Chapter 9

Value Function Approximation

Far better an approximate answer to the right question, which is often vague, than an exact answer to the wrong question . . .[1]

John Tukey, American statistician, 1915-2000.

9.1 Introduction

In models with large state spaces or action sets or both, direct evaluation and storage of policy and optimal value functions or state-action value functions may be prohibitive. As Tukey suggests, approximations may be preferable in such instances. To address this issue, this chapter describes methods in which value functions are often approximated by lower-dimensional functions such as polynomials, splines or neural networks. Doing so presents several challenges that are discussed below. Approaches based on function approximations are often referred to as *approximate dynamic programming* (ADP).

As an example, consider the medical appointment scheduling model of Section 3.7, analyzed in Section 9.6 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[2] arriving on a particular day. Actions represent the number of incoming appointment requests of each type to book in available appointment slots on each day in the booking horizon. The objective in this application

[1] Tukey [1962].

[2] In the application that motivated development of these models, type is an urgency level determined by a medical professional. Each urgency type has a different wait time target.

is to find a policy for assigning appointment slots to incoming appointment requests that uses resources efficiently but meets wait time targets.

In a realistic application of this model, the booking horizon $N = 30$, the number of appointment types $K = 3$, the daily capacity $C = 20$ and the maximum number of daily arrivals of each type $M = 10$. Subject to this specification the model will have $(21)^{30} \times (11)^3$ 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 alternatives.

To motivate the approach, let $\hat{v}^*_\lambda(s)$ denote an approximation to the optimal value function $v^*_\lambda(s)$ in an infinite horizon discounted model. This chapter considers approaches in which the approximation is a lower-dimensional *parametric* function of model characteristics referred to as *basis functions* or *features*. Instead of storing $v^*_\lambda(s)$ in tabular form, the methods herein store only the parameter estimates. Using this information, a "good" action in a specific state s' is chosen as follows. Evaluate $\hat{v}^*_\lambda(j)$ in states j that can be reached from s' in one transition and choose an action $a^*_{s'}$ greedily based on the one-step Bellman update

$$a^*_{s'} \in \arg\max_{a \in A_{s'}} \left\{ r(s', a) + \lambda \sum_{j \in S} p(j|s', a) \hat{v}^*_\lambda(j) \right\}. \tag{9.1}$$

Since in most applications, there are few successor states to s', this calculation can be very efficient. Recall that this is equivalent[3] to solving a one-period problem (see Chapter 2) with terminal reward $\hat{v}^*_\lambda(s)$. Note that such an approach was used in Section 8.4.5 in which the POMDP optimal value function was approximated by supports obtained on a small set of belief points.

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

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

Moreover, approximations can be also used to directly represent policies in terms of model states such as in the policy gradient methods of Section 11.5.

Note that a policy based on greedy action choice using an approximation to the value function need not be optimal. The quality of the action depends on the accuracy of the approximation.

Returning to the appointment scheduling problem, suppose one has an approximation $\hat{v}^*_\lambda(j)$ to the optimal value function. Instead of a finding and storing the decision rule for all states, an action is chosen as needed as follows:

1. Observe the system state s', namely the number of appointments previously scheduled for each day in the booking horizon and the number of arrivals of each type.

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

2. Obtain an action to implement by evaluating the expression in braces in (9.1) for each $a \in A_{s'}$ and choosing the maximizing action,, $a^*_{s'}$.

3. Assign patients to future appointment dates according to $a^*_{s'}$.

In this case, as in most implementations of Markov decision processes, the optimal action in all states is not needed a priori; it can be determined on an as-needed basis using the above approach or one based on an approximation $\hat{q}^*(s,a)$. The advantage of using $\hat{q}^*(s,a)$ will become apparent in model-free implementations described in Chapters 10 and 11.

Questions arising are:

1. How to obtain good value function approximations?

2. How close to optimal is the stationary policy based on decision rule $\hat{d}^*(s) = a^*_s$?

Note that in most of the literature, approximation and simulation are discussed together. To identify issues particular to each, these discussions are separated. This chapter focuses on approximation. Chapter 10 discusses simulation methods. Then, Chapter 11 combines simulation and approximation, which is the cornerstone of *reinforcement learning*. Moreover, the focus in this chapter is on discounted models. Extensions to average and total reward models are left to the reader.

9.2 Approximating policy value functions

This section describes approaches for approximating value functions and subsequently generalizes them to state-action value functions. Moreover, the focus will be on approximating the expected discounted reward of a *fixed* stationary deterministic policy. The following example provides motivation.

Example 9.1. Recall that in the queuing service rate control model (Section 3.4.1) 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. There is a delay cost of $f(s) = s^2$ and a cost per period of serving at rate a_k is $m(a_k) = 5k^3$.

To illustrate approximations, the unbounded state space is truncated to $S = \{0, 1, \ldots, 50\}$ and the discount rate is set to $\lambda = 0.98$. Approximations to the value function of stationary policy $\pi = d^\infty$, which uses the deterministic decision

rule

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

are shown.

Since the state space is ordered and interpretable, simple parametric functions of the state can be used as value function approximators. To investigate the quality of approximations, the exact policy value function[a] is evaluated using methods of Chapter 5. Approximations to the exact policy value function based on linear, quadratic and exponential functions are considered. Parameter estimates for the polynomials are obtained using least squares linear regression, while those for the exponential function are obtained using nonlinear least squares. Regression is based on the data $\{(s, v(s))|s \in S\}$ or a subset thereof. In this example, this data was obtained from exact methods. In practice the exact $v(s)$ will not be available.

Figure 9.1a shows the true value function and its linear and quadratic approximations. Clearly the linear fit is poor while the quadratic fit appears to be better. It is left as an exercise to evaluate the quality of cubic polynomial and spline approximations. The equations of the two fitted models are

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

and

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

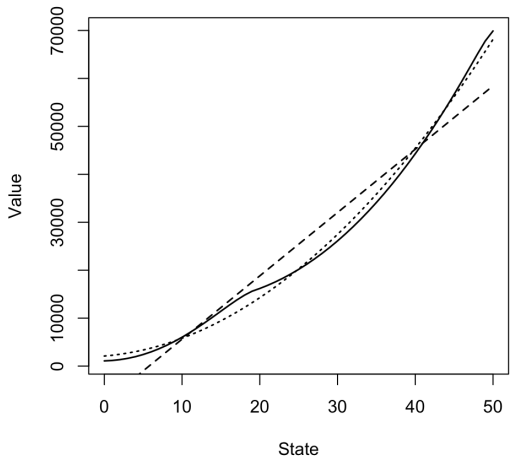
In addition the value function is approximated by the nonlinear function $\beta_0 + \beta_1 e^{\beta_2 s}$, which requires software that implements nonlinear least squares. The analysis here uses the *nls* package in R [R Core Team, 2024], which implements Gauss-Newton iteration to obtain parameter estimates. Given the sensitivity of nonlinear least squares to starting values, experimentation was required to obtain the fitted equation:

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

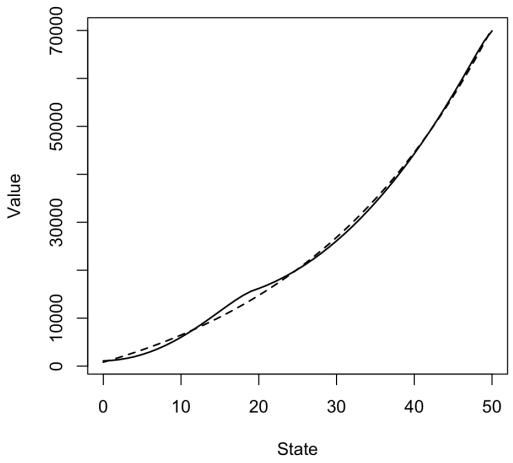
Figure 9.1b shows the actual value (solid line) and fitted value (dashed line). It appears that the exponential fit is superior to the quadratic fit.

Of course, value function approximation is not necessary in such a small instance; the above calculations are provided to motivate for the material in this chapter. In practice $v^{\pi}_{\lambda}(s)$ would not be known for all $s \in S$, so approximation would be necessary. As an aside, if the focus was the countable state version of this model, the approximation on $s \leq 50$ could be extrapolated to $s > 50$.

[a] Of course, if the exact policy value function is available, such as in this small example, there is no need to approximate it.



(a) Value function for d^∞ (solid line) with linear (dashed line) and quadratic (dotted line) approximations.



(b) Value function for d^∞ (solid line) with exponential approximation (dashed line).

Figure 9.1: Example 9.1 value functions and approximations.

9.2.1 Features

It is customary to use features of the state space as variables when approximating value functions. Feature generation is similar to the choice of independent variables in regression models. Features are usually low dimension summaries of key aspects of the state space that can be used to well approximate a value function. More formally:

Definition 9.1. A *feature* or *basis function* is a real-valued function defined on S.

Denote the set of features evaluated at s by the vector[4] $\mathbf{b}(s) = \big(b_0(s), b_1(s), \ldots, b_I(s)\big)$ Often $b_0(s) := 1$ to correspond to a constant in a regression model.

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

$$\mathbf{B} := \begin{bmatrix} b_0(s_1) & \ldots & b_I(s_1) \\ \cdot & \ldots & \cdot \\ \cdot & \ldots & \cdot \\ \cdot & \ldots & \cdot \\ b_0(s_M) & \ldots & b_I(s_M) \end{bmatrix}. \tag{9.2}$$

[4]Recall that Chapter 8 used the vector $\mathbf{b}$ to denote a belief state. Hopefully this will not cause confusion.

Assume that features are constructed so that the columns of $\mathbf{B}$ are linearly independent (in the linear algebra sense[5]) so that the matrix $\mathbf{B}^\mathsf{T}\mathbf{B}$ is invertible.

Specifying features

Feature choice depends on the setting:

1. If the state space is a set of **real numbers or ordered integers** possible feature choices include powers of the state, B-spline components, trigonometric functions or exponentials. In Example 9.1 when using a quadratic value function approximation, $I = 2$ and $b_0(s) = 1$, $b_1(s) = s$ and $b_2(s) = s^2$ for all $s \in S$.

2. If the state space is **vector-valued**, possible feature choices include component values, powers of component values, interactions between components or nonlinear 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

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

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

   A possible set of features is:

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

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

3. In **more general settings**, application and subject matter knowledge dictates feature choice. For example, in checkers[6] (Figure 9.2) there are 32 squares on the board that can be occupied (numbered in some way) and the two players, referred to as "Red" and "Black", each start with 12 pieces. Thus

$$S = \left\{(x_1, \ldots, x_{32}) \;\middle|\; x_i \in \{E, B, R, BK, RK\} \text{ for } i = 1, \ldots, 32 \text{ and } \sum_{i=1}^{32} I_{\{B,BK\}}(x_i) \le 12,\ \sum_{i=1}^{32} I_{\{R,RK\}}(x_i) \le 12\right\},$$

[5] This means that the only solution of $\mathbf{Bx} = \mathbf{0}$ is $\mathbf{x} = \mathbf{0}$. Equivalently the rank of $\mathbf{B}$ is $I + 1$ (since it is assume $I + 1 < M$).

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

Figure 9.2: Checkerboard showing positions of pieces at the start of a game. Credit: gmatsuno/E+ via Getty Images.

where $I_A(x)$ is the indicator function for $x \in A$, E indicates that the square is empty, B denotes it is occupied by a single black checker, R denotes it is occupied by a single red checker, BK denotes it is occupied by a black king, and RK denotes it is occupied by a red king. The summations indicate that there are at most 12 pieces of each color on the board. Since $|S| = 5^{32}$, approximations are necessary to analyze this game.

Possible features include

$$b_0((x_1, \ldots, x_{32})) = 1;$$

$$b_1((x_1, \ldots, x_{32})) = \sum_{i=1}^{32} I_{\{B\}}(x_i); \quad b_2((x_1, \ldots, x_{32})) = \sum_{i=1}^{32} I_{\{BK\}}(x_i);$$

$$b_3((x_1, \ldots, x_{32}) = \sum_{i=1}^{32} I_{\{R\}}(x_i); \quad b_4((x_1, \ldots, x_{32})) = \sum_{i=1}^{32} I_{\{RK\}}(x_i).$$

The above features can be interpreted as follows: $b_1(\cdot)$ and $b_3(\cdot)$ denote the number of single checkers of each color and $b_2(\cdot)$ and $b_4(\cdot)$ denote the number of kings of each color. Other possible features include the number of pieces of each color that are within κ rows of becoming kings for specific choices of κ.

In summary, possible features include:

1. the function 1,

2. powers and cross products of state variables,
3. expressions that interpolate between designated states, and
4. indicator functions of subsets of states.

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 $i = 1, \ldots, M$, is equivalent to a *tabular* representation of the model. This means that expressing the value function as a linear combination of such indicator functions is equivalent to using the standard Markov decision process formulation in terms of $v(s_i)$ for $i = 1, \ldots, M$. In this case the methods of earlier chapters apply directly.

9.2.2 Feature-based value function approximators

This section proposes some value function approximations based on features. These approximations may be linear or nonlinear with neural networks representing an important and widely used class of nonlinear approximations. Some authors refer to the specified functional form as an *architecture*.

Linear architectures

The expression *linear approximation*[7], refers to a value function approximation of the form:

$$v(s) \approx \beta_0 b_0(s) + \beta_1 b_1(s) + \ldots + \beta_I b_I(s), \tag{9.3}$$

where $b_0(s) = 1$ 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 linear approximations include the ability to estimate parameters using least squares regression (i.e., where estimates are available in closed form; see the chapter appendix) and having a functional form that can be directly used in a linear programming model. Moreover, when meaningful, the parameters can be interpreted as marginal values or costs.

Linear approximations can be nicely represented using matrix notation as follows:

$$\mathbf{v} \approx \mathbf{B}\boldsymbol{\beta}, \tag{9.4}$$

where

$$\mathbf{B} = \begin{bmatrix} b_0(s_1) & \ldots & b_I(s_1) \\ \cdot & \ldots & \cdot \\ \cdot & \ldots & \cdot \\ \cdot & \ldots & \cdot \\ b_0(s_M) & \ldots & b_I(s_M) \end{bmatrix} \quad \text{and} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \cdot \\ \cdot \\ \cdot \\ \beta_I \end{bmatrix} \tag{9.5}$$

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

and $S = \{s_1, \ldots, s_M\}$. In statistical literature, the matrix $\mathbf{B}$ is often referred to as the *design matrix*[8].

Note that the above expression does not explicitly include an error term ϵ because the models in this chapter do not involve statistical sampling. The only source of imprecision is the inaccuracy of the approximation.

Nonlinear architectures

Although in many applications, a linear approximation will suffice, a nonlinear function f of features can also be used to represent value functions. In general, nonlinear architectures can be written as

$$v(s) \approx f(b_0(s), b_1(s), \ldots, b_I(s)|\boldsymbol{\beta}), \tag{9.6}$$

where the unknown vector of parameters in state s is 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 $I + 1$. For example, with two features $b_0(s)$ and $b_1(s)$ a possible choice for $f(\cdot)$ is

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

where $\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2)$. In this case there are two features and three parameters. 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 9.1.

Neural networks

Artificial neural networks or neural networks (NNs)[9] 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, state-action value functions or even the probability of selecting an action. However they also have been used in numerous significant machine learning implementations including image recognition, recommendation systems, forecasting, and natural language processing.

The most basic type of neural network is a *feedforward neural network* or FNN. A FNN consists of *nodes* organized in *layers*, uni-directional *arcs* connecting nodes from "left" to "right" and *activation functions* corresponding to nodes in interior *hidden* layers. Layers are classified as input layers, hidden layers and output layers. Figure 9.3 depicts a feedforward neural network with an input layer, two hidden layers and an output layer.

[8]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.

[9]Some authors refer to neural networks as *multi-layer perceptrons*.

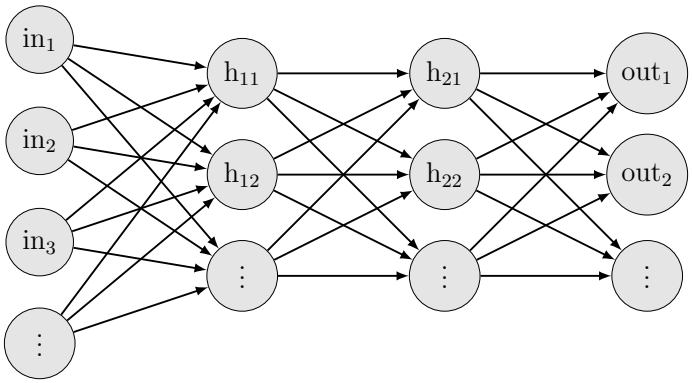


Figure 9.3: A generic feedforward neural network. The nodes labeled in_j denote nodes in the input layer, those labeled $\text{h}_{i,j}$ denote nodes in hidden layer j, and those labeled out_j denote nodes in the output layer.

The FNN processes information as follows: 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 nonlinearly by the activation function to produce outputs. These outputs are then weighted and summed and fed to the next layer, and so on.

Activation functions mimic the behavior of neurons that "fire" when the incoming signal reaches a threshold. They can be modeled 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 non-decreasing continuous functions $f : \mathbb{R} \to [0, 1]$ such as logistic or sigmoidal functions.

Other types of neural networks include:

1. **recurrent neural networks**, which generate outputs by combining inputs with previous hidden states, and

2. **convolutional neural networks**, which include filters that extract features from grid-like data such as camera images.

Details of these models are beyond the scope of this book but are important for applications in robotics and autonomous vehicle guidance. From the perspective of this chapter, neural networks are viewed as "black box" nonlinear function approximators.

An example

This example applies a single-layer FNN to the queuing service rate control example analyzed in Example 9.1 and discusses some practical issues arising when applying neural networks.

Example 9.2. Consider the queuing service rate control model analyzed in Example 9.1. This example approximates the value function of the specified stationary policy with $\lambda = 0.98$ using the FNN represented in Figure 9.4. This network consists of an input layer, a hidden layer with two nodes plus a node denoted by 1, which is often referred to as an *offset* or *bias*, and a single node in the output layer. The objective is to estimate parameters of this neural network to accurately approximate the value function. This may be regarded as using a neural network to perform nonlinear regression with features $b_0(s) = 1$ and $b_1(s) = s$.

In the hidden layer, the FNN model transforms a linear combination of inputs using the activation function

$$f(x) = \frac{1}{1 + e^{-x}} \tag{9.7}$$

and outputs the value function approximation using a linear combination of outputs from the hidden layer plus a constant. The model has seven parameters, one corresponding to each arc denoted by $w^k_{i,j}$, where i denotes the node at the start of the arc, j denotes the destination of the arc and $k = 2$ or 3 denotes the layer of the destination.

The FNN may be expressed as a single function as follows. The input to h_1 in the hidden layer is a linear function of the inputs 1 and s

$$w^1_{11} + w^1_{21} s$$

and the input to node 2 in the hidden layer is

$$w^1_{12} + w^1_{22} s.$$

The activation function transforms these inputs to

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

Finally the input to the output layer can be represented by the linear function of outputs from the hidden layer as follows:

$$w^2_{01} 1 + w^2_{11} \frac{1}{1 + e^{-(w^1_{11} + w^1_{21} s)}} + w^2_{21} \frac{1}{1 + e^{-(w^1_{12} + w^1_{22} s)}}. \tag{9.8}$$

Note that this function contains the constant w^2_{01}. In light of the von Neumann quote in the Part III introduction, one would expect an accurate approximation of the value function using this architecture.

The *nnet* package in R produced the parameter estimates shown on the arcs in Figure 9.4. Achieving reliable estimates required scaling both the state and the

value function to the interval [0, 1] prior to training, followed by a reverse transformation to the original scale. Furthermore, the inclusion of a small amount of random noise during training proved beneficial. Without scaling, the estimation process produced unreliable results.

Figure 9.5 compares the true value function (solid line) and its FNN approximation (dashed line). The RMSE was 395 and while the fit looks good, the residuals had a sinusoidal pattern suggesting a systematic lack of fit. Ultimately, the quality of the approximation is determined based on the policy derived from the approximate value function.

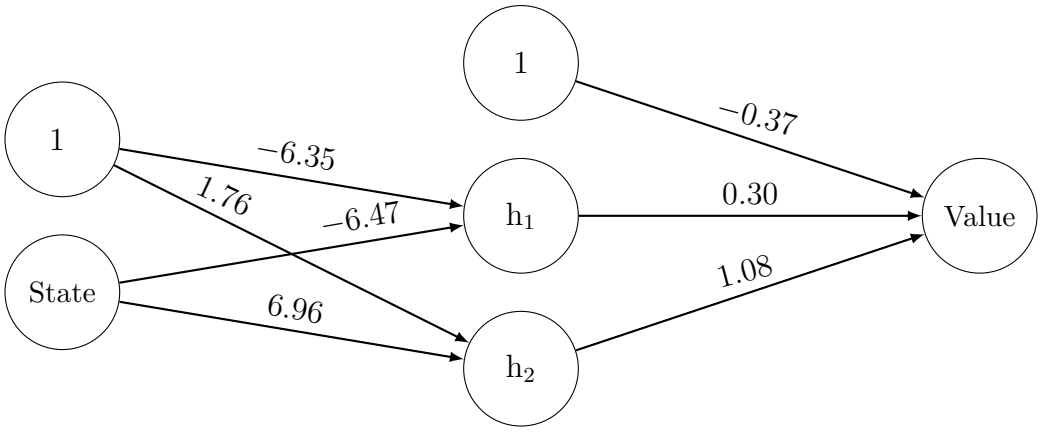


Figure 9.4: Feedforward neural network used in queuing service rate control approximation. Values on the arcs are estimated weights (parameter estimates).

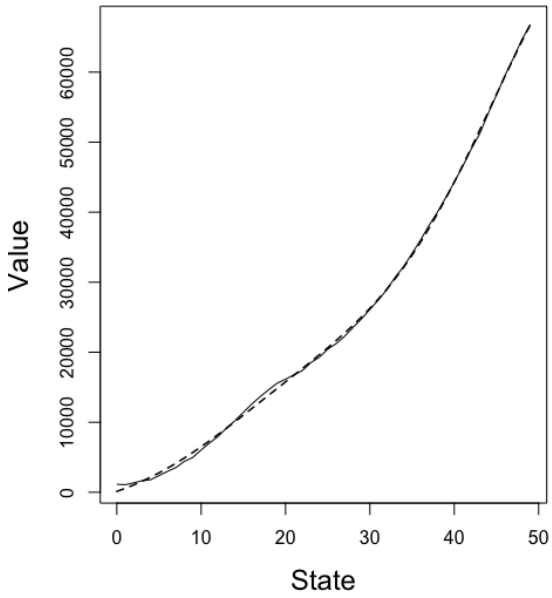


Figure 9.5: FNN approximation to the queuing system service rate control value function for the specified stationary policy. Solid line is the true value function while the dashed line is the approximation based on the FNN.

9.2.3 State-action value functions

The above sections describe how to approximate value functions using basis functions and linear or nonlinear architectures. Alternatively these constructs can be used to approximate state-action value functions $q(s,a)$. While such an approach may be of use in reinforcement learning environments when the model components are unknown, in this chapter, which assumes knowledge of $r(s,a)$ and $p(j|s,a)$, it is preferable to approximate $q(s,a)$ for all $a \in A_s$ by

$$\hat{q}(s,a) := r(s,a) + \lambda \sum_{j \in S} p(j|s,a)\hat{v}(j),$$

where $\hat{v}(s)$ is a value function approximation based on basis functions. Thus, the focus of this chapter will be on value function approximations.

9.3 Parameter estimation for policy value functions: Linear architectures

This section provides methods for finding good approximations to the value function of a *fixed* stationary policy in the form of linear combinations of basis functions. The

proposed methods are based on least squares estimation and linear programming. The next section considers nonlinear approximations. We strongly recommend referring to the Appendix of this chapter, which reviews least squares regression emphasizing matrix methods.

Motivation

Let d^∞ denote a stationary randomized policy. To determine its value, $\mathbf{v}_\lambda^{d^\infty}$, involves solving the following linear system of $|S|$ equations, which appeared previously in vector form in (5.31) as

$$\mathbf{v} = \mathbf{r}_d + \lambda \mathbf{P}_d \mathbf{v} = L_d \mathbf{v}. \tag{9.9}$$

Recall that when $\lambda < 1$, this system has a unique solution that can be represented by

$$\mathbf{v}_\lambda^{d^\infty} = (\mathbf{I} - \lambda \mathbf{P}_d)^{-1} \mathbf{r}_d. \tag{9.10}$$

Using matrix notation, the general form of a policy value function estimator[10] based on a linear combination of basis functions can be written as $\mathbf{v} = \mathbf{B}\boldsymbol{\beta}$, where the columns of $\mathbf{B}$ are vectors of basis functions evaluated for each $s \in S$. That is, a row of $\mathbf{B}$ is of the form $\big(b_0(s), \ldots, b_I(s)\big)$ for some $s \in S$. Let $\text{col}(\mathbf{B})$ denote the subspace of $\mathbb{R}^M$ spanned by the columns of $\mathbf{B}$, that is, it contains all real-valued functions that can be written as linear combinations of basis functions.

The benefit of seeking approximations in $\text{col}(\mathbf{B})$ is that there are $I+1$ parameters to determine, as opposed to $|S|$. In applications where approximation is necessary, I will be orders of magnitude smaller than $|S|$. But replacing exact calculation by an approximation introduces *approximation error*. Moreover the methods below seek to do this without first computing $\mathbf{v}_\lambda^{d^\infty}$.

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

To motivate approximation methods and introduce some notation, assume first that $\mathbf{v}_\lambda^{d^\infty}$ has been evaluated[11] and one seeks a good approximation $\text{col}(\mathbf{B})$. Thus one wants to find

$$\hat{\boldsymbol{\beta}} \in \underset{\boldsymbol{\beta} \in \mathbb{R}^{I+1}}{\arg\min} \, \|\mathbf{B}\boldsymbol{\beta} - \mathbf{v}_\lambda^{d^\infty}\|, \tag{9.11}$$

where $\|\cdot\|$ is either the Euclidean norm

$$\|\mathbf{v}\|_2 := \left(\sum_{s \in S} v(s)^2 \right)^{\frac{1}{2}} = (\mathbf{v}^\mathsf{T}\mathbf{v})^{\frac{1}{2}}$$

[10] The statistical expression *estimator* is used, even though there is no randomness in the model.

[11] Of course if this quantity were known, there would no need to approximate it unless one intended to interpolate or extrapolate, as is the case when truncating a countable state model.

or the weighted Euclidean norm

$$\|\mathbf{v}\|_2^w := \left(\sum_{s\in S} w(s)v(s)^2\right)^{\frac{1}{2}} = (\mathbf{v}^\mathsf{T}\mathbf{W}\mathbf{v})^{\frac{1}{2}}, \tag{9.12}$$

where $w(s) > 0$ for all $s \in S$ and $\mathbf{W}$ is a diagonal matrix with entries $w(s)$ on the diagonal.

The ordinary least squares (OLS) parameter and policy value function estimates (equations (9.68) and (9.70) in the chapter Appendix) become

$$\hat{\boldsymbol{\beta}}_{\text{OLS}} = (\mathbf{B}^\mathsf{T}\mathbf{B})^{-1}\mathbf{B}^\mathsf{T}\mathbf{v}_\lambda^{d^\infty} = \boldsymbol{\Gamma}\mathbf{v}_\lambda^{d^\infty} = \boldsymbol{\Gamma}(\mathbf{I} - \lambda\mathbf{P}_d)^{-1}\mathbf{r}_d \tag{9.13}$$

and

$$\hat{\mathbf{v}}_{\text{OLS}} = \mathbf{B}(\mathbf{B}^\mathsf{T}\mathbf{B})^{-1}\mathbf{B}^\mathsf{T}\mathbf{v}_\lambda^{d^\infty} = \boldsymbol{\Pi}\mathbf{v}_\lambda^{d^\infty} = \boldsymbol{\Pi}(\mathbf{I} - \lambda\mathbf{P}_d)^{-1}\mathbf{r}_d, \tag{9.14}$$

respectively, where

$$\boldsymbol{\Gamma} := (\mathbf{B}^\mathsf{T}\mathbf{B})^{-1}\mathbf{B}^\mathsf{T} \quad \text{and} \quad \boldsymbol{\Pi} := \mathbf{B}\boldsymbol{\Gamma} = \mathbf{B}(\mathbf{B}^\mathsf{T}\mathbf{B})^{-1}\mathbf{B}^\mathsf{T}. \tag{9.15}$$

The matrix $\boldsymbol{\Pi}$ is the projection of $\mathbb{R}^M$ onto the $(I+1)$-dimensional subspace $\text{col}(\mathbf{B})$. Note that $\boldsymbol{\Gamma}\mathbf{v}$ is a vector of parameters and $\boldsymbol{\Pi}\mathbf{v}$ is a vector of values.

Note that this approach was used to illustrate concepts earlier in Example 9.1. It is often referred to as a *direct method*.

Direct methods will not be used in practice since the goal of approximate dynamic programming is to approximate the value function without first computing it.

Least squares estimation: $\mathbf{v}_\lambda^{d^\infty}$ unknown

In contrast to the direct method described above, *indirect methods* approximate a value function by appealing to the Bellman equation and a variant. Suppose $\mathbf{v}'$ is an estimate of $\mathbf{v}_\lambda^{d^\infty}$. Then its accuracy can be measured with respect to the Bellman equation according to

$$\|\mathbf{v}' - L_d\mathbf{v}'\| \tag{9.16}$$

or with respect to the *projected Bellman evaluation equation*

$$\mathbf{v} = \boldsymbol{\Pi}L_d\mathbf{v} \tag{9.17}$$

and the corresponding error

$$\|\mathbf{v}' - \boldsymbol{\Pi}L_d\mathbf{v}'\|, \tag{9.18}$$

where as above, the norm can be either the Euclidean or weighted Euclidean norm.

Thus, setting $\mathbf{v}' = \mathbf{B}\boldsymbol{\beta}'$, the two above approaches for choosing a value of $\boldsymbol{\beta}'$ lead to:

1. **Bellman[12] residual minimization:** Choose $\hat{\boldsymbol{\beta}}_{\text{BR}}$ according to

$$\hat{\boldsymbol{\beta}}_{\text{BR}} \in \underset{\boldsymbol{\beta} \in \mathbb{R}^{I+1}}{\arg\min} \|\mathbf{B}\boldsymbol{\beta} - L_d \mathbf{B}\boldsymbol{\beta}\| \tag{9.19}$$

2. **Projection error minimization:** Choose $\hat{\boldsymbol{\beta}}_{\text{PE}}$ according to

$$\hat{\boldsymbol{\beta}}_{\text{PE}} \in \underset{\boldsymbol{\beta} \in \mathbb{R}^{I+1}}{\arg\min} \|\mathbf{B}\boldsymbol{\beta} - \boldsymbol{\Pi} L_d \mathbf{B}\boldsymbol{\beta})\|, \tag{9.20}$$

where as above, the norms may be either the Euclidean or weighted Euclidean norm.

Geometric interpretation

Geometrically these two approaches minimize different quantities as shown in Figure 9.6. 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($\mathbf{B}$). On the other hand the projection error minimization approach minimizes the difference between $\mathbf{B}\boldsymbol{\beta}$ and $\boldsymbol{\Pi}L_d\mathbf{B}\boldsymbol{\beta}$, where both expressions lie in col($\mathbf{B}$). Note that least squares regression theory (see chapter appendix) establishes that $\boldsymbol{\Pi}L_d\mathbf{B}\boldsymbol{\beta}$ represents the element of col($\mathbf{B}$) closet to $L_d\mathbf{B}\boldsymbol{\beta}$ in the least squares sense.

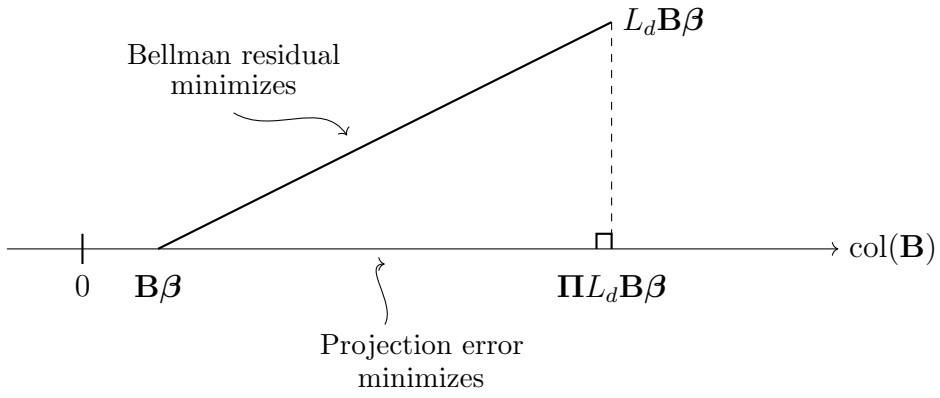


Figure 9.6: Graphical comparison of two approaches for estimating $\boldsymbol{\beta}$. This figure assumes that $\mathbf{B}$ contains a single column (corresponding to a single basis function) so that col($\mathbf{B}$) is a one-dimensional subspace of $\mathbb{R}^2$.

In principle, these two estimates can be found by varying $\boldsymbol{\beta}$ so as to make each of the distances as small as possible. In the one-dimensional example shown in Figure 9.6, this would correspond to altering the value of the scalar $\boldsymbol{\beta}$, or equivalently sliding $\mathbf{B}\beta$ along the horizontal axis and noting how $\|\mathbf{B}\boldsymbol{\beta} - L_d\mathbf{B}\boldsymbol{\beta}\|$ and $\|\mathbf{B}\boldsymbol{\beta} - \boldsymbol{\Pi}L_d\mathbf{B}\boldsymbol{\beta}\|$ change.

To distinguish these methods and gain familiarity with this notation, consider the following example.

[12]Note that as stated here, this approach corresponds to minimizing the error using the policy evaluation equation, which is a special case of the Bellman equation.

Example 9.3. Let $S = \{1, 2\}$,

$$\mathbf{r}_d = \begin{bmatrix} 2 \\ 8 \end{bmatrix} \quad \text{and} \quad \mathbf{P}_d = \begin{bmatrix} 0.25 & 0.75 \\ 0.10 & 0.90 \end{bmatrix}.$$

Choose a single basis function $b(s) = s$ so that $\mathbf{B} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$. Consequently $\boldsymbol{\beta}$ is a scalar denoted by β so that approximation of $v_\lambda^{d^\infty}(s)$ is of the form

$$\mathbf{B}\beta = \begin{bmatrix} \beta \\ 2\beta \end{bmatrix}.$$

The Bellman residual error is given by

$$\|\mathbf{B}\beta - L_d\mathbf{B}\beta\| = \left\| \begin{bmatrix} \beta \\ 2\beta \end{bmatrix} - \left(\begin{bmatrix} 2 \\ 8 \end{bmatrix} + \lambda \begin{bmatrix} 1.75\beta \\ 1.9\beta \end{bmatrix} \right) \right\| = \left\| \begin{bmatrix} (1 - 1.75\lambda)\beta - 2 \\ (2 - 1.90\lambda)\beta - 8 \end{bmatrix} \right\|.$$

On the other hand noting that

$$\mathbf{\Pi} = \mathbf{B}(\mathbf{B}^\mathsf{T}\mathbf{B})^{-1}\mathbf{B}^\mathsf{T} = \frac{1}{5}\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix},$$

the least squares projection error is given by

$$\left\| \begin{bmatrix} \beta \\ 2\beta \end{bmatrix} - \frac{1}{5}\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \left(\begin{bmatrix} 2 \\ 8 \end{bmatrix} + \lambda \begin{bmatrix} 1.75\beta \\ 1.90\beta \end{bmatrix} \right) \right\| = \left\| \begin{bmatrix} (1 - 1.11\lambda)\beta - 2.4 \\ (2 - 2.22\lambda)\beta - 4.8 \end{bmatrix} \right\|.$$

Setting $\lambda = 0.6$ and using the Euclidean norm gives

$$\mathbf{v}_\lambda^{d^\infty} = \begin{bmatrix} 18.40 \\ 25.10 \end{bmatrix}, \quad \mathbf{\Pi}\mathbf{v}_\lambda^{d^\infty} = \begin{bmatrix} 10.09 \\ 20.18 \end{bmatrix}, \quad \mathbf{B}\hat{\beta}_{\text{BR}} = \begin{bmatrix} 9.14 \\ 18.27 \end{bmatrix} \quad \text{and} \quad \mathbf{B}\hat{\beta}_{\text{PE}} = \begin{bmatrix} 10.78 \\ 21.55 \end{bmatrix}.$$

Because the value function is **not** in col($\mathbf{B}$) neither value function estimate accurately approximates $\mathbf{v}_\lambda^{d^\infty}$. However, the estimate based on projection error minimization provides a better estimate of the projection of the value function on col($\mathbf{B}$) as represented by $\mathbf{\Pi}\mathbf{v}_\lambda^{d^\infty}$.

9.3.1 Least squares policy evaluation

This section describes a flexible iterative algorithm for projection error minimization referred to as *least squares policy evaluation (LSPE)*. It is based on iteratively solving the *projected* policy evaluation equation $\mathbf{v} = \mathbf{\Pi}L_d\mathbf{v}$. Its convergence depends on contraction properties of $\mathbf{\Pi}L_d$. Bellman residual minimization will be briefly discussed in a subsequent starred section and not be explored beyond that.

The following iterative algorithm (stated in matrix form) corresponds to applying value iteration to a fixed decision rule. 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 as described below, and
4. easily generalizes to nonlinear architectures and optimization problems.

Note that the algorithm is structured so that value function approximations remain in col($\mathbf{B}$).

Algorithm 9.1. Least squares policy evaluation: linear architectures

1. **Initialize:** Specify $\boldsymbol{\beta}$ arbitrary, $\epsilon > 0$, and $\Delta > \epsilon$.
2. **Iterate:** While $\Delta \geq \epsilon$:
   (a) $\mathbf{v}' \leftarrow \mathbf{B}\boldsymbol{\beta}$.
   (b) $\mathbf{v} \leftarrow \mathbf{r}_d + \lambda \mathbf{P}_d \mathbf{v}'$.
   (c) $\boldsymbol{\beta}' \leftarrow \boldsymbol{\Gamma}\mathbf{v}$[a].
   (d) $\Delta \leftarrow \|\boldsymbol{\beta}' - \boldsymbol{\beta}\|_2$.
   (e) $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$.
3. **Terminate:** Return $\boldsymbol{\beta}$.

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

Closed form representation of the LSPE estimate

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

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

Step 2(c) represents projection (least squares regression of $\mathbf{v}$ on the basis vectors) on col($\mathbf{B}$) 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 it follows that LSPE generates $\boldsymbol{\beta}'$ from $\boldsymbol{\beta}$ according to

$$\boldsymbol{\beta}' = \boldsymbol{\Gamma}\Big(\mathbf{r}_d + \lambda \mathbf{P}_d \mathbf{B}\boldsymbol{\beta}\Big) = \boldsymbol{\Gamma}\mathbf{r}_d + \lambda \boldsymbol{\Gamma}\mathbf{P}_d\mathbf{B}\boldsymbol{\beta}. \tag{9.21}$$

As a result of (9.21), if $\lambda\boldsymbol{\Gamma}\mathbf{P}_d\mathbf{B}$ is a contraction, LSPE converges to the solution of

$$(\mathbf{I} - \lambda\boldsymbol{\Gamma}\mathbf{P}_d\mathbf{B})\boldsymbol{\beta} = \boldsymbol{\Gamma}\mathbf{r}_d, \tag{9.22}$$

where $\mathbf{I}$ is the $(I+1)\times(I+1)$ identity matrix. In this case, $(\mathbf{I} - \lambda\boldsymbol{\Gamma}\mathbf{P}_d\mathbf{B})$ is invertible so that the fixed point of the projection equation can be represented by

$$\hat{\boldsymbol{\beta}} = (\mathbf{I} - \lambda\boldsymbol{\Gamma}\mathbf{P}_d\mathbf{B})^{-1}\boldsymbol{\Gamma}\mathbf{r}_d. \tag{9.23}$$

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

$$\mathbf{v}' = \mathbf{B}\boldsymbol{\Gamma}(\mathbf{r}_d + \lambda\mathbf{P}_d\mathbf{v}) = \boldsymbol{\Pi}(\mathbf{r}_d + \lambda\mathbf{P}_d\mathbf{v}) = \boldsymbol{\Pi}L_d\mathbf{v} \tag{9.24}$$

in agreement with Figure 9.6 where $\mathbf{v} = \mathbf{B}\boldsymbol{\beta}$ and $\mathbf{v}' = \boldsymbol{\Pi}L_d\mathbf{B}\boldsymbol{\beta}$. As a result of this observation, the above algorithm may also be referred to as "projected value iteration".

Examples

The following example[13] shows that the inverse in (9.23) need not exist.

Example 9.4. Let $S = \{s_1, s_2\}$, $\mathbf{r}_d$ be arbitrary, $\lambda = 5/5.4 = 0.926$,

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

Hence

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

so that

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

This implies that $\lambda\boldsymbol{\Gamma}\mathbf{P}_d\mathbf{B}$ is not a contraction so that $\mathbf{I} - \lambda\boldsymbol{\Gamma}\mathbf{P}_d\mathbf{B}$ is not invertible.

In this example, the iterative scheme implicit in (9.21) reduces to

$$\boldsymbol{\beta}' \leftarrow \boldsymbol{\Gamma}\mathbf{r}_d + \boldsymbol{\beta}.$$

Hence, if $\boldsymbol{\Gamma}\mathbf{r}_d \neq \mathbf{0}$, LSPE diverges and if $\boldsymbol{\Gamma}\mathbf{r}_d = \mathbf{0}$, it does not uniquely determine $\boldsymbol{\beta}$.

[13] This example appears in de Farias and Roy [2000].

This example is an *edge case* that is unlikely to occur in practice. In most settings, LSPE will converge as shown in the example below. Note that under the conditions that $\mathbf{P}_d$ is recurrent and that the basis vectors are linearly independent, $\mathbf{\Pi}L_d$ is a contraction mapping[14] with respect to the weighted Euclidean norm (9.12) with weight function equal to the (positive) steady state distribution of $\mathbf{P}_d$.

Example 9.5. This example illustrates the calculations described above in the context of the queuing service rate control model analyzed in Example 9.1. It approximates the value function of the previously specified stationary policy d^∞ by linear and quadratic functions of the state and compares the estimates based on LSPE (or its fixed point representation based on (9.23)) to the exact value function and its direct OLS estimate.

Set $c(s,a) := f(s) + m(a)$. Step 2(b) of LSPE is implemented component-wise using the expressions:

$$v(0) = c(0, d(0)) + \lambda((1-b)v'(0) + bv'(1)),$$

$$v(50) = c(50, d(50)) + \lambda(d(50)v'(49) + (1 - d(50))v'(50)) \tag{9.25}$$

and

$$v(s) = c(s, d(s)) + \lambda((1-b)v'(s-1) + (1-b-d(s))v'(s) + d(s)v'(s)),$$

for $s = 1, \ldots, 49$ where, $d(s)$ denotes the service rate in state s.

Note that there is no need to distinguish the truncation state 50 as in (9.25) since the approximation $v'(51)$ is available. Choosing such an approach may be regarding as evaluating the infinite state model with $S = \{0, 1, \ldots\}$ by an approximation of $\boldsymbol{\beta}$ based on the reduced set of states $S' = \{0, \ldots, 50\}$.

The analysis below chooses $v'(s)$ to be either the linear approximation

$$v'(s) = \beta_0 + \beta_1 s$$

or the quadratic approximation

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

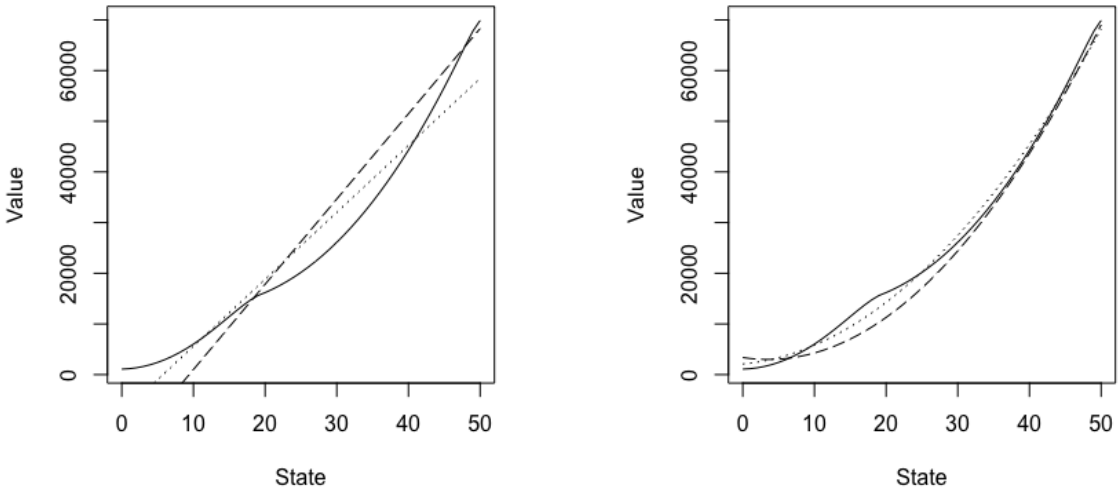
Applying LSPE for the linear approximation yields $\hat{\beta}_0 = -15825.3$ and $\hat{\beta}_1 = 1682.6$ compared to OLS values $\hat{\beta}_0 = -7603.3$ and $\hat{\beta}_1 = 1320.9$. Graphical comparisons are provided in Figure 9.7a. The main takeaways are that these two estimates are quite different and neither approximates $v_\lambda^{d^\infty}(s)$ well.

Using the quadratic approximation, 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 $\hat{\beta}_2 = 23.8$.

[14] See Bertsekas [2012], pp. 423-427.

Graphical comparisons of fitted values appear in Figure 9.7b. 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.

Note that improved fits may be possible by taking into account that $d(s)$ has a step change at $s = 20$. However letting the form of a particular policy determine the approximation will have downsides when turning to optimization. It is left to the reader to compare approximations that use cubic polynomials or a splines with knots at 20.



(a) Linear approximation.

(b) Quadratic approximation.

Figure 9.7: Linear and quadratic approximations to the exact value function for the specified policy in Example 9.5. 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.

Using subsets of S in LSPE and other iterative algorithms

The numerical examples above applied LSPE using basis functions and updates defined for **all** $s \in S$. But in practice, especially when $S = \{s_1, \ldots, s_M\}$ is large, one would seek to obtain estimates based on basis functions and updates using a small subset $S' = \{s'_1, \ldots, s'_n\} \subset S$. This affects Algorithm 9.1 and all other iterative methods in this chapter as follows.

To begin with, $\mathbf{B}$ has n rows with entries in each column corresponding to the values of basis functions on S'. Then instead of applying steps 2(a) and 2(b) for all $s \in S$, the algorithm generates $v(s')$ for $s' \in S'$ according to

$$v(s') \leftarrow r\big(s', d(s')\big) + \lambda \sum_{j \in S} \left(p\big(j|s', d(s')\big) \sum_{i=0}^{I} \beta_i b_i(j) \right). \tag{9.26}$$

Note that summation over j involves states that are not in S'. Therefore in such states the update uses the current value of $\boldsymbol{\beta}$ to approximate the value in these states by $\sum_{i=0}^{I} \beta_i b_i(j)$. Fortunately in most practical examples, such as queuing control or Gridworld navigation, $p(j|s', d(s')) > 0$ is positive for only a small number of successor states so computing this sum is not burdensome.

Applying (9.26) for all $s' \in S'$ results in a "data set" suitable for regression. This data, in matrix form, consists of

$$\mathbf{v} = \begin{bmatrix} v(s'_1) \\ \vdots \\ v(s'_n) \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} b_0(s'_1) & \dots & b_I(s'_1) \\ \vdots & \vdots & \vdots \\ b_0(s'_n) & \dots & b_I(s'_n) \end{bmatrix}.$$

Thus estimates of $\boldsymbol{\beta}'$ can be obtained in closed form as

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

or by using a regression package. In the nonlinear setting, $f(b_0(j), b_1(j), \dots, b_I(j); \boldsymbol{\beta})$ replaces $\sum_{i=0}^{I} \beta_i b_i(j)$ in the update.

Bellman residual minimization*

As noted above, Bellman residual minimization obtains 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

$$(\mathbf{I} - \lambda\mathbf{P}_d)\mathbf{B}\boldsymbol{\beta} = (\mathbf{B} - \lambda\mathbf{P}_d\mathbf{B})\boldsymbol{\beta} \approx \mathbf{r}_d. \tag{9.27}$$

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 (9.27) become

$$\hat{\mathbf{v}} = \mathbf{G}_d(\mathbf{G}_d^\mathsf{T}\mathbf{G}_d)^{-1}\mathbf{G}_d^\mathsf{T}\mathbf{r}_d \tag{9.28}$$

and

$$\hat{\boldsymbol{\beta}}_{\text{BR}} = (\mathbf{G}_d^\mathsf{T}\mathbf{G}_d)^{-1}\mathbf{G}_d^\mathsf{T}\mathbf{r}_d. \tag{9.29}$$

Note that in the above, the matrix $\mathbf{G}_d = \mathbf{B} - \lambda\mathbf{P}_d\mathbf{B}$ is $M \times (I+1)$ so that $\mathbf{G}_d^\mathsf{T}\mathbf{G}_d$ is $(I+1) \times (I+1)$.

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

$$\mathbf{G}_d^\mathsf{T}\mathbf{G}_d\hat{\boldsymbol{\beta}} = \mathbf{G}_d^\mathsf{T}\mathbf{r}_d. \tag{9.30}$$

directly.

Analysis of the quality of this approximation is left to the reader.

9.3.2 Linear programming approximations for policy value functions

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

Approximate linear programming – primal model

Chapter 5 formulated the following (primal) LP to determine optimal value functions:

$$\begin{aligned} \text{minimize} \quad & \sum_{s\in S} \alpha(s)v(s) \\ \text{subject to} \quad & v(s) - \lambda \sum_{j\in S} p(j|s,a)v(j) \geq r(s,a), \quad a \in A_s, s \in S. \end{aligned}$$

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

$$\text{minimize} \quad \sum_{s\in S} \alpha(s)v(s) \tag{9.31a}$$

$$\text{subject to} \quad v(s) - \lambda \sum_{j\in S} p(j|s,d(s))v(j) \geq r(s,d(s)), \quad s \in S. \tag{9.31b}$$

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

$$v(s) = \beta_0 b_0(s) + \beta_1 b_1(s) + \ldots + \beta_I b_I(s).$$

Substituting this expression into (9.31a) and (9.31b) and rearranging terms gives the approximate primal linear program:

Approximate primal linear program (APLP) – fixed decision rule

$$\text{minimize} \quad \sum_{i=0}^{I} \beta_i \sum_{s\in S} \alpha(s) b_i(s) \tag{9.32a}$$

$$\text{subject to} \quad \sum_{i=0}^{I} \beta_i b_i(s) - \lambda \left(\sum_{i=0}^{I} \beta_i \sum_{j\in S} p(j|s,d(s)) b_i(j) \right) \geq r(s,d(s)), \quad s \in S. \tag{9.32b}$$

Observe that this LP has $I+1$ variables $\beta_0, \beta_1, \ldots, \beta_I$ and $|S|$ constraints. Since in most applications $|S| \gg I$, this system has many more constraints than variables. The following example illustrates simple cases of this approximation and provides insight into the geometry underlying the LP approximation.

Example 9.6. 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. Assuming $\sum_{s\in S} \alpha(s) = 1$, the APLP becomes

$$\begin{aligned} \text{minimize} \quad & \beta_0 \\ \text{subject to} \quad & (1-\lambda)\beta_0 \geq r(s, d(s)), \quad s \in S. \end{aligned} \tag{9.33}$$

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

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

Now consider as a special case the two-state ($S = \{0, 1\}$) version of the queuing service rate control model previously analyzed in this chapter. 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 the LP formulation, represent it as a reward maximization problem with $r(0, d(0)) = -5$ and $r(1, d(1)) = -41$.

Therefore the approximate linear program (9.33) 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 $\hat{\beta}_0^{\text{LP}} = -10$. Consequently the LP approximation to the value function is $\hat{v}_{\text{LP}}(0) = \hat{v}_{\text{LP}}(1) = -10$. Note that if S were instead truncated at $N = 50$, there would be 51 inequalities of a similar form to those above.

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, $\hat{\beta}_0^{\text{OLS}} = -41.5$, so that the least squares approximation is $\hat{v}_{\text{OLS}}(0) = \hat{v}_{\text{OLS}}(1) = -41.5$.

Figure 9.8 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, this cannot be known priori, so care must be taken when choosing basis functions.

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

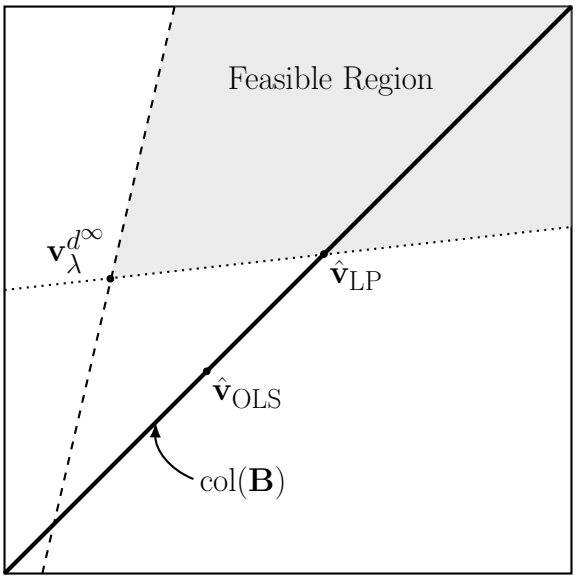


Figure 9.8: Graphical representation of the value function and two approximations for Example 9.6. It shows the feasible region for the exact LP, its solution $\mathbf{v}_\lambda^{d^\infty}$, and the subspace col($\mathbf{B}$). The quantities $\hat{\mathbf{v}}_{\text{LP}}$ and $\hat{\mathbf{v}}_{\text{OLS}}$ denote the LP and OLS estimates in that subspace. Since the LP minimizes β_0, its value is the minimum value of the intersection of col($\mathbf{B}$) and the feasible region of the LP. The OLS estimate is the orthogonal projection of the exact value function on to that subspace.

The approximate LP model will be discussed further below when it is used to find approximate optimal values.

9.4 Parameter estimation for a fixed policy: Nonlinear architectures

This section modifies Algorithm 9.1 to use nonlinear architectures. This requires replacing computation of $\mathbf{v}'$ in step 2(a) by

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

written as $\mathbf{v}' \leftarrow \mathbf{f}(\boldsymbol{\beta})$. Step 2(c), replaces the computation of $\mathbf{v}'$ by its least squares estimate obtained using Gauss-Newton iteration or some other method expressed 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 9.2. Least squares policy evaluation: nonlinear architectures

1. **Initialize:** Specify $\boldsymbol{\beta}$ arbitrary, $\epsilon > 0$, and $\Delta > \epsilon$.
2. **Iterate:** While $\Delta \geq \epsilon$:
   (a) $\mathbf{v}' \leftarrow \mathbf{f}(\boldsymbol{\beta})$.
   (b) $\mathbf{v} \leftarrow \mathbf{r}_d + \lambda \mathbf{P}_d \mathbf{v}'$.
   (c) $\boldsymbol{\beta}' \in \arg\min_{\boldsymbol{\beta}} \|\mathbf{f}(\boldsymbol{\beta}) - \mathbf{v}\|_2$.
   (d) $\Delta \leftarrow \|\boldsymbol{\beta}' - \boldsymbol{\beta}\|_2$.
   (e) $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$.
3. **Terminate:** Return $\boldsymbol{\beta}$.

The following example applies Algorithm 9.2 to fit a nonlinear exponential approximation to the value function in the queuing service rate control model.

Example 9.7. This example illustrates the nonlinear LSPE algorithm above in the queuing service rate control model in Example 9.1. In it, the value function for the specified stationary policy d^∞ is approximated by an exponential function. It compares the estimate based on LSPE to the exponential fit when the value function is known. To implement the algorithm, 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 $\mathbf{v} = \mathbf{r}_d$. Step 2(c) used the *nls* function in R. Estimation required some adjustment of starting values to ensure convergence at every iteration of step 2(c).

With $\epsilon = 0.001$, the algorithm terminated in 386 iterations with the approxi-

mation

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

which is shown in Figure 9.9. Observe that $\hat{\beta}_1$ and $\hat{\beta}_2$ differ from that in Example 9.1 but $\hat{\beta}_3$ is almost identical to that obtained when the value function was known. Moreover the approximation was considerably more accurate than those based on linear and quadratic approximations.

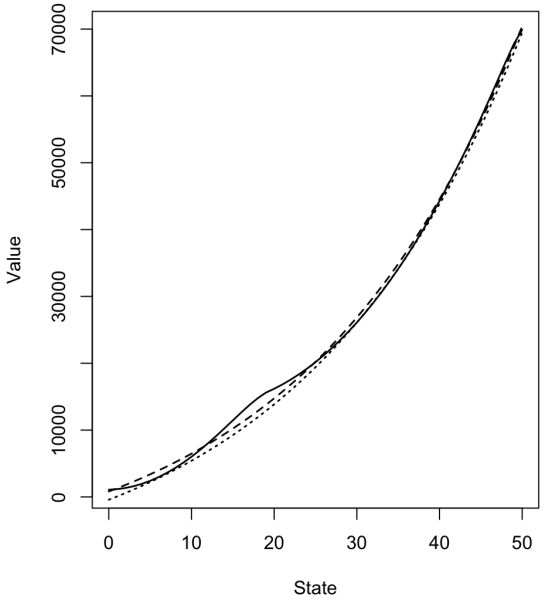


Figure 9.9: Value function (solid line) for queuing service rate control model with nonlinear least squares approximation (dashed line) and nonlinear LSPE approximation (dotted line).

This example suggest that applying nonlinear architectures requires considerably more care than in the linear case because of the sensitivity of nonlinear 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.

Note that Algorithm 9.2 can also be used to fit a neural network iteratively. In this case step 2(b) would require repeated training of the neural network.

9.5 Combining approximation and optimization

This section generalizes the previous section by providing methods that seek good policies in a Markov decision process based on approximations to the optimal value

function. Such methods are usually referred to as approximate dynamic programming (ADP). It presents value iteration, policy iteration and modified policy iteration algorithms based on linear, nonlinear and neural network architectures and a linear programming approach, 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.

9.5.1 Iterative methods for linear architectures

The following algorithm is an optimization version of LSPE. It is then generalized to obtain policy iteration and modified policy iteration algorithms. The approaches all assumes a pre-specified set of basis functions and a set of states at which to evaluate them so that the matrix $\mathbf{B}$ can be specified a priori.

Least squares value iteration (LSVI) – linear architectures

The following algorithm generalizes Algorithm 9.1 to incorporate optimization. The only change is in step 2(b) where the Bellman operator replaces the one-step policy update $\mathbf{v} \leftarrow \mathbf{r}_d + \lambda \mathbf{P}_d \mathbf{v}'$.

Algorithm 9.3. Least squares value iteration: linear architectures

1. **Initialize:** Specify $\boldsymbol{\beta}$ arbitrary, $\epsilon > 0$, and $\Delta > \epsilon$.
2. **Iterate:** While $\Delta \geq \epsilon$:
   (a) $\mathbf{v}' \leftarrow \mathbf{B}\boldsymbol{\beta}$.
   (b) $\mathbf{v} \leftarrow \text{c-max}_{d \in D^{\text{MD}}} \{\mathbf{r}_d + \lambda \mathbf{P}_d \mathbf{v}'\}$.
   (c) $\boldsymbol{\beta}' \leftarrow \boldsymbol{\Gamma}\mathbf{v}$.
   (d) $\Delta \leftarrow \|\boldsymbol{\beta}' - \boldsymbol{\beta}\|_2$.
   (e) $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$.
3. **Terminate:** Return $\boldsymbol{\beta}$.

As noted in Example 9.4 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 (LSPI) – linear architectures

This section describes a policy iteration method for finding "good" policies. Instead of choosing the value in step 2(b) of Algorithm 9.3, it obtains the arg c-max, evaluates the resulting policy and then finds the best approximation. The algorithm can be initiated with either:

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

The order of steps in the first pass through the algorithm depends on how it is initialized. With linear architectures it appeared best to begin with a parameter or value; with nonlinear architectures, methods based on initialization with a value seemed to work best. In the latter case, the first pass through step 2 begins with an improvement.

Algorithm 9.4. Least squares policy iteration: linear architectures

1. **Initialize:** Specify $d' \in D^{\text{MD}}$, $\boldsymbol{\beta}$ arbitrary, $\epsilon > 0$, and $\Delta > \epsilon$.
2. **Iterate:** While $\Delta \geq \epsilon$:
   (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:** $\mathbf{v}' \leftarrow \mathbf{B}\boldsymbol{\beta}'$.
   (c) **Improvement:**
   $$d' \leftarrow \underset{d \in D^{\text{MD}}}{\arg\,\text{c-max}} \{\mathbf{r}_d + \lambda \mathbf{P}_d \mathbf{v}'\}.$$
   (d) $\Delta \leftarrow \|\boldsymbol{\beta}' - \boldsymbol{\beta}\|_2$.
   (e) $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$.
3. **Terminate:** Return d' and $\boldsymbol{\beta}$.

Some comments about this algorithm follow.

1. This algorithm can be applied to a subset of S using methods described in the section on LSPE.
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. Our calculations found no difference in algorithmic behavior between these criteria in examples. Note if a stopping criterion is based on decision rules, the specification "set $d' = d$ if possible" to avoid cycling needs to be added.

3. Note that the only reason to carry $\boldsymbol{\beta}'$ forward in step 2(e) is to evaluate the stopping criterion at the next iteration, it is not used in steps 2(a)-2(c).

4. Step 2(a) provides the closed-form representation for $\boldsymbol{\beta}$ 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, this step would be replaced by LSPE.

5. Recall that the matrix $\boldsymbol{\Gamma} = (\mathbf{B}^\mathsf{T}\mathbf{B})^{-1}\mathbf{B}^\mathsf{T}$.

6. When the algorithm is initialized with a starting value for $\boldsymbol{\beta}$, 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.

Example 9.8. This example applies LSPI to the queuing service rate control model with $S = \{0, \ldots, 50\}$. It compares linear, quadratic and cubic polynomial approximations. The algorithm was initiated with the previously considered decision rule and used $\epsilon = 0.0001$ for the stopping criterion. All instances required 3 iterations for convergence.

Figure 9.10 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. The policy identified by the cubic approximation was optimal.

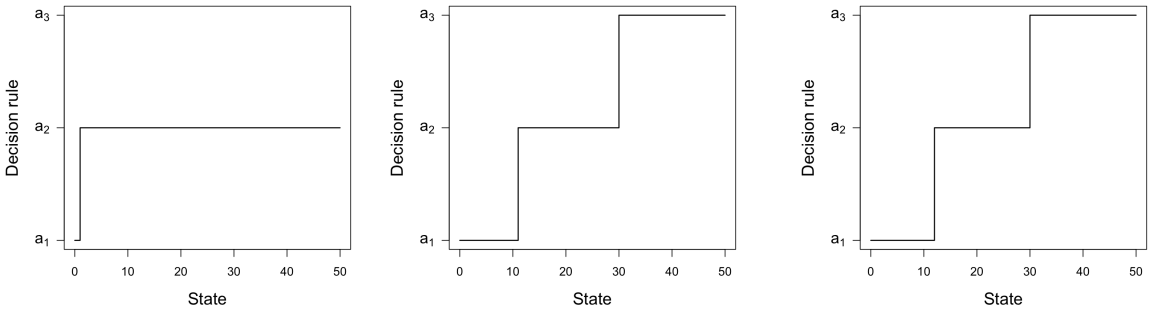
A larger model: Additional calculations revealed that the policies identified by the cubic approximation agreed with the optimal policy for $S = \{0, \ldots, 500\}$ and $\lambda = 0.9$[a]. When $\lambda = 0.98$, the policy obtained from the cubic approximation d^∞_{cub} and the optimal policy d^∞_{opt} differ as follows:

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

To gain further insight into the effect of approximation, the values of these two policies were compared. Note that LSPI does not provide the value of d^∞_{cub}; this requires computing $(\mathbf{I} - \lambda \mathbf{P}_{d_{\text{cub}}})^{-1}\mathbf{r}_{d_{\text{cub}}}$. These calculations show that values

differ most at low occupancy levels. Bounds in the next subsection will provide further insight. However, it appears that more accurate approximations may be needed when λ is close to 1.

[a]When solving the larger instance, the approximating polynomials were centered 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, this step could be replaced by iterative policy evaluation such as in Algorithm 9.5 below.



(a) Linear (b) Quadratic (c) Cubic

Figure 9.10: Stationary policies obtained using LSPI with linear, quadratic and cubic approximations. The policy obtained using a cubic approximation was optimal. In these figures, $S = \{0, \ldots, 50\}$ and $\lambda = 0.9$.

The following example shows that LSPI may not converge when using linear approximations.

Example 9.9. This example applies LSPI to the two-state example from Section 2.5 using the single basis function $b(s_1) = 0.05$ and $b(s_2) = 1$, and setting $\lambda = 0.9$. In this case,

$$\mathbf{B} = \begin{bmatrix} 0.05 \\ 1 \end{bmatrix} \quad \text{and} \quad \mathbf{B}\beta = \begin{bmatrix} 0.05\beta \\ \beta \end{bmatrix}.$$

The Bellman update (in component notation) applied to the approximation becomes:

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

It is left as an 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 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.

On the other hand, for other basis functions and starting values of β 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 cycling phenomenon is explored further in Bertsekas [2011] and Bertsekas [2012].

[a] In LSVI, the resulting policy is defined as the greedy decision rule based on the value approximation, once a convergence criterion is met.

LSPI bounds

This section applies the bounds from Section 5.5.3 to LSPI. Step 2(c) of the algorithm implicitly computes the quantity

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

Therefore with minimal additional computation the bounds from Theorem 5.12 can be applied to obtain

$$\begin{aligned} \mathbf{v}' + \frac{1}{(1-\lambda)}\underline{(L\mathbf{v}' - \mathbf{v}')}\mathbf{e} \leq L\mathbf{v}' + \frac{\lambda}{(1-\lambda)}\underline{(L\mathbf{v}' - \mathbf{v}')}\mathbf{e} \leq \mathbf{v}_\lambda^{(d')^\infty} \leq \mathbf{v}_\lambda^* \\ \leq L\mathbf{v}' + \frac{\lambda}{(1-\lambda)}\overline{(L\mathbf{v}' - \mathbf{v}')}\mathbf{e} \leq \mathbf{v}' + \frac{1}{(1-\lambda)}\overline{(L\mathbf{v}' - \mathbf{v}')}\mathbf{e}, \end{aligned} \tag{9.34}$$

where as before $\overline{\mathbf{u}} = \max_{s \in S} u(s)$, $\underline{\mathbf{u}} = \min_{s \in S} u(s)$ and

$$d' \in \arg \underset{d \in D^{\mathrm{MD}}}{\text{c-max}}\{\mathbf{r}_d + \lambda \mathbf{P}_d \mathbf{v}'\}.$$

Most importantly, since $\mathbf{v}_\lambda^{(d')^\infty}$ is never computed using LSPI[15], the following bound provides a measure of how close it is to the optimal value:

$$\|\mathbf{v}_\lambda^{(d')^\infty} - \mathbf{v}_\lambda^*\| \leq \frac{\lambda}{(1-\lambda)}\mathrm{sp}(L\mathbf{v}' - \mathbf{v}'), \tag{9.35}$$

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

$$L\mathbf{v}' + \frac{\lambda}{2(1-\lambda)}\mathrm{sp}(L\mathbf{v}' - \mathbf{v}')\mathbf{e} \approx \mathbf{v}_\lambda^*. \tag{9.36}$$

[15] It only computes the approximation $\mathbf{v}' = \mathbf{B}\boldsymbol{\beta}'$.

The beauty of these bounds is that they can be applied to any $\mathbf{v}'$ and $L\mathbf{v}'$ regardless of how they have been obtained. Moreover, (9.35) provides an alternative stopping criterion for LSPI (or LSVI). However, if the stopping criterion is too strict or the basis functions are inappropriate, it need not be attained. Also, the factor of $(1-\lambda)^{-1}$ may result in the bounds not being very tight.

The following example applies these bounds to the queuing control example.

Example 9.10. This example computes the bounds in (9.34) for the queuing service rate control model with capacity 50.

Since $\mathbf{v}'$ is computed in step 2(b) and $L\mathbf{v}'$ can be easily obtained from step 2(c), the quantity $L\mathbf{v}'-\mathbf{v}'$ and all of the bounds can be easily computed. Using a cubic approximation, at termination of LSPI, $\underline{L\mathbf{v}'-\mathbf{v}'} = -113.9$ and $\overline{L\mathbf{v}'-\mathbf{v}'} = 38.2$. Using these values gives the inner bounds in (9.34), which are shown in Figure 9.11b.

Since $\text{sp}(L\mathbf{v}'-\mathbf{v}') = 152.1$, (9.35) yields the estimate

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

Since in fact $(d')^\infty$ is optimal, this bound is not very accurate. However, the average absolute error of the approximation in (9.36) is 38.8 indicating that the approximation based on (9.36) is very accurate.

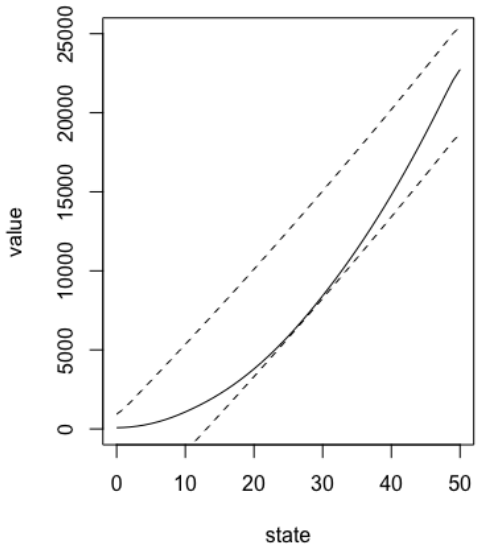
Further calculations show that bounds based on a quadratic approximation are only slightly less tight than those based on the cubic approximation but, those based on a linear approximation are considerably less accurate (Figure 9.11a).

Least squares modified policy iteration – linear architectures

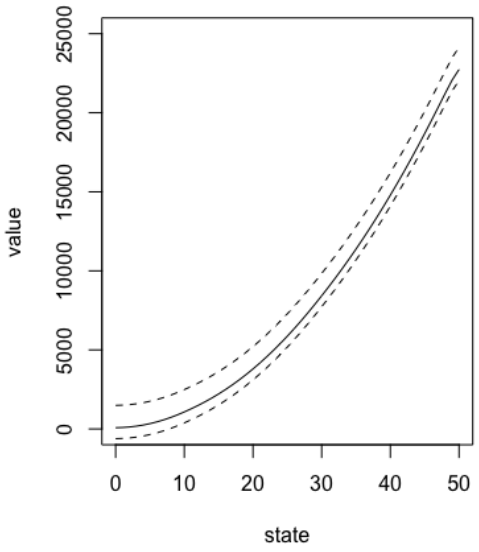
In large applications, it might be challenging to construct and solve the linear system implicit in step 2(a) of the LSPI iteration algorithm. The following generalization of MPI, referred to as *least squares modified policy iteration* and denoted by LSMPI addresses this issue.

Algorithm 9.5. Least squares modified policy iteration: linear architectures

1. **Initialize:** Specify $M \in \mathbb{Z}_+$, $\boldsymbol{\beta}$ arbitrary, $\epsilon > 0$, and $\Delta > \epsilon$.
2. **Iterate:** While $\Delta \geq \epsilon$:
   (a) $\mathbf{v} \leftarrow \mathbf{B}\boldsymbol{\beta}$.



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



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

Figure 9.11: Bounds (dashed lines) for value function (solid line) derived from the LSPI solution using inner inequalities in (9.34).

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

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

(d) $m \leftarrow 1$.

(e) While $m \leq M$:

i. $\boldsymbol{\beta}' \leftarrow \boldsymbol{\Gamma} \mathbf{v}'$.

ii. $\mathbf{v}' \leftarrow \mathbf{r}_{d'} + \lambda \mathbf{P}_{d'} \mathbf{B} \boldsymbol{\beta}'$.

iii. $m \leftarrow m + 1$.

(f) $\boldsymbol{\beta}' \leftarrow \boldsymbol{\Gamma} \mathbf{v}'$.

(g) $\Delta \leftarrow \|\boldsymbol{\beta}' - \boldsymbol{\beta}\|_2$.

(h) $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$.

3. **Terminate:** Return d' and $\boldsymbol{\beta}$.

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(e) corresponds to terminating LSPE after $M+1$ iterations. If $M=0$, step 2(e) is skipped and the algorithm is identical LSVI and if $M=\infty$, it is equivalent to LSPI.

3. The integer M denotes the order of the algorithm. It can vary from iteration to iteration. See the discussion in Chapter 5 for guidance on selection of M. Choosing M adaptively to ensure that the norm of successive values of $\boldsymbol{\beta}'$ computed in 2(e)i is less than some pre-specified small tolerance is equivalent to LSPI.

4. It may be easier (especially when applying to LSMPI to nonlinear architectures[16]) to initialize the algorithm with some $\tilde{\mathbf{v}}$ instead of $\boldsymbol{\beta}$. In this case step 2(a) is not necessary at the first pass through the step 2.

5. Steps 2(b) and 2(c) are implemented together avoiding double computation.

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

7. The concluding observations in Example 9.9 apply as well to LSMPI.

9.5.2 Iterative methods for nonlinear architectures

This brief section generalizes LSVI and LSPI by developing similar methods for finding good policies using nonlinear approximations based on features.

Least squares value iteration – nonlinear architectures

This section modifies the LSVI algorithm (Algorithm 9.3) to allow for nonlinear approximation architectures $\mathbf{v}\approx\mathbf{f}(\boldsymbol{\beta})$. The main difference with the linear architecture variant is that the estimate of $\boldsymbol{\beta}$ cannot be obtained in closed form. That is, the closed form representation in step 2(c) of Algorithm 9.3 is replaced by an estimate derived by nonlinear least squares. This step is implemented using appropriate code.

Algorithm 9.6. Least squares value iteration: nonlinear architectures

1. **Initialize:** Specify $\boldsymbol{\beta}$ arbitrary, $\epsilon>0$, and $\Delta>\epsilon$.

2. **Iterate:** While $\Delta\geq\epsilon$:

   (a) $\mathbf{v}'\leftarrow\mathbf{f}(\boldsymbol{\beta})$.

[16] In complicated nonlinear models such as neural networks, it may be challenging to accurately estimate parameter values.

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

(c) $\boldsymbol{\beta}' \in \arg\min_{\boldsymbol{\beta}} \|\mathbf{f}(\boldsymbol{\beta}) - \mathbf{v}\|_2$.

(d) $\Delta \leftarrow \|\boldsymbol{\beta}' - \boldsymbol{\beta}\|_2$.

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

3. **Terminate:** Return $\boldsymbol{\beta}$.

Least squares policy iteration – nonlinear architectures

The following generalizes LSPI (Algorithm 9.4) to nonlinear architectures.

Algorithm 9.7. Least squares policy iteration: nonlinear architectures

1. **Initialize:** Specify $d \in D^{\text{MD}}$, $\boldsymbol{\beta}$ arbitrary, $\epsilon > 0$, and $\Delta > \epsilon$.
2. **Iterate:** While $\Delta \geq \epsilon$:

(a) $\mathbf{v}' \leftarrow \mathbf{f}(\boldsymbol{\beta})$.

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

(c) Apply LSPE using d' to return $\boldsymbol{\beta}'$.

(d) $\Delta \leftarrow \|\boldsymbol{\beta}' - \boldsymbol{\beta}\|_2$.

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

3. **Terminate:** Return d' and $\boldsymbol{\beta}$.

Note that step 2(c) produces a parameter estimate corresponding to policy $(d')^\infty$. Its value is computed at the next iteration by $\mathbf{f}(\boldsymbol{\beta})$. Note also that both of these algorithms can be applied on subsets of S using the modification described in the LSPE section.

The following example illustrates the use of nonlinear LSPI in the queuing service rate control model.

Example 9.11. As in Example 9.7 the value function is approximated by

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

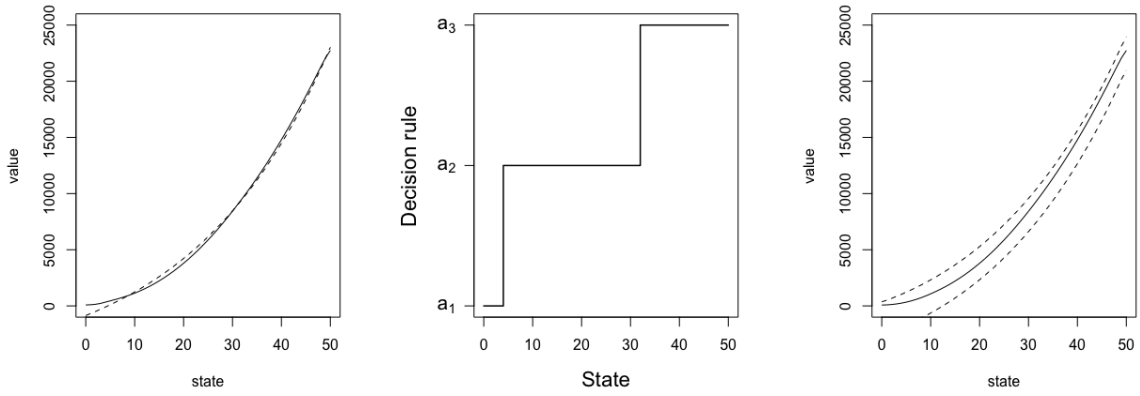
and $\lambda = 0.9$.

Applying the stopping criterion "stop if $d' = d$", nonlinear 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 $\|\boldsymbol{\beta} - \boldsymbol{\beta}'\|_2$ would also terminate after three iterations.

Figure 9.12a shows the optimal value function and the exponential approximation found with Algorithm 9.7. Figure 9.12b shows the stationary policy based on the optimal exponential approximation. Figure 9.12c shows bounds on the optimal value function based on the exponential approximation.

Note 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 9.10). Therefore one can conclude that the cubic polynomial approximation is preferable for this instance.



(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.

Figure 9.12: Value function, policy and bounds for Example 9.11.

9.5.3 Optimization using linear programming: Linear architectures

The linear programming approach in Section 9.3.2 was used to approximate the policy value function of a fixed policy. This section extends this approach to obtain approximations to the optimal value function. As noted above, this estimate differs from the least squares estimates obtained using LSVI, LSPI or LSMPI. And of course, this approach is only appropriate for approximations that are linear functions of the parameters.

Primal LP approximation

In a reward maximization problem, the primal LP approximation in component form can be written as follows:

Approximate primal linear program (APLP): component form

$$\text{minimize} \quad \sum_{i=0}^{I} \beta_i \sum_{s \in S} \alpha(s) b_i(s) \tag{9.37a}$$

$$\text{subject to} \quad \sum_{i=0}^{I} \beta_i b_i(s) - \lambda \left(\sum_{i=0}^{I} \beta_i \sum_{j \in S} p(j|s,a) b_i(j) \right) \geq r(s,a), \quad a \in A_s, s \in S. \tag{9.37b}$$

In a cost minimization problem this APLP becomes:

$$\text{maximize} \quad \sum_{i=0}^{I} \beta_i \sum_{s \in S} \alpha(s) b_i(s) \tag{9.38a}$$

$$\text{subject to} \quad \sum_{i=0}^{I} \beta_i b_i(s) - \lambda \left(\sum_{i=0}^{I} \beta_i \sum_{j \in S} p(j|s,a) b_i(j) \right) \leq r(s,a), \quad a \in A_s, s \in S. \tag{9.38b}$$

Observe that both models have $I+1$ variables and $\sum_{s \in S} |A_s|$ constraints. Note that the difference between formulation (9.37) and formulation (9.32) 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 it is considerably easier to approach it from a matrix perspective. Recall that the (exact) primal model in Section 5.9 can be written as:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\alpha}^{\mathsf{T}} \mathbf{v} \\ \text{subject to} \quad & \mathbf{A}\mathbf{v} \geq \mathbf{r}, \end{aligned} \tag{9.39}$$

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 $\boldsymbol{\alpha}$ is an arbitrary positive $|S|$-dimensional vector. Substituting the linear approximation $\mathbf{v} \approx \mathbf{B}\boldsymbol{\beta}$ into (9.39) yields:

Approximate primal linear program: vector form

$$\begin{aligned} \text{minimize} \quad & (\boldsymbol{\alpha}^\mathsf{T}\mathbf{B})\boldsymbol{\beta} \\ \text{subject to} \quad & (\mathbf{AB})\boldsymbol{\beta} \geq \mathbf{r}. \end{aligned} \tag{9.40}$$

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

Algorithm 9.8. 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 (9.39).
3. Transform the model to (9.40).
4. Solve (9.40) to obtain $\hat{\boldsymbol{\beta}}$.
5. Set $\hat{\mathbf{v}} \leftarrow \mathbf{B}\hat{\boldsymbol{\beta}}$.
6. Choose
$$\hat{d} \in \underset{d \in D^{\text{MD}}}{\arg\,\text{c-max}} \{\mathbf{r}_d + \lambda \mathbf{P}_d \hat{\mathbf{v}}\}.$$
7. Obtain bounds on $\mathbf{v}_\lambda^*$ using (9.34).

The beauty of this approach is that $\mathbf{B}$ can easily be modified to try different approximations. The bounds in the last step can be used to assess the quality of the approximation[17]. Note 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 the case for an arbitrary $\mathbf{B}$ and $\hat{\boldsymbol{\beta}}$).

Dual LP approximation

Note that the approximate **dual** LP has the following form:

$$\begin{aligned} \text{maximize} \quad & \mathbf{r}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & (\mathbf{AB})^\mathsf{T}\mathbf{x} = \mathbf{B}^\mathsf{T}\boldsymbol{\alpha} \\ & \mathbf{x} \geq \mathbf{0}. \end{aligned} \tag{9.41}$$

This dual linear program has $I + 1$ equality constraints and $\sum_{s \in S} |A_s|$ variables. Whether the primal or dual is more efficiently solved is largely a function of the problem instance. Modern solvers are capable of deciding the best approach to solving a given

[17] 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.

LP formulation. Note that if the dual is solved, the primal variables are also available because they are the shadow prices of the dual constraints (and vice versa). This is important because the primal variables are the quantities needed to construct the approximation to the optimal value function. Assuming the dual has many more columns than variables, it may also be solved using column generation[18] methods.

An example

The following example uses linear programming to find linear approximations to the optimal value function in the queuing service rate control model.

Example 9.12. Recall that in this model $A_s = \{a_1, a_2, a_3\}$ 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 $m(a_k) = 5k^3$. The state space is truncated at $N = 50$, $\lambda = 0.9$ and approximations of the form

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

are considered.

Since the objective is to minimize costs, formulation (9.38) is used. Because the above quadratic approximation has three parameters, the primal LP has 3 variables and 153 constraints. Note that the corresponding dual linear program, has 153 variables and 3 constraints (plus non-negativity 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):

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

Using the matrix formulation of the APLP (9.40) avoids the error-prone and inflexible manipulations used to derive the above constraints in component form. As noted previously the matrices of basis function values for linear and quadratic

[18] Desrosiers and Lübbecke [2005] provides a primer on column generation in LPs.

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. Hence the primal LP model becomes:

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

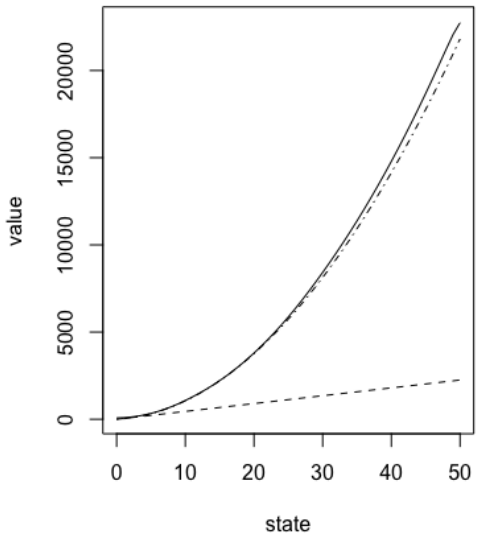
Solving it using the command *simplex* in R provides the estimates $\hat{\boldsymbol{\beta}} = (0, 45)$ for the linear approximation and $\hat{\boldsymbol{\beta}} = (0, 23.2, 8.26)$ for the quadratic approximation.

Figure 9.13a shows the resulting linear and quadratic approximations to the optimal value function. Observe that the linear approximation is inadequate and the quadratic approximation fits well. In both cases, the approximations lie below the true optimal value function. This is because the LP maximizes the greatest lower bound to the optimal value function that satisfies all of the constraints. Moreover the APLP objective function based on the quadratic approximation equals 7532 compared with the optimal objective function of 7538 obtained from the exact LP.

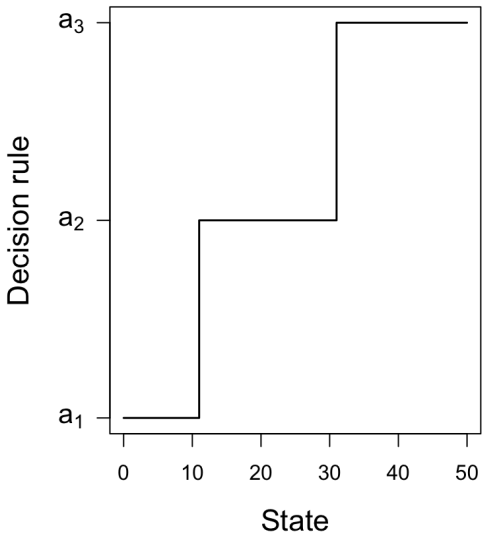
Figure 9.13b shows the stationary policy obtained using the above algorithm based on the quadratic approximation. It is monotone and only differs from the optimal policy in states 30 and 31 where it chooses actions a_2 instead of a_3.

Note also 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, $\lambda = 0.999$ was used. In this case the quadratic and cubic approximations differed, but the greedy policy based on each was the same.

Based on the observations, the linear programming approach described in Algorithm 9.8 offers an attractive and flexible approach for obtaining approximations to the optimal value function.



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



(b) Stationary policy obtained using quadratic LP approximation.

Figure 9.13: Results of using the approximate LP in the queuing service rate control model with $N = 50$.

9.6 Application: strategic scheduling

This section applies the above methods to the following simplified version of the advanced scheduling problem in Section 3.7. The lengthy discussion in the later part of this section illustrates the type of insights that can be gained by solving a Markov decision process and issues arising when using approximations.

In the simplified model, requests for appointments arrive randomly throughout the current day (today). At the end of the day, appointment requests are scheduled for service on some day in the future (tomorrow and subsequent days). Once scheduled, an appointment cannot be rescheduled or delayed. On account of these assumptions, appointment requests arriving today cannot be scheduled until tomorrow.

There are two appointment request types (Class 1 and Class 2) and two types of service (regular time and overtime[19]). Class 1 appointments are urgent and Class 2 appointments are less urgent. 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 tomorrow (if space is available after assigning the Class

[19]In the medical setting, overtime is often referred to as *surge capacity*.

1 requests), between 2 and N days into the future or through overtime tomorrow[20]. 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 is scheduled for overtime, the cost is C. If a Class 2 appointment request is scheduled for regular service on day n in the future, the cost is c_n, $n = 1, \ldots, N$[21] and the cost is C if it is served through overtime. It is reasonable to assume c_n is non-decreasing in n and that $C > c_N$[22]. Figure 9.14 illustrates the cost structure and decision problem symbolically.

The number of Class 1 and Class 2 appointment requests arriving each day are random and independent. For each day, the probability of j Class 1 requests is p_j and the probability of k Class 2 requests in q_k. The vectors $\mathbf{p} = (p_0, \ldots, p_J)$ and $\mathbf{q} = (q_0, \ldots, q_K)$ represent the probability distributions. Assume a capacity of M regular time appointments each day, no limit on overtime and at most J Class 1 arrivals and K Class 2 arrivals on any day. Moreover assume demand on successive days is independent.

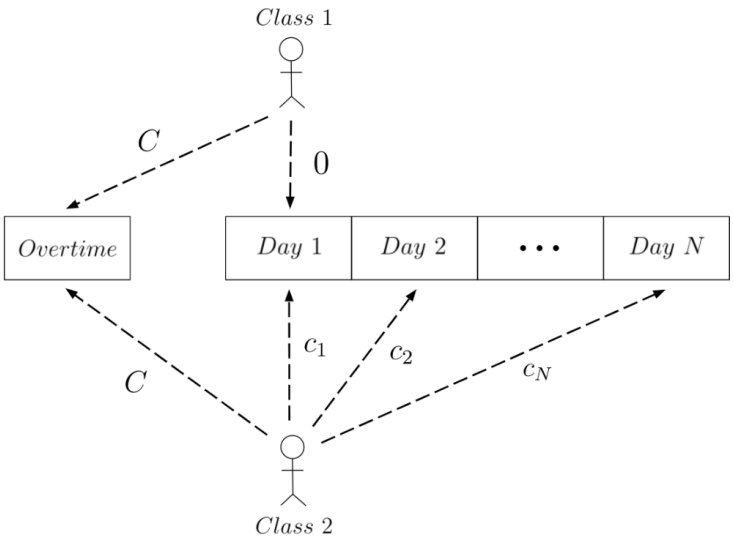


Figure 9.14: Symbolic representation of simplified advance appointment scheduling problem to be analyzed in this section. Note that Day 1 corresponds to "tomorrow".

Model formulation

A cost minimization Markov decision process formulation follows.

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

[21] Note $n = 1$ corresponds to tomorrow. It seems reasonable to set $c_1 = 0$ but it is left arbitrary to simplify notation.

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

Decision epochs: Decision epochs correspond to the specific point of time at the end of a day when the scheduler assigns requests to appointments. Although there is a finite booking horizon, the problem is ongoing so that an infinite horizon formulation is appropriate. Therefore

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

States: A state is a vector $(s_1, \ldots, s_N, d_1, d_2)$ where s_n for $n = 1, \ldots, N$ denotes the number of previously booked appointments on days n and d_i for $i = 1, 2$ denotes the number of Class i appointment requests to schedule. Therefore $|S| = (M+1)^N \times (J+1) \times (K+1)$. Note that an alternative state description would replace the number of booked appointments on day n by the number of available appointment slots on day n.

Actions: Actions are the number of Class 2 requests to assign to day n or to overtime. More formally let a_n denote the number of Class 2 requests assigned to day n, $n = 1, \ldots, N$ and a_0 denote the number of Class 2 appointments assigned to overtime. Then the following rather complicated expression describes the feasible assignments given demands d_1 and d_2:

$$A_{(s_1,\ldots,s_n,d_1,d_2)} = \{(a_0, \ldots, a_N) \,|\, a_0 \geq 0;\ a_1 = (M - s_1 - d_1)^+;$$
$$0 \leq a_n \leq M - s_n, n = 2, \ldots, N;\ \sum_{n=0}^{N} a_n = d_2\}.$$

An interpretation of these constraints follows:

1. The condition $a_0 \geq 0$ means that there is no bound on the number of units served through overtime.

2. The condition $a_1 = (M - s_1 - d_1)^+$ means that if after scheduling Class 1 demand for day 1 (tomorrow) there is any capacity available, as much Class 2 demand as possible will be assigned to day 1. Since it is least expensive to schedule demand on day 1, any optimal policy will use this assignment. Moreover it is the only way in which Class 1 demand explicitly interacts with Class 2 demand.

3. Since $M - s_n$ represents the number of available appointment slots on day n, the condition $0 \leq a_n \leq M - s_n$ constrains the number of Class 2 appointments that can be assigned on day n.

4. The condition $\sum_{n=0}^{N} a_n = d_2$ means that all Class 2 demand has to be assigned to appointment slots (including overtime).

This is one of the few examples in the book where the action varies with the state. This complicates coding because it requires deriving and storing or generating action sets as needed. Note that there is no decision to make for Class 1 appointments because they are served in regular time if capacity is available and otherwise through overtime.

Costs: Costs are incurred on the basis of the number of appointments booked in overtime time, C, and the number of Class 2 appointments booked on day n, c_n. That is

$$c\big((s_1,\ldots,s_N,d_1,d_2),(a_0,\ldots,a_N)\big) = C\big(d_1-(M-s_1)\big)^+ + Ca_0 + \sum_{n=1}^{N} c_n a_n.$$

In this expression, the first term corresponds to overtime charges for Class 1 appointments where $M - s_1$ denotes the number of free spots on day 1. If d_1 exceeds this number, overtime costs are incurred. Recall that it is assumed Class 1 appointment requests incur no cost if assigned to regular time on day 1. Note that the model is formulated in terms of a cost $c(s,a)$ instead of a negative reward $-r(s,a)$.

Transition probabilities: Transition probabilities can be expressed as:

$$p((s_1',\ldots,s_N',d_1',d_2') \mid (s_1,\ldots,s_N,d_1,d_2),(a_0,\ldots,a_N)) = \begin{cases} p_{d_1'}q_{d_2'}, & s_n' = s_{n+1}+a_{n+1}, n=1,\ldots,N-1, s_N'=0 \\ 0, & \text{otherwise.} \end{cases}$$

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

The Bellman equation

The Bellman equation for the infinite horizon discounted cost minimization version is given by:

$$v\big((s_1,\ldots,s_N,d_1,d_2)\big) = \min_{(a_0,\ldots,a_N)\in A_{(s_1,\ldots,s_n,d_1,d_2)}} \Big\{ C\Big(\big(d_1-(M-s_1)\big)^+ + a_0\Big) + \sum_{n=1}^{N} c_n a_n + \lambda \sum_{j=0}^{J}\sum_{k=0}^{K} p_j q_k v\big((s_2+a_2,\ldots,s_N+a_N,0,j,k)\big) \Big\} \tag{9.42}$$

for all $(s_1,\ldots,s_N,d_1,d_2) \in S$. Note that the argument of $v(\cdot)$ inside the "min" represents tomorrow's state after allocating appointment requests that arrived today. The probabilities p_j and q_k only impact the last two components of the state vector.

> 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. The greedy policies based on approximations to the optimal value function do the latter.

9.6.1 A numerical example

This section formulates a small numerical example chosen so that an ϵ-optimal solution can be obtained without resorting to approximations. Let $N = 3, M = 4, J = K = 4$ so that the model has $5^3 \times 5 \times 5 = 3125$ 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.

A comparison of ϵ-optimal policies and values found with value iteration to those found using LSVI with linear and quadratic approximations follows. Results are for an instance with $\lambda = 0.95, c_n = 2n, C = 20, \mathbf{p} = (0, 0.2, 0.2, 0.3, 0.3)$ and $\mathbf{q} = (0.3, 0, 0, 0.3, 0.4)$. The arrival probabilities were chosen so that the total expected daily demand of $5.2 = 2.7 + 2.5$ exceeds the capacity of 4, making strategic scheduling a necessity. Results and interpretations appear in the following sections. Because of the complexity of the action set, policy iteration and linear programming are left to the reader.

9.6.2 Value iteration without approximation

Value iteration, using the recursion implicit in iterating the Bellman equation (9.42), was the easiest to code[25] and as well converged quickly. For this instance, the iterates of value iteration achieved the stopping criterion $\text{sp}(\mathbf{v}' - \mathbf{v}) < 0.0001$ in 11 iterations. Columns 2 and 3 of Table 9.1 gives actions and values obtained using value iteration for selected states.

To interpret these results, consider the case $s = (1, 1, 0, 2, 4)$. This means that there are 3 units of regular-time capacity available on days 1 and 2 and 4 units available on day 3. Since the Class 1 demand is 2, it can be scheduled on day 1 leaving 1 unit of capacity available for Class 2 demand. Thus 3 units of Class 2 demand remain to be scheduled. The greedy action based on the ϵ-optimal value function, assigns one unit to day 2 and two units to day 3. Thus at the next decision epoch, $(s'_1, s'_2, s'_3) = (2, 2, 0)$.

[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] It is left as an exercise to determine the exact number of actions.

[25] Generating the set of actions for each state and manipulating arrays was challenging to code. Complexities included a "feature" of R that converted one-dimensional matrices to vectors.

State $(s_1, s_2, s_3, d_1, d_2)$	ϵ-optimal action	ϵ-optimal value	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 9.1: This table gives results for the value iteration solution (columns 2 and 3) and that found using LSVI with 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 $i = 1, 2, 3$.

In contrast, the greedy policy based on the linear approximation schedules the 3 units of Class 2 demand to day 3, perhaps causing challenges in the future.

9.6.3 LSVI with linear value function approximation

LSVI was used to obtain parameter estimates based on a linear architecture and basis functions that are linear in the state. That is,

$$\hat{v}((s_1, s_2, s_3, d_1, d_2)) = \beta_0 + \beta_1 s_1 + \beta_2 s_2 + \beta_3 s_3 + \beta_4 d_1 + \beta_5 d_2. \tag{9.43}$$

The advantages of using this approximation are:

1. It is easy to obtain parameter estimates using readily available linear regression packages.
2. The number of quantities to approximate and store is reduced from the 3125 components of $\mathbf{v}$ (in its tabular representation) to the six components of $\boldsymbol{\beta}$.
3. The parameter estimates provide insight into the effect demand and capacity have on cost. In (9.43), $\beta_i, i = 1, 2, 3$ represents the *marginal cost* of one less unit of 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.

LSVI[26] as stated in Algorithm 9.3 was implemented as follows. It was initialized

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

with $\boldsymbol{\beta} = \mathbf{0}$ and $\epsilon = 0.0001$. Steps 2(a) and 2(b) were implemented component-wise instead of in vector form. Step 2(a) used representation (9.43) instead of the matrix $\mathbf{B}$. The right hand side of step 2(b) becomes

$$\min_{(a_0,a_1,a_2,a_3)\in A_{(s_1,s_2,s_3,d_1,d_2)}} \Big\{ c\big((s_1,s_2,s_3,d_1,d_2),(a_0,a_1,a_2,a_3)\big) + \lambda \sum_{j=0}^{4}\sum_{k=0}^{4} p_j q_k \hat{v}\big((s_2+a_2,s_3+a_3,0,j,k)\big) \Big\}. \tag{9.44}$$

After substituting the linear approximation, the summation above becomes

$$\begin{aligned} &\sum_{j=0}^{4}\sum_{k=0}^{4} p_j q_k \hat{v}((s_2+a_2,s_3+a_3,0,j,k)) \\ &\quad = \sum_{j=0}^{4}\sum_{k=0}^{4} p_j q_k \big(\beta_0 + \beta_1(s_2+a_2) + \beta_2(s_3+a_3) + \beta_3 0 + \beta_4 j + \beta_5 k\big) \\ &\quad = \beta_0 + \beta_1(s_2+a_2) + \beta_2(s_3+a_3) + \beta_4 E(D_1) + \beta_5 E(D_2), \end{aligned} \tag{9.45}$$

where the random variables D_1 and D_2 represent the amount of Class 1 and Class 2 demand, respectively. Since after substitution, several of the expressions in (9.44) do not involve the action explicitly, the greedy action based on the linear approximation chooses

$$(\hat{a}_0,\hat{a}_1,\hat{a}_2,\hat{a}_3) \in \operatorname*{arg\,min}_{(a_0,a_1,a_2,a_3)\in A_{(s_1,s_2,s_3,d_1,d_2)}} \Big\{ Ca_0 + c_1 a_1 + (c_2+\lambda\beta_1)a_2 + (c_3+\lambda\beta_2)a_3 \Big\} \tag{9.46}$$

The example was solved using the representation (9.44) to allow easy generalization to other approximations 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 of the difference $\boldsymbol{\beta}' - \boldsymbol{\beta}$ exceeded that at a previous iteration. This is consistent with the observation above that the LSVI operator need not be a contraction mapping.

The linear approximation to the optimal value function obtained by substituting the value of $\hat{\boldsymbol{\beta}}$ is given by

$$\hat{v}\big((s_1,s_2,s_3,d_1,d_2)\big) = -43.98 + 12.59 s_1 + 6.54 s_2 + 4.28 s_3 + 12.60 d_1 + 8.98 d_2. \tag{9.47}$$

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 because of the greater flexibility in scheduling Class 2 demand.

The structure of a greedy policy based on the linear approximation can be obtained as follows. Recall that the daily capacity is $M = 4$ units, overtime cost is $C = 20$ and the cost of scheduling a unit of Class 2 demand on day $n = 1, 2, 3$ is $2n$. Substituting the parameter estimates in (9.47) into (9.46) implies that

$$(\hat{a}_0, \hat{a}_1, \hat{a}_2, \hat{a}_3) \in \underset{(a_0,a_1,a_2,a_3)\in A_{(s_1,s_2,s_3,d_1,d_2)}}{\arg\min} \{20a_0 + 2a_1 + 10.21a_2 + 10.06a_3\}. \tag{9.48}$$

where $10.21 = 4+0.95\cdot 6.54$ and $10.06 = 6+0.95\cdot 4.28$. The action set can be expressed as

$$A_{(s_1,s_2,s_3,d_1,d_2)} = \left\{ (a_0, a_1, a_2, a_3) \;\middle|\; a_0 \geq 0; a_1 = (4 - s_1 - d_1)^+; \right. \\ \left. 0 \leq a_n \leq 4 - s_n, n = 2, 3; \sum_{n=0}^{3} a_n = d_2 \right\}. \tag{9.49}$$

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

As a result of (9.48) and (9.49) it follows that the greedy action based on the linear approximation has the following structure:

> In each state in which $d_2 > 0$, assign as much Class 2 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 Class 2 demand to overtime when there is no other capacity available.

This policy can be summarized succinctly as: after using up capacity on day 1, fill capacity starting from the end of the booking horizon and working backwards to the current day. Note that this characterization avoids the need to compute the optimal action in each state although it is done so in Table 9.1 for comparison.

9.6.4 LSVI with quadratic value function approximation

LSVI was also applied using a value function approximation of the form:

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

A more general quadratic model would include 10 additional cross product terms of the form $s_i s_j$, $s_i d_j$ and $d_1 d_2$. Since the model is linear in the parameters, they can be easily estimated using linear regression. In contrast to the approximation in the previous section, 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 LSVI converged in 13 iterations and the norms of the differences of successive parameter estimates were not monotone.

The estimated value function is given by

$$\hat{v}((s_1, s_2, s_3, d_1, d_2)) = -34.93 + 5.93s_1 + 3.55s_2 + 1.92s_3 + 5.93d_1 + 5.07d_2 + 1.68s_1^2 + 0.85s_2^2 + 0.70s_3^2 + 1.68d_1^2 + 1.04d_2^2. \quad (9.50)$$

Using the same approach as in the case of the linear value function approximation, substituting $\hat{v}(s)$ into the right hand side of the Bellman equation, the analogous expression to (9.48), after some algebra, becomes

$$20a_0 + 2a_1 + \left(7.37a_2 + 0.81(s_2 + a_2)^2\right) + \left(7.82a_3 + 0.66(s_3 + a_3)^2\right). \quad (9.51)$$

Given a state, choosing the action to minimize this expression gives the greedy action. Since the action set is finite, this minimization can be done by enumeration. For example, in state $(2, 1, 1, 2, 1)$ there are three feasible actions as shown in Table 9.2. Evaluating the above expression for each action shows that the greedy action is $(0, 0, 1, 0)$; that is, assign 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 (9.51)
$(1, 0, 0, 0)$	20.00
$(0, 0, 1, 0)^*$	11.21
$(0, 0, 0, 1)$	11.27

Table 9.2: Greedy action choice in state $(2, 1, 1, 2, 1)$ based on quadratic value function approximation. The greedy action is indicated with an asterisk.

9.6.5 Comparison of policies and values

The three left most columns of Table 9.1 above 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 in these states Class 2 demand is never served with overtime.

Columns 4 and 5 of Table 9.1 show the greedy action obtained using the linear value function approximation and the value $\hat{v}$. Observe that in 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[27]. 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. Overall, the action based on the linear approximation differed from that chosen by the ϵ-optimal policy in 632 out of 3125 states.

Remarkably, the greedy actions chosen using the 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 was considerably better and might be appropriate in larger instances of the model.

9.6.6 Further insights

This section provides some additional insights that can be gained by investigating other aspects of the model and solution methods.

Using overtime strategically

To test whether overtime was ever used strategically, overtime costs were chosen to be class specific. Setting the overtime cost for Class 1 to 40 and for Class 2 to 10, made it more essential to reserve capacity for Class 1 demand. For example the ϵ-optimal policy in state $(2, 3, 0, 2, 4)$ chose action $(2, 0, 0, 2)$. This means that it assigns 2 units of Class 2 demand to overtime and 2 units as late as possible, that is to day 3. Thus the ϵ-optimal policy uses overtime strategically by reserving capacity for Class 1 demand.

Note that the greedy action corresponding to the linear value function approximation remained $(0, 0, 0, 0, 4)$ but the quadratic approximation again identified the optimal action. This provides further support for using a quadratic approximation.

State sampling

In the above analysis, LSVI approximations were based on estimating parameters using value function updates for all states. However, good results may be possible if LSVI is applied to only a subset of the 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

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

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

needs to be evaluated at each iteration of LSVI.

The following offline approach was used to apply LSVI using a subset of S. A subset of states was selected by:

1. specifying the fraction of states to sample,
2. randomly sampling the specified number of states,
3. discarding duplicates[28], and
4. applying LSVI as described in Section 9.3.1 using the designated sample.

Thus after selecting a random subset of states, LSVI uses it for all iterations. The next two chapters consider online versions of these algorithms, which sample states sequentially and avoid computing the expectation $\sum_{j\in S} p(j|s,a)\hat{v}(j)$.

The following analysis is based on using the above quadratic approximation. It applies LSVI to a large number of samples, in which the fraction of states sampled was varied between 0.1 to 0.9. The quality of fit was compared on the basis of

$$\text{RMSE} = \left(\frac{1}{11}\sum_{i=0}^{10}(\hat{\beta}_i - \hat{\beta}_i^{\text{all}})^2\right)^{0.5},$$

where β_i^{all} denotes the parameter estimates based on using all states.

Results of this experiment appear in Figure 9.15, which shows two things:

1. the average quality of the fits improve with sample size, and
2. there is considerable variability between samples.

We speculate that by instead choosing the state samples to be representative of the entire state space, much of this between sample variability can be reduced. Moreover, it appears that the accuracy based on sampled fractions greater than 0.6 are similar.

The quadratic value function approximation based on a random sample of half of the states is given by

$$\begin{aligned}\hat{v}((s_1,s_2,s_3,d_1,d_2)) = -36.86 + 5.61s_1 + 3.75s_2 + 2.21s_3 + 7.21d_1 + 5.12d_2 \\ + 1.73s_1^2 + 0.80s_2^2 + 0.70s_3^2 + 1.37d_1^2 + 1.03d_2^2. \quad (9.52)\end{aligned}$$

Comparing it to (9.50) obtained using **all** states shows that the estimates are reasonably close and the order of magnitude of the corresponding parameters is similar. Derivation of the corresponding greedy policy and comparison to that obtained using all states is left to the reader.

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

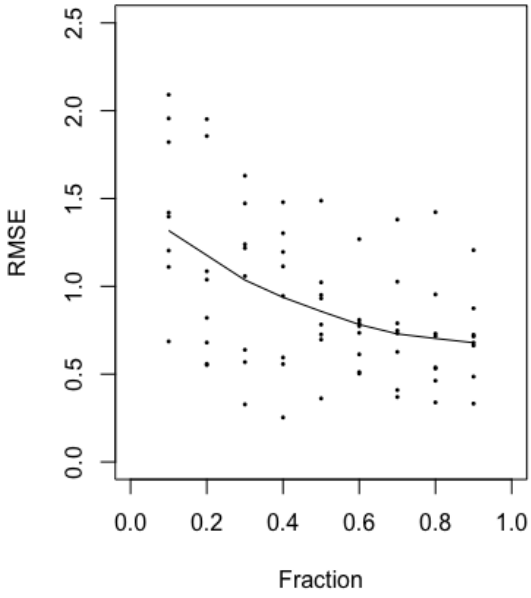


Figure 9.15: RMSE as a function of the fraction of states sampled over multiple state samples. The dots correspond to a specific sample and the line is a Lowess smoothing of the individual RMSEs.

Numerical example: Neural network approximation

Obtaining good results using a neural network approximation proved challenging when using the package "neuralnet" in R. When incorporated in LSVI, the algorithm either oscillated and did not satisfy a stopping criterion or the parameter estimation algorithm did not converge. Several alternatives were explored. Most notably, when the ϵ-optimal value function on the entire state space was approximated by several different neural network structures, the greedy policy was similar to that found with a linear value function approximation. In this application, we believe that neural networks **do not** provide an attractive approximation architecture but leave it to the reader to prove the supposition false.

Approximate linear programming

An analysis based on linear programming is not presented 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 approximation 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.

9.7 Application: Golf strategy

Often, in real-life situations, transition probabilities, reward functions and the policy used are not known. This section provides a case study of the estimation of a value function based on multiple realizations of finite sequences of states and rewards that terminate when a *goal state* is reached. This case study, drawn from the game of golf, illustrates this approach and provides a transition to the next two chapters.

9.7.1 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

Golf is a game where players strive to propel a 1.68-inch diameter golf ball into a 4.25-inch diameter cup (often referred to as a "hole") in the fewest number of strokes. A typical golf course comprises 18 holes, each featuring a tee box, a putting green (where the cup is located), a fairway, and possibly some rough, sand traps, obstructed areas, out-of-bounds areas, and natural obstacles like woods, deserts, or water. The distance from the tee to the cup typically ranges from 100 to 600 yards.

Play begins from the tee, and subsequent shots are played from the point where the previous shot came to rest. A penalty stroke is added if a ball is lost or deemed unplayable. Playing **each hole** in a round of golf may be viewed as a Markov decision process as follows.

Decision epochs: Decision epochs correspond to the instance before a shot is hit:

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

Note that an infinite horizon formulation is appropriate because the number of shots required to complete a hole of golf is not fixed; it is determined by the quality of play. 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[29].

[29] In men's professional (PGA) golf, the highest recorded hole score to date is 16 in 2011 by Kevin Na at the Valero Open.

States: The state represents the distance from the hole $\delta \in \mathcal{D}$ and the terrain type $\tau \in \mathcal{T}$ (fairway, rough, etc.) 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,

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

On most golf courses $\mathcal{D} = \{100, 101, \ldots, 600\}$ and

$$\mathcal{T} = \{\text{tee, fairway, rough, sand trap, hazard, obstructed, green}\}.$$

Actions: Each action represents the type of shot a player tries to hit and its intended target. These vary by location, wind and course configuration.

$$A_{(\delta,\tau)} = \{\text{Possible targets and shot types from } (\delta, \tau)\}$$
$$A_{(0,\text{green})} = \{a_0\},$$

where a_0 indicates that the hole is completed.

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 that at this level, shot types are not described explicitly because they vary greatly with the skill of the player and the terrain. For example, if the ball is behind a tree then the player may hit a recovery shot. If the ball is on the green, the player will putt. In the fairway, the player may pick a target 10 feet short of the hole and attempt to hit a low draw[30].

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. One would expect that shots from closer to the hole will have less variable outcomes than shots from greater distances.

Since the hole terminates when $(0, \text{green})$ is reached, $p((0, \text{green})|(0, \text{green}), a_0) = 1$.

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[31].

Rewards: It requires one stroke to hit a shot so that:

$$r((\delta, \tau), a, (\delta', \tau')) = -1 \quad \text{for all} \quad (\delta, \tau) \in S$$

and

$$r((0, \text{green}), a_0, (0, \text{green})) = 0.$$

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

[31] Unless of course the golfer is really frustrated after hitting a poor shot, since mood may affect execution of future shots.

Because the objective in golf is to minimize the number of shots on each hole, the reward represents the "cost" of a shot. Because this is formulated as an infinite horizon model, the state $(0, \text{green})$ is the zero-cost (reward-free) absorbing state.

Note that it is preferable to model this as a minimum cost problem, in which case $r((\delta, \tau), a, (\delta', \tau')) = -1$ is replaced by $c((\delta, \tau), a, (\delta', \tau')) = 1$.

Value functions in golf

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, that is

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

Of course $v(\delta, \tau) \geq 0$ and $v(0, \text{green}) = 0$. Moreover for a fixed τ, $v(\delta, \tau)$ should be non-decreasing in δ.

As an example, for an average male professional golfer[32],

$$v(100, \text{fairway}) = 2.80 \quad \text{and} \quad v(100, \text{rough}) = 3.02.$$

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. Thus, on average a player adds 0.22 shots if a shot ends up in the rough as opposed to the fairway. This may not sound like much but over a tournament made up of four 18 hole rounds, 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 9.16). Such a figure can also guide player strategy as discussed below. Knowing this value from every distance and terrain is fundamental to developing a performance metric referred to as *strokes gained*, described below.

A case study

The following case study is based on Marty's collaboration with the University of British Columbia (UBC) women's golf team. The objective of this study was to establish value functions for female golf team members. Unlike in the case of male professional golfers, such specialized information was not available at the time of the study.

The following approach was 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 for illustrative purposes only shots from the fairway are analyzed here. Therefore the component τ will be dropped from the state description

[32] Broadie [2014] p. 85.

Figure 9.16: Graphical representation of golf value functions for male professional golfers based on Broadie [2014]. For example, on average it requires 2.8 shots to hole out from anywhere on the black arc at 100 yards from the green.

for the remainder of this section. Hence, the objective of this analysis is to estimate $v(\delta)$; the expected number of shots to hole out from distance δ on the fairway.

The complete data set, which contained the starting location (state) and ending location (state) for 10,100 shots executed during the 2016/2017 golf season, was entered by team members in a custom application. 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 $(\delta, v) = (200, 4)$ indicates that 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.

A value function approximation

The value function was approximated by a linear combination of the basis functions $b_0(\delta) = 1, b_1(\delta) = \delta, b_2(\delta) = \delta^2$ and $b_3(\delta) = \delta^3$. That is, it was approximated by a cubic polynomial. In contrast to other examples in this chapter based on an MDP model,

an observed value was represented by

$$v(\delta) = \beta_0 + \beta_1\delta + \beta_2\delta^2 + \beta_3\delta^3 + \epsilon, \tag{9.53}$$

where ϵ represented an unobservable random disturbance.

The reason for developing an approximation using basis functions as opposed to separate estimates of $v(\delta)$ for each δ based on the average of shots from that distance was to account for data sparsity and to obtain a value function approximation that was non-decreasing 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 $v(\delta)$ should be non-decreasing in δ.

Using least squares regression resulted in the following fitted model:[33]

$$\hat{v}(\delta) = 1.95 + 0.015\,\delta - 0.000040\,\delta^2 - 0.000000049\,\delta^3. \tag{9.54}$$

The data together with the fitted model are shown in Figure 9.17. Note that more complicated models or including random effects had little effect on estimated values.

Based on the fitted model, $\hat{v}(100) = 3.11$, which is 0.31 higher than for a male professional golfer from that location or equivalent to the value for a male professional from 185 yards.

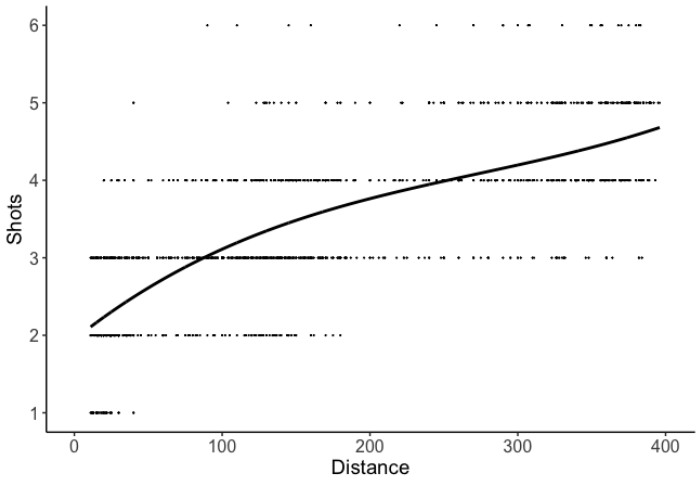


Figure 9.17: Data and fitted model for (9.54). The data values are discrete since they represented the number of shots required to "hole out" from each distance on the fairway. Note that the fitted function is monotone increasing.

[33] An alternative is to fit a Poisson regression since the dependent variable represents a count. Thanks to Antoine Sauré for this suggestion.

Golf strategy based on value functions

In the context of the Markov decision process framework, the value functions can be used to determine strategy. Strategic choices in golf include anticipated shot distance and shot type. Outcomes are impacted by shot dispersion patterns represented by probability distributions. Shorter shots have less dispersion than longer shots.

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 try to reach green in one shot. Choice 2 is to "lay up", which means to be conservative and hit a shorter shot that most likely 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 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 in the fairway where the value is $v(60, \text{fairway}) = 2.70$. Thus the expected value of the action "go for the green" is

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

Suppose the alternative action is to aim for a target 80 yards from the green where there is a 0.9 chance of being close to 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 the action "lay up" is $v_2 = 2.77$. Since the goal is to minimize the number of strokes taken to hole out, the decision to go for the green is optimal if $v_1 \leq v_2$, otherwise the golfer should lay up. Therefore noting that $3.70 - 1.56p = 2.77$ implies $p = 0.596$, the optimal decision rule is

$$d(240, \text{fairway}) = \begin{cases} \text{go for green}, & p \geq 0.596 \\ \text{lay up}, & p < 0.596. \end{cases}$$

Hence if the player thinks there is greater than a 59.6% chance of hitting the green, then he[35] should go for the green; otherwise he should lay up. In reaching this decision, a player must take into account factors including the weather, the lie in the fairway and his stress level.

Using value functions: Strokes gained

Suppose a golfer scores a 3 on a par 4 hole. To which shot (or shots) can this outcome be attributed? Was it the result of an outstanding drive, a precise approach shot or

[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].

an excellent putt or was it some combination of these shots? This is the question that the concept of *strokes gained* seeks to answer.

Determining which action contributes to success in an episodic model is referred to in the machine learning literature to as the *the credit assignment problem*[36]. This challenge arises in domains such as the games chess and Go, robotic navigation and natural language processing. Value functions and state-action value functions provide a way to approach this problem in general.

Solving the credit assignment problem is crucial for effective learning. If a learning system cannot assign credit (or blame) to actions, then it will not be able to learn which actions are beneficial and which are detrimental. Without being able to make this distinction, the learner will not be able to improve performance over time.

Returning to golf, *strokes gained* measures the difference between the expected value of a shot and its result. Let X_D be a random variable denoting the distance of the ball to the hole after the current shot and X_T be a random variable denoting the terrain of the ball after the current shot. Also, let

$$
\begin{aligned}
(\delta', \tau') &= \text{Actual location of the ball after the current shot} \\
(\delta, \tau) &= \text{Actual location of the ball immediately prior to the current shot.}
\end{aligned}
$$

Then strokes gained, SG[37], is defined as

$$\text{SG}(\delta, \tau) = E_{(\delta,\tau)}[v(X_D, X_T)] - v(\delta', \tau'). \tag{9.55}$$

Recalling the MDP formulation of golf above, the policy evaluation equation

$$v(s) = r(s) + \sum_{j \in S} p(j|s)v(j)$$

expressed in terms of random variables and costs becomes

$$v(\delta, \tau) = 1 + E_{(\delta,\tau)}[v(X_D, X_T)]. \tag{9.56}$$

So rearranging terms and substituting (9.56) into (9.55) yields

$$\text{SG}(\delta, \tau) = v(\delta, \tau) - (v(\delta', \tau') + 1) \tag{9.57}$$

The first quantity on the right hand side of (9.57) 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.

[36] A nice discussion of this issue appears in Bennett [2023].

[37] see Broadie [2014]

To interpret this quantity, recall that the goal in golf is to *minimize* the expected number of strokes to hole out. If the player hits a better than average shot, $v(\delta', \tau') + 1$ will be less than $v(\delta, \tau)$ and strokes gained will be positive. If the shot outcome is worse than average, strokes gained will be negative. When a player hits an "average" shot, $v(\delta, \tau) = v(\delta', \tau') + 1$ so that strokes gained will be zero.

Example 9.13. To illustrate this concept consider the following scenario where values are based on the fitted curve (9.54). Suppose a women's golf team member is at 275 yards in the fairway, then $v(275, \text{fairway}) = 4.09$. After hitting an average shot from this location, her strokes gained would be $4.09 - 1 = 3.09$. This is equivalent to a distance of 98 yards in the fairway, so an average shot to the fairway would be $275 - 98 = 177$ yards. Hitting the ball to 75 yards in the fairway results in a value of 2.88 and the corresponding strokes gained is $4.09 - 2.88 - 1 = 0.21$. If instead she hits the ball to 125 yards in the fairway, the value is 3.31 corresponding to a strokes gained of -0.22. Note that hitting the ball into the rough or a sand trap at 125 yards would result in more negative stokes gained.

This calculation ignores the possibility that player may have made a strategic choice to lay up to specific distance so as to avoid a hazard.

From the credit assignment problem perspective, strokes gained averaged over several rounds identifies a player's strengths and weaknesses and which areas need improvement. Hence, this information can be used to develop a practice plan. For example, if average strokes gained for putts in the range 8 to 12 feet is negative, then this is an area where the player needs to improve. This was the objective of the UBC golf team project.

9.8 Technical appendix: Regression and nonlinear optimization

To effectively apply and interpret value function approximation, familiarity with the basics of linear regression as described in this appendix is crucial.

9.8.1 Linear regression

Given observations of a dependent variable y_j and independent variables $x_{j,1}, \ldots, x_{j,I}$ for $j = 1, \ldots, J$, the objective of linear regression is to find a set of *regression coefficients*, *weights* or *parameters* β_i, $i = 0, \ldots, I$ so that

$$f(x_{j,1}, \ldots, x_{j,I}; \beta_0, \ldots, \beta_I) := \beta_0 + \beta_1 x_{j,1} + \cdots + \beta_I x_{j,I} \tag{9.58}$$

well approximates y_j for $j = 1, \ldots, J$. This is referred to as a *linear* regression problem because $f(x_{j,1}, \ldots, x_{j,I}; \beta_0, \ldots, \beta_I)$ is a linear function of its parameters.

The standard approach[38] to estimation is to obtain parameters that minimize the sum of squared errors

$$g(\beta_0, \ldots, \beta_I) := \sum_{j=1}^{J} \left(y_j - (\beta_0 + \beta_1 x_{j,1} + \cdots + \beta_I x_{j,I})\right)^2 \tag{9.59}$$

$$= \sum_{j=1}^{J} g_j(\beta_0, \ldots, \beta_I)^2 \tag{9.60}$$

where

$$g_j(\beta_0, \ldots, \beta_I) := y_j - (\beta_0 + \beta_1 x_{j,1} + \cdots + \beta_I x_{j,I})$$

for $j = 1, \ldots, J$.

The function $g_j(\cdot)$ is introduced to provide more elegant expressions and simplify the derivation of Gauss-Newton iteration below. Note that this notation expresses dependence on observation $(y_j, x_{j,1}, \ldots, x_{j,I})$ through the subscript j.

Parameters that minimize $g(\beta_0, \ldots, \beta_I)$ are referred to as *least squares* or *ordinary least squares (OLS)* estimates. Using a matrix formulation leads to a closed form representation. Parameter estimates are often denoted by $\hat{\beta}_i, i = 0, \ldots, I$.

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

$$y_j = \beta_0 + \beta_1 x_{j,1} + \cdots + \beta_I x_{j,I} + \epsilon_j, \tag{9.61}$$

where ϵ_j 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 $E[\epsilon_j \epsilon_k] = 0$ for $j \neq k$, 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_i, $i = 0, \ldots, I$ are also *maximum likelihood estimates* and consequently have many desirable theoretical properties that allow for statistical inference, such as the computation of confidence intervals and hypothesis testing. Usually, these statistical concepts are not taken into account in approximate dynamic programming.

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

$$y_j \sim NID(\beta_0 + \beta_1 x_{j,1} + \cdots + \beta_I x_{j,I}, \sigma^2), \tag{9.62}$$

where the expression $NID(\mu, \sigma^2)$ corresponds to a set of independent normally distributed random variables with mean μ and variance σ^2. From this perspective, the mean of y_j varies from observation to observation and the variance is constant. The beauty of writing the model this way is that it allows generalization to other distributions such as Poisson or binomial, or different non-constant variance, or auto-correlated errors.

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

9.8.2 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 formulae for predicted values and parameter estimates. To do so define the following vectors and matrices:

$$\mathbf{y} := \begin{bmatrix} y_1 \\ \cdot \\ \cdot \\ \cdot \\ y_J \end{bmatrix}, \ \mathbf{X} := \begin{bmatrix} 1 & x_{1,1} & \dots & x_{1,I} \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ 1 & x_{J,1} & \dots & x_{J,I} \end{bmatrix}, \ \boldsymbol{\beta} := \begin{bmatrix} \beta_0 \\ \cdot \\ \cdot \\ \cdot \\ \beta_I \end{bmatrix} \text{ and } \boldsymbol{\epsilon} := \begin{bmatrix} \epsilon_1 \\ \cdot \\ \cdot \\ \cdot \\ \epsilon_J \end{bmatrix}. \tag{9.63}$$

In (9.63), $\mathbf{y}$ denotes the $J \times 1$ column vector of observations of the dependent variable, $\mathbf{X}$ is the $J \times (I+1)$ matrix of values of the independent variables, $\boldsymbol{\epsilon}$ denotes the $J \times 1$ column vector of random disturbances and $\boldsymbol{\beta}$ is the $(I+1) \times 1$ column vector of parameters. It is also convenient to denote the j-th row of $\mathbf{X}$ by the vector $\mathbf{x}_j$, that is

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

for $j = 1, \dots, J$. When written as a row vector, it will be expressed as $\mathbf{x}_j^\mathsf{T}$.

Using this matrix notation, the data generating model, which provides an elegant way of representing (9.61) for all j in a single equation, is given by

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{9.64}$$

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

$$g(\boldsymbol{\beta}) = \sum_{j=1}^{J} g_j(\boldsymbol{\beta})^2 = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\mathsf{T}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = \mathbf{y}^\mathsf{T}\mathbf{y} - 2\boldsymbol{\beta}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{y} + \boldsymbol{\beta}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\boldsymbol{\beta}. \tag{9.65}$$

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

$$g(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2. \tag{9.66}$$

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"

$$\mathbf{X}^\mathsf{T}\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^\mathsf{T}\mathbf{y}. \tag{9.67}$$

When $\mathbf{X}^\mathsf{T}\mathbf{X}$ is invertible[39] the least square parameter estimates can be written in closed form as:

[39] 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.

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y} := \boldsymbol{\Gamma}\mathbf{y}. \tag{9.68}$$

Note that $\hat{\boldsymbol{\beta}}$ determined by either (9.67) or (9.68) can also be written as

$$\hat{\boldsymbol{\beta}} \in \underset{\boldsymbol{\beta}\in\mathbb{R}^{I+1}}{\arg\min} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2. \tag{9.69}$$

The vector of predicted (fitted) values $\hat{\mathbf{y}}$ can be expressed as:

$$\hat{\mathbf{y}} = \mathbf{X}(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y} := \boldsymbol{\Pi}\mathbf{y}. \tag{9.70}$$

The matrices $\boldsymbol{\Gamma}$ and $\boldsymbol{\Pi}$ defined in (9.68) and (9.70) are fundamental to the discussion of iterative algorithms in this chapter. 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 $\mathbf{y}$. It is also referred to as a *projection matrix* for reasons described below.

The following example illustrates this notation.

Example 9.14. Quadratic regression: Suppose the dependent variable y_i is represented by a quadratic function of the independent variable x_i written as

$$y_j = \beta_0 + \beta_1 x_j + \beta_2 x_j^2 + \epsilon_j$$

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

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ 1 & x_J & x_J^2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}.$$

There is no necessity that the x_j 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

A beautiful geometric representation underlies least squares estimation and is fundamental to the LSVI and LSPI algorithms.

Since $\mathbf{\Pi}^2 = \mathbf{\Pi}$, $\mathbf{\Pi}$ is a projection matrix and it maps $\mathbf{y}$ onto the space of vectors spanned by the columns of $\mathbf{X}$ denoted col($\mathbf{X}$) (see Figure 9.18). In other words $\mathbf{\Pi y}$ is the linear combination of the columns of $\mathbf{X}$ that is closest in the Euclidean norm sense to $\mathbf{y}$.

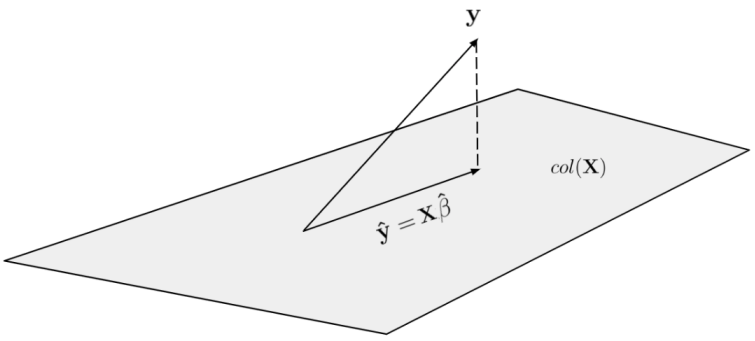


Figure 9.18: Illustration of the geometry underlying least squares regression. It shows the plane col($\mathbf{X}$), the vector $\mathbf{y}$ which lies outside the plane and its projection $\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$ on col($\mathbf{X}$). The dashed line corresponds to the vector of residuals $\mathbf{y} - \hat{\mathbf{y}}$.

Furthermore, rewriting the normal equations (9.67) 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 9.18. What this means is that $\mathbf{X}\hat{\boldsymbol{\beta}}$ is the closest point to $\mathbf{y}$ 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, for example when the variance of y_i is a function of the independent variables, it might be preferable to minimize the weighted sum of squared errors

$$g_{\mathbf{w}}(\boldsymbol{\beta}) := \sum_{j=1}^{J} w_j g_j(\boldsymbol{\beta})^2, \tag{9.71}$$

where $\mathbf{w} = (w_1, \ldots, w_J)$ is a vector of positive scalar weights. Defining the *squared weighted Euclidean norm* of $\mathbf{u} \in \mathbb{R}^J$ as

$$\|\mathbf{u}\|_{\mathbf{w}}^2 := \sum_{j=1}^{J} w_j u_j^2, \tag{9.72}$$

then

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

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 $w_1, \ldots, w_J$.

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

$$\boldsymbol{\beta}_{\text{WLS}} = (\mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{y}. \tag{9.73}$$

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

There are several reasons one might want to use the weighted sum of squares as an optimality criterion:

1. Some regions of the state space might be more important than others, so it is desirable that estimates 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 $j = 1, \ldots, J$,

   $$y_j = \beta_0 + \beta_1 x_j + \epsilon_j$$

   and $\text{var}[\epsilon_j] = \sigma^2 x_j$, then it would be appropriate to use WLS with $w_j = 1/x_j$. 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 when the data is either an average or a total of the observations at each state.

4. In the body of this chapter it is noted that the operator corresponding to LSPE is a contraction mapping with respect to the weighted *supremum* norm with weight function equal to the stationary distribution of the underlying Markov chain.

9.8.3 Nonlinear regression

In some cases it may be preferable to represent value functions by nonlinear models, which can be denoted in scalar notation as[40]

$$y_j = f(x_{j,1}, \ldots, x_{j,I}; \beta_0, \ldots, \beta_I) + \epsilon_j \tag{9.74}$$

for $j = 1, \ldots, J$ or in vector notation as

$$\mathbf{y} = \mathbf{f}(\mathbf{X}; \boldsymbol{\beta}) + \boldsymbol{\epsilon}, \tag{9.75}$$

where

$$\mathbf{f}(\mathbf{X}; \boldsymbol{\beta}) := \begin{bmatrix} f(x_{1,1}, \ldots, x_{1,I}; \beta_0, \ldots, \beta_I) \\ \cdot \\ \cdot \\ \cdot \\ f(x_{J,1}, \ldots, x_{J,I}; \beta_0, \ldots, \beta_I) \end{bmatrix} = \begin{bmatrix} f(\mathbf{x}_1; \boldsymbol{\beta}) \\ \cdot \\ \cdot \\ \cdot \\ f(\mathbf{x}_J; \boldsymbol{\beta}) \end{bmatrix}.$$

As in the case of linear regression, the least squares estimates of the parameters $\beta_0, \ldots, \beta_I$ minimize the squared error

$$g(\beta_0, \ldots, \beta_I) = \sum_{j=1}^{J} (y_j - f(x_{j,1}, \ldots, x_{j,I}; \beta_0, \ldots, \beta_I))^2, \tag{9.76}$$

which can be expressed in matrix form as

$$g(\boldsymbol{\beta}) = \sum_{j=1}^{J} g_j(\boldsymbol{\beta})^2 = \mathbf{g}(\boldsymbol{\beta})^\mathsf{T} \mathbf{g}(\boldsymbol{\beta}), \tag{9.77}$$

where $g_j(\boldsymbol{\beta}) = y_j - f(\mathbf{x}_j; \boldsymbol{\beta})$ and $\mathbf{g}(\boldsymbol{\beta})$ denotes an $J \times 1$ vector with components $g_j(\boldsymbol{\beta})$. Finding $\boldsymbol{\beta}$ such that $g(\boldsymbol{\beta})$ is minimized can be done by applying nonlinear optimization methods to (9.77) such as those described in the next subsection.

9.8.4 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, which are described here.

Gradient descent

Let $g(\boldsymbol{\beta})$ denote a real-valued function of the $(I+1)$-dimensional parameter vector $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_I)$. *Gradient descent* refers to algorithms based on recursions of the form

$$\boldsymbol{\beta}' \leftarrow \boldsymbol{\beta} - \tau \nabla_{\boldsymbol{\beta}} g(\boldsymbol{\beta}). \tag{9.78}$$

[40] Recall that in general the number of independent variables does not need to equal I.

Written in terms of its iterates this recursion is expressed as:

$$\boldsymbol{\beta}^{n+1} = \boldsymbol{\beta}^n - \tau_n \nabla_{\boldsymbol{\beta}} g(\boldsymbol{\beta}^n), \tag{9.79}$$

where the gradient of g evaluated at $\boldsymbol{\beta}$ is defined by

$$\nabla_{\boldsymbol{\beta}} g(\boldsymbol{\beta}) := \begin{bmatrix} \frac{\partial g(\boldsymbol{\beta})}{\partial \beta_0} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial g(\boldsymbol{\beta})}{\partial \beta_I} \end{bmatrix} \tag{9.80}$$

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

The motivation for this method is that $-\nabla_{\boldsymbol{\beta}} g(\boldsymbol{\beta}^n)$ represents the direction of steepest descent of g at $\boldsymbol{\beta}^n$ so that decreasing $\boldsymbol{\beta}$ 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 i-th component of the gradient of g evaluated at $\boldsymbol{\beta}$ is given by

$$\frac{\partial g(\boldsymbol{\beta})}{\partial \beta_i} = -2 \sum_{j=1}^{J} \left(\big(y_j - f(x_{j,1}, \ldots, x_{j,I}; \beta_0, \ldots, \beta_I)\big) \frac{\partial f(x_{j,1}, \ldots, x_{j,I}; \beta_0, \ldots, \beta_I)}{\partial \beta_i} \right)$$

for $i = 0, \ldots, I$. In matrix form this can be written as

$$\nabla_{\boldsymbol{\beta}} g(\boldsymbol{\beta}) = -2\mathbf{J}(\boldsymbol{\beta})^{\mathsf{T}} \mathbf{g}(\boldsymbol{\beta}), \tag{9.81}$$

where $\mathbf{J}(\boldsymbol{\beta})$ denotes the $J \times (I+1)$ *Jacobian matrix* of $\mathbf{f}(\mathbf{X}; \boldsymbol{\beta})$ with components

$$\mathbf{J}(\boldsymbol{\beta})_{j,i} := \frac{\partial f(\mathbf{x}_j; \boldsymbol{\beta})}{\partial \beta_i}$$

for $j = 1, \ldots, J$ and $i = 0, \ldots, I$. Alternatively,

$$\mathbf{J}(\boldsymbol{\beta}) = \begin{bmatrix} \nabla_{\boldsymbol{\beta}} f(\mathbf{x}_1; \boldsymbol{\beta})^{\mathsf{T}} \\ \cdot \\ \cdot \\ \cdot \\ \nabla_{\boldsymbol{\beta}} f(\mathbf{x}_J; \boldsymbol{\beta})^{\mathsf{T}} \end{bmatrix}.$$

The following example illustrates the above elements used to specify gradient descent.

Example 9.15. The expressions below are used to fit the nonlinear[a] model

$$y_j = f(x_j; \beta_0, \beta_1, \beta_2) + \epsilon_j = \beta_0 + \beta_1 e^{\beta_2 x_j} + \epsilon_j$$

$j = 1, \ldots, J$ to data using gradient descent. In this model, $\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2)$. Let

$$g(\boldsymbol{\beta}) = \sum_{j=1}^{J} (y_j - f(x_j; \beta_0, \beta_1, \beta_2))^2 = \sum_{j=1}^{J} g_j(\boldsymbol{\beta})^2,$$

where $g_j(\boldsymbol{\beta}) = y_j - f(x_j; \beta_0, \beta_1, \beta_2)$.

For this model

$$-\frac{1}{2}\nabla_{\boldsymbol{\beta}}\, g(\boldsymbol{\beta}) = \begin{bmatrix} \sum_{j=1}^{J} (y_j - (\beta_0 + \beta_1 e^{\beta_2 x_j})) \\ \sum_{j=1}^{J} (y_j - (\beta_0 + \beta_1 e^{\beta_2 x_j})) e^{\beta_2 x_j} \\ \sum_{j=1}^{J} (y_j - (\beta_0 + \beta_1 e^{\beta_2 x_j})) \beta_1 x_j e^{\beta_2 x_j} \end{bmatrix} = \mathbf{J}(\boldsymbol{\beta})^{\mathsf{T}} \mathbf{g}(\boldsymbol{\beta}),$$

where

$$\mathbf{J}(\boldsymbol{\beta}) = \begin{bmatrix} 1 & e^{\beta_2 x_1} & \beta_1 x_1 e^{\beta_2 x_1} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ 1 & e^{\beta_2 x_J} & \beta_1 x_J e^{\beta_2 x_J} \end{bmatrix} \quad \text{and} \quad \mathbf{g}(\boldsymbol{\beta}) = \begin{bmatrix} y_1 - (\beta_0 + \beta_1 e^{\beta_2 x_1}) \\ \cdot \\ \cdot \\ \cdot \\ y_J - (\beta_0 + \beta_1 e^{\beta_2 x_J}) \end{bmatrix}.$$

[a] If the errors were multiplicative and $\beta_0 = 0$, this model would be linearizable by taking logarithms.

Newton's method

Because gradient descent may converge slowly, Newton's method and its variants provide attractive alternative approaches 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

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

For a J-dimensional vector-valued function $(\mathbf{u})$ with components $f_j(\mathbf{u})$ for $j = 1, \ldots, J$ and $\mathbf{u} = (u_1, \ldots, u_J)$ 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 $\mathbf{f}(\mathbf{u})$.

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

$$\boldsymbol{\beta}^{n+1} = \boldsymbol{\beta}^n - (\nabla^2_{\boldsymbol{\beta}} g(\boldsymbol{\beta}^n))^{-1} \nabla_{\boldsymbol{\beta}} g(\boldsymbol{\beta}^n), \tag{9.82}$$

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

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

for $i = 0, \ldots, I$ and $k = 0, \ldots, I$.

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

Gauss-Newton iteration

Gauss-Newton is based on approximating the real-valued function $g_j(\boldsymbol{\beta})$ as defined in (9.77) by the first-order Taylor series approximation

$$g_j(\boldsymbol{\beta}) \approx g_j(\boldsymbol{\beta}^n) + (\nabla_{\boldsymbol{\beta}} g_j(\boldsymbol{\beta}^n))^\mathsf{T} (\boldsymbol{\beta} - \boldsymbol{\beta}^n). \tag{9.83}$$

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

$$g(\boldsymbol{\beta}) \approx (\mathbf{g}(\boldsymbol{\beta}^n) + \mathbf{J}(\boldsymbol{\beta}^n)(\boldsymbol{\beta} - \boldsymbol{\beta}^n))^\mathsf{T} (\mathbf{g}(\boldsymbol{\beta}^n) + \mathbf{J}(\boldsymbol{\beta}^n)(\boldsymbol{\beta} - \boldsymbol{\beta}^n)), \tag{9.84}$$

where $\mathbf{J}(\boldsymbol{\beta}^n)$ denotes the $J \times (I+1)$ Jacobian matrix of $\mathbf{g}(\boldsymbol{\beta}^n)$. Note that in (9.84) $\boldsymbol{\beta}^n$ is fixed and $\boldsymbol{\beta}$ is variable so that $\mathbf{g}(\boldsymbol{\beta}^n)$ is a fixed $J \times 1$ vector and $\mathbf{J}(\boldsymbol{\beta}^n)$ is a fixed $J \times (I+1)$ 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:

$$\boldsymbol{\beta}^{n+1} = \boldsymbol{\beta}^n - (\mathbf{J}(\boldsymbol{\beta}^n)^\mathsf{T} \mathbf{J}(\boldsymbol{\beta}^n))^{-1} \mathbf{J}(\boldsymbol{\beta}^n)^\mathsf{T} \mathbf{g}(\boldsymbol{\beta}^n). \tag{9.85}$$

The following comments are worth noting:

1. The recursion in (9.85) involves matrix inversion. Instead of inverting this matrix, one can solve the system of equations

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

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

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 $\mathbf{J}(\boldsymbol{\beta}^n)^\mathsf{T}\mathbf{J}(\boldsymbol{\beta}^n)$ 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.

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 do not believe the authors anticipated how significant and widely used this approach would become.

The texts by Bertsekas [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 LSVI and LSPI appear to be new and useful, although other bounds exist in the literature.

de Farias and Roy [2003] provides an in depth study of the approximate LP model. Moreover they analyze an instance of the queuing admission control model used for examples in this chapter and elsewhere.

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]. Sauré et al. [2015] also analyzes the queuing service rate control model. The shots gained metric was developed by Broadie [2014]. A similar metric, points gained, was developed for American football by Chan et al. [2021].

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

Exercises

1. Consider the queuing service rate control model analyzed throughout this chapter. Describe the basis functions you would use if you intended to approximate the value function by:

   (a) Aggregating states into subgroups.

   (b) A piecewise linear approximation that interpolates between a subgroup of states.

   (c) A piecewise quadratic approximation that interpolates between a subgroup of states.

2. Consider a two-state model with a single policy $d(s)$, with $\lambda = 1/2$,

$$\mathbf{r}_d = \begin{bmatrix} 4 \\ 2 \end{bmatrix} \quad \text{and} \quad \mathbf{P}_d = \begin{bmatrix} 0.5 & 0.5 \\ 1 & 0 \end{bmatrix}. \tag{9.86}$$

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

   (b) Represent these estimates graphically as in Figure 9.6. Include the coordinates for all quantities.

3. Consider the model in Example 9.3 using an approximation based on $b(s) = s$.

   (a) Obtain parameter estimates based on LSPE and compare them to those in the text.

   (b) Repeat your analysis using APLP.

   (c) Repeat your analysis with $b(s) = s^{0.5}$.

4. Compute linear and quadratic approximations to the value function in Example 9.5 using Bellman residual minimization and compare them to those in the text.

5. Repeat the analysis in Example 9.5 for $S = 0, \ldots, 100$ and a decision rule that always uses action a_2.

   (a) Compare the quality of quadratic and cubic approximations obtained using OLS, LSPE and Bellman residual minimization to the exact value function and its projection on col($\mathbf{B}$).

6. Consider the queuing service rate control model with $S = \{0, \ldots, 60\}$.

   (a) Solve the model exactly using a method of your choice.

   (b) **Extrapolation.** Find a quadratic value function approximation using LSVI or LSPI on $S' = \{0, \ldots, N\}$ for $N = 20, 30, 40$ and use them to obtain greedy policies for all of S. How do these policies compare to the optimal policy?

(c) Use the value function approximations obtained using the above subsets to obtain bounds on the value functions on all of S using (9.35).

(d) **State sampling.** Find a quadratic value function approximation by sampling a fraction of states and then applying LSVI or LSPI on this subset. Compare the resulting greedy policies and values functions to the optimal values and policies as in Figure 9.15. What can you conclude?

(e) **State aggregation.** Consider the effect of state aggregation on the accuracy of approximate value functions and corresponding greedy policies where basis functions are indicator functions of disjoint sequences of m consecutive states. Investigate the impact of m on the quality of the approximations.

7. **Oscillation of LSPI.**[41] 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 it 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 5.10.2 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 each.

9. Solve the instance of the advanced scheduling problem in Section 9.6 using policy iteration, linear programming and evaluate quadratic approximations obtained using LSPI and approximate linear programming.

10. For the advanced appointment scheduling model in Section 9.6, compare the greedy policy based on (9.52) obtained from a random sample of half the states to that in (9.50) based on applying LSVI to the entire state space.

11. (Mini project.) This open-ended problem asks you to carry out your own analysis of the advanced appointment scheduling model analyzed in Section 9.6. To do

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

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?

12. (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.

    Customer 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, LSMPI and linear programming. Choose suitable basis 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.

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

14. Show that all equivalences in (9.65) hold.

15. Consider the model in Example 9.15.

    (a) Write out the Newton's method and Gauss-Newton recursions for this model.

(b) Fit this model to the data in Table 9.3 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.

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

Table 9.3: Data for Exercise 15. The data generating model was $y_j = 2 + 3e^{0.3x_j} + \epsilon_j$ where $\epsilon_j \sim NID(0, 1)$.

16. Formulate the game of Scrabble as an MDP. What features would you use when approximating a value function?

17. Formulate the game of Tetris as an MDP. What features would you use when approximating a value function?