REINFORCEMENT LEARNING AND STOCHASTIC OPTIMIZATION

A unified framework for sequential decisions

Warren B. Powell

August 22, 2021

WILEY-
INTERSCIENCE

A JOHN WILEY & SONS, INC., PUBLICATION

Library of Congress Cataloging-in-Publication Data:

Optimization Under Uncertainty: A unified framework
Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

PART IV - POLICY SEARCH

Policy search is a strategy where we define a class of functions that determine a decision, and then search for the best function within that class. Policies in the policy search class can be divided into two subclasses:

Policy function approximations (PFAs) - PFAs are analytical functions that relate information in the state variables to decisions. PFAs come in three (overlapping) forms: lookup tables, parametric models, and nonparametric (or locally parametric) models, which are the same classes of functions used in machine learning. PFAs are typically limited to scalar actions or low-dimensional controls.

PFAs are covered in chapter 12, along with a general discussion of methods for policy search.

Cost function approximations (CFAs) - Parametric CFAs are parameterized optimization problems, where the parameterization guides the optimization problem to produce decisions that work well a) over time and b) under uncertainty. We first saw a parametric CFA in chapter 7 in the form of policies for multiarmed bandit problems such as an interval estimation policy

$$X^\pi(S_t|\theta) = \arg\max_{x \in \mathcal{X}} \left(\bar{\mu}_x^n + \theta \bar{\sigma}_x^n\right)$$

where $\mathcal{X} = \{x_1, \ldots, x_M\}$ is a discrete set of alternatives (ads, drugs) and where $\bar{\mu}_x^n$ is our current estimate of the performance of alternative x after n experiments, and $\bar{\sigma}_x^n$ is the standard deviation of $\bar{\mu}_x^n$. The parameter θ has to be tuned to optimize the policy.

The presence of the "$\arg\max$" operator opens the door to using optimization solvers which means the modified optimization problem can be a large linear, nonlinear or integer program. Now, x can be a high dimensional vector, with thousands, even hundreds of thousands, of variables. An example is scheduling flights for an airline where we have to introduce schedule slack for weather delays, or the scheduling of energy generators for the power grid, where schedules have to be set given the possibility of outages.

CFAs are covered in chapter 13.

Policy search applied to finding analytical policy function approximations has been widely studied in the academic literature. There are close parallels between policy search and classical machine learning: machine learning minimizes some distance metric between a model $f(x^n|\theta)$ and the corresponding observation y^n and requires a training dataset $(x^n, y^n), n = 1, \ldots, N$, while policy search requires a performance metric $C(S_t, x_t)$ and a model of the system given by the transition function $S_{t+1} = S^M(S_t, x_t, W_{t+1})$ and a model of the exogenous information process.

Parametric cost function approximations, on the other hand, represent a powerful strategy that has been widely used in practice (usually in an ad hoc manner), but almost completely ignored by the research literature, where it is viewed as a "deterministic heuristic." Our position is that it is just as valid as any parametric model used in machine learning. This book is the first to treat this approach as a valid algorithmic strategy for certain classes of stochastic optimization problems.

The policy search class of policies are simpler than the lookahead classes, and as a result they are quite popular. The academic literature places far more attention on the lookahead classes, but the policy search class is much more widely used in practice. The problem is that the price of simplicity is tunable parameters, and tuning is hard.

CHAPTER 12

POLICY FUNCTION APPROXIMATIONS AND POLICY SEARCH

A policy function approximation (PFA) is any analytical function mapping a state to an action. These "analytical functions" come in three broad (and overlapping) flavors:

Lookup tables - These consist of discrete inputs, and produce a discrete output. Examples are: "If the chess board is in this state, I take this move" or "If this is a male patient, over 50, never smoked, high blood sugar, then take this medication."

Parametric functions - These can be linear or nonlinear models, including neural networks. The user has to specify the structure of the model which is assumed to be governed by a vector of parameters θ, and then algorithms search for the best values of the parameters.

Nonparametric functions - Nonparametric functions might be locally constant approximations, locally linear defined over regions, or high-dimensional nonlinear functions such as deep neural networks.

What distinguishes policy function approximations from the other classes of policies we introduce later in the book is that each of the remaining classes has an imbedded optimization problem within the policy. As a result, PFAs are the simplest class of policy and the easiest to compute, but require a human (typically) to specify the architecture. Not surprisingly, given the wide range of decisions that we encounter throughout life, most decisions are made with simple rules that can be characterized as PFAs, so PFAs are arguably the most widely used class of policy in day-to-day decision making.

Most of our attention will be devoted to parametric functions that are characterized by a set of parameters which we denote by θ. Some examples are listed below.

EXAMPLE 12.1

A basic inventory policy is to order product when the inventory goes below some value θ^{min} where we order up to some upper value θ^{max}. If S_t is the inventory level, this policy might be written

$$X^\pi(S_t|\theta) = \begin{cases} \theta^{max} - S_t & \text{if } S_t < \theta^{min}, \\ 0 & \text{otherwise.} \end{cases}$$

EXAMPLE 12.2

If S_t is a scalar variable giving, for example, the rainfall over the last week, we might set a policy for releasing water from a reservoir using

$$X^\pi(S_t|\theta) = \theta_0 + \theta_1 S_t + \theta_2 S_t^2.$$

EXAMPLE 12.3

A popular strategy in the engineering community is to train a policy $U^\pi(S_t|\theta)$ for controlling a robot (or a rocket like SpaceX) using a neural network which is characterized by a set of layers and a set of weights that are captured by θ (we provided a brief description of neural networks in section 3.9.3) which takes as input a state variable S_t and outputs a control u_t.

Each of these examples involves a policy parameterized by a parameter vector θ. In principle, we can represent a lookup table using this notation where there is a parameter θ_s for each discrete state s. However, most problems exhibit a large (potentially infinite) number of states, which translates to an equally large (and potentially infinite) number of parameters. There are techniques for optimizing over high-dimensional parameter vectors as long as we can compute gradients exactly (which we develop later in this chapter). However, most applications will be lower-dimensional, and can be optimized using the methods of chapters 5 and 7.

We begin by describing different classes of policies where we focus on policies that have attracted some attention in the literature. Afterward, we turn our attention to the much harder task of optimizing these parameters. The foundation of this process starts with one of our objective functions such as

$$\max_{\theta \in \Theta^\pi} \mathbb{E} \left\{ \sum_{t=0}^{T} C(S_t, X^\pi(S_t|\theta)) | S_0 \right\}, \tag{12.1}$$

where $S_{t+1} = S^M(S_t, X^\pi(S_t|\theta), W_{t+1})$, and where the expectation is over the beliefs in S_0 (if applicable) and the different possible sequences $W_1, \ldots, W_T$. The search is over some space Θ^π that corresponds to the class of policy we have chosen. As we show, this disarmingly simple formulation can be quite hard to solve. However, we have to remember that PFAs are likely the most widely used class of policy in the vast range of sequential decision problems.

12.1 POLICY SEARCH AS A SEQUENTIAL DECISION PROBLEM

All policy search methods start from the basic idea of simulating a sample path ω giving us a performance metric such as

$$\hat{F}^{\pi}(\theta,\omega) = \sum_{t=0}^{T} C\big(S_t(\omega), X^{\pi}(S_t(\omega)|\theta)\big), \tag{12.2}$$

where $S_{t+1}(\omega) = S^M(S_t(\omega), X^{\pi}(S_t(\omega)|\theta), W_{t+1}(\omega))$, where we are following a sample path $W_1(\omega),\ldots,W_T(\omega)$. If we let $W=(W_1,\ldots,W_T)$ represent the entire sequence of random variables (dropping the index ω), we can write this problem using the standard form of a stochastic search problem given by

$$\max_{\theta} F^{\pi}(\theta) = \mathbb{E}F^{\pi}(\theta, W). \tag{12.3}$$

Of course, we only work with simulations of $\hat{F}^{\pi}(\theta,\omega)$, but the form in equation (12.3) is the standard way of writing stochastic search problems.

The objective function in equation (12.3) describes a sequential decision problem characterized by our five elements: 1) the state S_t, 2) the policy $X^{\pi}(S_t|\theta)$, 3) the exogenous information process W_t, 4) the transition function $S_{t+1} = S^M(S_t, X^{\pi}(S_t|\theta), W_{t+1})$, and 5) the objective function (12.3), just as we outlined in chapter 9.

The problem of searching for θ is its own sequential decision problem, which consists of the same five components:

1) The state of the algorithm $S^{\theta,n}$, which includes our belief B^n about the function $F^{\pi}(\theta)$.

2) The decision θ^n which is determined by the θ-policy $\theta^n = \Theta^{\pi}(S^{\theta,n})$.

3) The exogenous information, which would be the outcome of a simulation of the policy $\hat{F}^{\pi}(\theta,\omega)$ from equation (12.3).

4) The transition function

$$S^{\theta,n+1} = S^{\theta,M}(S^{\theta,n}, \theta^n, \hat{F}^{\pi}(\theta,\omega^{n+1})),$$

which are the equations for updating the belief B^n given the point θ^n at which we observed the function, and the observed performance $\hat{F}^{\pi}(\theta^n,\omega^{n+1})$, where ω^{n+1} is the sample path used for the $n+1^{st}$ simulation.

5) The objective function, where we use the terminal performance of our learning policy π^{lrn} for learning θ after N iterations:

$$\max_{\pi^{lrn}} \mathbb{E}_{S^0}\mathbb{E}_{W^1,\ldots,W^N|S^0}\mathbb{E}_{\widehat{W}|S^0}\left\{F(\theta^{\pi,N},\widehat{W})|S^{\theta,0}|S^0\right\}$$

See equation (7.5) in chapter 7 for an in-depth discussion of this objective function.

Now we face the same issues as we do designing an implementation policy $X^{\pi}(S_t|\theta)$. This is the challenge we address in this chapter by reviewing both derivative-based and derivative-free methods of performing parameter tuning for any PFA-based policy.

12.2 CLASSES OF POLICY FUNCTION APPROXIMATIONS

A policy function approximation can, quite simply, use any of the strategies used in machine learning that we reviewed in chapter 3: lookup tables, parametric functions (which includes neural networks), and nonparametric functions (including deep neural networks), as well as any hybrids. The only difference between machine learning and policy function approximations is the objective function, as well as the data requirements. The reader is encouraged to flip back to section 1.6.2 where we made this connection. The main point is that machine learning involves solving the search problem over functions $f \in \mathcal{F}, \theta \in \Theta^f$ which we write as

$$\min_{\theta=(f\in\mathcal{F},\theta\in\Theta^f)} \frac{1}{N}\sum_{n=1}^{N}\left(y^n - f(x^n|\theta)\right)^2,$$

where we need the training dataset $(x^n, y^n),\ n = 1,\ldots,N$. By contrast, policy search involves solving

$$\min_{\theta=(f\in\mathcal{F},\theta\in\Theta^f)} \mathbb{E}\sum_{t=0}^{t} C(S_t, X^\pi(S_t|\theta)),$$

where we do not need a training dataset, but we do need the system model $S_{t+1} = S^M(S_t, X^\pi(S_t|\theta), W_{t+1})$ and the model of the exogenous information process S_0, $W_1,\ldots,W_T$. Otherwise, both are searching over the same classes of functions $f \in \mathcal{F}$ which includes lookup tables, parametric and nonparametric functions, and any associated parameters $\theta \in \Theta^f$.

12.2.1 Lookup table policies

A lookup table policy is a function where for a particular discrete state s we return a discrete action $x = X^\pi(s)$. This means we have one parameter (an action) for each state. We exclude from this class any policies that can be parameterized by a smaller number of parameters.

Lookup tables are relatively common in practice, since they are easy to understand. Some examples are:

- The Transportation Safety Administration (TSA) has specific rules that determine when and how a passenger should be searched.
- Call-in centers use specific rules to govern how a call should be routed.
- Expert chess players are able to look at a board (in the initial stages of a game) and know exactly what move to make.
- A doctor will often take a set of symptoms and patient characteristics to determine the right treatment.

Lookup tables are easy to understand, and easy to enforce. But in practice, they can be very hard to optimize since there is a value (the action) for each state. So, if we have $|\mathcal{S}| = 1000$ states, searching directly for the best policy would mean searching over a 1000-dimensional parameter space (the action to be taken in each state).

One attraction of lookup table policies is that they are very easy to compute in production; imagine a real-time setting where decisions have to be made with exceptional speed. In business, lookup table policies are widely used where they are known as business rules, although these rules may often be parameterized. In practice these rules are not optimized using formal methods; this chapter will indicate how to do this.

12.2.2 Boltzmann policies for discrete actions

A Boltzmann policy chooses a discrete action $x \in \mathcal{X}_s$ according to the probability distribution

$$f(x|s,\theta) = \frac{e^{\theta \bar{C}(s,x)}}{\sum_{x' \in \mathcal{X}} e^{\theta \bar{C}(s,x)}}.$$

where $\bar{C}(s,x)$ is some sort of contribution to be maximized. This could be our estimate of a function $\mathbb{E}F(x,W)$ as we did in chapter 7, or an estimate of the one-step contribution plus a downstream value, as in

$$\bar{C}(S^n,x) = C(S^n,x) + \mathbb{E}\{\overline{V}^n(S^{n+1})|S^n,x\},$$

where $\overline{V}^n(S)$ is our current estimate of the value of being in state S.

Let $F(x|S^n,\theta)$ be the cumulative distribution of our probabilities

$$F(x|s,\theta) = \sum_{x' \le x} f(x'|s,\theta).$$

Let $U \in [0,1]$ be a uniformly distributed random number. Our policy $X^\pi(s|\theta)$ could be written

$$X^\pi(s|\theta) = \arg\max_x \{F(x|s,\theta)|F(x|s,\theta) \le U\}.$$

This is an example of a so-called "stochastic policy," but we handle it just as we would any other policy.

Boltzmann policies are often referred to as "soft-max" because the actions with the highest estimated value are given the highest probability of being accepted. As θ increases, the probability of choosing the decision x with the highest $\bar{C}(s,x)$ quickly approaches 1.0. The purpose of using values of θ so that there is a reasonable probability of choosing actions with less attractive values is that we can observe how well the decision performs, and update our estimate of $\bar{C}(s,x)$.

12.2.3 Linear decision rules

A linear decision rules (also known as an "affine policy") is any policy that is linear in the unknown parameters. Thus, a linear decision rule policy might be of the form

$$X^\pi(S_t|\theta) = \theta_0 + \theta_1\phi_1(S_t) + \theta_2\phi_2(S_t).$$

A simple illustration might be a rule for setting the insulin dosage x of a drug given the blood sugar h_t of a patient. We might propose a dosing strategy given by

$$X^\pi(S_t|\theta) = \theta_0 + \theta_1 h_t + \theta_2 h_t^2 + \theta_3 h_t^3.$$

Now the challenge is determining the vector θ that keeps blood sugar within a specified range.

We first saw linear decision rules in chapter 4 when we presented the linear quadratic control problem which, in our notation, is given by

$$\min_{\theta} \mathbb{E} \sum_{t=0}^{T} \big((S_t)^T Q_t S_t + (X^\pi(S_t|\theta))^T R_t X^\pi(S_t|\theta)\big). \tag{12.4}$$

After considerable algebra, it is possible to show that the optimal policy $X_t^*(S_t)$ is given by

$$X_t^*(S_t) = K_t S_t,$$

where K_t is a suitably dimensioned matrix that is a function of the matrices Q_t and R_t. Of course, we assume that S_t and x_t are continuous vectors. Thus, $X^*(S_t)$ is a linear function of S_t with coefficients determined by the matrix K_t. See section 14.11 for more details.

This result requires that the objective function be quadratic (or a mixture of quadratic and linear) functions of the state S_t and control x_t. It also requires that the problem be unconstrained, which can be a reasonable starting point for many problems in robotic controls where forces x_t can be positive or negative, and where some constraints (such as the maximum force) would simply not be binding.

Linear decision rules have been applied to other problems, but care has to be used. Linear approximations of functions can be quite useful in a particular region of the function, but a policy $X^\pi(S_t)$ has to work well over the entire range of states S_t that we might actually encounter. Low-dimensional linear models (such as a quadratic approximation) can incur fitting errors, while higher-dimensional models are harder to fit, especially when experiments are expensive.

12.2.4 Monotone policies

There are a number of problems where the decision increases, or decreases, with the state variable. If the state variable is multidimensional, then the decision (which we assume is scalar) increases, or decreases, with *each* dimension of the state variable. Policies with this structure are known as *monotone policies*. Some examples include:

- There are a number of problems with binary actions that can be modeled as $x \in \{0, 1\}$. For example
  - We may hold a stock ($x_t = 0$) or sell ($x_t = 1$) if the price p_t falls below a smoothed estimate $\bar{p}_t$ which we compute using

    $$\bar{p}_t = (1 - \alpha)\bar{p}_{t-1} + \alpha p_t.$$

    Our policy is then given by

    $$X^\pi(S_t|\theta) = \begin{cases} 1 & \text{if } p_t \leq \bar{p}_t - \theta \\ 0 & \text{otherwise.} \end{cases}$$

    The function $X^\pi(S_t|\theta)$ decreases monotonically in p_t (as p_t increases, $X^\pi(S_t|\theta)$ goes from 1 to 0).

  - A shuttle bus waits until there are at least R_t customers on the bus, or it has waited τ_t. The decision to dispatch goes from $x_t = 0$ (hold the bus) to $x_t = 1$ (dispatch the bus) as R_t exceeds a threshold θ^R or as τ_t exceeds θ^τ, which means the policy $X^\pi(S_t|\theta)$ increases monotonically in both state variables $S_t = (R_t, \tau_t)$.

- A battery is being used to buy power from the grid when electricity prices p_t fall below a lower limit θ^{min}, or sell when the price goes above θ^{max}. The battery does nothing when $\theta^{min} < p_t < \theta^{max}$. We write the policy as

$$X^\pi(S_t|\theta) = \begin{cases} \text{-1} & \text{if } p_t \leq \theta^{min}, \\ 0 & \text{if } \theta^{min} < p_t < \theta^{max}, \\ 1 & \text{if } p_t \geq \theta^{max}. \end{cases} \tag{12.5}$$

  We see that $X^\pi(S_t|\theta)$ increases monotonically in the state $S_t = p_t$.

- Dosages for blood sugar control increase with both the weight of the patient, and with the patient's glycemic index. The policy is in the form of a lookup table, with different dosages for each range of weight and glycemic index.

Each of these policies is controlled by a relatively small number of parameters, although this is not always the case. For example, if we use a fine discretization of the patient's weight and glycemic index, we could find that we need to specify hundreds of dosages. However, monotonicity can dramatically reduce the search process.

12.2.5 Nonlinear policies

The term "nonlinear policy" pretty much covers any policy that has a single parametric form, which is not linear in the parameters θ that can be tuned. This includes:

- There are many problems that have specific structure. Our decision might be a continuous quantity such as the amount of water to apply to a wildfire, or the dosage of a drug to be given to a patient. We might feel that the policy will have a S-curve behavior with respect to a variable such as the intensity of the wildfire, or the weight of a patient, which can be described by

$$X^\pi(S_t|\theta) = \frac{1}{1 + e^{\theta_0 + \theta_1\phi_1(S_t) + \ldots + \theta_F \theta_F \phi_F(S_t)}}.$$

  The term $\phi_1(S_t)$ might capture the intensity of the fire or weight of the patient, while the other terms might capture other variables that shift the S-curve.

- A "buy low, sell high" policy such as the one in equation (12.5) is a kind of nonlinear policy. It is not smooth, since the function increases in steps as the price increases.

- Neural networks - A neural network (even a small neural network) is a high-dimensional nonlinear model that can have thousands to millions of parameters. The advantage of neural networks is that they can fit virtually any functional form, which seems to suggest that we do not have to know the form. Neural networks have actually been used for decades in primarily deterministic engineering control problems where the decision might be a three-dimensional force on a device.

  Neural networks have three weaknesses:

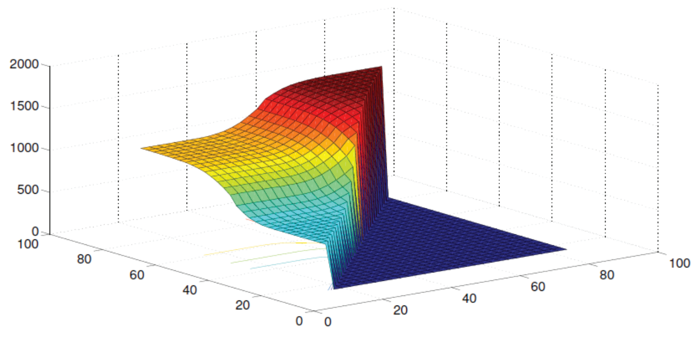


Figure 12.1 Illustration of a complex nonlinear (monotone) function.

– Neural networks are very high-dimensional architectures, which means they need a lot of data. This problem is magnified when there is noise (most uses of neural networks are applied to deterministic problems such as pattern recognition or robotic control).

– Neural networks are very flexible (they can fit virtually any function) which means they can overfit, which means that they struggle with noisy data, as can easily happen when simulating a policy.

– It is hard to make neural networks reflect structure such as monotonicity (the higher the price, the lower the demand).

As of this writing, neural networks have attracted considerable attention from the computer science community (and they have been used for a long time in engineering control problems), but care has to be used given the issues listed above. They have attracted considerable attention in the context of optimizing games, which are low noise (you just have the behavior of your opponent) and it is possible to run millions of simulated games to train the policies.

12.2.6 Nonparametric/locally linear policies

The problem with parametric models is that sometimes functions are simply too complex to fit with low-order parametric models. For example, imagine that our policy looks like the function shown in figure 12.1. Simple quadratic fits will not work, and higher-order polynomials will struggle due to overfitting unless the number of observations is extremely large.

We could handle very general functions if we could use lookup tables (which may require that we discretize any continuous parameters). However, lookup tables can become extremely large when we have three or more dimensions in our state variable. Even three dimensional lookup tables quickly grow to thousands to millions of elements. The problem is compounded when the search algorithm has to evaluate actions for each state many times to handle noise.

A surprisingly powerful strategy for many problems with continuous states and actions is to assume locally linear responses. For example, S_t may capture the level of a reservoir, or the current speed and altitude of a helicopter. The control x_t could be the rate at which water is released from the reservoir, or the forces applied to the helicopter. Assume that we use our understanding of the problem to create a family of regions $\mathcal{S}_1, \ldots, \mathcal{S}_I$, which are most likely going to be a set of rectangular regions (or intervals if there is only one dimension). We might then create a family of linear (affine) policies of the form

$$X_i^\pi(S_t|\theta) = \theta_{i0} + \theta_{i1}\phi_1(S_t) + \theta_{i2}\phi_2(S_t),$$

for $S_t \in \mathcal{S}_i$ where $\mathcal{S}_i$ is a user-defined region of the state space (there are only a few of these).

This approach has been found to be very effective in some classes of control problems. In practice, the regions $\mathcal{S}_i$ are designed by someone with an understanding of the physics of the problem. Further, instead of tuning one vector θ, we have to tune $\theta_1, \ldots, \theta_I$. While this involves considerable testing and tuning, the approach can work quite well and offers the important feature that the resulting policy can be computed extremely quickly.

12.2.7 Contextual policies

Imagine that we have designed a policy $X^\pi(S_t|\theta)$ that depends on S_t and is parameterized by θ. This policy is actually the solution to the problem

$$\max_{\pi=(f\in\mathcal{F},\theta\in\Theta^f)} \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, X^\pi(S_t|\theta))|S_0\right\}. \tag{12.6}$$

Recall that $f \in \mathcal{F}$ reflects the search over policy classes (and we may have fixed this to one class), while $\theta \in \Theta^F$ captures the search over any tunable parameters for that class (and we always have this search). For now, let's assume that we have also fixed the policy to some class $f \in \mathcal{F}$, so that we are only optimizing over θ.

The attentive reader will note that we always write our objective function in this way, which means we express the explicit dependence on the initial state S_0, which captures deterministic parameters, initial values of dynamic parameters, and prior beliefs. This means that our optimized θ actually should be written $\theta(S_0)$. In other words, if we change our initial conditions, we may have to retune θ.

Some communities refer to the initial state S_0 as the "context" of the problem. If S_0 captures stable, static parameters, then it is unlikely to change very much. However, it could happen that we re-tune our policy every quarter (as happens in financial settings), so that the policy picks up current market conditions, which can be very complex. If our sequential decision problem is a search algorithm, then S_0 might be the starting point of the algorithm, while θ governs the behavior of the stepsize policy.

12.3 PROBLEM CHARACTERISTICS

When designing a policy search method it is important to understand the characteristics of how the system responds to the parameterized policy. Some important dimensions of problem characteristics include:

Computational complexity - How you approach policy search will be quite different if a single simulation of a policy is a fraction of a second, or hours, or days, or longer. Methods based on viewing the simulator as a black-box tend to require more function evaluations.

Level of noise - Policy simulations can be reasonably stable, especially for very large systems where aggregate behavior is more stable, but they can also be extremely random.

The response surface - It may be concave, smooth but only unimodular, nonconcave with local maxima, and it may feature jumps (think about buy low, sell high policies).

Parameter dimensionality - We can divide parameters into three classes

- Scalar models - There are a number of applications with a single scalar parameter (think of our Boltzmann policy)
- Low-dimensional continuous models - In many applications the number of tunable parameters is less than five or 10, which may mean that it is too large to do a full grid search (discretizing each dimension and searching over all values), but it may simplify computing numerical derivatives.
- High-dimensional continuous models - It is easy to create policies with tens to hundreds of parameters, or even hundreds of thousands. A good example of a high-dimensional policy is a neural network, but it could also arise with a linear model with a high-dimensional state variable (as might arise in the management of complex resources).

Stationarity - The process we are controlling may be:

- Stationary, which means the parameters of the underlying process are not changing over time.
- Periodic (such as time of day patterns).
- Nonstationary, which also comes in different forms. For example, imagine we are controlling a basic inventory problem. This problem may feature:
  - Smooth transitions as demand for product steadily increases or decreases, or exhibits smooth seasonal transitions.
  - Bursts, when a product suddenly gets popular for a period of time.
  - Shifts, such as a sudden increase in demand following an advertising campaign or change in price.
  - Spikes, such as a spike in electricity prices which encourages sudden selling.

12.4 FLAVORS OF POLICY SEARCH

Given a parametric (or locally parametric) function parameterized by θ (typically a vector, but not always), we now face the challenge of finding the best value of θ. There are different dimensions to the policy search problem:

Derivative-based vs. derivative free - These include:

Derivative-based methods - Derivative-based methods are attractive when optimizing vectors of continuous parameters and where we feel comfortable that $\mathbb{E}F^\pi(\theta, W)$ is smooth (note that the expectation often helps considerably to smooth functions). The vast majority of derivative-based methods use the classical first-order, stochastic gradient algorithm described in chapter 5:

$$\theta^{n+1} = \theta^n + \alpha_n \nabla_\theta F^\pi(\theta^n, W^{n+1}). \tag{12.7}$$

We can divide derivative-based methods into two broad categories:

- Numerical derivatives - Numerical derivatives are estimates of derivatives which use only the simulations $F^\pi(\theta, \omega)$, without requiring any actual derivative information of $\nabla_\theta F^\pi(\theta, \omega)$. These methods have been described in chapter 5, but we will review methods based on numerical derivatives.
- Exact derivatives - These methods exploit the underlying structure of a sequential decision problem to compute derivatives exactly, avoiding the need for expensive numerical derivatives.

Derivative-free methods - These methods all view the policy simulator as a black box, and use the methods described in chapter 7. Let $S^{\theta,n}$ be what we know about $\mathbb{E}F^\pi(\theta, W)$ (not to be confused with S_t within our sequential decision problem), and let $\Theta^\pi(S^{\theta,n})$ be our policy for choosing the parameter vector θ^n based on what we know in $S^{\theta,n}$. The update rule $\Theta^\pi(S^{\theta,n})$ can be any of the four classes of policies described in chapter 7.

Online vs. offline learning - In online learning, we are learning in an environment where updates come to us. As a rule, we have to live with the performance of our policy, which means we are maximizing the cumulative reward. Most policy search uses some form of adaptive algorithm, although this can be done in a laboratory where we use one policy, the *learning policy*, to find the best policy to implement, called the *implementation policy*. Some in the reinforcement learning community refer to the learning policy as the *behavior policy* while the implementation policy is the *target policy*. Many refer to the learning policy as an algorithm; we think the relationship to policies creates a bridge to our entire framework with the four classes of policies.

Performance-based vs. supervised learning - Most policy search uses as a goal to maximize the total reward (either the final reward or cumulative reward), but there are settings where we have an "expert" (the supervisor) who will specify what to do, allowing us to fit our policies to the choices of the supervisor. The expert decisions could come from a physician making decisions, a financial trader making traders, or a dispatcher assigning drivers to loads. This turns a policy search problem (maximizing a performance metric) into a machine learning (effectively "predicting" what the supervisor will do), or a hybrid, where we balance a performance metric against matching the decisions of an exogenous decision-maker.

As we review the methods, there will be a clear tradeoff between efficiency and complexity. We are going to start with the simplest methods, which are those which treat $F^\pi(\theta)$ as a black box and, as a result, do not exploit any structural properties of the underlying problem. This includes derivative-free methods, and derivative-based methods using numerical derivatives.

We then progress to derivative-based methods where we work with analytical derivatives of the gradient which exploit the structure of the underlying problem. For our presentation, these come in two flavors:

Discrete dynamic programs - These are problems where we are at a node (state) s, choose a discrete action a and then transition to a node s' with probability $P(s'|s,a)$ (which we represent but generally cannot compute). An important subclass of graph problems are those where actions are chosen at random (known as a stochastic policy), but transitions are made deterministically. Here, we wish to optimize a parameterized policy $A^\pi(s|\theta)$, where action $a_t = A^\pi(S_t|\theta)$ is discrete.

Continuous control problems - In this setting we choose a continuous control x_t (which may be vector-valued) that impacts the state S_{t+1} in a continuous way through a known (and differentiable) transition function.

Both problem classes have attracted considerable attention, and illustrate different methods for computing gradients.

Our presentation will proceed from the simplest methods to the most sophisticated:

- Derivative-based policy search using numerical derivatives - Section 12.5.
- Derivative-free policy search - Section 12.6.
- Derivative-based with exact derivatives: continuous dynamic programs - Section 12.7.
- Derivative-based with exact derivatives: discrete dynamic programs - Section 12.8.

The first two methods treat the policy simulator as a black box, and makes virtually no assumptions about the internal structure of the problem. These methods are simplest, but the price you pay is that you will have to deal with the potentially high noise of a policy simulator. Also, while simulating a policy can be quite fast, there are many applications where this is computationally intensive, requiring several minutes to hours to days or more for numerical simulations. In addition, there are settings where we do not have access to a simulator, and have to do our policy search in the field, where it takes a day to observe a day.

The third method derives an explicit formula for the gradient, which requires knowing specific relationships within the model. The derivatives required to compute the gradient are all computed for a specific sample path, and therefore avoid any complexities associated with expectations.

The fourth method is designed for discrete dynamic programs, and works directly from the expectation-based form of the objective function. This is a mathematically advanced presentation for readers with a strong probability background (which is the reason that it is marked with a **).

12.5 POLICY SEARCH WITH NUMERICAL DERIVATIVES

Any "black box" model starts with our assumption that we can perform a simulation of the policy $X^\pi(S_t|\theta)$ by simulating a sample path to get an estimate of

$$\hat{F}(\theta,\omega) = \sum_{t=0}^{T} C(S_t(\omega), X^\pi(S_t(\omega)|\theta)). \tag{12.8}$$

While there are different ways of estimating derivatives numerically, we are going to focus on the SPSA algorithm ("simultaneous perturbation stochastic approximation") which is designed for settings where θ is a vector, which we first presented in section 5.4.4. In theory SPSA can produce estimates of the gradient $\nabla_\theta F(\theta, \omega)$, regardless of the dimension of θ, with just two simulations. In practice, these estimates can be quite noisy, motivating using multiple simulations and averaging.

The method works as follows:

1) Let $Z_k, k = 1, \ldots, K$ be a vector of zero-mean random variables, and let Z^n be a sample of this vector at iteration n.

2) Create perturbed values of θ^n using $\theta^{n+} = \theta^n + \eta^n Z^n$ and $\theta^{n-} = \theta^n - \eta^n Z^n$, where η^n is a scaling sequence (it is typically chosen as a constant that does not vary with n).

3) Let $W^{n+1,+}$ and $W^{n+1,-}$ represent two different samples of the random variables driving the simulation (these can be generated in advance or on the fly). There is no meaning to the $+$ and $-$ in the superscript other than to indicate that these are the samples that are run to evaluate θ^{n+} and θ^{n-}.

4) Run the simulation twice, once to find $\hat{F}^{n+} = F(\theta^{n+}, W^{n+1,+})$, and once to find $\hat{F}^{n-} = F(\theta^{n-}, W^{n+1,-})$.

5) It is common that we have to run multiple simulations and take an average. Let $W_m^{n+1,+}$ be m^{th} sample of the random information series and let

$$\hat{F}_m^{n+1,+}(\theta^{n+1,+}) = F(\theta^{n+1,+}, W_m^{n+1,+}),$$

represent the performance of the m^{th} simulation which we run m^{batch} times (this is called a "mini-batch"). Let $\hat{F}_m^{n+1,-}(\theta^{n+1,-})$ be parallel sets of runs. We then take an average

$$\bar{F}^{n+1,+}(\theta^{n+1,+}) = \frac{1}{m^{batch}} \sum_{m=1}^{m^{batch}} \hat{F}_m^{n+1,+}(\theta^{n+1,+}).$$

$\bar{F}^{n+1,-}(\theta^{n+1,-})$ is computed similarly.

6) Compute the estimate of the gradient using

$$g^{n+1}(\theta^n) = \begin{bmatrix} \frac{\bar{F}^{n+1,+}(\theta^{n+1,+}) - \bar{F}^{n+1,-}(\theta^{n+1,-})}{2\eta^n Z_1^n} \\ \frac{\bar{F}^{n+1,+}(\theta^{n+1,+}) - \bar{F}^{n+1,-}(\theta^{n+1,-})}{2\eta^n Z_2^n} \\ \vdots \\ \frac{\bar{F}^{n+1,+}(\theta^{n+1,+}) - \bar{F}^{n+1,-}(\theta^{n+1,-})}{2\eta^n Z_P^n} \end{bmatrix} \tag{12.9}$$

We then use this in our stochastic gradient algorithm

$$\theta^{n+1} = \theta^n + \alpha_n g^{n+1}(\theta^n). \tag{12.10}$$

While the basic gradient updating formula (12.10) is disarmingly simple (hence the reason we presented it first), it hides the need to experiment with stepsize formulas (covered in chapter 6), tuning parameters required by the stepsize formula, and tuning the size of the mini-batch (which may need to vary by iteration).

Stochastic gradients can be effective and easy to implement, but be prepared to spend some time tuning the algorithm to get good results.

12.6 DERIVATIVE-FREE METHODS FOR POLICY SEARCH

In this section we provide a tour through chapter 7 on methods that only require that we be able to perform simulations of a policy. We remind the reader of the four classes of policies that can be used to perform derivative-free stochastic search:

Policy function approximation (PFA) - Section 7.4 - These are simple rules, and below we suggest one that accelerates a simple statistical learning method. Of course, the price of simplicity is (yet another) tunable parameter.

Cost function approximation (CFA) - Section 7.5 - Simple CFAs include upper confidence bounding and interval estimation for problems with discrete alternatives. Below we suggest a strategy for applying these ideas to policy search.

Value function approximation (VFA) - Section 7.6 - VFA-based policies are relatively complex and have not yet been demonstrated to significantly outperform simpler methods. For this reason we do not cover these methods here.

Direct lookahead (DLA) - Section 7.7 - The knowledge gradient is a one-step lookahead (easily modified to be a restricted multistep lookahead) which has proven useful in the context of expensive function evaluations requiring smaller budgets.

12.6.1 Belief models

We can draw on a number of the different belief structures presented in chapter 3. Some that are likely to be useful in the representation of continuous vectors for the parameter vector θ include:

- Lookup table with correlated beliefs - Also known as Gaussian process regression (technically this is one form of GPR), this could work well for vectors θ with one to three dimensions. GPR does not impose any structural assumptions other than smoothness, but this also means that it is not able to produce functions that are known to be concave, convex or unimodular.

- Low-dimensional linear models (e.g. quadratic) - Low dimensional linear models can be used in a number of settings, spanning anywhere from one to dozens of variables. Particularly useful are methods that work to fit a low-dimensional model in the vicinity of the optimum (which, of course, we are trying to find).

- Sparse linear models - These models extend the linear models to the domain of high-dimensional vectors, but where we think that many of the elements of θ may be zero.

- Sampled belief models - There are problems with special structure that suggest a particular type of nonlinear model, such as logistic regression for a pricing or recommendation system. If the nonlinear function $f(x|\theta)$ is parameterized by an unknown vector θ, we might represent the uncertainty in our belief by a family of possible values $\theta \in \{\theta^1, \ldots, \theta^K\}$.

- Neural networks - We have described gradient-based search models using neural network policies (which can be very high dimensional), but the biggest strength of neural networks, which is their flexibility to replicate any functions, is also their biggest weakness, since this flexibility requires very large datasets. Their flexibility also means that they can overfit noisy data.

It is helpful, even important, to represent not only our best estimate of the belief, but also the uncertainty in the belief. We can do this for lookup tables (including with correlated beliefs) and linear models. We can do this for nonlinear models using the technique of using a sampled belief model, where we maintain a population of possible values of the unknown parameter vector θ and the probability that each is the true value. However, we are not able to do this with neural networks.

We encourage the reader to review the different policies in chapter 7, but we provide some simple illustrations that have proven useful.

12.6.2 Learning through perturbed PFAs

One of the most popular heuristics for optimizing an unknown function is to use the first n observations, $(x^0, y^1), (x^1, y^2), \ldots, (x^{n-1}, y^n)$ to create a belief $\bar{f}^n(x|\bar{\theta}^n)$ using any of the methods in chapter 7. Then, we could compute

$$x^n = \arg\max_x \bar{f}^n(x|\bar{\theta}^n), \tag{12.11}$$

which we then use to run a simulation to obtain the updated sample

$$y^{n+1} = F(x^n, W^{n+1}).$$

It turns out that this simple idea is surprisingly ineffective, as illustrated in figure 12.2 for the setting of learning how sales responds to price, where we need to learn the relationship between sales and price, while maximizing revenue. Figure 12.2(a) shows three different possible sales response curves, where we are making the simplistic assumption that this relationship is linear in price (remember that *any* parametric model is at best going to be locally accurate).

Now assume that we use our best estimate of the sales response to create a best estimate of revenue as a function of price, and then set the price to maximize the revenue (as we would if we used equation (12.11) to determine the next point to observe). The problem with this is that we end up testing prices near the apparent optimum, as illustrated in figure 12.2(b). The problem with these observations is that it requires that we learn the sales response from a series of noisy observations that are clustered together, which makes it virtually impossible to get a reliable estimate of the sales curve.

The best way to learn the sales curve is to make observations that are as far from the center as possible, as shown in figure 12.2(c). There are two problems with this strategy. First, our sales response model is only an approximation; in this example we assume it is linear in price, which is clearly accurate only near the middle. The second problem is that if we are learning in the field, these would be points where we perform poorly (that is, we would expect to receive very low revenue).

The most effective strategy is illustrated in figure 12.2(d), showing observations that are not too close to the optimum, but not too far. This is known as “sampling the shoulders” of the function.

The idea of sampling a function in a region *around* the optimum, rather than the optimum itself, is supported by an analysis of the value of information from sampling each point. Figure 12.3 shows the value of information for a scalar function, $\bar{f}^n(x|\bar{\theta}^n)$ computed using the knowledge gradient (see section 7.7.2 and section 7.8), which shows that there are peaks to the value of information that is some distance from the optimum.

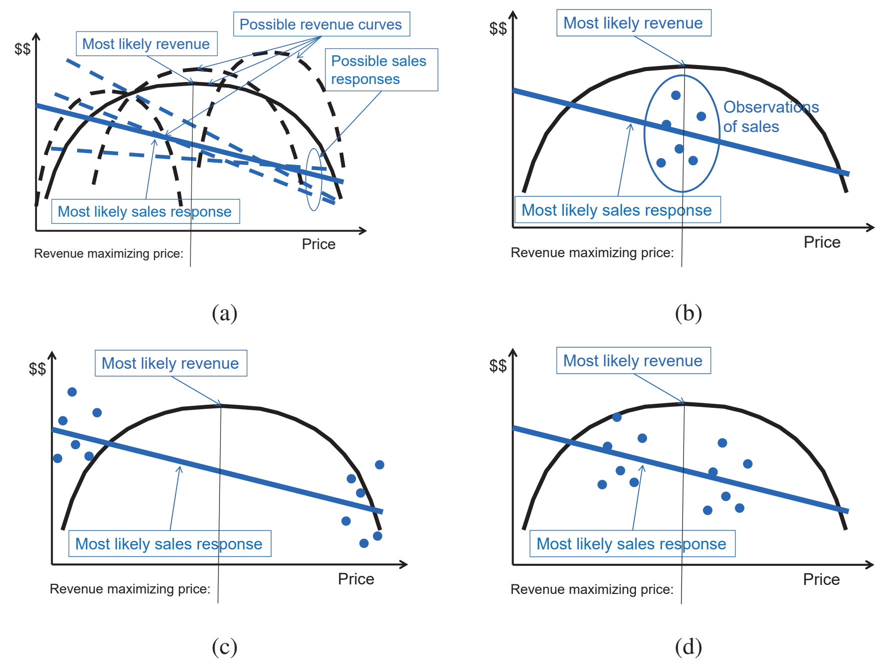


Figure 12.2 Actively learning a demand response function: (a) Three possible sales response lines and corresponding revenue curves, (b) Observed price-sales combinations if we use prices that appear to maximize revenue, (c) Observing extreme prices (high and low) to improve learning of sales response, and (d) Balancing learning (observing away from the middle) and earning (observing prices near the middle).

This raises the question: how to find this peak? The calculations used to compute the knowledge gradient are more complex, and still require knowing something about the behavior of the true function, which would never be true in practice. For this reason, an interesting strategy is to take this insight and design a simple policy (which falls in the PFA class). In fact, we suggest two policies:

An optimum-deviation policy - The idea here is to pick a point x^n that is a distance ρ from the optimum $\bar{x}^n = \arg\max_x \bar{f}^n(x|\bar{\theta}^n)$. If x is a $k-$dimensional vector, this deviation can be created by sampling k normally distributed random variables $Z_1, \ldots, Z_K$, each with mean 0 and variance 1, and then normalizing them so that

$$\sqrt{\sum_{k=1}^{K} Z_k^2} = \rho.$$

Let $\bar{Z}^n$ be the resulting $k-$dimensional vector. Now compute the sampling point using

$$x_k^n = \bar{x}_k^n + \bar{Z}_k^n.$$

Note that in one dimension, we would have $\bar{Z}^n = \pm\rho$.

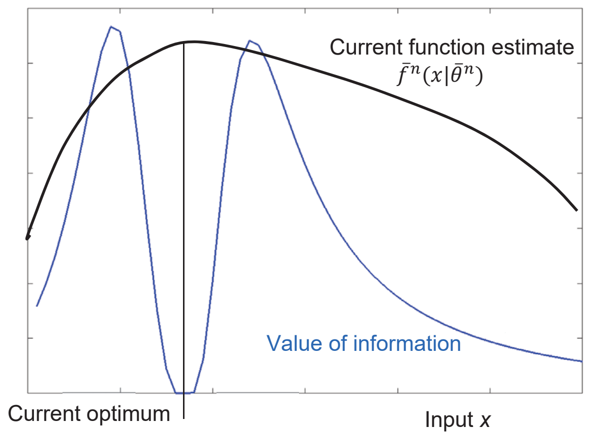


Figure 12.3 Plot of the value of information from sampling x over a range, showing the highest value of information that is some distance from the current apparent optimum.

An excitation policy - Here we again generate a $k-$dimensional perturbation vector Z^n, where each element has mean 0 and variance 1, and then set

$$x_k^n = \bar{X}_k^n + \rho Z_k^n.$$

While the optimum-deviation policy forces x^n to be a distance ρ from the optimum $\bar{x}^n$, an excitation policy simply introduces a random perturbation with mean 0, which means the most likely point to sample is the optimum of $\bar{f}^n(x|\bar{\theta}^n)$.

The excitation policy is more natural in a setting where we are learning in the field using a cumulative reward objective, providing an additional incentive to sample in the vicinity of the apparent optimum, while still forcing some exploration. The optimum-deviation policy will produce faster learning, but at a price to how well we do while we are learning, which is best if we are using a final-reward objective.

We have to remind ourselves that these policies are designed for tuning the parameter vector θ of an implementation policy, but we now have a new tunable parameter, ρ. Fortunately, we may be able to pick a reasonable value of ρ a-priori. We first note that virtually any search algorithm benefits from an assumption that the data x can be scaled. For example, we may assume that we can scale each dimension of x to be between 0 and 1, or normally distributed with mean 0 and variance 1. When we do this, we might feel that ρ will likely be between 0.1 and 0.5.

12.6.3 Learning CFAs

Section 7.5 describes a number of CFA policies for derivative-free stochastic search that can all be used for parameter search. We illustrate two policies that work through the same mechanism which highlights an important characteristic of active learning policies (which describes *any* policy where decisions affect a belief about unknown parameters).

Interval estimation - Start by assuming that we can represent the feasible region for x by a finite (or sampled) set $\mathcal{X} = \{x^1, \ldots, x^K\}$. Let $\bar{\mu}_x^n$ be our estimate of

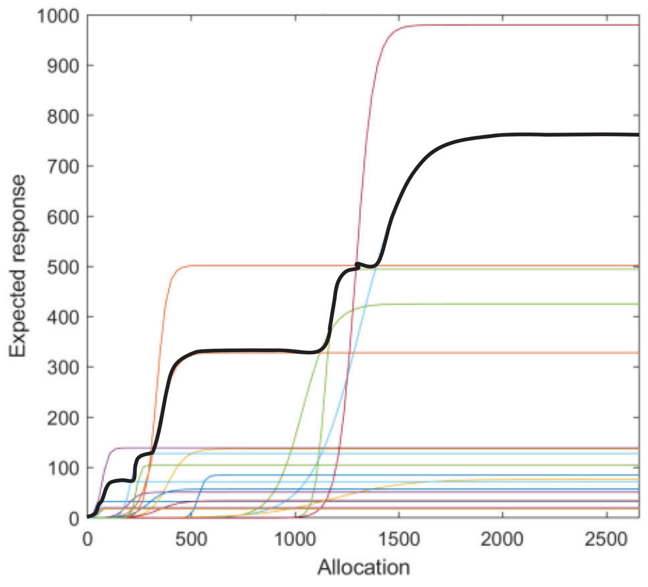


Figure 12.4 Sampled belief model, with 95th percentile highlighted (second highest belief).

$f(x) = \mathbb{E}F(x,W)$ for $x \in \mathcal{X}$ after n experiments. Since $f(x)$ is a continuous surface, it makes sense to use correlated beliefs (also known as Gaussian process regression) which we introduced in section 3.4.2. Recall that we would maintain a covariance matrix Σ^n.

A basic interval estimation policy is given by

$$X^{IE}(S^n|\theta^{IE}) = \arg\max_{x \in \mathcal{X}} \left(\bar{\mu}_x^n + \theta^{IE}\bar{\sigma}_x^n\right), \tag{12.12}$$

where $\bar{\sigma}_x^n = \sqrt{\Sigma_{xx}^n}$. Note that our state S^n is our belief $B^n = (\bar{\mu}^n, \Sigma^n)$.

Sampled θ-percentile - A policy closely related to interval estimation is to explicitly capture the θ-percentile. Figure 12.4 shows a sampled belief model with 20 possible beliefs. If we set $\theta = 0.95$, this means taking the second-highest belief, which is shown as the solid black line. As above, we still have to tune θ.

Both the interval estimation policy and the sampled θ-percentile policies make recommendations based on an optimistic estimate of the estimated function. With interval estimation, the random variable $\bar{\mu}_x^n$ will be normally distributed (from the central limit theorem), so if we pick $\theta^{IE} = 2$, for example, we will be making our choices based on the 95th percentile of the function.

Similarly, if we use $K = 20$ samples in our sampled belief model, we could use the 19th highest sample (as we did in figure 12.4) and again obtain a 95th percentile estimate. Of course, the percentile we use is a tunable parameter that depends on the size of our experimental budget, and whether we are doing offline (final reward) or online (cumulative reward) learning. For expensive functions (and small learning budgets), the best value of θ will likely be a declining function of the number of experiments that have been completed.

There is a substantial literature that analyzes policies based on the principle of using optimistic estimates using the broad term of *upper confidence bounding*. The idea is that learning improves when using optimistic estimates of the function, since the current estimate may, as a result of experimental noise, underestimate the true function.

12.6.4 DLA using the knowledge gradient

A form of direct lookahead is the knowledge gradient which we first introduced in section 7.7.2 (see also section 7.8) which is a one-step lookahead. The knowledge gradient has been found to be particularly useful for functions that are relatively expensive to evaluate, which limits the size of our experimental budget.

Figure 12.5 illustrates the knowledge gradient on a two-dimensional surface which is estimated using correlated beliefs (see section 7.8.5 for a summary of how to compute the knowledge gradient with correlated beliefs, and section 3.4.2 for the updating equations for correlated beliefs). Note that the knowledge gradient (on the right) is highest in regions of the function farthest from prior measurements, while the knowledge gradient is smallest at points that have just been evaluated (which minimizes uncertainty).

12.6.5 Comments

There is a general theme that runs through these policies (and throughout the literature on active learning problems), which is that you want to perform function evaluations that strike a balance between maximizing uncertainty, while simultaneously maximizing the *possibility* that the point in the function may prove to be best. This means that it is not enough to maintain a belief about the function $\bar{f}^n(x|\bar{\theta}^n)$; we also have to maintain a belief about our *uncertainty* in the function at each point. This section highlights the methods that we have found to be most effective in our own work.

12.7 EXACT DERIVATIVES FOR CONTINUOUS SEQUENTIAL PROBLEMS*

We are now going to derive an exact gradient (technically, a stochastic gradient in the style of chapter 5) of the performance of a policy with respect to the parameters θ that govern the performance of the policy. This section will focus on problems where the state S_{t+1} is a differentiable function with respect to the state S_t and decision x_t, as might arise when we are managing resources (water, blood, money).

We return again to our basic sequential optimization problem

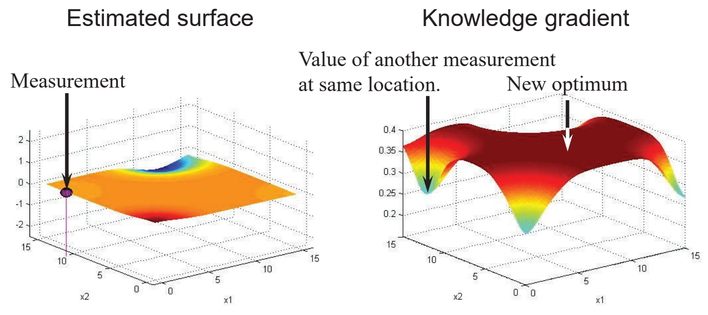
$$F^\pi(\theta) = \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, X_t^\pi(S_t|\theta))|S_0\right\}, \tag{12.13}$$

where our dynamics evolve (as before) according to

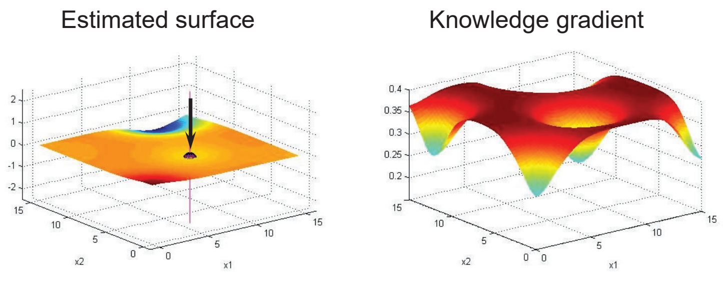
$$S_{t+1} = S^M(S_t, x_t, W_{t+1}),$$

where we are given an initial state S_0 and access to observations of the sequence $W = (W_1, \ldots, W_T)$. Our goal in this section is to find the gradient $\nabla_\theta F^\pi(\theta, \omega)$ exactly for a particular sample path ω (rather than using a numerical derivative).

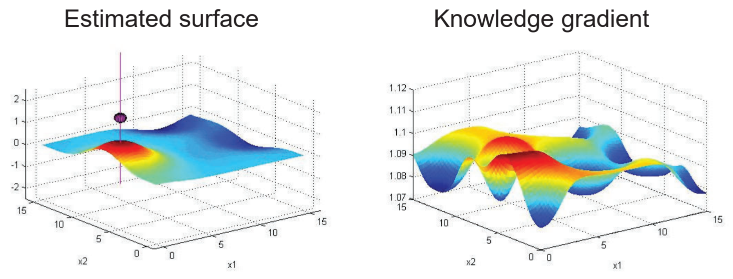
We have written our policy $X_t^\pi(S_t)$ in a time-dependent form for generality, but this means estimating time-dependent parameters θ_t that characterize the policy. In most



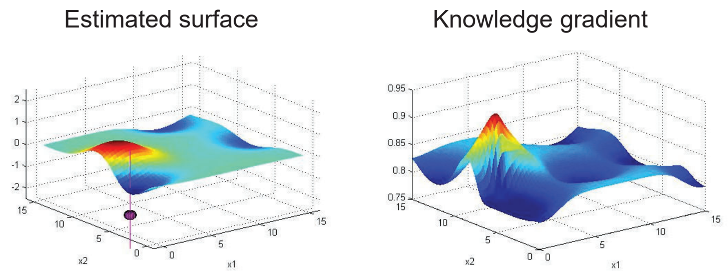
After four samples



After five samples



After seven samples



After nine samples

Figure 12.5 The knowledge gradient with correlated beliefs being applied to a two-dimensionsional surface. The plots on the left are the beliefs after n samples, while the plots on the right plot the knowledge gradient at each point.

applications we would use the stationary version $X^\pi(S_t)$, with a single set of parameters θ. However, when we can compute the gradient exactly, we can handle high-dimensional parameters much more efficiently than methods based on numerical derivatives can (SPSA may seem like magic, but it isn't!).

Continuous sequential problems problems are distinguished from discrete dynamic programs specifically because we assume we can compute $\partial S_{t+1}/\partial x_t$. With discrete dynamic programs, we assumed the actions a were categorical (e.g. left/right or red/green/blue). In that setting, we had to consider the downstream impact of a decision made now by capturing the effect of changing the policy parameter θ on the probability of which state we would visit. Now we can capture this impact directly.

There are two approaches for minimizing $F^\pi(\theta)$ over the parameter vector θ:

Batch learning - Here we replace (12.13) with an average over N samples, giving us

$$\bar{F}^\pi(\theta) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T} C(S_t(\omega^n), X_t^\pi(S_t(\omega^n)|\theta)), \tag{12.14}$$

where $S_{t+1}(\omega^n) = S^M(S_t(\omega^n), X^\pi(S_t(\omega^n)), W_{t+1}(\omega^n))$ is the sequence of states generated following sample path ω^n. This is a classical statistical estimation problem.

Adaptive learning - Rather than solving a single (possibly very large) batch problem, we can use our standard stochastic gradient updating logic (from chapter 7)

$$\theta^{n+1} = \theta^n + \alpha_n \nabla_\theta F^\pi(\theta^n, W^{n+1}).$$

This update is executed following each forward pass through the simulation.

Both approaches depend on computing the gradient $\nabla_\theta F^\pi(\theta, \omega)$ for a given simple path ω from which we generate a sequence of state $S_{t+1} = S^M(S_t, x_t, W_{t+1}(\omega))$ where $x_t = X^\pi(S_t)$. Normally we would write $S_t(\omega)$ or $x_t(\omega)$ to indicate the dependence on sample path ω, but we suppress this here for notational compactness.

We find the gradient by differentiating (12.13) with respect to θ, which requires a meticulous application of the chain rule, recognizing that the contribution $C(S_t, x_t)$ is a function of both S_t and x_t, the policy $X^\pi(S_t|\theta)$ is a function of both the state S_t and the parameter θ, and the state S_t is a function of the previous state S_{t-1}, the previous control x_{t-1}, and the most recent exogenous information W_t, which is assumed to be independent of the control, although this could be handled. This gives us

$$\begin{aligned}\nabla_\theta F^\pi(\theta,\omega) \;=\; & \left(\frac{\partial C_0(S_0,x_0)}{\partial x_0}\right)\left(\frac{\partial X_0^\pi(S_0|\theta)}{\partial \theta}\right) + \sum_{t'=1}^{T}\left[\left(\frac{\partial C_{t'}(S_{t'}, X_{t'}^\pi(S_{t'}|\theta))}{\partial S_{t'}}\frac{\partial S_{t'}}{\partial \theta}\right)\right. \\ & \left. + \frac{\partial C_{t'}(S_{t'},x_{t'})}{\partial x_{t'}}\left(\frac{\partial X_{t'}^\pi(S_{t'}|\theta)}{\partial S_{t'}}\frac{\partial S_{t'}}{\partial \theta} + \frac{\partial X_{t'}^\pi(S_{t'}|\theta)}{\partial \theta}\right)\right],\end{aligned} \tag{12.15}$$

where

$$\frac{\partial S_{t'}}{\partial \theta} = \frac{\partial S_{t'}}{\partial S_{t'-1}}\frac{\partial S_{t'-1}}{\partial \theta} + \frac{\partial S_{t'}}{\partial x_{t'-1}}\left[\frac{\partial X_{t'-1}^\pi(S_{t'-1}|\theta)}{\partial S_{t'-1}}\frac{\partial S_{t'-1}}{\partial \theta} + \frac{\partial X_{t'-1}^\pi(S_{t'-1}|\theta)}{\partial \theta}\right]. \tag{12.16}$$

The derivatives $\partial S_{t'}/\partial \theta$ are computed using (12.16) by starting at $t' = 0$ where

$$\frac{\partial S_0}{\partial \theta} = 0,$$

and stepping forward in time.

Equations (12.15) and (12.16) require that we be able to take derivatives of the cost function, the policy, and the transition function. We assume this is possible, although the complexity of these derivatives is highly problem dependent.

12.8 EXACT DERIVATIVES FOR DISCRETE DYNAMIC PROGRAMS**

This section is going to take a significant step up in complexity in its derivation of analytical derivatives for policy parameters in the context of discrete dynamic programs. These are problems where decisions are categorical: left-right, color, or a product recommendation. For the advanced (and determined) reader, this presentation provides a different perspective into the mathematics of sequential decision problems that works directly with expectations rather than the simulation-based strategy used in section 12.7.

To emphasize the use of discrete actions, we are going to switch notation to action a rather than our usual decision x. We assume that we are going to maximize the single-period expected reward in steady state. We use the following notation

$r(s,a)$	$=$	reward if we are in state $s \in \mathcal{S}$ and take action $a \in \mathcal{A}_s$,
$A^\pi(s\|\theta)$	$=$	policy that determines the action a given that we are in state s, which is parameterized by θ,
$P_t(s'\|s,a)$	$=$	probability of transitioning to state s' given that we are in state s and take action a at time t (we use $P(s'\|s,a)$ if the underlying dynamics are stationary),
$d^\pi_t(s\|\theta)$	$=$	probability of being in state s at time t while following policy π.

This notation reflects the classical notation of the reinforcement learning community, which has adopted the notation from Markov decision processes (we will see this in much more detail in chapter 14). Normally we would use a transition function, but here we are using the one-step transition matrix (we showed how to calculate the one-step transition matrix from the transition function in section 9.7). Also, we are using for the first time the probability of being in a state while following policy π, given by $d^\pi_t(s|\theta)$, although we previously used the idea of computing the expectation over the states in section 9.11 (look at the objective function for class 4).

We first introduce a parameterized stochastic policy which is typically required for problems where decisions are discrete with no particular structure (e.g. red-green-blue). We note that the parameters that we are optimizing are primarily controlling the balance of exploration and exploitation. We then present the objective function (there is more than one way to write this, as we show later). Finally, we describe a computable method for taking the gradient of this objective function.

12.8.1 A stochastic policy

We follow the standard practice in the literature of using what is called a stochastic policy, where an action a is chosen probabilistically. We represent our policy using

$p^\pi_t(a\|s,\theta) =$	the probability of choosing action a at time t, given that we are in state s, where θ is a tunable parameter (possibly a vector).

Most of the time we will use a stationary policy that we denote $\bar{p}^\pi(a|s,\theta)$ which can be viewed as a time-averaged version of our policy $p_t^\pi(a|s,\theta)$ which we might compute using

$$\bar{p}^\pi(a|s,\theta) = \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} p_t^\pi(a|s,\theta).$$

A particularly popular policy (especially in computer science) assumes that actions are chosen at random according to a Boltzmann distribution (also known as Gibbs sampling). Assume at time t that we have

$$\bar{Q}_t(s,a) \quad = \quad \text{estimated value at time } t \text{ of being in state } s \text{ and taking action } a.$$

Next define the probabilities (using our familiar Boltzmann distribution)

$$p_t^\pi(a|s,\theta) = \frac{e^{\theta \bar{Q}_t(s,a)}}{\sum_{a'\in\mathcal{A}_s} e^{\theta\bar{Q}_t(s,a')}}. \tag{12.17}$$

We can compute the values $\bar{Q}_t(s,a)$ using $\bar{Q}_t(s,a) = r(s,a)$, although this means choosing actions based on immediate rewards. Alternatively, we might use

$$\bar{Q}_t(s,a) \quad = \quad r(s,a) + \max_{a'} \bar{Q}_{t+1}(s',a'),$$

where s' is chosen randomly from simulating the next step (or sampling from the transition matrix $P_t(s'|s,a)$ if this is available). We first saw methods for computing Q-values under the umbrella of reinforcement learning in section 2.1.6.

If we are modeling a stationary problem, it is natural to transition to a stationary policy. Let $\bar{p}^\pi(a|s,\theta)$ be our stationary action probabilities where we replace the time-dependent values $\bar{Q}_t(s,a)$ with stationary values $\bar{Q}(s,a)$ computed using

$$\bar{Q}^\pi(s,a|\theta) \quad = \quad r(s,a) + \mathbb{E}\left\{\sum_{t'=1}^{T} r(S_{t'}, A^\pi(S_{t'}|\theta))|S_0 = s, a_0 = a\right\}. \tag{12.18}$$

This is the total reward over the horizon from starting in state s and taking action a (note that we could use average or discounted rewards, over finite or infinite horizons). We remind the reader we are never going to actually compute these expectations. Using these values, we can create a stationary distribution for choosing actions using

$$\bar{p}^\pi(a|s,\theta) = \frac{e^{\theta\bar{Q}^\pi(s,a|\theta)}}{\sum_{a'\in\mathcal{A}_s} e^{\theta\bar{Q}^\pi(s,a'|\theta)}}. \tag{12.19}$$

Finally, our policy $A^\pi(s|\theta)$ is to choose action a with probability given by $p_t^\pi(a|s,\theta)$. The development below does not require that we use the Boltzmann policy, but it helps to have an example in mind.

12.8.2 The objective function

To develop the gradient, we have to start by writing out our objective function which is to maximize the average reward over time, given by

$$F^\pi(\theta) = \lim_{T\to\infty}\frac{1}{T}\left\{\sum_{t=0}^{T}\sum_{s\in\mathcal{S}}\left(d_t^\pi(s|\theta)\sum_{a\in\mathcal{A}_s} r(s,a)p_t^\pi(a|s,\theta)\right)\right\}. \tag{12.20}$$

A more compact form involves replacing the time-dependent state probabilities with their time averages (since we are taking the limit). Let

$$\bar{d}^\pi(s|\theta) = \lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T} d_t^\pi(s|\theta).$$

We can then write our average reward per time period as

$$F^\pi(\theta) = \sum_{s\in\mathcal{S}} \bar{d}^\pi(s|\theta) \sum_{a\in\mathcal{A}_s} r(s,a)\bar{p}^\pi(a|s,\theta). \tag{12.21}$$

12.8.3 The policy gradient theorem

We are now ready to take derivatives. Differentiating both sides of (12.21) and applying the chain rule gives us

$$\nabla_\theta F^\pi(\theta) \;=\; \sum_{s\in\mathcal{S}}\left(\nabla_\theta \bar{d}^\pi(s|\theta)\sum_{a\in\mathcal{A}_s} r(s,a)\bar{p}^\pi(a|s,\theta) + \bar{d}^\pi(s|\theta)\sum_{a\in\mathcal{A}_s} r(s,a)\nabla_\theta\bar{p}^\pi(a|s,\theta)\right). \tag{12.22}$$

While we cannot compute probabilities such as $d^\pi(s)$, we can simulate them (we show this below). We also assume we can compute $\nabla_\theta\bar{p}^\pi(a|s,\theta)$ by differentiating our probability distribution in (12.19). Derivatives of probabilities such as $\nabla_\theta\bar{d}^\pi(s|\theta)$, however, are another matter.

This is where the development known as the *policy gradient theorem* helps us. This theorem tells us that we can calculate the gradient of $F^\pi(\theta)$ with respect to θ using

$$\frac{\partial F^\pi(\theta)}{\partial\theta} \;=\; \sum_s d^\pi(s|\theta)\sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}Q^\pi(s,a), \tag{12.23}$$

where $Q^\pi(s,a)$ is given by

$$Q^\pi(s,a|\theta) \;=\; \sum_{t=1}^{\infty}\mathbb{E}\{r(s_t,a_t) - F^\pi(\theta)|s_0 = s, a_0 = a\}.$$

This is the expected difference between rewards earned each time period from a starting state, and the expected reward (given by $F^\pi(\theta)$) earned each period when we are in steady state. We will not be able to compute this derivative exactly, but we show below that we can produce an unbiased estimate without too much difficulty. What is most important is that, unlike equation (12.22), we do not have to compute (or even approximate) $\nabla_\theta\bar{d}^\pi(s|\theta)$. We pick this derivation up in the appendix in section 12.10.1. If you are willing to trust that equation (12.23) is true, read on!

12.8.4 Computing the policy gradient

As is always the case in stochastic optimization, the challenge boils down to computation. To help the discussion, we repeat the policy gradient result:

$$\frac{\partial F^\pi(\theta)}{\partial\theta} \;=\; \sum_s d^\pi(s|\theta)\sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}Q^\pi(s,a). \tag{12.24}$$

We start by assuming that we have some analytical form for the policy which allows us to compute $\partial \bar{p}^\pi(a|s,\theta)/\partial\theta$ (which is the case when we use our Boltzmann distribution). This leaves the stationary probability distribution $d^\pi(s|\theta)$, and the marginal rewards $Q^\pi(s,a)$.

Instead of computing $d^\pi(s|\theta)$ directly, we instead simply simulate the policy, depending on the fact that over a long simulation, we will visit each state with probability $d^\pi(s|\theta)$. Thus, for large enough T, we can compute

$$\nabla_\theta F^\pi(\theta) \approx \frac{1}{T}\sum_{t=1}^{T}\sum_{a}\frac{\partial \bar{p}^\pi(a|s_t,\theta)}{\partial\theta}Q^\pi(s_t,a), \tag{12.25}$$

where we simulate according to a known transition function $s_{t+1} = S^M(s_t, a, W_{t+1})$. We may simulate the process from a known transition function and a model of the exogenous information process W_t (if this is present), or we may simply observe the policy in action over a period of time.

This then leaves us with $Q^\pi(s_t,a)$. We are going to approximate this with estimates that we call $\bar{Q}^\pi_t(S_t|\theta)$, which we will compute by running a simulation starting at time t until T (or some horizon $t+H$). This requires running a different simulation that can be called a roll-out simulation, or a lookahead simulation. To avoid confusion, we are going to let $\tilde{S}_{tt'}$ be the state variable at time t' in a roll-out simulation that is initiated at time t. We let $\widetilde{W}_{tt'}$ be the simulated random information between $t'-1$ and t' for a simulation that is initiated at time t. Recognizing that $\tilde{S}_{tt} = S_t$, we can write

$$\bar{Q}^\pi_t(S_t|\theta) = \mathbb{E}_W \frac{1}{T-t}\sum_{t'=t}^{T-1} r(\tilde{S}_{tt'}, A^\pi(\tilde{S}_{tt'}|\theta)),$$

where $\tilde{S}_{t,t'+1} = S^M(\tilde{S}_{tt'}, A^\pi(\tilde{S}_{tt'}|\theta), \widetilde{W}_{t,t'+1})$ represents the transitions in our lookahead simulation. Of course, we cannot compute the expectation, so instead we use the simulated estimate

$$\bar{Q}^\pi_t(S_t|\theta) \approx \frac{1}{T-t}\sum_{t'=t}^{T-1} r(\tilde{S}_{tt'}, A^\pi(\tilde{S}_{tt'}|\theta)). \tag{12.26}$$

We note that while we write this lookahead simulation as spanning the period from t to T, this is not necessary. We might run these lookahead simulations over a fixed interval $(t, t+H)$, and adjust the averaging accordingly.

We now have a computable estimate of $F^\pi(\theta)$ which we obtain from (12.26) by replacing $Q^\pi_t(S_t|\theta)$ with $\bar{Q}^\pi_t(S_t|\theta)$, giving us a sampled estimate of policy π using

$$F^\pi(\theta) \quad \approx \quad \sum_{t=0}^{T-1}\hat{Q}^\pi_t(S_t|\theta).$$

The final step is actually computing the derivative $\nabla_\theta F^\pi(\theta)$. For this, we are going to turn to numerical derivatives. Assume the lookahead simulations are fairly easy to compute. We can then obtain estimates of $\nabla_\theta \hat{Q}^\pi_t(S_t|\theta)$ using the finite difference. We can do this by perturbing each element of θ. If θ is a scalar, we might use

$$\nabla_\theta \hat{Q}^\pi_t(S_t|\theta) = \frac{\hat{Q}^\pi_t(S_t|\theta+\delta) - \hat{Q}^\pi_t(S_t|\theta-\delta)}{2\delta}. \tag{12.27}$$

If θ is a vector, we might do finite differences for each dimension, or turn to simultaneous perturbation stochastic approximation (SPSA) (see section 5.4.3 for more details).

This strategy was first introduced under the name of the REINFORCE algorithm. It has the nice advantage of capturing the downstream impact of changing θ on later states, but in a very brute force manner. This is actually a form of direct lookahead policy which we cover in depth in chapter 19.

Phew! Now you see why we marked this section with a **!

12.9 SUPERVISED LEARNING

An entirely different approach to developing PFAs is to take advantage of the presence (if available) of an external source of decisions which we call "the supervisor." This might be a domain expert (such as a doctor making medical decisions, radiologists interpreting X-rays, or drivers operating a car), or perhaps simply a different optimization-based policy such as a deterministic lookahead. Supervised learning for decision problems is exactly analogous to supervised learning for machine learning.

Imagine that we have a set of decisions x^n from an external source (human or computer). Let S^n be our state variable, representing information available when the n^{th} decision was made. For the moment, assume that we have access to a dataset $(S^n, x^n)_{n=1}^N$ from past history. Now we face a classical machine learning problem of fitting a function (policy) to this data. Start by assuming that we are going to use a simple linear model of the form

$$X^\pi(S|\theta) = \sum_{f\in\mathcal{F}} \theta_f \phi_f(S),$$

where $(\phi_f(S))_{f\in\mathcal{F}}$ is a set of features designed by a human (there is a vast machinery of statistical learning tools we can bring to bear on this problem). We can use our batch dataset to estimate $X^\pi(S|\theta)$, although more often we can use the tools in chapter 3 to adapt to new data in an online fashion.

Several issues arise when pursuing this approach:

- Our policy is never better than our supervisor, although in many cases a policy that is as good as an experienced supervisor might be quite good.

- In a recursive setting, we need to design algorithms that allow the policy to adapt as more data becomes available. Using a neural network, for example, can result in significant overfitting, producing unexpected results as the function adapts to noisy data.

- If our supervisor is a human, we are going to be limited in the number of times we can query our domain expert, raising the problem of efficiently designing questions.

Supervised learning can be a powerful strategy for finding an initial policy, and then using policy search methods (derivative-based or derivative-free) to further improve the policy. However, we face the issue of collecting data from our supervisor. If we have an extensive database of decisions and the corresponding state variables that capture the information we would use to make a decision, then we simply have a nice statistical challenge (albeit, not necessarily an easy one). However, it is often the case that we have to work with data arriving sequentially in an online manner. We can approach our policy estimation in two ways:

Active policy search - Here we are actively involved in the operation of the process to design better policies. We can do this in two ways:

Active policy adjustment - This involves adjusting the parameters controlling the policy, as we described above with policy search.

Active state selection - We may choose the state that then determines the decision. This might be in the form of choosing hypothetical situations (e.g. patient characteristics) and then asking the expert for his/her decision.

Passive policy search - In this setting, we are following some policy, and then selectively using the results to update our policy.

Active state selection is similar to derivative-free stochastic search (chapter 7). Instead of choosing x to obtain a noisy observation of $F(x) = \mathbb{E}F(x, W)$, we are choosing a state S to get a (possibly noisy) observation of an action x from some source. Active state selection can only be done in an offline setting (we cannot choose the characteristics of a patient walking into the hospital, but we can pose the characteristics of a hypothetical patient), but we are limited in terms of how many questions we can pose to our supervisor, especially if it is human (but also if it is a time consuming optimization model).

Passive policy search is an approach where we use our policy $X^\pi(S_t)$ to make decisions x_t that are then used to update the policy. Of course, if all we did was feed our own decisions back into the same function that produced the decisions, then we would not learn anything. However, it is possible to perform a weighted statistical fit, where we put a higher weight on decisions that perform better.

12.10 WHY DOES IT WORK?

12.10.1 Derivation of the policy gradient theorem

We are going to provide the detailed derivation of

$$\frac{\partial F^\pi(\theta)}{\partial\theta} = \sum_s d^\pi(s|\theta) \sum_a \frac{\partial \bar{p}^\pi(a|s,\theta)}{\partial\theta} Q^\pi(s,a). \tag{12.28}$$

that we started in section 12.8.3.

We begin by defining two important quantities:

$$\begin{aligned}
Q^\pi(s,a|\theta) &= \sum_{t=1}^{\infty} \mathbb{E}\{r(s_t,a_t) - F^\pi(\theta)|s_0 = s, a_0 = a\}, \\
V^\pi(s|\theta) &= \sum_{t=1}^{\infty} \mathbb{E}\{r(s_t,a_t) - F^\pi(\theta)|s_0 = s\}, \\
&= \sum_{a\in\mathcal{A}} \bar{p}^\pi(a_0 = a|s,\theta) \sum_{t=1}^{\infty} \mathbb{E}\{r(s_t,a_t) - F^\pi(\theta)|s_0 = s, a_0 = a\}, \\
&= \sum_a \bar{p}^\pi(a|s,\theta) Q^\pi(s,a).
\end{aligned} \tag{12.29}$$

Note that $Q^\pi(s,a|\theta)$ is quite different than the quantities $\bar{Q}^\pi(s,a|\theta)$ used above for the Boltzmann policy (which is consistent with Q-learning, which we first saw in section 2.1.6).

$Q^\pi(s,a|\theta)$ sums the difference between the reward each period and the steady state reward per period (a difference that goes to zero on average), given that we start in state s and initially take action a. $V^\pi(s|\theta)$ is simply the expectation over all initial actions actions a as specified by our probabilistic policy

We next rewrite $Q^\pi(s,a)$ as the first term in the summation, plus the expected value of the remainder of the infinite sum using

$$\begin{aligned} Q^\pi(s,a) &= \sum_{t=1}^{\infty} \mathbb{E}\{r_t - F^\pi(\theta)|s_0 = s, a_0 = a\}, \\ &= r(s,a) - F^\pi(\theta) + \sum_{s'} P(s'|s,a)V^\pi(s'), \quad \forall s,\ a, \qquad (12.30) \end{aligned}$$

where $P(s'|s,a)$ is the one-step transition matrix (recall that this does not depend on θ). Solving for $F^\pi(\theta)$ gives

$$F^\pi(\theta) = r(s,a) + \sum_{s'} P(s'|s,a)V^\pi(s') - Q^\pi(s,a). \qquad (12.31)$$

Now, note that $F^\pi(\theta)$ is not a function of either s or a, even though they both appear in the right hand side of (12.31). Noting that since our policy must pick some action, $\sum_{a\in\mathcal{A}} \bar{p}^\pi(a|s,\theta) = 1$, which means

$$\sum_{a\in\mathcal{A}} \bar{p}^\pi(a|s,\theta)F^\pi(\theta) = F^\pi(\theta), \quad \forall a.$$

This means we can take the expectation of (12.31) over all actions, giving us

$$F^\pi(\theta) = \sum_a \bar{p}^\pi(a|s,\theta)\left(r(s,a) + \sum_{s'} P(s'|s,a)V^\pi(s') - Q^\pi(s,a)\right), \qquad (12.32)$$

for all states s. Taking a deep breath, we can now take derivatives using the following steps (explanations follow the equations):

$$\begin{aligned} \frac{\partial F^\pi(\theta)}{\partial\theta} &= \frac{\partial}{\partial\theta}\left(\sum_a \bar{p}^\pi(a|s,\theta)\left(r(s,a) + \sum_{s'} P(s'|s,a)V^\pi(s') - Q^\pi(s,a)\right)\right) \qquad (12.33) \\ &= \sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta} r(s,a) + \sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}\sum_{s'} P(s'|s,a)V^\pi(s') \\ &\quad + \sum_a \bar{p}^\pi(a|s,\theta)\sum_{s'} P(s'|s,a)\frac{\partial V^\pi(s')}{\partial\theta} - \frac{\partial}{\partial\theta}\left(\sum_a \bar{p}^\pi(a|s,\theta)Q^\pi(s,a)\right) \qquad (12.34) \\ &= \sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}\left(r(s,a) + \sum_{s'} P(s'|s,a)V^\pi(s')\right) \\ &\quad + \sum_a \bar{p}^\pi(a|s,\theta)\sum_{s'} P(s'|s,a)\frac{\partial V^\pi(s')}{\partial\theta} - \frac{\partial V^\pi(s)}{\partial\theta} \qquad (12.35) \\ &= \sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}\left(Q^\pi(s,a) + F^\pi(\theta)\right) \\ &\quad + \sum_a \bar{p}^\pi(a|s,\theta)\sum_{s'} P(s'|s,a)\frac{\partial V^\pi(s')}{\partial\theta} - \frac{\partial V^\pi(s)}{\partial\theta} \qquad (12.36) \\ &= \sum_a \frac{\partial\bar{p}^\pi(a|s,\theta)}{\partial\theta}Q^\pi(s,a) + \sum_a \bar{p}^\pi(a|s,\theta)\sum_{s'} P(s'|s,a)\frac{\partial V^\pi(s')}{\partial\theta} - \frac{\partial V^\pi(s)}{\partial\theta}. \\ &\qquad (12.37) \end{aligned}$$

Equation (12.33) is from (12.32); (12.34) is the direct expansion of (12.33), where two terms vanish because $r(s,a)$ and $P(s'|s,a)$ do not depend on the policy $\bar{p}^\pi(a|s,\theta)$; (12.33) uses (12.29) for the last term; (12.36) uses (12.30); (12.29) uses the fact $F^\pi(\theta)$ is constant over states and actions, and $\sum_a \bar{p}^\pi(a|s,\theta) = 1$. Finally, note that equation (12.37) is true for all states.

We proceed to write

$$\frac{\partial F^\pi(\theta)}{\partial\theta} = \sum_s d^\pi(s|\theta)\frac{\partial F^\pi(\theta)}{\partial\theta} \tag{12.38}$$

$$= \sum_s d^\pi(s|\theta)\left(\sum_a \frac{\partial \bar{p}^\pi(a|s,\theta)}{\partial\theta}Q^\pi(s,a) + \sum_a \bar{p}^\pi(a|s,\theta)\sum_{s'} P(s'|s,a)\frac{\partial V^\pi(s')}{\partial\theta} - \frac{\partial V^\pi(s)}{\partial\theta}\right). \tag{12.39}$$

Expanding gives us

$$\frac{\partial F^\pi(\theta)}{\partial\theta} = \sum_s d^\pi(s|\theta)\sum_a \frac{\partial \bar{p}^\pi(a|s,\theta)}{\partial\theta}Q^\pi(s,a) + \sum_s d^\pi(s|\theta)\sum_a \bar{p}^\pi(a|s,\theta)\sum_{s'} P(s'|s,a)\frac{\partial V^\pi(s')}{\partial\theta} - \sum_s d^\pi(s|\theta)\frac{\partial V^\pi(s)}{\partial\theta} \tag{12.40}$$

$$= \sum_s d^\pi(s|\theta)\sum_a \frac{\partial \bar{p}^\pi(a|s,\theta)}{\partial\theta}Q^\pi(s,a) + \sum_s d^\pi(s|\theta)\frac{\partial V^\pi(s)}{\partial\theta} - \sum_s d^\pi(s|\theta)\frac{\partial V^\pi(s)}{\partial\theta} \tag{12.41}$$

$$= \sum_s d^\pi(s|\theta)\sum_a \frac{\partial \bar{p}^\pi(a|s,\theta)}{\partial\theta}Q^\pi(s,a). \tag{12.42}$$

Equation (12.38) uses $\sum_s d^\pi(s|\theta) = 1$; (12.39) uses the fact (12.37) holds for all s; (12.40) simply expands (12.39); (12.41) uses the property that since $d^\pi(s)$ is the stationary distribution, then $\sum_s d^\pi(s|\theta)P(s'|s,a) = d^\pi(s'|\theta)$ (after substituting this result, then just change the index from s' to s). Equation (12.42) is the policy gradient theorem we first presented in equation (12.28) (and equation (12.23) in the body of the chapter).

12.11 BIBLIOGRAPHIC NOTES

Section 12.1 - The idea of modeling stochastic search algorithms (whether it is derivative-based or derivative-free) was first done (to our knowledge) in Powell (2019).

Section 12.2 - The concept that the search over policy function approximations is over the same classes of functions as would take place in any machine learning exercise seems to be new.

Section 12.5 - 12.6 - The concept of optimizing parameterized policies, which has been described as "policy search," has been actively studied since the 1990s. It is the reason we named this class the "policy search" class. Our presentation of policy search using numerical derivatives, or the methods of derivative-free stochastic search (both of which depend purely on simulating a policy as a black box) is well known in the reinforcement learning community (see Sigaud & Stulp (2019) for a recent and thorough review). We note that this review is specifically for continuous actions, but a parameterized policy can be used for discrete actions, and optimized using the same methods.

Section 12.7 - Both sections 12.5 and 12.6 depend purely on function approximations to perform stochastic search. There is a large class of dynamic programs where the future state S_{t+1} is a continuous function of S_t and x_t. These include, for example, resource allocation problems for managing money, water, blood, inventory and electric power, where inventories of resources R_t are being allocated through decisions x_t to produce updated inventories R_{t+1}. The core equations ((12.15)-(12.16)) are little more than elaborate exercises in the chain rule that have long been used in control problems and neural networks (where it is referred to as backpropagation). See any standard treatment of discrete time optimal control (such as Kirk (2012), Stengel (1986), Sontag (1998), and Lewis & Vrabie (2012)). Our adaptation for parameterized policies was derived here from first principles, but the approach is straightforward.

Section 12.8 - Policy gradient methods have received considerable attention in the reinforcement learning community for problems with discrete states and actions. This section describes a method for computing policy gradients for discrete dynamic programs using a concept that has become known as the "policy gradient method," introduced in Sutton et al. (2000), and described nicely in the second edition of their book Sutton & Barto (2018)[Chapter 13].

EXERCISES

Review questions

12.1 Policy search is a sequential decision problem. Write out the elements of a policy search algorithm using our modeling framework.

12.2 What is an "affine policy"? Write out a general form for an affine policy. Imagine that we are managing an inventory storage problem where the state $S_t = (R_t, p_t)$ depends on the inventory we are holding R_t and the price we can sell the inventory p_t. Let x_t be the amount of our inventory to sell at time t. If we write our policy as

$$X^\pi(S_t|\theta) = \theta_0 + \theta_1 R_t + \theta_2 R_T^2 + \theta_3 p_t + \theta_4 p_t^2 + \theta_5 R_t p_t,$$

is this an affine policy? Why?

12.3 To do policy search it is critical that you know how to write out the objective function that you use to evaluate the performance of the policy.

a) What is the objective function if you are tuning your policy in a simulator? Carefully explain each source of uncertainty (or randomness).

b) What is the objective function if you are tuning your policy in the field?

Modeling questions

12.4 Assume we are going to search for policies for a simple inventory problem where the inventory R_t evolves according to

$$R_{t+1} = \max\{0, R_t + x_{t-\tau} - \hat{D}_{t+1}\}$$

where the random demand $\hat{D}_{t+1}$ follows a discrete uniform distribution from 1 to 10 with equal probability. An order x_t arrives at time $t + \tau$, which we will specify below. Assume $R_0 = 10$, and use the contribution function

$$C(S_t, x_t) = p_t \min\{R_t + x_{t-\tau}, \hat{D}_{t+1}\} - 15x_t.$$

where the price p_t is drawn from a uniform distribution between 16 and 25 with equal probability.

We are going to place our orders according to the order-up-to policy

$$X^{Inv}(S_t|\theta) = \begin{cases} \theta^{max} & \text{if } R_t < \theta^{min}, \\ 0 & \text{otherwise.} \end{cases}$$

We want to choose θ to solve

$$\max_\theta F(\theta) = \mathbb{E}_W \left\{ \sum_{t=0}^{100} C(S_t, x_t)|S_0 \right\} \tag{12.43}$$

where $W = (W_1, \ldots, W_{100})$ is the vector of realizations of prices and demands.

a) What is the state variable S_t at time t?

b) What is the decision variable at time t? Does it matter that the decision at time t does not have any impact on the system until τ time periods later?

c) What are the elements of the exogenous information variable W_t?

d) What is the transition function? Recall that you need an equation for each element of S_t.

e) The objective function in (12.43) maximizes the cumulative reward, but we are optimizing the policy in an offline simulator, which means we want to optimize the final reward, not the cumulative reward. Make the argument that (12.43) is still the correct objective. [Hint: look at table 9.3 and identify which of the four classes of objective functions that equation (12.43) falls in.

12.5 Assume you are tuning the parameters θ of a policy $X^\pi(S^n|\theta)$ to find $x^{\pi,N}$ in N iterations to maximize $\mathbb{E}F(\theta, W)$ using a gradient-based search algorithm. This means you have access to the gradient $\nabla_\theta F(\theta, W)$.

a) Write out the five elements of a sequential decision problem (state variables, decision variables, exogenous information, transition function, objective function).

b) What is the exogenous information for this problem?

c) Recalling the menu of stepsize policies that we can draw from (see section 6.2.3), what is meant by searching over policies?

Computational exercises

The next two exercises will optimize the policy modeled in exercise 12.4 using derivative-based methods.

12.6 Implement the basic stochastic gradient algorithm based on finite differences (see section 5.4.3). Use the harmonic stepsize

$$\alpha_n = \frac{\theta^{step}}{\theta^{step} + n - 1}, \tag{12.44}$$

which means we also have to tune θ^{step}. Start by assuming $\tau = 1$.

a) Run the algorithm 100 iterations for $\theta^{step} = 1, 5, 10, 20$ (just one sample path each) and report which one works best, and the value of θ that the algorithm returns.

b) Run the algorithm 100 iterations for $\theta^{step} = 10$, and plot the objective function over the iterations for each value of θ^{step}. Repeat this 20 times to demonstrate the range of sample paths the algorithm can take. How many samples do you think you would need to reliably estimate which value of θ^{step} works best?

c) Using the best value of θ^{step}, find the best value of θ when $\tau = 1, 5, 10$.

12.7 You are going to optimize the policy modeled in exercise 12.4 using the SPSA algorithm. Assume $\tau = 1$.

a) Implement the simultaneous perturbation stochastic approximation (SPSA) algorithm (see section 5.4.4). Use the harmonic stepsize (see equation (12.44)), which means we also have to tune the stepsize parameter θ^{step}. Use a mini-batch of 1 for computing the gradient. Run the algorithm 100 iterations for $\theta^{step} = 1, 5, 10, 20$, where you run 20 repetitions for each value of θ^{step} and average the results. Report which one works best.

b) Run the algorithm using $\theta^{step} = 10$ and mini-batch sizes of 1, 5, 10 and 20, and compare the performance over 100 iterations.

The next two exercises will optimize the policy modeled in exercise 12.4 using derivative-free methods. For each method, enumerate a set Θ of possible values for the two-dimensional ordering policy θ by varying θ^{min} over the values $2, 4, \ldots, 10$, and varying θ^{max} over the range $6, 8, \ldots, 20$ while excluding any combination where $\theta^{min} \geq \theta^{max}$. Let Θ be the set of allowable combinations of θ. Assume $\tau = 1$ throughout.

12.8 Lookup table with correlated beliefs: After building the set Θ, do the following:

a) Initialize your belief by running five simulations for five different values of $\theta \in \Theta$. Average these results and set $\bar{\mu}^0_\theta$ to this average for all $\theta \in \Theta$. Compute the variance $\sigma^{2,0}$ of these five observations, and initialize the precision of the belief at $\beta^0_\theta = 1/\sigma^{2,0}$ for all $\theta \in \Theta$. Let

$$\bar{F}^0 = \max_{\theta \in \Theta} \bar{\mu}^0_\theta$$

and report $\bar{F}^0$ (of course, $\bar{\mu}^0_\theta$ is the same for all θ, so you can just pick any θ.

b) Assume that the estimates $\bar{\mu}^0_\theta$ are related according to

$$Cov(\bar{\mu}^0_\theta, \bar{\mu}^0_{\theta'}) = \sigma^0 e^{-\rho|\theta-\theta'|}.$$

Compute $Cov(\bar{\mu}^0_\theta, \bar{\mu}^0_{\theta'})$ by running 10 simulations for each combination of $\theta = (4,6), (4,8), (4,10), (4,12), (4,14)$. Now find the value of ρ that produces the best fit of $Cov(\bar{\mu}^0_\theta, \bar{\mu}^0_{\theta'})$ using these five datapoints. Now, fill out the matrix Σ^0 where

$$\Sigma^0_{\theta,\theta'} = \sigma^0 e^{-\rho|\theta-\theta'|}$$

for all $theta, \theta' \in \Theta$, and using the value of ρ that you determined above.

c) White out the equations for updating $\bar{\mu}^n_\theta$ using correlated beliefs (see section 3.4.2).

d) Now use the interval estimation policy

$$\Theta^\pi(S^n|\theta^{IE}) = \arg\max_{\theta\in\Theta} \left(\bar{\mu}^n_\theta + \theta^{IE}\bar{\sigma}^n_\theta\right)$$

where $\bar{\sigma}^n_\theta = \Sigma^n_{\theta,\theta}$. Of course, we have now introduced another tunable parameter θ^{IE} in our policy to tune the parameters in our ordering policy $X^\pi(S_t|\theta)$. Get used to it - this happens a lot. Using $\theta^{IE} = 2$, execute the policy $\Theta^\pi(S^n|\theta^{IE})$ for 100 iterations, and report the simulated performance of the objective (12.43) as you progress. On a two-dimensional graph showing all the combinations of Θ, report how many times you sample each of the combinations.

e) Repeat your search for $\theta^{IE} = 0, .5, 1, 2, 3$. Prepare a graph showing the performance of each value of θ^{IE}.

12.9 Response surface methods: In this exercise we are going to optimize θ by creating a statistical model of the function $F(\theta)$. After building the set Θ, do the following:

a) Randomly pick 10 elements of Θ, simulate the policy and then use these five points to fit the linear model

$$\bar{F}^0(\theta) = \rho^0_0 + \rho^0_1\theta^{min} + \rho^0_2(\theta^{min})^2 + \rho^0_3\theta^{max} + \rho^0_4(\theta^{max})^2 + \rho^0_5\theta^{min}\theta^{max}.$$

Use the methods in section 3.7 to fit this model.

b) At iteration n, find

$$\theta^n = \arg\max_\theta \bar{F}^n(\theta).$$

We then run the policy using $\theta = \theta^n$ to obtain $\hat{F}^{n+1}(\theta^n)$. Add $(\theta^n, \hat{F}^{n+1})$ to the data used to fit the approximation to obtain the updated approximation $\bar{F}^{n+1}(\theta)$, and repeat. Run this for 20 iterations, and repeat 10 times. Report the average and the spread.

c) Repeat the algorithm, but this time replace the policy for computing θ^n with

$$\begin{aligned} \hat{\theta}^n &= \arg\max_\theta \bar{F}^n(\theta), \\ \theta^n &= \hat{\theta}^n + \delta^n, \end{aligned}$$

where

$$\theta^n = \begin{pmatrix} \theta_1^n \\ \theta_2^n \end{pmatrix}$$

and

$$\delta^n = \begin{pmatrix} \delta_1^n \\ \delta_2^n \end{pmatrix}.$$

The vector δ is a perturbation of magnitude r where

$$\begin{aligned} \delta_1^n + \delta_2^n &= 0, \\ \sqrt{(\delta_1^n)^2 + (\delta_2^n)^2} &= r. \end{aligned}$$

These equations imply that

$$\delta_1^n = -\delta_2^n = r/\sqrt{2},$$

or

$$\delta_2^n = -\delta_1^n = r/\sqrt{2}.$$

This algorithm exploits the property that it is better to sample points that are displaced from the optimum. As is often the case, this simple policy involves another tunable parameter, the perturbation radius r. Start with $r = 4$. Run this algorithm for 20 iterations, and then do a final evaluation with $\delta^n = 0$ to see the performance based on the value of θ that is best given our approximate function. Repeat for $r = 0, 2, 6, 8$ and report which performs the best.

Problem solving questions

12.10 Imagine we have an asset selling problem where the policy is given by

$$X^\pi(S_t|\theta) = \begin{cases} 1 = \text{"sell"} & \text{if } p_t \geq \theta, \\ 0 = \text{"hold"} & \text{if } p_t < \theta. \end{cases} \tag{12.45}$$

a) Is this an affine policy? Why or why not?

b) Now imagine that we do not know that this might be the right structure of the policy, and you want to design an affine policy. What might this look like? Do you think your affine policy might work well?

c) What is meant by a monotone policy? Is the policy in (12.45) monotone?

d) Imagine that you believe that your policy is monotone in price, but other than this, you do not know the shape of the function. Suggest an approximation strategy you might propose that allows you to require that the function be monotone in p_t, and sketch a method for estimating this function.

Sequential decision analytics and modeling

These exercises are drawn from the online book *Sequential Decision Analytics and Modeling* available at `http://tinyurl.com/sdaexamplesprint`.

12.11 Review the asset selling problem in chapter 2 up through 2.4. Three policies are suggested, but in this exercise we are going to focus on the tracking policy, which involves tuning a single parameter. We will be using the Python module "AssetSelling" at `http://tinyurl.com/sdagithub`, which contains the code to simulate the tracking policy. This exercise will focus on derivative-free methods for performing the parameter search.

a) Run 20 simulations of the pricing model and determine from these runs the largest and smallest prices. Divide this range into 20 segments. Now implement an interval estimation policy

$$X^{IE}(S^n|\theta^{IE}) = \arg\max_{x\in\mathcal{X}}(\bar{\mu}^n_x + \theta^{IE}\bar{\sigma}^n_x). \tag{12.46}$$

where $\mathcal{X}$ is the 20 possible values of the tracking parameter, $\bar{\mu}^n_x$ is our estimate of the performance of the tracking parameter when it takes value $x \in \mathcal{X}$. For this exercise, set $\theta^{IE} = 2$ (although this is a parameter that would also need tuning). Show your estimates $\bar{\mu}^N_x$ for each value of x when your experimentation budget is $N = 20$, and then when $N = 100$.

b) This time, we are going to create a quadratic belief model where

$$\bar{F}^n(x) = \bar{\theta}^n_0 + \bar{\theta}^n_1 x + \bar{\theta}^n_2 x^2,$$

where x is still the value of the tracking parameter. Test three policies for choosing x^n (these are all presented in chapter 7):

i) A greedy policy where $x^n = \arg\max_x \bar{F}^n(x)$.

ii) An excitation policy $x^n = \arg\max_x \bar{F}^n(x) + \varepsilon^{n+1}$ where $\varepsilon^{n+1} \sim N(0, \sigma^2)$, where σ^2 is the noise in the exploration process, which is a parameter that has to be tuned.

iii) A parameterized knowledge gradient policy where $x^n = \arg\max_x \bar{F}^n(x) + Z$ where $Z = \pm r$, where r is a parameter that needs to be tuned.

Simulate each policy for 100 iterations, and compare the performance of each policy.

12.12 Review the asset selling problem in chapter 2 up through 2.4. Three policies are suggested, but in this exercise we are going to focus on the tracking policy, which involves tuning a single parameter. We will be using the Python module "AssetSelling" at `http://tinyurl.com/sdagithub`, which contains the code to simulate the tracking policy. This exercise will focus on derivative-based methods for performing the parameter search.

a) Produce an estimate of a stochastic gradient by running a simulation where the tracking parameter is set at x, and then again at $x + \delta$ where $\delta = 1$. Use a harmonic stepsize

$$\alpha_n = \frac{\theta^{step}}{\theta^{step} + n - 1},$$

where we leave the tuning of θ^{step} to you. Run this algorithm for 100 iterations, and find θ^{step} to produce the best solution $x^{\pi,N}$.

b) Repeat (a), but this time repeat the simulation using a mini-batch m for $m = 1, 5, 10, 20$. Note that the best value of θ^{step} is likely to depend on m. Run the stochastic gradient algorithm for each value of m for $N = 100$ iterations, and compare the results.

Diary problem

The diary problem is a single problem you chose (see chapter 1 for guidelines). Answer the following for your diary problem.

12.13 Pick a particular decision in your diary problem (if there is more than one) and try to design a policy function approximation to make the decision. This will typically involve a tunable parameter (if you state a PFA without a tunable parameter, try to introduce one). Then, show how to tune the policy in the following settings:

a) Offline, in a simulator. Remember that you will have both the tuning of the parameter(s), followed by testing. Write out the objective function using final reward formulation (if you do not remember this by now, flip back to equation (7.2)). Explicitly describe any uncertainties in your initial state S^0, along with the exogenous information W_t and the testing random variable $\widehat{W}_t$.

b) Online, in the field. This means optimizing using the cumulative reward formulation (see equation (7.3)). Again - clearly define all the random variables.

Bibliography

Kirk, D. E. (2012), *Optimal Control Theory: An introduction*, Dover, New York.

Lewis, F. L. & Vrabie, D. (2012), *Design Optimal Adaptive Controllers*, 3 edn, John Wiley & Sons, Hoboken, NJ.

Powell, W. B. (2019), 'A unified framework for stochastic optimization', *European Journal of Operational Research* **275**(3), 795–821.

Sigaud, O. & Stulp, F. (2019), 'Policy search in continuous action domains: An overview', *Neural Networks* **113**, 28–40.

Sontag, E. (1998), 'Mathematical Control Theory, 2nd ed.', *Springer* pp. 1–544.

Stengel, R. F. (1986), *Stochastic optimal control: theory and application*, John Wiley & Sons, Hoboken, NJ.

Sutton, R. S. & Barto, A. G. (2018), *Reinforcement Learning: An Introduction*, 2nd edn, MIT Press, Cambridge, MA.

Sutton, R. S., McAllester, D., Singh, S. P. & Mansour, Y. (2000), 'Policy gradient methods for reinforcement learning with function approximation', *Advances in neural information processing systems* **12**(22), 1057–1063.