e} \le L\tilde{\mathbf{v}} + \frac{\lambda}{(1-\lambda)}\underline{(L\tilde{\mathbf{v}} - \tilde{\mathbf{v}})}\mathbf{e} \le \mathbf{v}_\lambda^{(d')^\infty} \le \mathbf{v}_\lambda^* \\ \le L\tilde{\mathbf{v}} + \frac{\lambda}{(1-\lambda)}\overline{(L\tilde{\mathbf{v}} - \tilde{\mathbf{v}})}\mathbf{e} \le \tilde{\mathbf{v}} + \frac{1}{(1-\lambda)}\overline{(L\tilde{\mathbf{v}} - \tilde{\mathbf{v}})}\mathbf{e}\end{aligned} \tag{1.28}$$

where as before $\overline{\mathbf{u}} = \max_{s\in S} u(s)$, $\underline{\mathbf{v}} = \min_{s\in S} u(s)$ and $d' \in \arg\,\text{c-max}_{d\in D^{\text{MD}}}\{\mathbf{r}_d + \lambda\mathbf{P}_d\tilde{\mathbf{v}}\}$.

Note $\mathbf{v}_\lambda^{(d')^\infty}$ is never computed using LSPI[16] but we can use the bounds to derive the following estimate of how accurately it estimates the optimal value as follows:

$$\|\mathbf{v}_\lambda^{(d')^\infty} - \mathbf{v}_\lambda^*\| \le \frac{\lambda}{(1-\lambda)}\text{sp}(L\tilde{\mathbf{v}} - \tilde{\mathbf{v}}), \tag{1.29}$$

where as before $\text{sp}(\mathbf{v}) = \overline{\mathbf{v}} - \underline{\mathbf{v}}$. Moreover, averaging the inner upper and lower bounds gives the approximation

$$L\tilde{\mathbf{v}} + \frac{\lambda}{2(1-\lambda)}\text{sp}(L\tilde{\mathbf{v}} - \tilde{\mathbf{v}})\mathbf{e} \approx \mathbf{v}_\lambda^*. \tag{1.30}$$

The beauty of these bounds is that they can be applied to any $\mathbf{v}$ regardless how it has been obtained, provided we have computed $L\mathbf{v}$. Unfortunately, the factor of $(1-\lambda)^{-1}$ results in bounds that may not be very tight. We now illustrate these bounds in the context of the queuing control example.

[16]It only computes the approximation $\tilde{\mathbf{v}}$ based on a linear combination of basis functions.

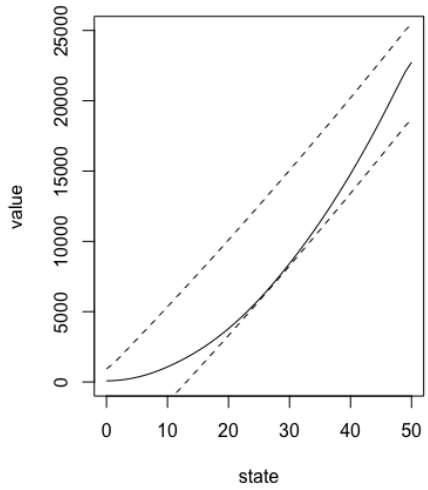
Example 1.10. We evaluate the bounds in (1.28) for the queuing service rate control model with capacity 50 based on approximation with a cubic polynomial. Since $\tilde{\mathbf{v}}$ is computed in step 2(b) and $L\tilde{\mathbf{v}}$ can be easily obtained from step 2(c), we can easily evaluate $L\tilde{\mathbf{v}} - \tilde{\mathbf{v}}$ and all of the bounds. We find $\underline{L\tilde{\mathbf{v}} - \tilde{\mathbf{v}}} = -113.9$ and $\overline{L\tilde{\mathbf{v}} - \tilde{\mathbf{v}}} = 38.2$ from which we obtain the inner bounds shown in Figure 1.12b.

Since $\text{sp}(L\tilde{\mathbf{v}} - \tilde{\mathbf{v}}) = 152.1$, (1.29) yields

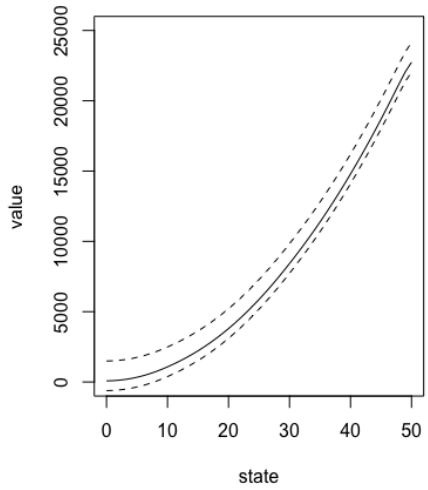
$$\|\mathbf{v}_\lambda^{(d')^\infty} - \mathbf{v}_\lambda^*\| \leq 1369.5$$

and the average absolute error of the approximation in (1.30) is 38.8. Hence in this example, the approximation based on (1.30) is very accurate.

Further calculations show that bounds based on a quadratic approximation are only slightly less tight than those based on the cubic approximation but, those based on a linear approximation are considerably less accurate (Figure 1.12a).



(a) Bounds based on a linear approximation to the value function.



(b) Bounds based on a cubic approximation to the value function.

Figure 1.12: Bounds (dashed lines) for value function (solid line) derived from the LSPI solution using inner inequalities in (1.28).

Least squares modified policy iteration – linear architectures

In large applications, it might be awkward to construct and solve the linear system implicit in step 2(c) of the LSPI iteration algorithm. Instead, we can use the following

generalization of MPI which we refer to as least squares modified policy iteration and denote by LSMPI.

Algorithm 1.5. Least-squares modified policy iteration – linear architectures

1. Specify $\epsilon > 0$, $\boldsymbol{\beta}$ arbitrary, $K > 0$ and $\Delta > \epsilon$.
2. Do until $\Delta < \epsilon$:
   (a) $\tilde{\mathbf{v}} \leftarrow \mathbf{B}\boldsymbol{\beta}$
   (b) $\mathbf{v}' \leftarrow \text{c-max}_{d \in D^{\text{MD}}}\{\mathbf{r}_d + \lambda \mathbf{P}_d \tilde{\mathbf{v}}\}$.
   (c) $d' \leftarrow \arg\text{c-max}_{d \in D^{\text{MD}}}\{\mathbf{r}_d + \lambda \mathbf{P}_d \tilde{\mathbf{v}}\}$.
   (d) Do K times:
      i. $\boldsymbol{\beta}' \leftarrow \boldsymbol{\Gamma}\mathbf{v}'$
      ii. $\tilde{\mathbf{v}} \leftarrow \mathbf{B}\boldsymbol{\beta}'$
      iii. $\mathbf{v}' \leftarrow \mathbf{r}_{d'} + \mathbf{P}_{d'}\tilde{\mathbf{v}}$
   (e) $\boldsymbol{\beta}' \leftarrow \boldsymbol{\Gamma}\mathbf{v}'$
   (f) $\Delta = \|\boldsymbol{\beta}' - \boldsymbol{\beta}\|_2$
   (g) $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$.
3. Return $d_{LSMPI} = d'$ and $\boldsymbol{\beta}_{LSMPI} = \boldsymbol{\beta}$.

Some comments about using this algorithm follow:

1. The algorithm is presented in a way that enables its implementation on a *subset* of S.
2. Step 2(d) corresponds to terminating LSPE after $K + 1$ iterations.
3. The integer K denotes the order of the algorithm. It can vary from iteration to iteration. See the discussion in Chapter **??** for guidance on selection of K. Choosing K adaptively to ensure that the norm of successive values of $\boldsymbol{\beta}'$ computed in 2(d)i. is less than some pre-specified small tolerance is equivalent to LSPI.
4. We include the approximation steps 2(d)i. and 2(d)ii. in order to avoid evaluating $v'(s)$ for all $s \in S$ in 2(d)iii.
5. It may be easier (especially when applying to LSMPI to non-linear architectures[17]) to initialize the algorithm with $\tilde{\mathbf{v}}$ instead of $\boldsymbol{\beta}$. In this case 2a. is not

[17]In complicated non-linear models such as neural networks, it may be challenging to accurately estimate parameter values.

necessary at the first pass through the step 2.

6. Steps 2b. and 2c. are implemented together avoiding double computation.

7. As an alternative to a parameter based stopping criterion, one could use the value based stopping criterion $\|\mathbf{v}' - \tilde{\mathbf{v}}\| < \epsilon$. We found this modification necessary when using neural network approximations in which parameter estimates were highly unstable.

8. We note that the conclusion observations in Example 1.9 apply as well to LSMPI.

LSVI: Nonlinear Architectures

We now modify the above LSVI algorithm (Algorithm 1.4) to allow for non-linear approximation architectures $\mathbf{v} \approx \mathbf{f}(\boldsymbol{\beta})$. The main difference with the linear architecture variant is that the estimate of $\tilde{\boldsymbol{\beta}}$ cannot be obtained in closed form. That is, the closed form representation in Step 2(c) of Algorithm 1.3 is replaced by an estimate derived by non-linear least squares. This step is implemented using appropriate code.

Algorithm 1.6. Least-squares value iteration – nonlinear architectures

1. Specify $\epsilon > 0$, $\boldsymbol{\beta}$ and $\Delta > \epsilon$.
2. Do while $\Delta > \epsilon$:
   (a) $\tilde{\mathbf{v}} \leftarrow \mathbf{f}(\boldsymbol{\beta})$.
   (b) $\mathbf{v}' \leftarrow \text{c-max}_{d \in D^{\text{MD}}} \{\mathbf{r}_d + \lambda \mathbf{P}_d \tilde{\mathbf{v}}\}$.
   (c) $\boldsymbol{\beta}' \in \arg\min_{\boldsymbol{\beta}} \|\mathbf{f}(\boldsymbol{\beta}) - \mathbf{v}'\|_2$
   (d) $\Delta = \|\boldsymbol{\beta}' - \boldsymbol{\beta}\|_2$
   (e) $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$
3. Return $\boldsymbol{\beta}_{LSVI} = \boldsymbol{\beta}$.

LSPI: Nonlinear Architectures

We provide a generalization of LSPI for non-linear architectures.

Algorithm 1.7. Least-squares policy iteration – nonlinear architectures

1. Specify $d \in D^{\text{MD}}$, $\epsilon > 0$, $\delta > 0$, $\boldsymbol{\beta}$ arbitrary and $\Delta > \epsilon$.
   (a) Do until $\Delta < \epsilon$:
   (b) Apply LSPE using d to return $\tilde{\boldsymbol{\beta}}$.

(c) $\tilde{\mathbf{v}} \leftarrow \mathbf{f}(\tilde{\boldsymbol{\beta}})$.

(d) $d' \leftarrow \arg\text{c-max}_{d \in D^{\text{MD}}}\{\mathbf{r}_d + \lambda \mathbf{P}_d \tilde{\mathbf{v}}\}$.

(e) $\Delta = \|\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}\|_2$

(f) $d \leftarrow d'$ and $\boldsymbol{\beta} \leftarrow \tilde{\boldsymbol{\beta}}$.

2. Return $d_{LSPI} = d$ and $\mathbf{v}_{LSPI}$.

We illustrate LSPI with an example.

Example 1.11. As in Example 1.6 we use the approximation

$$v(s) = \beta_0 + \beta_1 e^{\beta_2 s}.$$

Using the stopping criterion "stop if $d' = d$, non-linear LSPI terminates in three iterations. Note that since the algorithm identified the same decision rule at two successive iterations, it also generated the same values of $\boldsymbol{\beta}$ at these iterations so a stopping rule based on $\|\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}\|_2$ would also terminate after 3 iterations.

Figure 1.13a shows the optimal value function and the exponential approximation found with Algorithm 1.7, Figure 1.13b shows the stationary policy based on the optimal exponential approximation, and Figure 1.13c shows bounds on the optimal value function based on the exponential approximation.

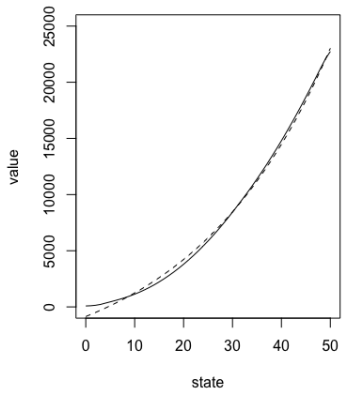
We observe that the approximation does not fit well for low state values and the policy obtained using the approximation deviates from the optimal policy in states 4 to 13 and in state 31. (see Figure 1.11a). Therefore we prefer the cubic polynomial approximation for this instance.

1.5.2 Linear Programming: Linear Architectures

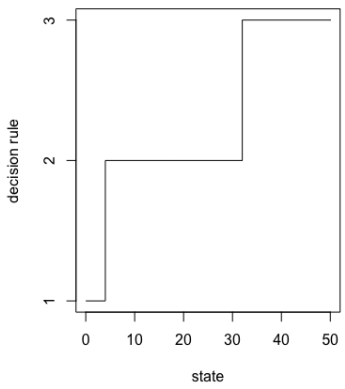
The linear programming approach in Section 1.3.2 was used to find an approximate the value function of a fixed policy. This approach extends easily to finding optimal policies.

Primal LP Approximation

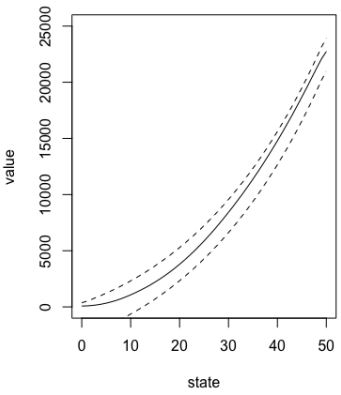
First, we write out the approximate LP in component form.



(a) Comparison of exponential approximation (dashed line) and the true value function (solid line).



(b) Stationary policy obtained using exponential approximation.



(c) Bounds on optimal value function based on exponential approximation.

Approximate primal linear program (APLP): component form

$$\text{minimize} \quad \sum_{k=0}^{K} \beta_k \sum_{s \in S} \alpha(s) b_k(s) \tag{1.31a}$$

$$\text{subject to} \quad \sum_{k=0}^{K} \beta_k b_k(s) - \lambda \left(\sum_{k=0}^{K-1} \beta_k \sum_{j \in S} p(j|s,a) b_k(j) \right) \geq r(s,a), \quad a \in A_s, s \in S. \tag{1.31b}$$

In a cost minimization problem this APLP becomes:

$$\text{minimize} \quad \sum_{k=0}^{K} \beta_k \sum_{s \in S} \alpha(s) b_k(s) \tag{1.32a}$$

$$\text{subject to} \quad \sum_{k=0}^{K} \beta_k b_k(s) - \lambda \left(\sum_{k=0}^{K-1} \beta_k \sum_{j \in S} p(j|s,a) b_k(j) \right) \leq r(s,a), \quad a \in A_s, s \in S. \tag{1.32b}$$

Observe that both models have $K+1$ variables and $\sum_{s \in S} |A_s|$ constraints. Note that the difference between formulation (1.31) and formulation (1.27) is that the constraints are for all $a \in A_s$ rather than for a fixed policy d. Hence, there are $\sum_{s \in S} |A_s|$ constraints instead of $|S|$ constraints.

Matrix version of the approximate linear program

Instead of implementing the model in component notation we find it is considerably easier to approach it from a matrix perspective. We rewrite the (exact) primal model from Section **??** as

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\alpha}^{\mathsf{T}}\mathbf{v} \\ \text{subject to} \quad & \mathbf{A}\mathbf{v} \geq \mathbf{r}, \end{aligned} \tag{1.33}$$

where $\mathbf{r}$ is a column vector with entries $r(s,a)$ listed in the same order as the constraints represented in $\mathbf{A}$ and α is an arbitrary positive $|S|$-dimensional vector. Substituting the linear approximation $\mathbf{v} \approx \mathbf{B}\boldsymbol{\beta}$ into (1.33) yields:

The primal approximate linear program: matrix form

$$\begin{aligned} \text{minimize} \quad & (\boldsymbol{\alpha}^{\mathsf{T}}\mathbf{B})\boldsymbol{\beta} \\ \text{subject to} \quad & (\mathbf{A}\mathbf{B})\boldsymbol{\beta} \geq \mathbf{r}. \end{aligned} \tag{1.34}$$

This suggests the following approach for finding policies based on the APLP.

Algorithm 1.8. Using APLP to find an approximate optimal policy

1. Specify an approximation $\mathbf{v} \approx \mathbf{B}\boldsymbol{\beta}$.
2. Formulate the exact primal LP (1.33).
3. Transform the model to (1.34).
4. Solve (1.34) to obtain $\hat{\boldsymbol{\beta}}$.
5. Set $\hat{\mathbf{v}} = \mathbf{B}\hat{\boldsymbol{\beta}}$.
6. Choose $\hat{d} \in \arg\,\text{c-max}_{d \in D^{\text{MD}}}\{\mathbf{r}_d + \lambda \mathbf{P}_d \hat{\mathbf{v}}\}$.
7. Obtain bounds on $\mathbf{v}_\lambda^*$ using (1.28).

The beauty of this approach is that we can easily modify $\mathbf{B}$ to try different approximations and use the bounds in the last step to determine the quality of the approximation[18]. We emphasize that $\hat{\mathbf{v}}$ only equals $\mathbf{v}_\lambda^{d^\infty}$ when $\hat{\mathbf{v}}$ is the optimal value (which it most likely will not be for an arbitrary $\mathbf{B}$ and $\hat{\boldsymbol{\beta}}$.

[18] We believe the idea of using these bounds in the linear programming context is new. Moreover they can be obtained when finding an approximately optimal policy in the previous step.

APLP example

We now illustrate this approach by using linear programming to find approximations in the queuing service rate control model.

Example 1.12. We consider the queuing service rate control model, analyzed frequently in this chapter. Recall that $A_s = \{a_1, a_2, a_3\}$ with $a_1 = 0.2$, $a_2 = 0.4$ and $a_3 = 0.6$, the probability of an arrival $b = 0.2$ and the cost $c(s, a_k) = s^2 + 5k^3$, which is the sum of the delay cost $f(s) = s^2$ and the serving cost per period $m(a_k) = k^3$. We truncate the state space at $N = 50$ and consider an approximation of the form

$$v(s) = \beta_0 + \beta_1 s + \beta_2 s^2.$$

Since we seek to minimize costs, we use formulation (1.32). In this case, the primal approximate LP has 3 variables and 153 constraints.

Choosing $\alpha(s) > 0$ so that $\sum_{s\in S} \alpha(s) = 1$ and letting $M_1 = \sum_{s\in S} \alpha(s)s$ and $M_2 = \sum_{s\in S} \alpha(s)s^2$ the APLP becomes (after considerable algebra):

$$\begin{aligned}
\text{maximize} \quad & \beta_0 + M_1\beta_1 + M_2\beta_2 \\
\text{subject to} \quad & (1-\lambda)\beta_0 - \lambda b\beta_1 - \lambda b\beta_2 \le c(s, a_k), \quad s = 0, k = 1, 2, 3, \\
& (1-\lambda)\beta_0 + (s + \lambda(a_k - b))\beta_1 + (s^2 - \lambda(b + a_k) - 2\lambda(b - a_k)s)\beta_2 \\
& \quad \le c(s, a_k), \quad s \in \{1, \ldots, 49\}, k = 1, 2, 3, \\
& (1-\lambda)\beta_0 + (s + \lambda a_k)\beta_1 + (s^2 - \lambda a_k + 2\lambda a_k s)\beta_2 \\
& \quad \le c(s, a_k), \quad s = 50, k = 1, 2, 3.
\end{aligned}$$

Clearly the above algebra is not necessary, difficult to modify, and error prone. We prefer using the matrix formulation based on the LP formulation in Section **??**. In this case the basis matrices for linear and quadratic approximations are given by

$$\mathbf{B} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ 1 & 50 \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 2^2 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ 1 & 50 & 50^2 \end{bmatrix},$$

respectively. We follow the approach suggested by Algorithm 1.8. Since the goal

is to minimize costs, we represent the approximate LP model by

$$\begin{aligned} \text{maximize} \quad & (\boldsymbol{\alpha}^\mathsf{T}\mathbf{B})\boldsymbol{\beta} \\ \text{subject to} \quad & (\mathbf{AB})\boldsymbol{\beta} \leq \mathbf{r}. \end{aligned}$$

We solve the model using the command *simplex* in R [2021] to obtain $\hat{\boldsymbol{\beta}}^\mathsf{T} = (0, 45)$ for the linear approximation and $\hat{\boldsymbol{\beta}}^\mathsf{T} = (0, 23.2, 8.26)$ for the quadratic approximation.

Figure 1.14a shows the resulting linear and quadratic approximations to the value function. Observe that the linear approximation is inadequate and the quadratic approximation fits well. In both cases, the approximations lie below the true value function. This is because the LP maximizes the greatest lower bound to the value function that satisfies all of the constraints. Moreover the APLP objective function based on the quadratic approximation equalled 7532 compared with the optimal objective function of 7538 obtained from the exact LP.

Figure 1.14b shows the stationary policy obtained using the above algorithm based on the quadratic approximation. It is monotone and only differs from the optimal policy in states 30 and 31 where it chooses actions a_2 instead of a_3.

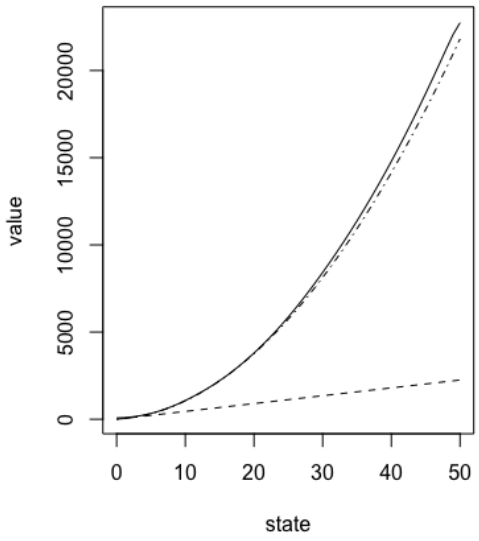
We also note that for this instance, a cubic approximation resulted in identical values to the quadratic approximation because the coefficient of the cubic term was zero. To further explore this issue, we increased λ. For $\lambda = 0.999$, the quadratic and cubic approximations differed, but the greedy policy based on each was the same.

1.6 Application: Advanced Appointment Scheduling

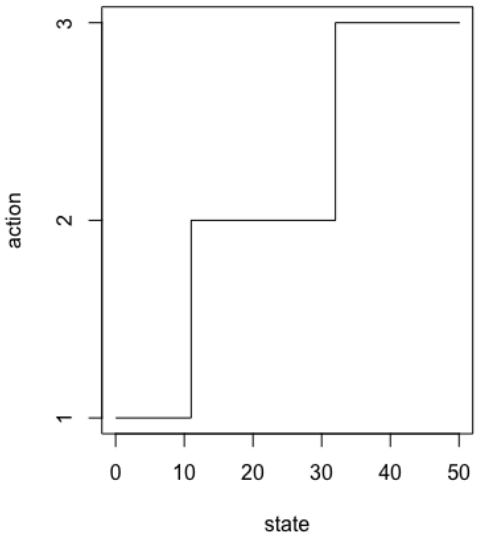
We consider the following simplified version of the advanced scheduling problem in Section **??**. Requests for appointments arrive randomly throughout the current day (today). At the end of the day appointment requests are scheduled for service at some day in the future (tomorrow and subsequent days). We assume the once scheduled, an appointment cannot be rescheduled or delayed.

There are two request types (class 1 and class 2) and two types of service (regular time and overtime). Class 1 appointments must be served the next day (tomorrow) through either regular time or overtime and class 2 appointments can be scheduled for regular time service tomorrow (if space is available), between 2 and N days into the future or through overtime tomorrow[19]. We refer to the quantity N as the booking horizon, no appointments can be assigned to an appointment beyond that day. If a class 1 appointment is scheduled for regular service tomorrow, no cost is incurred, if it

[19]If an appointment is to be scheduled for overtime, it is better to do so tomorrow instead of incurring delay costs.



(a) Linear and quadratic approximations to exact value function (solid line) obtained using APLP.



(b) Stationary policy obtained using quadratic LP approximation.

Figure 1.14: Results of using the approximate LP in the queuing service rate control model with N=50.

is scheduled for overtime, the cost is C. If a class 2 appointment request is scheduled for regular service on day i in the future, the cost is c_i, $i = 1, \ldots, N$[20] and the cost is C if it is served through overtime. It is reasonable to assume c_i is non-decreasing in i and that $C > c_N$[21] Figure 1.15 represents the cost structure symbolically.

The number of class 1 and class 2 appointment requests arriving each day are random and independent. The probability of k class 1 requests is p_k and q_k denotes the probability of k class 2 requests on a day. We represent them by the vectors $\mathbf{p}$ and $\mathbf{q}$. Assume a capacity of M regular time appointments each day, no limit on overtime and at most K arrivals of each class.

Model formulation:

We formulate the model this model as a cost minimization Markov decision process as follows:

[20]Note $i = 1$ corresponds to tomorrow. It seems reasonable to set $c_1 = 0$ but we leave it arbitrary to simplify notation.

[21]Otherwise it would be preferable to use overtime rather than schedule an appointment on or before day N.

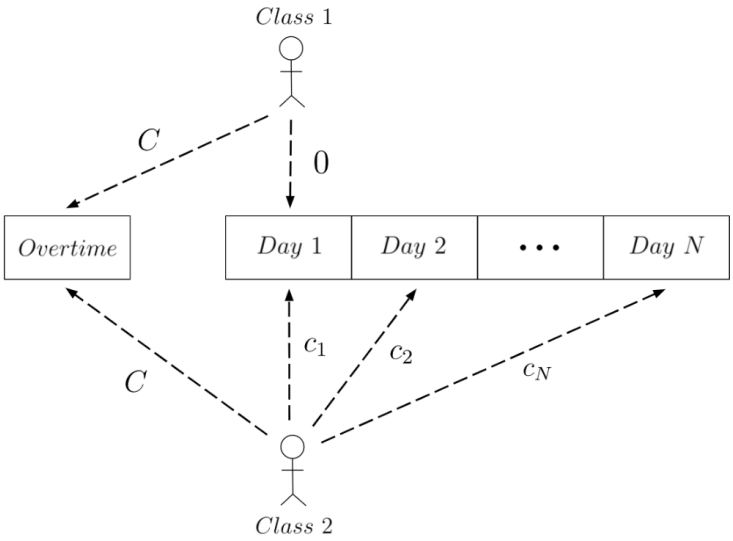


Figure 1.15: Symbolic representation of advanced appointment scheduling problem. Note that Day 1 corresponds to "tomorrow".

Decision epochs: The end of each day so that $T = \{1, 2, \ldots\}$.

States: The state is a vector $(s_1, \ldots, s_N, d_1, d_2)$ where s_i denotes the number of previously booked appointments on days 1 to N and d_j denotes the number of class j appointment requests. Therefore $S = M^N \times K^2$.

Actions: The number of class 2 requests assigned to day n or to overtime. More formally let a_i denote the number of class 2 requests assigned to day i, $i = 1, \ldots, N$ and a_0 denote the number of class 2 appointments assigned to overtime. Then

$$A_{(s_1,\ldots,s_n,d_1,d_2)} = \{(a_0, \ldots, a_N) : 0 \leq a_0,\, 0 \leq a_i \leq M - s_i, i = 1, \ldots, N,\, \sum_{i=0}^{N} a_i = d_2\}. \tag{1.35}$$

We interpret the constraints to mean that you can only book as many class 2 appointments as there are open slots on day i and you have to book all class 2 appoints either through regular time or overtime.[22]

Costs: Number of appointments booked to overtime times C plus the sum of the costs c_i times the number of class 2 appointments booked on day i. That is

$$c((s_1, \ldots, s_N, d_1, d_2), (a_0, \ldots, a_n)) = C(d_1 - (M - s_1))^+ + Ca_0 + \sum_{i=1}^{N} c_i a_i.$$

[22]This is one of the few examples we consider where the action varies with the state. This complicates coding because it requires deriving and storing, or generating action sets as needed.

In this expression, the first term corresponds to overtime charges for class 1 appointments, $M - s_1$ denotes the number of free spots on day 1. If d_1 exceeds this number, then overtime costs are incurred. Note we write $c(s, a)$ instead of $-r(s, a)$.

Transition probabilities:

$$p((s'_1, \ldots, s'_N, d'_1, d_2)' | ((s_1, \ldots, s_N, d_1, d_2), (a_0, \ldots, a_n)) = \begin{cases} p_{d'_1} q_{d'_2} & s'_i = s_{i+1} + a_{i+1}, i = 1, \ldots, N-1, s'_N = 0 \\ 0 \quad \text{otherwise} \end{cases}.$$

Transition probabilities are simpler than they look. When there is a transition from today to tomorrow, tomorrow's state becomes today's state after scheduling appointment requests and state on day N becomes 0 since no appointments were previously scheduled on that day. The only probabilistic term corresponds to arrival of class 1 and class 2 appointment requests today.

We express the Bellman equation for infinite horizon discounted cost minimization as :

$$v((s_1, \ldots, s_N, d_1, d_2)) = \min_{\sum_{i=0}^{N} a_i = d_2; s_i + a_i \leq M, i=1\ldots,N} \Bigg\{ C \sum_{k=0}^{K} p_k (k - (M - s_1))^+ + C a_0 + \sum_{i=1}^{N} c_i a_i + \lambda \sum_{k'=1}^{K} \sum_{k=1}^{K} p_{k'} q_k v((s_2 + a_2, \ldots, s_N + a_N, 0, k', k)) \Bigg\} \tag{1.36}$$

defined for all $(s_1, \ldots, s_N, d_1, d_2) \in S$. The first term in the brackets denotes the expected overtime cost; it is independent of the action and can be moved outside of the minimization. Note that the argument of $v(\cdot)$ inside the "min" represents tomorrow's state after allocating appointment requests that arrived today. The probabilities $p_{k'}$ and q_k only impact the the last two components of the state vector and the actions do not depend on d_1.

> The challenge faced by the scheduler is to determine on which day to assign class 2 demand so as to retain sufficient capacity for future random class 1 demand and at the same time, not waste capacity.

Scheduling in practice presents many challenges, most notably anticipating future demand when scheduling current demand. Schedulers often act myopically and schedule appointments at the earliest possible date instead of trying to reserve capacity to meet future demand. We find below that our ϵ-optimal policies do the latter.

Numerical example: formulation and solutions

We now formulate a small numerical example chosen so that we can find an ϵ-optimal solution easily. We set $N = 3, M = 4, K = 4$ so the model has $5^3 \times 5^2 = 3125$ states. The number of actions in each state varies from 1 to 35[23] so that the number of state-action pairs is large[24]. Clearly increasing any of these quantities to practical levels will result in a model that requires solution by approximation.

We now compare the ϵ-optimal policies and values found with value iteration to those found using LSVI with linear and quadratic approximations. We describe implementation of each approach followed by a comparison of value functions and policies. We solved many instances of the problem but will describe results for one in which $\lambda = 0.95, c_n = 2n, C = 20, \mathbf{p} = (0, 0.2, 0.2, 0.3, 0.3)$ and $\mathbf{q} = (0.3, 0, 0, 0.3, 0.4)$. The arrival probabilities were chosen so that the total expected daily demand exceeded capacity so that strategic scheduling was necessary.

Value iteration– no approximation: Value iteration used the recursion based on the Bellman equation (1.36) since it was the easiest to code[25] and as well converged quickly. For this instance, the iterates of value iteration achieved the stopping criterion $sp(\mathbf{v}' - \mathbf{v}) < .0001$ in 11 iterations. Columns 2 and 3 of Table 1.1 gives actions and values obtained using value iteration for selected states.

State $(s_1, s_2, s_3, d_1, d_2)$	ϵ-optimal action	Value at termination	Linear approx. action	Linear approx. value
(2,1,2,4,2)	(0,0,1,1)	63.39	(0,0,0,2)	66.06
(2,3,0,2,4)	(0,0,0,4)	46.64	(0,0,0,4)	48.11
(1,1,0,2,4)	(0,1,1,2)	28.98	(0,1,0,3)	31.45
(1,2,2,2,3)	(0,1,0,2)	30.58	(0,1,0,2)	33.09
(0,0,0,4,4)	(0,0,2,2)	30.98	(0,0,0,4)	33.04
(0,0,0,2,4)	(0,2,1,1)	19.22	(0,2,0,2)	19.82

Table 1.1: This table gives results for the value iteration solution (columns 2 and 3) and that found using LSVI and a linear value function approximation (columns 4 and 5) for the instance described in the text. The first component of the action, a_0, corresponds to class 2 demand assigned to overtime and the next three components of the action (a_1, a_2, a_3) represent the number of units of class 2 demand scheduled on day $i = 1, 2, 3$. Columns 4 and 5 provide the action and value obtained using the linear value function approximation (1.37)

LSVI - linear approximation: We now describe our approach to finding a so-

[23]In state $(0, 0, 0, 0, 4)$ there are 35 actions and in any state with $d_2 = 0$ or no available capacity, there is only one action.

[24]We leave it as an exercise to determine the exact number of actions.

[25]We found generating the set of actions for each state and manipulating arrays challenging to code. Complexities included a "feature" of R [2021] which converted one-dimensional matrices to vectors

lution based on approximating the value function with a linear architecture and basis functions that are linear in the state. That is, we set

$$\hat{v}((s_1, s_2, s_3, d_1, d_2)) = \beta_0 + \beta_1 s_1 + \beta_2 s_2 + \beta_3 s_3 + \beta_4 d_1 + \beta_5 d_2. \tag{1.37}$$

The advantages of using this approximation are:

1. It is easy to obtain parameter estimates from any linear regression routine embedded in the language used to code the algorithm.

2. The number of quantities to determine is reduced from the 3125 components of $\mathbf{v}$ (in its tabular representation) to the 6 components of $\boldsymbol{\beta}$. Moreover we need only store the 6-dimensional vector $\hat{\boldsymbol{\beta}}$ instead of an array representing the 3125 components of $\mathbf{v}$.

3. The parameters estimates provide insight into the effect demand and capacity has on cost. In (1.37), $\beta_i, i = 1, 2, 3$ represents the *marginal cost* of one unit less remaining capacity on day i, and β_4 and β_5 represent the marginal costs of an extra unit of Class 1 and Class 2 demand.

4. Greedy actions corresponding to the approximate value function can be easily determined.

5. Parameter estimates for this approximation can also be found using linear programming.

We now describe our approach to solving this model using LSVI[26]. We initialized the algorithm with $\boldsymbol{\beta} = \mathbf{0}$ and set $\epsilon = .0001$. We implemented steps 2a. and 2b. state by state instead of in vector form. Step 2a. used representation (1.37) directly instead of deriving the matrix $\mathbf{B}$. The right hand side of 2b. is given by

$$\min_{(a_0,a_1,a_2,a_3)\in A_{(s_1,s_2,s_3,d_1,d_2)}} \Big\{ c((s_1, s_2, s_3,d_1.d_2), (a_0, a_1.a_2, a_3)) + \lambda \sum_{j=0}^{4}\sum_{k=0}^{4} p_j q_k \hat{v}((s_2 + a_2, s_3 + a_3, 0, j, k)) \Big\} \tag{1.38}$$

where the summation in the brackets reduces to

$$\begin{aligned}
&\sum_{j=0}^{4}\sum_{k=0}^{4} p_j q_k \hat{v}((s_2 + a_2, s_3 + a_3, 0, j, k)) \\
&\quad = \sum_{j=0}^{4}\sum_{k=0}^{4} p_j q_k [\beta_0 + \beta_1(s_2 + a_2) + \beta_2(s_3 + a_3) + \beta_3 0 + \beta_4 j + \beta_5 k] \\
&\quad = \beta_0 + \beta_1(s_2 + a_2) + \beta_2(s_3 + a_3) + \beta_4 E(D_1) + \beta_5 E(D_2)
\end{aligned} \tag{1.39}$$

[26]We chose LSVI instead of LSPI because it was simpler to code and converged quickly.

and the random variables D_1 and D_2 denote the expected demand of class 1 and class 2 appointments respectively. Since several of the expressions in (1.38) do not involve the action explicitly, the greedy action based on the linear approximation chooses

$$(\hat{a}_0, \hat{a}_1, \hat{a}_2, \hat{a}_3) \in \underset{(a_0,a_1,a_2,a_3)\in A_{(s_1,s_2,s_3,d_1,d_2)}}{\arg\min} \left\{ Ca_0 + c_1 a_1 + (c_2 + \lambda\beta_1)a_2 + (c_3 + \lambda\beta_2)a_2 \right\} \tag{1.40}$$

We solved the model using the representation in (1.38) to allow easy generalization to other forms for $\hat{v}(\cdot)$. LSVI converged in 13 iterations. Curiously, convergence of the quantity $\|\boldsymbol{\beta}' - \boldsymbol{\beta}\|_2$ was not monotone; at one iteration the norm the difference $\boldsymbol{\beta}' - \boldsymbol{\beta}$ was greater than at the previous iteration. This is consistent with the observation above that the LSVI operator need not be a contraction mapping.

We obtained the estimated linear approximation:

$$\hat{v}((s_1, s_2, s_3, d_1, d_2)) = -43.98 + 12.59s_1 + 6.54s_2 + 4.28s_3 + 12.60d_1 + 8.98d_2. \tag{1.41}$$

Observe that all coefficients (except the constant) are positive meaning that the greater the occupancy or demand, the greater the expected discounted cost. More importantly, the expected discounted cost of an extra unit of occupancy decreases from day 1 to day 3 and an extra unit of class 1 demand contributes more to the expected discounted cost than a unit of class 2 demand.

We now determine the structure of greedy policies based on the linear approximation. Recall that the maximum capacity each day is $M = 4$ units, overtime cost $C = 20$ and the cost of scheduling a unit of class 2 demand on day $k = 1, 2, 3$ is $2k$. Substituting the parameter estimates in (1.41) into (1.39) implies that we choose

$$(\hat{a}_0, \hat{a}_1, \hat{a}_2, \hat{a}_3) \in \underset{(a_0,a_1,a_2,a_3)\in A_{(s_1,s_2,s_3,d_1,d_2)}}{\arg\min} \{20a_0 + 2a_1 + 10.21a_2 + 10.06a_2\}. \tag{1.42}$$

where $10.21 = 4 + 0.95 \times 6.54$ and $10.06 = 6 + 0.95 \times 4.28$ and the action set can be expressed as

$$A_{(s_1,s_2,s_3,d_1,d_2)} = \{0 \le a_0; 0 \le a_1 \le (4 - s_1 - d_1)^+; 0 \le a_k \le 4 - s_k, k = 2, 3; \sum_{k=0}^{3} a_k = d_2\}. \tag{1.43}$$

Observe further that the "arg min" only involves the current state through its constraints.

From (1.42) and (1.43) we easily obtain the **structure** of greedy action based on the linear approximation.

> In each state in which $d_2 > 0$ assign as much demand as possible to day 1, then assign as much as possible to day 3, then assign as much as possible to day 2. Only assign demand to overtime when there is no other capacity available.

This policy can be summarized succinctly as: after using up capacity on day 1, back fill capacity from the end of the booking horizon. Note that this characterization avoids the need to compute the optimal action in each state although we do so in Table 1.1 below for comparison purposes.

LSVI - quadratic approximation: We also approximated the value function with a quadratic model of the form:

$$\begin{aligned}\hat{v}((s_1, s_2, s_3, d_1, d_2)) =& \beta_0 + \beta_1 s_1 + \beta_2 s_2 + \beta_3 s_3 + \beta_4 d_1 + \beta_5 d_2 \\ &+ \beta_6 s_1^2 + \beta_7 s_2^2 + \beta_8 s_3^2 + \beta_9 d_1^2 + \beta_{10} d_2^2.\end{aligned}$$

A more general quadratic model would also include 10 additional cross product terms of the form $s_i s_j, s_i d_j$ or $d_1 d_2$. Since the model is linear in the parameters, they can be easily obtained using linear regression code. In this case the marginal costs vary with the state vector. For example the marginal cost of an additional unit of class 2 demand would be $\beta_5 + 2\beta_{10} d_2$. Again we found that LSVI converged in 13 iterations and the norms of the differences of successive parameter estimates were not monotone.

We obtained the estimated value function:

$$\begin{aligned}\hat{v}((s_1, s_2, s_3, d_1, d_2)) = -34.93 &+ 5.93 s_1 + 3.55 s_2 + 1.92 s_3 + 5.93 d_1 + 5.07 d_2 \\ &+ 1.68 s_1^2 + 0.85 s_2^2 + 0.70 s_3^2 + 1.68 d_1^2 + 1.04 d_2^2. \quad (1.44)\end{aligned}$$

Following our discussion above, the quantity in the the "argmin" in (1.38) after some algebra becomes

$$20 a_0 + 2 a_1 + [7.37 a_2 + 0.81 (s_2 + a_2)^2] + [7.82 a_3 + 0.66 (s_3 + a_3)^2]. \quad (1.45)$$

From this expression we can easily determine the greedy action in any state. To do so we would minimize this expression over the set of feasible actions. For example, in state $(2, 1, 1, 2, 1)$ there are three feasible actions given in Table 1.2. Evaluating the above expression for each action, we find that the greedy action is $(0, 0, 1, 0)$; that is, assigning the single unit of class 2 demand to day 2. This contrasts with the action based on the linear approximation which would assign this unit of class 2 demand to day 3.

Action	Value of (1.45)
$(1, 0, 0, 0)$	20
$(0, 0, 1, 0)$	11.21^*
$(0, 0, 0, 1)$	11.27

Table 1.2: Choosing greedy action in state $(2, 1, 1, 2, 1)$ based on quadratic approximation to the value function.

Numerical example: comparison of policies and values

We first describe the **structure** of the ϵ-optimal policy. The three left most columns of Table 1.1 give the action chosen by the ϵ-optimal policy and its value in some selected high demand states. Observe that in these states, the policy does not assign class 2 demand to the earliest day possible, instead it **reserves** capacity for future class 1 demand. For example, in state $(2, 1, 2, 4, 2)$, all day 1 capacity is used by class 1 demand and the two units of class 2 demand are split between days 2 and 3. In state $(0, 0, 0, 4, 4)$, all class 1 demand is served on day 1 and the 4 units of Class 2 demand are split between days 2 and 3. Moreover we observe that Class 2 demand is never served with overtime.

Columns 4 and 5 of Table 1.1 show the action obtained using the linear approximation[27] and the value of $\hat{v}$. Observe that the several of these states, the action based on the linear approximation differs from that using the ϵ-optimal policy. In cases where it differed, it assigned the class 2 demand to later appointments[28] For example in state $(0, 0, 0, 4, 4)$ the policy chosen using the linear approximation assigned **all** class 2 demand to day 3 while the ϵ-optimal policy split it between days 2 and 3. In total the action based on the linear approximation differed from that chosen by the ϵ-optimal policy in 632 out of 3125 states.

Remarkably we found that the actions chosen using a quadratic approximation were **identical** to those corresponding to the ϵ-optimal policy. Moreover the RMSE of the difference between the ϵ-optimal value and that found using a linear approximation was 1.99 and the RMSE of the difference between the ϵ-optimal value and that found using a quadratic approximation was 1.11. Hence the quadratic approximation fit considerably better and might be appropriate in larger instances of the model.

Numerical example - class specific overtime costs

We noted above that Class 2 demand is never served with overtime. To test whether overtime was ever used strategically, we assigned different overtime costs to each class. With the overtime cost for Class 1 equal 40 and the overtime cost for Class 2 equal to 10, we found that the ϵ-optimal policy in state $(2, 3, 0, 2, 4)$ became $(0, 0, 2, 2)$ instead of $(0, 0, 0, 0, 4)$, that is it assigned 2 units of Class 2 demand to day 3 and 2 units to overtime. When using the linear approximation with these costs, the corresponding action in this state remained $(0, 0, 0, 0, 4)$ but the quadratic approximation again identified the optimal action.

---

[27] This action is chosen greedily according to $\arg\min_{a \in A_s} \left\{ c(s,a) + \lambda \sum_{j_i n S} p(j|s,a)\hat{v}(j) \right\}$.

[28] This observation is similar to the policy found by Patrick et al. [2008] using linear programming with a linear approximation to the value function

Numerical example - state sampling

In the above analysis, LSVI approximations were based on using the entire state space in all calculations. Since the linear approximation had 6 parameters, the quadratic approximation had 11 parameters, the model had 3125 states and there was no variability, it seemed reasonable good results were possible if LSVI was applied to a subset of states. More importantly, in larger models obtaining approximations by updating the above recursion in all states would be computationally prohibitive.

The advantage of state sampling is that it reduces the number of times that the Bellman operator:

$$L\hat{v}(s) = \min_{a \in A_s} \{c(s,a) + \lambda \sum_{j_S} p(j|s,a)\hat{v}(j)\}$$

needs to be evaluated at each iteration of LSVI.

We adopted the following offline approach to applying LSVI on a subset of states. In our computations we selected a sample subset by:

1. specifying the fraction of states to sample,
2. randomly sample the specified number of states,
3. discarding duplicates[29] and
4. apply LSVI on the designated sample.

Hence our approach selects a random subset of states and use it at all iterations of LSVI. The next two chapters consider online versions of these algorithms which sample the states sequentially and avoid computing the quantity $\sum_{j_S} p(j|s,a)\hat{v}(j)$.

We used the quadratic approximation and applied LSVI a large number of samples varying the fraction of states sampled between 0.1 to 0.9. We compared the quality of fits based on

$$RMSE = (\frac{1}{11}\sum_{k=0}^{10}(\hat{\beta}_k - \hat{\beta}_k^{all})^2)^{.5}$$

where β_k^{all} denotes the parameter estimates based on using all states. Results of this process are displayed in Figure 1.16. It shows as expected, that the quality of the fits improves with the sample size and that there is considerable variability between samples. We further observed, as illustrated in the sample below, that relative magnitude of parameter estimates was similar in the sample to that in the full model.

The output of a typical sample based on half of the states was

$$\begin{aligned}\hat{v}((s_1,s_2,s_3,d_1,d_2)) = -36.86 &+ 5.61s_1 + 3.75s_2 + 2.21s_3 + 7.21d_1 + 5.12d_2 \\ &+ 1.73s_1^2 + 0.80s_2^2 + 0.70s_3^2 + 1.37d_1^2 + 1.03d_2^2. \quad (1.46)\end{aligned}$$

[29]Since there is no variability other than sampling,each replication of state s sample would yield the same value for $L\hat{v}(s)$ it was unnecessary to evaluate the same state more than once. In a simulation approach where the subsequent state is sampled, this would not be the case.

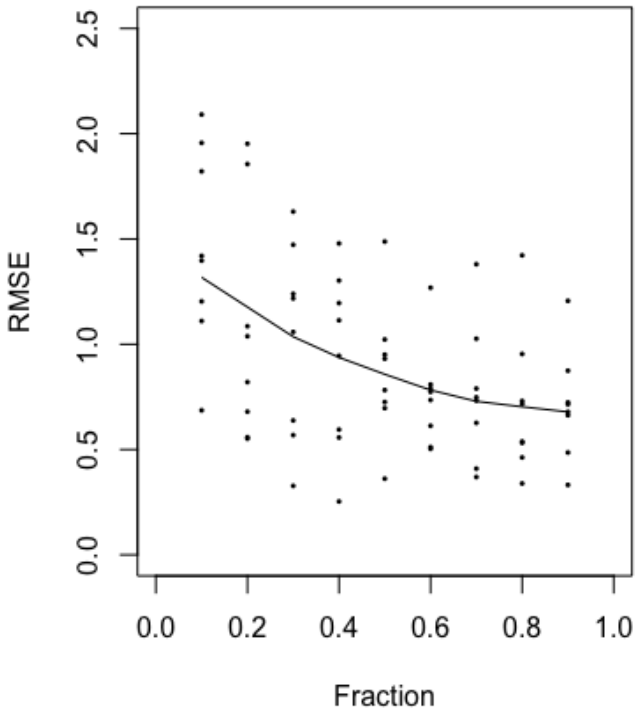


Figure 1.16: RMSE as a function of the fraction of states sampled over multiple state samples. The dots correspond to a specific sample and the smooth line to a lowess smooth of the individual runs.

This is similar to the estimates to the full model. We leave it as an exercise to compare the policy obtained using this sample and that using LSVI on all states.

Numerical example: Neural network approximation

Obtaining good results using a neural network approximation proved challenging. We used the package "neuralnet" in R [2021]. When incorporated in LSVI, the algorithm either oscillated and did not satisfy a stopping criterion or the parameter estimation algorithm did not converge. We explored several alternatives. Most notably, when we fit the optimal value function on the entire state space with several different neural network structures, the greedy policy was similar to that found with a linear value function approximation so it was not optimal. We conclude that in this application neural networks **do not** provide an attractive approximation architecture.

Approximate Linear programming*

We chose not to provide an analysis based on linear programming here. The main challenge was to construct constraint matrices. This model has been analyzed in considerable detail in Patrick et al. [2008]. Therein the linear program was solved using column generation on the dual which contained few rows and many columns. The coefficients for the linear and quadratic approximations were obtained from the corresponding primal solution. This work identified the same greedy policy as described above for the LSVI solution of the linear approximation. Moreover the parameter estimates obtain using LSVI and linear program were different and the LP solution for the quadratic app gave identical results to linear approximation in contract to what we observed above.

1.7 Application: Approximating value functions from data - Offline training*

In some settings, we may not explicitly know what policy is being used or the precise transition probabilities but instead have observed an initial state and a sequence of rewards from multiple realizations (or replications) of a decision making process. Such a situation is often referred to as offline training. To illustrate this we will consider a case study drawn from the game of golf. We model each hole played as an infinite horizon Markov decision process under the expected (undiscounted) total reward criterion.

1.7.1 Value functions and strokes gained in golf

This section describes dynamic programming based concepts developed by Broadie [2014] and now widely used in all levels of competitive golf.

Golf as a Markov decision process

In the game of golf, a player (a.k.a. golfer) strives to hit a 1.68 inch diameter golf ball into a 4.25 inch diameter cup (often ambiguously called "a hole") in as few as shots as possible. A golf course usually consists of 18 holes, each made up of a tee box, a green, a fairway, rough and sand traps, and woods, desert or water to penalize errant shots. Cups are located on greens. The distance from tee to the cup usually ranges between 100 and 600 yards. The first shot is played from the tee and subsequent shots are taken from the fairway, rough, sand trap, green or from close to an obstruction such as a tree or cactus[30]. When a ball is lost or unplayable, a penalty stroke is incurred.

Playing a hole of golf may be viewed as a Markov decision process as follows.

[30]Shots from an obstructed location are often referred to as *recovery shots*.

Decision epochs: The instance before a shot is hit:

$$T = \{1, 2, \ldots\}.$$

Note we use an infinite horizon formulation because the number of decision epochs to play a hole of golf is not fixed; in theory it could take an arbitrarily large number of strokes to hit the ball into the hole from any location of the golf course. In reality this is not supported by data but often it might seem like the case to a frustrated golfer[31].

States: The state represents the distance from the hole $\delta \in \mathcal{D}$ and the terrain type (fairway, rough, etc.) $\tau \in \mathcal{T}$ where the ball lies immediately before a shot is taken. Note that the state $(0, \text{green})$ is the zero-reward absorbing state corresponding to the ball being "In the hole". Therefore,

$$S = \{\mathcal{D} \times \mathcal{T}\}.$$

Actions: An action corresponds to the type of shot (intended target and trajectory) a player tries to hit.

$$\begin{aligned} A_{(\delta,\tau)} &= \{\text{All possible targets } (\delta', \tau') \text{ and shot types from } (\delta, \tau)\} \\ A_{(0,\text{green})} &= \{a_0\} \end{aligned}$$

where a_0 means "do nothing". Elements of $A_{(\delta,\tau)}$ can be thought of as capturing the choice of a golfer to aim the ball in a certain direction, to hit the ball at a certain speed and angle, and to attempt to give the shot a particular curvature. Note we do not specify the shots types explicitly because they vary greatly with the skill of the player and the terrain. For example, if the player is behind a tree then he/she may hit a recovery shot. If the player is on the green, he/she will putt. In the fairway, the player may pick a target 10 feet short of the hole and attempt to hit a high draw[32].

Transition probabilities: The transition probabilities $p((\delta', \tau')|(\delta, \tau), a)$ are determined from a probability distribution of shot outcomes given the target and shot type. Since $(0, \text{green})$ is absorbing, $p((0, \text{green})|(0, \text{green}), a_0) = 1$.

The assumption that the transition probability depends only on the current state and action makes sense since given the location of the ball and its terrain, the outcome of the current shot should not depend on the outcome of previous shots (unless of course the golfer is really frustrated after hitting a bad shot and mood effects execution of future shots).

[31] In men's professional (PGA) golf, the highest recorded hole score to date is 16 in 2011 by Kevin Na at the Valero Open. From a handicap perspective the maximum number of strokes allowable on a hole is determined by a complex formula so when that number of strokes is reached, the player is encouraged to pick up his/her ball and stop playing the hole.

[32] A shot that curves from the right to the left for a right-handed golfer.

Rewards: It requires one stroke to hit a shot so that: $r((\delta,\tau),a,(\delta',\tau')) = -1$ for all $(\delta,\tau) \in S$ and $r((0,\text{green}),a_0,(0,\text{green})) = 0$. Because the objective in golf is to to minimize the number of shots on each hole, the reward represents the "cost" of a shot. Since this is an infinite horizon formulation, (0,green) represents the zero-cost (reward-free) absorbing state. Note that it is preferable to model this as a minimum cost problem in which case we replace -1 by 1 and $r(\cdot,\cdot)$ by $c(\cdot,\cdot)$.

Value Functions in Golf

We define the value of each location as the expected number of shots to "hole out", that is get the ball into the cup, from that location. Both the distance and terrain affect this value. We will take value functions to be non-negative, that is

$$v(\delta,\tau) = \text{ the expected number of strokes to hole out from location } \delta \text{ and terrain } \tau.$$

For example, for an average male pro (PGA) golfer $v(100,\text{fairway}) = 2.80$ and $v(100,\text{rough}) = 3.02$ (See Broadie [2014] p.85.) This means that the expected number of shots for an average male professional golfer to hole out from 100 yards in the fairway is 2.80 and from 100 yards in the rough is 3.02. Comparing these values shows that on average a player adds 0.22 shots if he misses the fairway and ends up in the rough. Since a player is faced with many shots throughout a four-round[33] tournament, missing the fairway several times can have a significant impact on the player's final score.

The value function can be overlaid on a picture of a hole to show the expected number of shots from each location. (See Figure 1.17). Such a figure can also guide player decision making. Knowing this value from every distance and terrain will be fundamental to the development of a performance metric referred to as *strokes gained*, which we will describe below.

A Case Study

The following case study is based on a collaboration between author Puterman and the University of British Columbia (UBC) women's golf team. The objective of this study was to establish reference values (value functions) for female golf team members. Unlike the case of male professionals, such information was not available elsewhere at the time of the study.

We describe the approach we used to estimate values of shots from the fairway for distances between 10 and 350 yards from the hole. Note that the data contained shots from all terrains however we restrict our analysis to shots from the fairway for illustrative purposes. Therefore we eliminate τ from the state description for the remainder of this section.

[33] A round usually consists of playing 18 holes.

Figure 1.17: Graphical representation of golf value functions on a typical golf hole. For example, on average it requires 2.5 shots to hole out from anywhere on the red arc near the green.

The objective of this analysis is to estimate $\hat{v}(\delta)$; the expected number of shots to hole out from distance δ in the fairway.

The total data set, which contained the starting location and outcome of 10,100 shots from the 2016-2017 golf season, was entered by team members in an application we designed for that purpose. It was manipulated to create a data set consisting of a large number of pairs of distances and number shots to "hole out" from those distances. For example, the data point $(\delta, v) = (200, 4)$ means for a particular player, on a particular hole in a particular round it required 4 shots to "hole out" from 200 yards on the fairway.

We represented the state in terms of basis functions $b_0(\delta) = 1, b_1(\delta) = \delta, b_2(\delta) = \delta^2$ and $b_3(\delta) = \delta^3$. This means that we approximated the value function by a cubic polynomial. In contrast to other examples in this chapter, the model needed to take

into account random variability. Hence we modelled a data point by

$$v(\delta) = \beta_0 + \beta_1\delta + \beta_2\delta^2 + \beta_3\delta^3 + \epsilon \tag{1.47}$$

where ϵ represented an unobservable random disturbance.

Our purpose for developing an approximation using basis functions instead of a tabular representation for $\hat{v}(\delta)$ based on the average of shots from that distance was to account for data sparsity and to develop a value function approximation that was increasing in distance. For some distances, there were many records while for others there were none. Moreover it made sense that the further from the hole, the greater the number of strokes it should take to hole out so that $v(s)$ should be non-decreasing in s.

For the data described above we obtain the following fitted model:

$$\hat{v}(\delta) = 2.53 + 0.0054\,\delta + 0.0000076\,\delta^2 - 0.000000019\,\delta^3 \tag{1.48}$$

We display the data together with the fitted model graphically in Figure 1.18. Note that fitting more complicated models had little effect on estimated values.

Based on the fitted model we estimated that $\hat{v}(100) = 3.03$, which is 0.23 higher than for a male professional golfer from that location or equivalent to the value for a male professional in the rough at 100 yards.

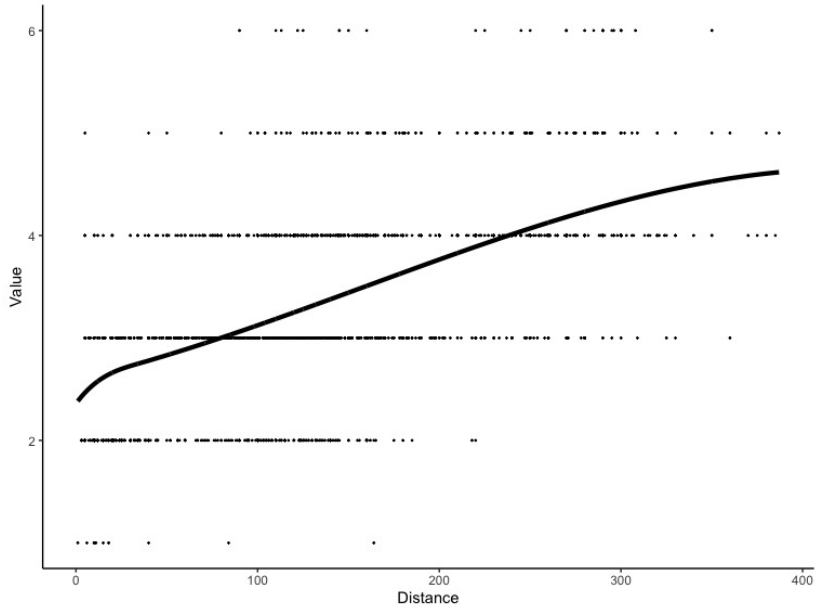


Figure 1.18: Data and fitted model for (1.48). The data values are discrete since they represented the number of shots required to "hole out" from each distance on the fairway. Note that the fitted function is monotone increasing and approximately linear between 50 and 300 yards.

Using value functions: strategy

In the context of the Markov decision process framework, we can use value functions to decide on strategy. Strategic choices in golf include aim point and shot type. Outcomes are impacted by shot dispersion patterns as modeled by probability distributions.

As an example, consider the option of a shot from the fairway when the hole is 240 yards away and there is a water hazard in front of the green. Suppose the player considers two choices. Choice 1 is to "go for the green", which means to be aggressive and get the ball on to the green in one shot. Choice 2 is to "lay up", which means to be conservative and hit a shorter shot that avoids the water, but will require a subsequent shot to get on the green.

To simplify exposition, suppose that under Choice 1 if the player aims at the green and lands on it, the ball ends up 50 feet from the hole which we assume occurs with probability p. The value from that point would be $v(50, \text{green}) = 2.14$[34]. However if the ball ends up in the water, which occurs with probability $1 - p$, the player incurs a one stroke penalty and must hit his next shot from 60 yards away from the green in the the fairway where the value is $v(60, \text{fairway}) = 2.70$. Thus the expected value of this alternative would be

$$v_1 = p \times 2.14 + (1 - p) \times (2.70 + 1) = 3.70 - 1.56p.$$

Alternatively under Choice 2 if the player aims at a target 80 yards from the green where there is a 0.9 chance of being near that target where $v(80, \text{fairway}) = 2.75$ and a 0.1 chance of being in the rough at the same distance where $v(80, \text{rough}) = 2.96$. The expected value of this outcome is $v_2 = 2.77$. Thus the decision to go for the green is optimal if $v_1 \leq v_2$ that is when $3.70 - 1.56p \leq 2.77$ or when $p \geq 0.6$.

Hence if the player thinks there is greater than a 60% chance of hitting the green, then he[35] should go for it.

Using value functions: strokes gained*

Strokes gained measures the difference between the expected value and the result of a shot taken from a specific location. Let

$(X_D, X_T) =$ Random variables denoting the location of the ball after the current shot
$(\delta', \tau') =$ Actual location of the ball after the current shot
$(\delta, \tau) =$ Actual location of the ball immediately prior to the current shot.

Broadie [2014] defines strokes gained, SG, to be

$$\text{SG}(\delta, \tau) = E_{(\delta,\tau)}[v(X_D, X_T)] - v(\delta', \tau'). \tag{1.49}$$

The Bellman equation expressed in terms of random variables and costs becomes

[34]Distance on the green are measured in feet and distances from other terrain are measured in yards.
[35]The calculations above use men's professional values from p.85 of Broadie [2014].

$$v(\delta, \tau) = 1 + E_{(\delta,\tau)}[v(X_D, X_T)] \tag{1.50}$$

So rearranging terms and substituting (1.50) into (1.49) yields

$$\mathrm{SG}(\delta, \tau) = v(\delta, \tau) - (v(\delta', \tau') + 1) \tag{1.51}$$

The first quantity on the right hand side of (1.51) denotes the expected number of shots to hole out prior the current shot. The second quantity is the expected number of shots to hole out from where the current shot ended up plus the "cost" of 1 stroke to reach this point.

Thus, if the player hits an "average" shot, $v(\delta, \tau) = v(\delta', \tau') + 1$ and the strokes gained will be zero. If the player hits a better than average shot, $v(\delta', \tau') + 1$ will be less than $v(\delta, \tau)$ and the strokes gained will be positive. If the shot outcome is worse than average, strokes gained will be negative.

We illustrate this concept using the fitted curve (1.48). Suppose a woman's golf team member is at 275 yards in the fairway, then $v(275, \text{fairway}) = 4.20$. After hitting an average shot from this location, her strokes gained would be $4.20 - 1 = 3.20$. This is equivalent to a distance of 112 yards in the fairway, so an average shot to the fairway would be $275 - 112 = 163$ yards. If she hits the ball to 75 yards in the fairway, we find a value of 2.97, corresponding to a strokes gained of $4.20 - 2.97 - 1 = 0.23$. If instead she hits the ball to 125 yards in the fairway, the value is 3.29 corresponding to a strokes gained of -0.09. Note that hitting the ball into the rough or a sand trap at 125 yards would result in more negative stokes gained.

Accumulated over several rounds, a player's strokes gained tabulated by terrain and distance indicate that player's strengths and weaknesses. Hence, a practice plan can be developed to reinforce strengths and improve weaknesses. This was the objective of the UBC golf team project.

1.8 Appendix - Regression and nonlinear optimization

In order to understand value function approximation, we regard it essential to be familiar with some basic concepts of linear regression.

1.8.1 Linear regression

The basic problem can be described as follows. Given N observations of a dependent variable y_i and independent variables $x_{i,1}, x_{i,2}, \ldots, x_{i,M}$ for $i = 1, \ldots, N$, the objective of linear regression is to find a set of *regression coefficients*, *weights* or *parameters* β_j $j = 1, \ldots, M$ so that

$$f(x_{i,1}, x_{i,2}, \ldots, x_{i,M} | \beta_1, \ldots, \beta_M) := \beta_1 x_{i,1} + \beta_2 x_{i,2} + \ldots + \beta_M x_{i,M} \tag{1.52}$$

well approximates y_i for $i = 1, \ldots, N$. This is referred to as a *linear* regression problem because $f(x_{i,1}, x_{i,2}, \ldots, x_{i,M})$ is a linear function of the parameters.

The standard approach[36] to estimation is to obtain parameters that minimize the sum of squared error function

$$g(\beta_1, \ldots, \beta_M) := \sum_{i=1}^{n} (y_i - (\beta_1 x_{i,1} + \beta_2 x_{i,2} + \ldots + \beta_M x_{i,M}))^2 \tag{1.53}$$

$$= \sum_{i=1}^{n} g_i(\beta_1, \ldots, \beta_M)^2 \tag{1.54}$$

where

$$g_i(\beta_1, \ldots, \beta_M) := y_i - (\beta_1 x_{i,1} + \beta_2 x_{i,2} + \ldots + \beta_M x_{i,M})$$

for $i = 1, \ldots, N$.

We introduce the function $g_i(\cdot)$, which is often referred to as the *ith residual*[37] to provide more elegant expressions and simplify the derivation of Gauss-Newton iteration below. Note that this notation expresses dependence of observation $(y_i; x_{i,1}, \ldots, x_{i,M})$ through the subscript i.

Parameters chosen so as to minimize $g(\beta_1, \ldots, \beta_M)$ are referred to as *least squares estimates*. Using a matrix formulation leads to a closed form representation. Parameter estimates are often denoted by $\hat{\beta}_j, j = 1, \ldots, M$.

Note that the statistical literature formulates the regression problem in the context of the statistical model

$$y_i = \beta_1 x_{i,1} + \beta_2 x_{i,2} + \ldots + \beta_M x_{i,M} + \epsilon_i \tag{1.55}$$

where ϵ_i represents an *unobservable* random disturbance (or error) drawn from a distribution with mean 0 and constant variance σ^2. Furthermore, it is usually assumed that $E[\epsilon_i \epsilon_j] = 0$ for $i \neq j$, that is, that the random disturbances are uncorrelated between observations. When in addition, the random disturbances are assumed to be normally distributed, the least squares estimates of β_j $j = 1, \ldots, M$ are also maximum likelihood estimates and consequently have many theoretical properties that allow for statistical inference, such as the computation of confidence intervals and hypothesis testing. Usually, these statistical concepts are not considered in approximate dynamic programming.

When the errors are assumed to be normal, the above model is often expressed as

$$y_i \sim NID(\beta_1 x_{i,1} + \beta_2 x_{i,2} + \ldots + \beta_M x_{i,M}, \sigma^2) \tag{1.56}$$

where the expression $NID(\mu, \sigma^2)$ corresponds to a set of independent normally distributed random variables with mean μ and variance σ^2. Note that in the regression

[36] Other choices for $g(\cdot)$ can be used including sums of absolute values or other weighting functions that may down weight outlying observations.

[37] To be rigorous, a residual refers to the situation when the parameters have been replaced by estimates.

model, the mean varies from observation to observation and the variance is constant. The beauty of writing the model this way is that it allows us to generalize by changing the distribution (for example Poisson or binomial) or assumptions about the variance (auto-correlated or non-constant).

1.8.2 Matrix Formulation

The most elegant approach for analyzing regression models is through its matrix formulation. It is widely used in approximate dynamic programming and reinforcement learning, and leads to elegant formula for predicted values and parameter estimates. We introduce the following vectors and matrices:

$$\mathbf{y} := \begin{bmatrix} y_1 \\ \cdot \\ \cdot \\ \cdot \\ y_N \end{bmatrix}, \mathbf{X} := \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,M} \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ x_{N,1} & x_{N,2} & \dots & x_{N,M} \end{bmatrix}, \boldsymbol{\beta} := \begin{bmatrix} \beta_1 \\ \cdot \\ \cdot \\ \cdot \\ \boldsymbol{\beta}_M \end{bmatrix} \text{ and } \boldsymbol{\epsilon} := \begin{bmatrix} \epsilon_1 \\ \cdot \\ \cdot \\ \cdot \\ \epsilon_N \end{bmatrix}. \quad (1.57)$$

In (1.57), $\mathbf{y}$ denotes the $N \times 1$ column vector of observations of the dependent variable, $\mathbf{X}$ is $N \times M$ matrix of values of the independent variables, $\boldsymbol{\epsilon}$ denotes the $N \times 1$ column vector of random disturbances and $\boldsymbol{\beta}$ is the $M \times 1$ column vector of parameters. We also find it convenient to denote the ith row of $\mathbf{X}$ by the vector $\mathbf{x}_j$, that is

$$\mathbf{x}_j = (x_{j,1}, \dots, x_{j,M})$$

for $j = 1, \dots, N$.

Using this matrix notation, the data generating model, which provides an elegant way of representing (1.55) for all N in a single equation, is given by

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \quad (1.58)$$

where $E[\boldsymbol{\epsilon}] = \mathbf{0}$ and $\boldsymbol{\Sigma} := \text{cov}[\boldsymbol{\epsilon}] = \sigma^2\mathbf{I}$.

The squared error criterion can be expressed in matrix form as

$$g(\boldsymbol{\beta}) = \sum_{i=1}^{n} g_i(\boldsymbol{\beta})^2 = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\mathsf{T}(\mathbf{y} - \mathbf{X}\beta) = \mathbf{y}^\mathsf{T}\mathbf{y} - 2\boldsymbol{\beta}^\mathsf{T}\mathbf{x}^\mathsf{T}\mathbf{y} + \boldsymbol{\beta}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\boldsymbol{\beta}. \quad (1.59)$$

Note that $g(\boldsymbol{\beta})$ is often written as

$$g(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2. \quad (1.60)$$

Computing the gradient of $g(\boldsymbol{\beta})$ and setting it equal to zero shows that $\hat{\boldsymbol{\beta}}$, the least squares estimator of $\boldsymbol{\beta}$, satisfies the "normal equation"

$$\mathbf{X}^\mathsf{T}\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^\mathsf{T}\mathbf{y}. \quad (1.61)$$

When $\mathbf{X}^\mathsf{T}\mathbf{X}$ is invertible[38] the least square parameter estimates can be written in closed form as

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y} := \boldsymbol{\Gamma}\mathbf{y}. \tag{1.62}$$

so that the vector of predicted (fitted) values $\hat{\mathbf{y}}$ can be expressed as

$$\hat{\mathbf{y}} = \mathbf{X}(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y} := \boldsymbol{\Pi}\mathbf{y}. \tag{1.63}$$

The matrices $\boldsymbol{\Gamma}$ and $\boldsymbol{\Pi}$ defined in (1.62) and (1.63) will be fundamental to our discussion of Markov decision process approximation algorithms. The matrix $\boldsymbol{\Pi}$ is sometimes referred to as the *hat matrix* because it maps the vector of observed values $\mathbf{y}$ into the vector $\hat{\mathbf{y}}$ of fitted values; that is, it puts a "hat" on $\mathbf{y}$.

Example 1.13. Quadratic regression: Suppose we model the dependent variable y_i by a quadratic function of the independent variable x_i written as

$$y_i = \beta_1 + \beta_2 x_i + \beta_3 x_i^2 + \epsilon_i$$

for $i = 1, , \ldots, N$. To form the matrix representation for this model, $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$, set

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ 1 & x_N & x_N^2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}.$$

Note that the constant β_1 results from multiplying the first component of $\boldsymbol{\beta}$ by a column of 1's. In most of the models considered in this book, $\mathbf{X}$ will contain a column of 1s.

There is no necessity that the x_i be distinct. Also, they can be pre-specified or randomly sampled. The subject *design of experiments* addresses how to specify these values so as to obtain the most precise estimates.

Least squares geometry

There is a beautiful geometric representation underlying least squares estimation. Since $\boldsymbol{\Pi}^2 = \boldsymbol{\Pi}$, $\boldsymbol{\Pi}$ is a projection matrix, it projects $\mathbf{y}$ onto the space of vectors spanned by the columns of $\mathbf{X}$ denoted $col(\mathbf{X})$ (see Figure 1.19). In other words $\Pi\mathbf{y}$ is the linear combination of the columns of $\mathbf{X}$ that is closest in the Euclidean norm sense to $\mathbf{y}$.

[38]This is true when the columns of $\mathbf{X}$ are linearly independent. Otherwise they are said to be *collinear*. In regression theory this phenomenon is referred to as *multi-collinearity*; several approaches most notably ridge regression have been developed to address this issue.

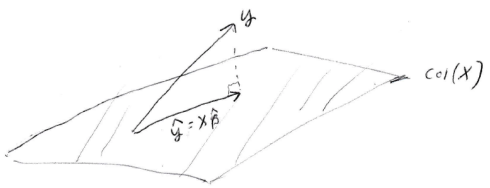


Figure 1.19: Illustration of the geometry underlying least squares regression. It shows the plane $col(\mathbf{X})$, the vector $\mathbf{y}$ which lies outside the plane and its projection $\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$ on $col(\mathbf{X})$. The dashed line corresponds to the vector of residuals $\mathbf{y} - \hat{\mathbf{y}}$.

Furthermore, rewriting the normal equations, (1.61) as

$$(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}})^{\mathsf{T}}\mathbf{X} = \mathbf{0}$$

shows that at the least squares estimator $\hat{\boldsymbol{\beta}}$, the vector of residuals, $\mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}$, is orthogonal to the space spanned by the columns of $\mathbf{X}$ as shown in Figure 1.19.

What this means is that $\mathbf{X}\hat{\boldsymbol{\beta}}$ is the closest point to *by* in the subspace $col(\mathbf{X})$. Moreover the vector of residuals $(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$ and the predicted values $\mathbf{y} = \mathbf{X}\boldsymbol{\beta}$ are uncorrelated, Hence the independent variables (represented by columns of $\mathbf{X}$ contain no additional explanatory power about the dependent variable.

Weighted Least Squares

In some situations we may prefer to minimize the weighted sum of squared errors

$$g_{\mathbf{w}}(\boldsymbol{\beta}) := \sum_{i=1}^{N} w_i g_i(\boldsymbol{\beta})^2 \tag{1.64}$$

where $\mathbf{w} = \{w_1, \ldots, w_N\}$ is a set of positive scalar weights. Defining the *squared weighted Euclidean norm* of $\mathbf{u} \in \Re^N$ as

$$\|\mathbf{u}\|_{\mathbf{w}}^2 := \sum_{i=1}^{N} w_i u_i^2, \tag{1.65}$$

we can write

$$g_{\mathbf{w}}(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_{\mathbf{w}}^2.$$

The matrix form of this expression is

$$g_{\mathbf{w}}(\boldsymbol{\beta}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathsf{T}}\mathbf{W}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}),$$

where $\mathbf{W} = \mathbf{I}\mathbf{w}$ is a diagonal matrix with entries $w_1, \ldots, w_N$.

Corresponding to this optimality criterion, the *weighted least squares (WLS) parameter estimates* are given by

$$\boldsymbol{\beta}_{WLS} = (\mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{y}. \tag{1.66}$$

Note that when $\mathbf{w} = \mathbf{I}$, these reduce to the ordinary least squares parameter estimates.

You might ask why we might want to use a weighted sum of squares as an optimality criterion. There are several reasons:

1. Some regions of the state space might be more important than others, so we would like estimates to be more precise in such regions.

2. In statistical models it might be the case that $\text{cov}[\boldsymbol{\epsilon}] = \sigma^2\mathbf{W}$. For example if for $i = 1, \ldots, n$,
$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$
and $\text{var}[\epsilon_i] = \sigma^2 x_i$, then we would use WLS with $w_i = 1/x_i$. This simple model applies widely, especially in economic data. It means observations become more variable the greater the value of the independent variable.

3. To adjust for unequal state sampling proportions and the data is either an average or a total of the observations at each state.

4. WLS has been used in reinforcement learning to ensure contraction properties[39].

1.8.3 Non-linear regression

Frequently we will represent value functions by nonlinear models, which we denote in scalar notation as

$$y_i = f(x_{i,1}, \ldots, x_{i,M}|\beta_1, \ldots, \beta_M) + \epsilon_i \tag{1.67}$$

for $i = 1, \ldots, N$ or in vector notation as

$$\mathbf{y} = \mathbf{f}(\mathbf{X}|\boldsymbol{\beta}) + \boldsymbol{\epsilon} \tag{1.68}$$

where

$$\mathbf{f}(\mathbf{X}|\boldsymbol{\beta}) := \begin{bmatrix} f(x_{1,1}, \ldots, x_{1,M}|\beta_1, \ldots, \beta_M) \\ \cdot \\ \cdot \\ \cdot \\ f(x_{N,1}, \ldots, x_{N,M}|\beta_1, \ldots, \beta_M) \end{bmatrix} = \begin{bmatrix} f(\mathbf{x}_1|\boldsymbol{\beta}) \\ \cdot \\ \cdot \\ \cdot \\ f(\mathbf{x}_N|\boldsymbol{\beta}) \end{bmatrix}$$

[39] Some operators may not be contractions with respect to the Euclidean norm, but are contractions with respect to a weighted Euclidean norm.

As in the case of linear regression, the least squares estimates of the parameters $\beta_1, \ldots, \beta_M$ minimize the squared error

$$g(\beta_1, \ldots, \beta_M) = \sum_{i=1}^{n} (y_i - f(x_{i,1}, \ldots, x_{i,M}|\beta_1, \ldots, \beta_M))^2, \tag{1.69}$$

which we express in matrix form as

$$g(\boldsymbol{\beta}) = \sum_{i=1}^{N} g_i(\boldsymbol{\beta})^2 = \mathbf{G}(\boldsymbol{\beta})^T \mathbf{G}(\boldsymbol{\beta}) \tag{1.70}$$

where $g_i(\boldsymbol{\beta}) = y_i - f(\mathbf{x}_i|\boldsymbol{\beta})$ and $\mathbf{G}(\boldsymbol{\beta})$ denotes an $N \times 1$ vector with components $g_i(\boldsymbol{\beta})$. Finding $\boldsymbol{\beta}$ such that $g(\boldsymbol{\beta})$ is minimized can be done by applying nonlinear optimization methods to (1.70). We discuss this in the next sub-section.

1.8.4 Nonlinear optimization

Many approaches can be used to find least squares estimates of nonlinear models. The most widely used are variants of gradient descent and Gauss-Newton iteration. We describe them here.

Gradient descent

Let $g(\boldsymbol{\beta})$ denote a function of the M-dimensional parameter vector $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_M)$. *Gradient descent* refers to algorithms based on recursions of the form

$$\boldsymbol{\beta}^{n+1} = \boldsymbol{\beta}^n - \tau_n \nabla g(\boldsymbol{\beta}^n) \tag{1.71}$$

where the gradient of g evaluated at $\boldsymbol{\beta}^n$ is given by

$$\nabla g(\boldsymbol{\beta}^n) := \begin{bmatrix} \frac{\partial g(\boldsymbol{\beta})}{\partial \beta_1} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial g(\boldsymbol{\beta})}{\partial \beta_M} \end{bmatrix} \tag{1.72}$$

and τ_n is the *step size* or learning rate at iteration n.

The motivation for this method is that $-\nabla g(\beta^n)$ represents the direction of steepest descent of g at β^n so that β in this direction will decrease g the fastest. Unfortunately, this method often converges very slowly and is very sensitive to starting values.

In the context of nonlinear regression, the jth component of the gradient of g evaluated at $\boldsymbol{\beta}$ is given by

$$\frac{\partial g(\boldsymbol{\beta})}{\partial \beta_j} = -2 \sum_{i=1}^{N} \left((y_i - f(x_{i,1}, \ldots, x_{i,M}|\beta_1, \ldots, \beta_M)) \frac{\partial f(x_{i,1}, \ldots, x_{i,M}|\beta_1, \ldots, \beta_M)}{\beta_j} \right)$$

for $j = 1, \ldots, M$. In matrix form this can be written as

$$\nabla g(\boldsymbol{\beta}) = -2\mathbf{J}(\boldsymbol{\beta})^{\mathsf{T}}, \mathbf{G}(\boldsymbol{\beta}) \tag{1.73}$$

where $\mathbf{J}(\boldsymbol{\beta})$ denotes the $M \times N$ *Jacobian matrix* of $F(\mathbf{X}|\boldsymbol{\beta})$ with components

$$\mathbf{J}(\boldsymbol{\beta})_{i,j} := \frac{\partial f(\mathbf{x}_i, \boldsymbol{\beta})}{\partial \beta_j} s$$

for $i = 1, \ldots, N$ and $j = 1, \ldots, M$. Alternatively

$$\mathbf{J}(\boldsymbol{\beta}) = \begin{bmatrix} \nabla f(\mathbf{x}_1|\boldsymbol{\beta})^{\mathsf{T}} \\ \cdot \\ \cdot \\ \cdot \\ \nabla f(\mathbf{x}_N|\boldsymbol{\beta})^{\mathsf{T}} \end{bmatrix}.$$

We illustrate the above calculations with an example.

Example 1.14. Suppose we wish to fit the nonlinear[a] model

$$y_i = f(x_i|\beta_1, \beta_2, \beta_3) + \epsilon_i = \beta_1 + \beta_2 e^{\beta_3 x_i} + \epsilon_i$$

$i = 1, \ldots, N$ to data. In this model, $\boldsymbol{\beta} = (\beta_1, \beta_2, \beta_3)$. Let

$$g(\boldsymbol{\beta}) = \sum_{i=1}^{N} (y_i - f(x_i|\beta_1, \beta_2, \beta_3)^2 = \sum_{i=1}^{N} g_i(\boldsymbol{\beta})^2$$

where $g_i(\boldsymbol{\beta}) = (y_i - f(x_i|\beta_1, \beta_2, \beta_3) = y_i - f(x_i|\boldsymbol{\beta})$.

For this model

$$-\frac{1}{2}\nabla g(\boldsymbol{\beta}) = \begin{bmatrix} \sum_{i=1}^{N} (y_i - (\beta_1 + \beta_2 e^{\beta_3 x_i})) \\ \sum_{i=1}^{N} (y_i - (\beta_1 + \beta_2 e^{\beta_3 x_i})) e^{\beta_3 x_i} \\ \sum_{i=1}^{N} (y_i - (\beta_1 + \beta_2 e^{\beta_3 x_i})) \beta_2 x_i e^{\beta_3 x_i} \end{bmatrix} = \mathbf{J}(\boldsymbol{\beta})^{\mathsf{T}} \mathbf{G}(\beta)$$

where

$$\mathbf{J}(\boldsymbol{\beta}) = \begin{bmatrix} 1 & e^{\beta_3 x_1} & \beta_2 x_1 e^{\beta_3 x_1} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ 1 & e^{\beta_3 x_N} & \beta_2 x_N e^{\beta_3 x_N} \end{bmatrix} \quad \text{and} \quad \mathbf{G}(\boldsymbol{\beta}) = \begin{bmatrix} y_1 - (\beta_1 + \beta_2 e^{\beta_3 x_1}) \\ \cdot \\ \cdot \\ \cdot \\ y_N - (\beta_1 + \beta_2 e^{\beta_3 x_N}) \end{bmatrix}.$$

[a]If the errors were multiplicative and $\beta_1 = 0$, this model would be linearizable by taking logarithms.

Newton's Method

Because gradient descent may converge slowly, Newton's method and its variants may be used to solve nonlinear least squares problems. Recall that to find a zero of the scalar function $f(u) = 0$, Newton's method uses the iteration scheme

$$u^{n+1} = x^n - \frac{f(u^n)}{f'(u^n)}$$

For a N-dimensional vector valued function $\mathbf{F}(\mathbf{u})$ with components $f_k(\mathbf{u})$ for $k = 1, \ldots, N$ and $\mathbf{u} = (u_1, \ldots, u_N)$ the iterates of Newton's method become

$$\mathbf{u}^{n+1} = \mathbf{u}^n - [\mathbf{J}(\mathbf{u}^n)]^{-1}\mathbf{F}(\mathbf{u}^n),$$

where as before $\mathbf{J}(\mathbf{u}^n)$, which is assumed to be invertible, denotes the Jacobian matrix of $\mathbf{F}(\mathbf{u})$.

We can solve a nonlinear least squares regression problem by applying Newton's method to the first order optimality condition $\nabla g(\boldsymbol{\beta}) = \mathbf{0}$. This results in the recursion

$$\boldsymbol{\beta}^{n+1} = \boldsymbol{\beta}^n - [\nabla^2 g(\boldsymbol{\beta}^n)]^{-1}\nabla g(\boldsymbol{\beta}^n) \tag{1.74}$$

where the *Hessian matrix* $\nabla^2 g(\boldsymbol{\beta}^n)$ has components

$$[\nabla^2 g(\boldsymbol{\beta}^n)]_{k,j} = \frac{\partial^2 g(\boldsymbol{\beta})}{\partial\beta_k \partial\beta_j}$$

for $k = 1, \ldots, M$ and $j = 1, \ldots, M$.

Gauss-Newton iteration provides an alternative to Newton's method that avoids computing the above second derivatives.

Gauss-Newton iteration

Gauss-Newton is based on approximating the real-valued function $g_i(\boldsymbol{\beta}^n)$ as defined in (1.70) by the first-order Taylor series approximation

$$g_i(\boldsymbol{\beta}) \approx g_i(\boldsymbol{\beta}^n) + [\nabla g_i(\boldsymbol{\beta}^n)]^\mathsf{T}(\boldsymbol{\beta} - \boldsymbol{\beta}^n). \tag{1.75}$$

Since $\mathbf{G}(\boldsymbol{\beta}) = (g_1(\boldsymbol{\beta}), \ldots, g_N(\boldsymbol{\beta}))$ and $g(\boldsymbol{\beta}) = \mathbf{G}(\boldsymbol{\beta})^\mathsf{T}\mathbf{G}(\boldsymbol{\beta})$

$$g(\boldsymbol{\beta}) = (\mathbf{G}(\boldsymbol{\beta}^n) + \mathbf{J}(\boldsymbol{\beta}^n)(\boldsymbol{\beta} - \boldsymbol{\beta}^n))^\mathsf{T}(\mathbf{G}(\boldsymbol{\beta}^n) + \mathbf{J}(\boldsymbol{\beta}^n)(\boldsymbol{\beta} - \boldsymbol{\beta}^n)) \tag{1.76}$$

where $\mathbf{J}(\boldsymbol{\beta}^n)$ denotes the $N \times M$ Jacobian matrix of $\mathbf{G}(\boldsymbol{\beta})$. We emphasize that in (1.76) the $\boldsymbol{\beta}^n$ is fixed and $\boldsymbol{\beta}$ is variable so that $\mathbf{G}(\boldsymbol{\beta}^n)$ is a fixed $N \times 1$ vector and

$\mathbf{J}(\boldsymbol{\beta}^n)$ is a fixed $N \times M$ matrix. Hence analogous to linear regression, this expression is minimized by choosing

$$\boldsymbol{\beta} - \boldsymbol{\beta}^n = -(\mathbf{J}(\boldsymbol{\beta}^n)^\mathsf{T}\mathbf{J}(\boldsymbol{\beta}^n))^{-1}\mathbf{J}(\boldsymbol{\beta}^n)^\mathsf{T}\mathbf{G}(\boldsymbol{\beta}^n).$$

Rearranging terms yields the Gauss-Newton recursion

$$\boldsymbol{\beta}^{n+1} = \boldsymbol{\beta}^n - (\mathbf{J}(\boldsymbol{\beta}^n)^\mathsf{T}\mathbf{J}(\boldsymbol{\beta}^n))^{-1}\mathbf{J}(\boldsymbol{\beta}^n)^\mathsf{T}\mathbf{G}(\boldsymbol{\beta}^n). \tag{1.77}$$

A few comments are relevant:

1. Note that the recursion in (1.77) involves a matrix inversion. Instead of inverting this matrix, one can solve the system of equations

   $$(\mathbf{J}(\boldsymbol{\beta}^n)^\mathsf{T}\mathbf{J}(\boldsymbol{\beta}^n))\Delta^{n+1} = -\mathbf{J}(\boldsymbol{\beta}^n)\mathbf{G}(\boldsymbol{\beta}^n).$$

   and set $\boldsymbol{\beta}^{n+1} = \boldsymbol{\beta}^n + \Delta^{n+1}$.

2. Often it might be inconvenient to compute $\mathbf{J}(\boldsymbol{\beta}^n)$ analytically, instead it can be approximated numerically.

3. Convergence of Gauss-Newton is sensitive to the starting value $\boldsymbol{\beta}^0$. A good choice can enhance convergence.

4. Modifications are available when $[\mathbf{J}(\boldsymbol{\beta}^n)^T\mathbf{J}(\boldsymbol{\beta}^n)]$ is close to singular.

5. Gauss-Newton can also be derived as a special case of Newton's method in which the Hessian is approximated by the Jacobian.

1.9 Bibliographic Remarks

The concept of using approximations in Markov decision processes appears to originate in the clear and concise paper Schweitzer and Seidmann [1985]. This paper, motivated by work on operations research problems, introduced the idea of approximating value functions by low order multi-variable polynomials and showed how to incorporate these approximations in linear programming and policy iteration. Moreover it considers both discounted and average reward formulations and like this chapter, did not consider the use of simulation in estimation. We don't believe the authors anticipated how significant and widely used this approach would become.

The texts by Bertseksas [2012], Bertsekas and Tsitsiklis [1996] and Powell [2007] provide comprehensive discussions of value function approximation.

The use of features to reduce state spaces appears to originate with Samuel [1959] who is also credited with coining the expression "Machine Learning" to represent the study of using computers to learn tasks without explicitly programming them to do so.

We base our discussion of least-squares parameter estimates in policy iteration on Lagoudakis and Parr [2003] and the surveys by Bertsekas [2011] and Buşoniu et al. [2012]. The bounds for LSPI appear to be new although other bounds exist in the literature.

de Farias and van Roy [2003] provides an in depth study of the approximate LP model and motivate Example 1.5 and especially Figure 1.8. Moreover they analyze the queuing admission control model we sue for examples in this chapter.

Nielsen [2015] provides a very readable step by step introduction to neural networks. Private communications with several researchers corroborate the difficulties we observed when incorporating neural networks in LSVI and LSPI. In fact, most analyses in the literature are based on linear approximations.

Our discussion and analysis of the advanced scheduling model draws on work of Patrick et al. [2008] and Sauré et al. [2015].

Draper and Smith [1998] provide a good introduction to and overview of regression.

1.10 Exercises

1. Show that all the equivalences in (1.59) holds.

2. a. Write out the Newton's method and Gauss-Newton recursions for the model in Example 1.14.

   b. Fit this model to the data in Table 1.3 using gradient descent, Newton's Method and Gauss-Newton. Compare the effort to set the recursion up, the rate of convergence and the sensitivity to starting values.

   Note that the data generating model was $y_i = 2+3e^{0.3x_i}+\epsilon_i$ where $\epsilon_i \sim NID(0,1)$.

Variable / Observation	1	2	3	4	5	6	7
y_i	8.37	15.39	4.93	8.92	33.74	7.14	62.95
x_i	2	5	1	3	8	2	10

Table 1.3: Data for non-linear regression example

3. Describe the state space for the game of Scrabble. What features would you use to summarize it?

4. Describe the state space for the game of Tetris. What features would you use to summarize it?

5. Carry out the analysis of the queuing service rate control model in Example 1.5 for value functions approximations of the form $\beta_0 + \beta_1 s$ and $\beta_0 + \beta_1 s + \beta_2 s^2$. Assume the $S = \{0, 1, \ldots, 10\}$ and $d(s) = a_1$ for $s \leq 5$ and $d(s) = a_2$ for $s > 5$.

(a) What are the features corresponding to each of these approximations.

(b) Obtain parameter estimates for each approximation using linear programming.

(c) Obtain parameter estimates for each approximation using LSVI.

(d) Compute the exact value function for this policy and compare it to the two approximations.

(e) Compare the greedy policies corresponding to each estimate to the optimal policy. What do you conclude?

6. Consider a two state model with a single policy $d(s)$, with $\lambda = \frac{1}{2}$,

$$\mathbf{r}_d = \begin{bmatrix} 4 \\ 2 \end{bmatrix} \quad \text{and} \quad \mathbf{P}_d = \begin{bmatrix} .5 & .5 \\ 1 & 0 \end{bmatrix}. \tag{1.78}$$

(a) Find $\hat{\boldsymbol{\beta}}_{OLS}$, $\hat{\boldsymbol{\beta}}_{LSVI}$ and $\hat{\boldsymbol{\beta}}_{BR}$ when the basis function is $b_0(s) = 1$.

(b) Represent these estimates graphically as in Figure 1.6.

7. **Oscillation of LSPI** [40] Let $S = \{s_1, s_2\}$, $A_{s_1} = \{a_{1,1}, a_{1,2}\}$, $A_{s_2} = \{a_{2,1}\}$, $r(s_1, a_{1,1}) = 0.1$, $r(s_1, a_{1,2}) = 0$, $r(s_2, a_{2,1}) = 0$, $p(s_1|s_1, a_{1,1}) = 0.8$, $p(s_2|s_1, a_{1,1}) = 0.2$, $p(s_2|s_1, a_{1,2}) = 1$, $p(s_1|s_2, a_{2,1}) = 1$ and $\lambda = 0.9$.

(a) Find the optimal policy by enumeration.

(b) Consider the basis function represented by $b(s_1) = 1, b(s_2) = 2$ so that the linear approximation to the value function is of the form $\tilde{\mathbf{v}} = \begin{bmatrix} \beta \\ 2\beta \end{bmatrix}$. For what values of β is optimal to use action $a_{1,1}$, respectively $a_{1,2}$, in s_1.

(c) Solve the problem using LSPI starting with $\beta = 1$. What behavior of the iterates do you observe?

8. Consider the replacement model in Section **??** with $M = 500, p = 0.1, \lambda = 0.95$, $K = 1200$ and a linear operating cost $2s$ in state s.

(a) Find an optimal policy by a method of your choosing and describe its structure.

(b) Find linear and quadratic value function approximations using LSVI,LSMPI, LSPI and linear programming. Compare the quality of the approximations and the greedy policies identified by them.

9. For the advanced appointment scheduling model in Section 1.6, compare the greedy policy based on (1.46) obtained from a random sample of half the states to that in (1.44) based on applying LSVI to the entire state space,

[40] This example is adopted from Example 1 in Bertsekas [2011].

10. (**mini-project**)This open-ended problem asks you to carry out your own analysis of the advanced appointment scheduling model analyzed in Section 1.6. To do so entails coding the model, solving it optimally, analyzing it with LSVI using polynomial and neural network approximations and comparing policies and the quality of approximations. You must specify λ, N, M and K, costs and demand distributions. Some questions to consider:

    (a) How large a problem can you solve exactly?

    (b) What is the structure of the greedy policies?

    (c) What is the impact of state sampling?

    (d) How does the quality of the approximations vary with cost structure?

    (e) How can you extend your analyses to three or more classes.

11. (**mini-project**) Consider an inventory system that manages L products it obtains from a single source. Every time an order is placed, there is a fixed charge of K dollars and a per unit charge of c_l for product l. Assume orders placed arrive prior to the next business day. The cost for storing one unit of product l for one day is h_l.

    Customers orders for products arrive daily and are filled immediately from stock on hand. Assume demand for product l is Poisson distributed with mean μ_l. The selling price for product l is r_l and orders that can not be filled from stock on hand are "lost", that is the customer seeks the product elsewhere.

    (a) Formulate this as a Markov decision problem where the objective is to maximize discounted expected revenue over an infinite horizon.

    (b) What quantities would need to be truncated in order to solve this problem. How would you truncate them?

    (c) Consider a model with $L = 5$, $K = 200$, $c_l = 50$, $r_l = 100$, $h_l = 2$ and $\mu_l = 5$. Find approximately optimal policies using LSVI, LSPI and linear programming. Choose suitable basic functions and architectures. Derive bounds to assess the quality of your approximations.

    (d) Is there any obvious structure to the approximate policies? Investigate how they vary when the model parameters change and the products are not identical.

12. State action value function approx example.

13. Example where approximation is an aggregation.

14. Neural net experiment.

15. Show LSPE operator is a contraction in weighted sup-norm with weighting function given by the stationary distribution of the underlying regular Markov chain.

Bibliography

D. Bertsekas. Approximate policy iteration: a survey and some new methods. *J. Control Theory Appl.*, 9:310–335, 2011.

D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

D.P. Bertseksas. *Dynamic Programming and Optimal Control, Volume 2, 4th Edition.* Athena Scientific, 2012.

M. Broadie. *Every Shot Counts.* Avery, 2014.

L. Buşoniu, A. Lazaric, and et al. Least-squares methods for policy iteration. In M. Wiering and M. Otterio, editors, *Reinforcement Learning: State of the Art*, pages 75–109. springer Berlin, 2012.

D. P. de Farias and B. van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51:850–865, 2003.

D. deFarias and B. van Roy. On the existence of fixed points for approximate value iteration and temporal-difference learning. *J. Opt. Theor. Appl.*, 105:589–608, 2000.

N. Draper and H. Smith. *Applied Regression Analysis, 3rd Edition.* Wiley-Interscience, 1998.

M. Lagoudakis and R. Parr. Least-squares policy evaluation. *J. Mach. Learning Res.*, 4:1107–1149, 2003.

M. Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015.

J. Patrick, M. L. Puterman, and M. Queyranne. Dynamic multi-priority patient scheduling. *Operations Research*, 56:1507–152, 2008.

W. Powell. *Approximate Dynamic Programming.* J.Wiley and Sons, 2007.

R. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2021. URL `https://www.R-project.org/`.

A. Samuel. Some studies in machine learning using the game of checkers. *IBM J. Res. and Dev.*, 3:210–229, 1959.

A. Sauré, J. Patrick, and M.L. Puterman. Simulation-based approximate policy iteration with generalized logistic functions. *INFORMS J. Computing*, 27:579–595, 2015.

P.J. Schweitzer and A. Seidmann. Genralized polynomial approximations in markov decision processes. *J. Math. Anal. Appl.*, 110:568–582, 1985.

Chapter 2

Index

1. xx