CHAPTER 14

OPTIMIZING OVER CONTINUOUS ALTERNATIVES*

There are many applications where we have to choose the best value of a continuous, multidimensional vector x. One example arises in the electricity sector, where utilities have to deal with the high variability of electricity spot prices. One strategy is to use a battery to store energy when prices are low, and release energy when prices are high. The idea is illustrated in Figure 14.2, where we store energy when the price goes below x^{store}, and we release energy when the price goes above $x^{withdraw}$. We now face the problem of deciding how to choose $x = (x^{store}, x^{withdraw})$. Let $\mu(x)$ be the expected profits per day that we obtain from using a policy fixed by x, where we assume we can only obtain noisy estimates of $\mu(x)$. Our challenge, as with elsewhere in this volume, is finding the best value of x as quickly as possible.

While we do not know the true function $\mu(x)$, we imagine it might look like the surface shown in Figure 14.2. We are going to assume it is smooth, but not necessarily concave (or convex, if we were solving a minimization problem). If x had only one or two dimensions, we could discretize it and use the techniques presented in the earlier chapters, although even two dimensions starts to become problematic if, for example, we would like to discretize each dimension into 100 intervals. If we have three or more dimensions, discretization quickly becomes cumbersome.

There are a number of problems which require tuning continuous but relatively low-dimensional parameter vectors. Some examples include:

Optimal Learning. By Warren B. Powell and Ilya O. Ryzhov

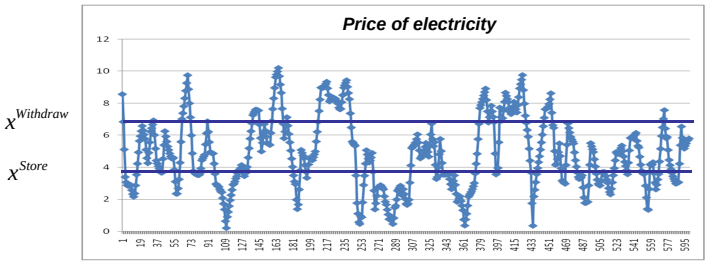

Figure 14.1 A battery charging problem, where we storage energy when the price goes below x^{store}, and withdraw it when the price goes above $x^{withdraw}$.

- Finding the best parameters that govern a business simulator - We might have a model that simulates the use of ambulances, or a manufacturing system that depends on the speed of different machines. A model of freight transportation may require assumptions on the time that drivers have to rest or the maximum time that a customer may be served early or late.
- Tuning the concentrations of different chemicals - We may be trying to maximize the yield of a chemical process that depends on the concentrations of different additives (we may also have to vary the temperature of different steps of the process).
- Design of devices - The design of an aerosol spray can requires tuning the diameter of the feed-in tube, the spacing between the feed-in tube and the aerosol spray tube, the diameter of the aerosol spray tube, and the pressure in the can.

In some cases, these parameters have to be tuned using field experiments or physical lab experiments, while in other cases they can be done using computer simulations. Computer simulations may run in a few minutes, but in some instances they can require hours or even days to complete a single run. Laboratory and field experiments can be even more time consuming, and are generally much more expensive. For this reason, we want to choose our experiments carefully so that we learn as much as possible from each function evaluation.

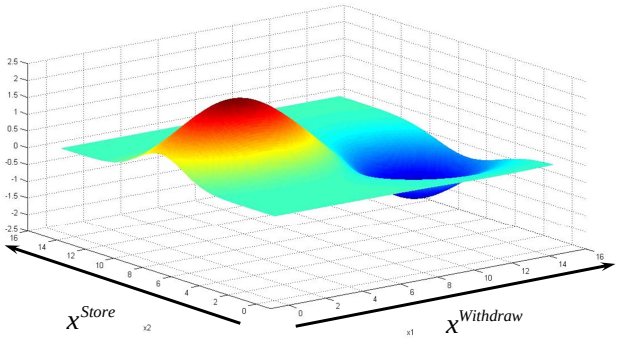

Figure 14.2 Illustration of the surface generated by varying x^{store} and $x^{withdraw}$.

We are going to continue to use our notation that $\mu(x)$ is the true value of the function at x. The only difference is that x is now a continuous vector. We are going to make a series of experiments $x^0, x^1, \ldots, x^n$, from which we make noisy observations of our function of the form

$$\hat{y}^{n+1} = \mu(x^n) + \varepsilon^{n+1}.$$

We assume that ε is a random variable that is normally distributed with mean 0 and variance λ. We wish to design a sequential search policy π that guides the selection of x^i, giving us an estimate of the value of our function which we represent as

$$F^\pi = \mu(x^N),$$

where $x^N = \arg\max_{x \in \mathcal{X}} \bar{\mu}^N(x)$, and $\bar{\mu}^N(x)$ is our estimate of $\mu(x)$ after N experiments obtained by following policy π. Our challenge is to find the policy π^* that solves

$$F^* = \max_\pi F^\pi.$$

This chapter addresses the problem of tuning these multidimensional parameter vectors for problems where we do not have access to derivatives. As always, we assume that observing $\mu(x)$ is time consuming and/or expensive, and noisy. Also, while we focus on multidimensional parameter vectors, our experimental work as of this writing is for vectors where the number of dimensions is less than 10.

14.1 THE BELIEF MODEL

In previous chapters, we have considered different ways of representing our belief in the function. We started with a lookup table, where we assumed that x was discrete, and there was an estimate θ_x^n for each x (see, for example, Chapters 3, 4 and 5). We have also considered models where we used linear regression to approximate $\mu(x)$ (as in Chapter 7). In this chapter, we need a different belief model since we have almost no idea about the structure of our function, but we do know it is continuous. For this reason, we are going to use an approach known as *Gaussian process regression* which is a class of nonparametric models which creates an approximation based on all prior observations.

We begin with a prior $\theta^0(x)$, which is now in the form of a continuous function. We are going to use our policy to generate a sequence of experiments $x^0, \ldots, x^{n-1}$, and we are going to use these points to approximate our function. Thus, after these n experiments $\hat{y}^1, \ldots, \hat{y}^n$, we will have a belief $\theta^n(x)$ at each of these points. We also need to capture the covariance in our beliefs about the function at each of these points. We define the $n \times n$ matrix Σ^n, where $\Sigma^n_{ij} = Cov^n(\mu(x^i), \mu(x^j))$ is the covariance in our belief about μ at x^i and x^j after n iterations. Note that each time we run a new experiment, θ^n grows by one element, while the matrix Σ^n adds a row and a column.

We are going to take advantage of the continuous structure of the problem and assume that we can write the covariance in our belief between any two points using

the function

$$Cov^0(\mu(x^i), \mu(x^j)) = \beta \exp(-\sum_{m=1}^{p} \alpha_i (x_m^i - x_m^j)^2), \quad \alpha > 0, \beta > 0. \tag{14.1}$$

It is common to refer to the p-dimensional vector α as the activity of μ, while the scalar parameter β captures the uncertainty of our belief about μ. The activity parameter α controls the degree of smoothness, where the function becomes smoother as α becomes smaller. For larger values of α, the covariance between two points shrinks with α, which means that we learn less about $\mu(x^i)$ and $\mu(x^j)$ as x^i and x^j become farther apart.

14.1.1 Updating equations

We have to learn how to update our beliefs about $\mu(x^i),\ i = 1, \ldots, n$ as we make more observations. We are going to use the property that the vector θ^n follows a multivariate normal distribution, where $\mathbb{E}^n \mu(x^i) = \theta^n(x^i)$ and the covariance matrix is given by Σ^n. Throughout this chapter $\mathbb{E}^n$ refers to the conditional information given the history of observations $\hat{y}^1, \ldots, \hat{y}^n$.

We start by calculating a vector $\tilde{y}^n$ which we call *experimental residuals* using

$$\tilde{y}^n = \begin{bmatrix} \hat{y}^1 \\ \vdots \\ \hat{y}^n \end{bmatrix} - \begin{bmatrix} \theta^0(x^0) \\ \vdots \\ \theta^0(x^{n-1}) \end{bmatrix}. \tag{14.2}$$

This is the difference between our observations $\hat{y}^i,\ i = 1, \ldots, n$, and our original belief $\theta^0(x^i),\ i = 1, \ldots, n$. We also define the *residual covariance* S^n which is updated using

$$S^n = \Sigma^0 + Diagonal([\lambda(x^0), ..., \lambda(x^{n-1})]). \tag{14.3}$$

Expanded into matrices, this is the same as

$$\begin{bmatrix} S_{11}^n & \cdots & S_{1i}^n & \cdots & S_{1n}^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ S_{i1}^n & \cdots & S_{ii}^n & \cdots & S_{in}^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ S_{n1}^n & \cdots & S_{ni}^n & \cdots & S_{nn}^n \end{bmatrix} = \begin{bmatrix} \Sigma_{11}^0 & \cdots & \Sigma_{1i}^0 & \cdots & \Sigma_{1n}^0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \Sigma_{i1}^0 & \cdots & \Sigma_{ii}^0 & \cdots & \Sigma_{in}^0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \Sigma_{n1}^0 & \cdots & \Sigma_{ni}^0 & \cdots & \Sigma_{nn}^0 \end{bmatrix} + \begin{bmatrix} \lambda(x^0) & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & 0 & \cdots & \vdots \\ \vdots & 0 & \lambda(x^i) & 0 & \vdots \\ \vdots & \vdots & 0 & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & \lambda(x^{n-1}) \end{bmatrix}$$

Finally, we are going to compute the *gain matrix*, denoted G^n using

$$G^n = \Sigma^0 [S^n]^{-1}. \tag{14.4}$$

As we see shortly below, the gain matrix is used to weigh new information, which plays the same role as a stepsize.

The updating equations for the mean vector θ^n and the covariance matrix S^n are now given by

$$\begin{bmatrix} \theta^n(x^0) \\ \vdots \\ \theta^n(x^{n-1}) \end{bmatrix} = \begin{bmatrix} \theta^0(x^0) \\ \vdots \\ \theta^0(x^{n-1}) \end{bmatrix} + G^n\tilde{y}^n, \tag{14.5}$$

$$\Sigma^n = (I_n - G^n)\Sigma^0. \tag{14.6}$$

where I_n is the $n \times n$ identity matrix.

Equations (14.5) and (14.6) update our beliefs around the points $x^0, \ldots, x^n$ that we have already measured, but we also need a belief for an arbitrary point x, since we need to search over all possible points to determine the next point that we *might* wish to evaluate. We begin by defining

$$\bar{\Sigma}^0 = \Sigma^0([x^0, ..., x^{n-1}, x]),$$
$$\bar{\Sigma}^n = \Sigma^n([x^0, ..., x^{n-1}, x]).$$

Let $\vec{0}$ be a column vector of zeroes. Our new gain matrix is given by

$$\bar{K}^n = \bar{\Sigma}^0 \begin{bmatrix} I_n \\ - \\ \vec{0}^T \end{bmatrix} [S^n]^{-1}. \tag{14.7}$$

We can now find θ^0 and $\bar{\Sigma}^0$ for an arbitrary $(n+1)$st point x using

$$\begin{bmatrix} \theta^n(x^0) \\ \vdots \\ \theta^n(x^{n-1}) \\ \theta^n(x) \end{bmatrix} = \begin{bmatrix} \theta^0(x^0) \\ \vdots \\ \theta^0(x^{n-1}) \\ \theta^0(x) \end{bmatrix} + \bar{K}^n\tilde{y}^n, \tag{14.8}$$

$$\bar{\Sigma}^n = (I_{n+1} - \bar{K}^n \begin{bmatrix} I_n & | & \vec{0} \end{bmatrix})\bar{\Sigma}^0. \tag{14.9}$$

If we want the distribution of $\mu(x)$ given our n observations at some arbitrary decision x, we can use (14.8) and (14.9) to obtain

$$\theta^n(x) = \theta^0(x) + \begin{bmatrix} \Sigma^0(x^0, x) & , \cdots , & \Sigma^0(x^{n-1}, x) \end{bmatrix} [S^n]^{-1}\tilde{y}^n, \tag{14.10}$$

$$\Sigma^n(x, x) = \Sigma^0(x, x) - \begin{bmatrix} \Sigma^0(x^0, x) & , \cdots , & \Sigma^0(x^{n-1}, x) \end{bmatrix} [S^n]^{-1} \begin{bmatrix} \Sigma^0(x^0, x) \\ \vdots \\ \Sigma^0(x^{n-1}, x) \end{bmatrix}. \tag{14.11}$$

Equation (14.10) is known as *Gaussian process regression* (GPR) in some communities, and regression kriging in others.

We have to choose the point x^n to measure before we have observed the outcome $\hat{y}^{n+1}$. The updated regression function, given the observations $\hat{y}^1, \ldots, \hat{y}^n$, is nor-

mally distributed with distribution

$$\begin{bmatrix} \theta^{n+1}(x^0) \\ \vdots \\ \theta^{n+1}(x^{n-1}) \\ \theta^{n+1}(x^n) \end{bmatrix} = \begin{bmatrix} \theta^{n}(x^0) \\ \vdots \\ \theta^{n}(x^{n-1}) \\ \theta^{n}(x^n) \end{bmatrix} + \tilde{\sigma}(\bar{\Sigma}^n, x^n)Z^{n+1}, \tag{14.12}$$

where $Z^{n+1} = \left(\hat{y}^{n+1} - \theta^n(x^n)\right) / \sqrt{\lambda(x^n) + \Sigma^n(x^n, x^n)}$, with

$$\tilde{\sigma}(\Sigma, x) = \frac{\Sigma e_x}{\sqrt{\lambda(x) + e_x^T \Sigma e_x}}. \tag{14.13}$$

Here e_x is a column vector of zeroes with a 1 at the row corresponding to decision x. It can be shown that Z^{n+1} is a standard normal random variate (mean 0, variance 1) because $\text{Var}(\hat{y}^{n+1} - \theta^n(x^n)|\mathcal{F}^n) = \lambda(x^n) + \Sigma^n(x^n, x^n)$.

14.1.2 Parameter estimation

Our model is characterized by the p-dimensional vector α, the scalar β and the noise λ, as well as our prior $\theta^0(x)$. There may be problems where we feel we can assume these are known, but in practice we are going to have to estimate them from data. As is so often the case with statistical estimation, there is more than one way to solve this problem, but a popular method that we have found works quite well is maximum likelihood estimation (MLE).

MLE starts by creating a *likelihood function* which is the product of the density (using the normal distribution) of all the prior observations given the unknown parameters. We take the log of this product (giving us the log-likelihood), and then find the values of the parameters to maximize the resulting sum.

We form the likelihood function using

$$\begin{aligned} &L_{\hat{y}}\left(\alpha, \beta, \lambda(x^0), ..., \lambda(x^{n-1}), \theta^0(x^0), ..., \theta^0(x^{n-1})\right) \\ &\quad = \quad (2\pi)^{-n/2} \left|S^n\right|^{-1/2} \exp\left(-\frac{1}{2}\begin{bmatrix} \hat{y}^1 - \theta^0(x^0) \\ \vdots \\ \hat{y}^n - \theta^0(x^{n-1}) \end{bmatrix}^T (S^n)^{-1} \begin{bmatrix} \hat{y}^1 - \theta^0(x^0) \\ \vdots \\ \hat{y}^n - \theta^0(x^{n-1}) \end{bmatrix}\right). \end{aligned}$$

Now, if we assume that the variance of the observation noise, $\lambda(\cdot)$, is a constant λ and $\theta^0(\cdot)$ is a constant θ^0, we can write the likelihood function as

$$L_{\hat{y}}(\alpha, \beta, \lambda, \theta^0) = (2\pi)^{-n/2}|\Sigma^0 + \lambda I_n|^{-1/2} \exp\left(-\frac{1}{2}(\hat{y} - \theta^0\mathbf{1})^T(\Sigma^0 + \lambda I_n)^{-1}(\hat{y} - \theta^0\mathbf{1})\right),$$

where $\mathbf{1}$ is a $n \times 1$ column vector of ones and $\hat{y} = \begin{bmatrix} \hat{y}^1 & \cdots & \hat{y}^n \end{bmatrix}^T$. Note that in this case we are estimating $p + 3$ parameters using n observations. We can write the log-likelihood function as:

$$\ell_{\hat{y}}(\alpha, \beta, \lambda, \theta^0) = -\frac{n}{2}\ln(2\pi) - \frac{1}{2}\ln(|\Sigma^0 + \lambda I_n|) - \frac{1}{2}(\hat{y} - \theta^0\mathbf{1})^T(\Sigma^0 + \lambda I_n)^{-1}(\hat{y} - \theta^0\mathbf{1}). \tag{14.14}$$

We can approximately maximize the likelihood over the parameters by using the function `patternsearch` in MATLAB started at multiple points chosen by a Latin hypercube sampling (LHS) design using the command `lhsdesign`. Also, in the above log-likelihood we can easily solve for θ^0 in terms of α, β, and λ, giving us the estimate

$$\hat{\theta^0} = \frac{\hat{y}^T(\Sigma^0 + \lambda I_n)^{-1}\mathbf{1}}{\mathbf{1}^T(\Sigma^0 + \lambda I_n)^{-1}\mathbf{1}}.$$

Finally, to prevent numerical issues, if $|\Sigma^0 + \lambda I_n|$ is very small in (14.14), a useful equivalent expression to $\ln(|\Sigma^0 + \lambda I_n|)$ is $trace(logm(\Sigma^0 + \lambda I_n))$ where $logm$ is the matrix logarithm.

We typically will have to perform an initial sample of experiments x to obtain a starting estimate of α, β and λ. If d is the dimensionality of our parameter vector (the sum of the dimensions of α, β and λ), a common rule of thumb is to perform a Latin hypercube design with $2d$ plus 2 points. In MATLAB, we can use `lhsdesign` to choose the points $x^1, \ldots, x^{2d+2}$. However, as the number of dimensions grows, the value of a LHS design diminishes, and it is better to simply choose the starting experiments at random.

14.2 SEQUENTIAL KRIGING OPTIMIZATION

Sequential kriging uses a form of meta-modeling that assumes the surface is represented by a linear model, a bias model and a noise term, which we can write using

$$\mu(x) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(x) + Z(x) + \varepsilon.$$

Here, $\mathcal{F}$ is a set of features, $\phi_f(x)$ is a basis function (also known as an independent variable) corresponding to feature f, and $Z(x)$ is a term that represents the systematic bias due to the limitations of the basis functions.

The kriging meta-model is based on the idea that the bias function $Z(x)$ is a realization of a stationary Gaussian stochastic process. If x is a d-dimensional vector, we assume that the covariance between $Z(x)$ and $Z(x')$ can be written

$$Cov(Z(x), Z(x')) = \beta \exp\left[-\sum_{i=1}^{d} \alpha_i (x_i - x_i')^2\right],$$

where, as before, β is the variance of the stochastic process while α_i scales the covariance function for each dimension.

Samples of one-dimensional Gaussian surfaces are illustrated in Figure 14.3, which shows curves generated from different values of the activity variable α. Smaller values of α produces slower, undulating surfaces, while larger values of α (which reduces the correlations between nearby points) produces surfaces that undulate with higher frequencies. If this is the bias, it means that the true surface is likely to be reasonably smooth. It is important to realize that $Z(x)$ is not a random noise

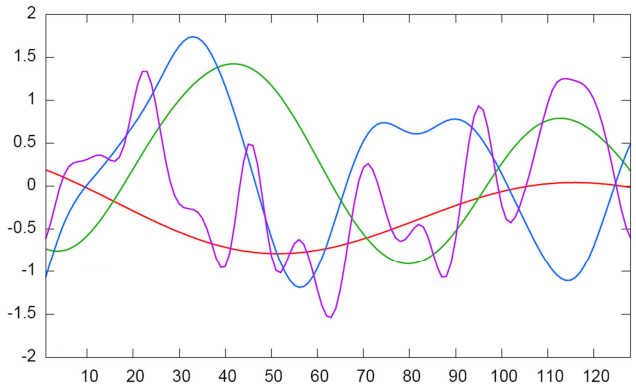

Figure 14.3 Illustration of sample realizations of one-dimensional Gaussian surfaces.

term with zero mean (which would imply that the approximation is unbiased). A sample realization of $Z(x)$ is the bias for a specific function $\mu(x)$ and a specific approximation represented by the set of basis functions ϕ and regression vector θ.

The best linear predictor $\bar{Y}^n(x)$ of $\mu(x)$ after n experiments is given by

$$\begin{aligned}\bar{Y}^n(x) &= \sum_{f\in\mathcal{F}} \theta_f^n \phi_f(x) + \\ &\quad \sum_{i=1}^{n} Cov(Z(x_i), Z(x)) \sum_{j=1}^{n} Cov(Z(x_i), Z(x_j))(\hat{y}_i - \sum_{f\in\mathcal{F}} \theta_f^n \phi_f(x)),\end{aligned}$$

where θ^n is the least squares estimator of the regression parameters after n observations.

The idea of sequential kriging is to use the expected improvement $I(x)$ resulting from measuring the function at x. We start by illustrating this for a function $f(x)$ that can be observed deterministically. Let x^n be the current best solution given our current approximation. After n experiments, let $F(x)$ represent our belief about $f(x)$ which is normally distributed with mean $\bar{Y}^n(x)$ and variance $\sigma^{2,n}(x)$. The expected improvement, given the history of n observations, can be written as

$$\mathbb{E}^n[I(x)] = \mathbb{E}^n \max\big(f(x^*) - F(x), 0\big). \tag{14.15}$$

We need to build on this idea to handle the issue of noisy experiments. Sequential kriging optimization (SKO) uses the following expression for the expected improvement

$$\mathbb{E}^n I(x) = \mathbb{E}^n\left[\max\left(\bar{Y}^n(x^{**}) - \mu(x), 0\right)\right]\left(1 - \frac{\sigma_\varepsilon}{\sqrt{\sigma^{2,n}(x) + \sigma_\varepsilon^2}}\right). \tag{14.16}$$

The first term on the right hand side of (14.16) mimics the expected improvement term used if we could measure the function deterministically in equation (14.15). The second term in (14.16) represents a heuristic adjustment term that rewards uncertainty.

If $\sigma^{2,n}(x) = 0$, then this adjustment term is zero and we would not place any value in measuring that point.

The expectation can be calculated analytically using

$$\begin{aligned}\mathbb{E}^n\left[\max\left(\bar{Y}^n(x^{**}) - \mu(x)\right)\right] &= \left(\bar{Y}^n(x^{**}) - \bar{Y}^n(x)\right)\Phi\left(\frac{\bar{Y}^n(x^{**}) - \bar{Y}^n(x)}{\sigma^n(x)}\right) \\ &\quad + \sigma^n(x)\phi\left(\frac{\bar{Y}^n(x^{**}) - \bar{Y}^n(x)}{\sigma^n(x)}\right),\end{aligned}$$

where ϕ and Φ are the standard normal density and cumulative distribution functions.

Define a utility function that captures the uncertainty associated with a point x after n observations. For example, we might choose

$$u^n(x) = -\left(\bar{Y}^n(x) + \sigma^n(x)\right).$$

We choose our effective best solution, x^{**}, by maximizing the utility. We could try to search over the entire region $x \in \mathcal{X}$, but a common shortcut is to limit the search over previously sampled locations $x^1, \ldots, x^n$, giving us the rule

$$x^{**} = \arg\max_{x^1,\ldots,x^n}[u^n(x)].$$

14.3 EFFICIENT GLOBAL OPTIMIZATION

While our interest is in solving problems where we can only obtain noisy observations of our function $\mu(x)$, an important contribution is a procedure known as *efficient global optimization* (EGO) which was developed for the case where there is no noise. EGO is similar to the knowledge gradient in that it uses the same idea of choosing to measure the point that maximizes the expected value of information. We have already seen this concept in Section **??**, where it was known as the expected improvement (EI) procedure. In the literature, the names EGO and EI are used interchangeably to refer to the same technique.

Following our standard notation, we assume that we choose an experiment x^n and then observe $\hat{y}^{n+1} = \mu(x^n)$. Our indexing system follows the style of the rest of the book, and is designed to handle uncertainty. Below, $\hat{y}^{n+1}$ will be a random variable when we choose x^n, but in this section it is deterministic.

We start by defining

$$I^{n+1}(x) = \max\left(\theta^{n+1}(x) - \max_{i=1,..,n}\hat{y}^i, 0\right). \tag{14.17}$$

If we have no observation noise, $\bar{\nu}^{KG,n}(x) \leq \mathbb{E}^n[I^{n+1}(x)]$. Furthermore, $\mathbb{E}^n[I^{n+1}(x)] = \mathbb{E}^n\left[\max_{i=0,..,n}\theta^{n+1}(x^i)|x^n = x\right] - \max_{i=0,..,n-1}\theta^n(x^i)$. We show this using

$$\begin{aligned}
\bar{\nu}^{KG,n}(x) &= \mathbb{E}^n\left[\max_{i=0,..,n}\theta^{n+1}(x^i)|x^n = x\right] - \max_{i=0,..,n}\theta^n(x^i)\Big|_{x=x^n} \\
&\leq \mathbb{E}^n\left[\max_{i=0,..,n}\theta^{n+1}(x^i)\Big|x = x^n\right] - \max_{i=0,..,n-1}\theta^n(x^i) \\
&= \mathbb{E}^n\left[\max\left(\theta^{n+1}(x^n), \max_{i=0,..,n-1}\theta^n(x^i)\right)\Big|x = x^n\right] - \max_{i=0,..,n-1}\theta^n(x^i) \\
&= \mathbb{E}^n\left[\max\left(\theta^{n+1}(x^n), \max_{i=1,..,n}\hat{y}^i\right)\Big|x = x^n\right] - \max_{i=1,..,n}\hat{y}^i \\
&= \mathbb{E}^n\left[\max\left(\theta^{n+1}(x^n) - \max_{i=1,..,n}\hat{y}^i, 0\right)\Big|x = x^n\right] \\
&= \mathbb{E}^n[I^{n+1}(x)]. \qquad (14.18)
\end{aligned}$$

In the third line, we used the fact that, given what we know at time n, $\hat{y}^{i+1} = \theta^n(x^i) = \theta^{n+1}(x^i)$ for $i = 0, ..., n-1$ since there is no observation noise. The EGO algorithm maximizes the expected improvement given by equation (14.18) at each iteration. If we assume that there is no experimental noise, the expectation of (14.17) has a closed-form solution

$$\nu^{EGO,n}(x) = \tilde{\sigma}_x\left(\Sigma^n, x\right) f\left(\frac{\theta^n(x) - \max_{i=1,...,n}\hat{y}^i}{\tilde{\sigma}_x\left(\Sigma^n, x\right)}\right),$$

much like its analog in (4.34). This policy closely parallels the knowledge gradient for the case where there is no observation noise, and does not account for correlations in the belief structure.

14.4 EXPERIMENTS

It will always be hard to draw firm conclusions about the performance of algorithms in real experiments. In this section, we draw on a series of experiments reported in Scott et al. (2011), where SKO and KGCP were compared using a series of standard test problems as well as new test problems that were generated directly from the Gaussian process model. The standard test problems are known as Ackley, Branin, Hartman3 and the Six Hump Camelback. The Gaussian process datasets generated scalar functions using $\alpha = .1, 1.0, 10.0$, which captures the structural deviation between the true surface and the approximation. Recall that for smaller values of α, these deviations are described by smooth undulations, while larger values of α produce high frequency ripples. All of the test problems were run with three different levels of experimental noise, where we used $\lambda = .1, 1.0$ and 10.0.

Table 14.1 summarizes seven test problems, along with the results of comparisons of SKO and KGCP using the expected opportunity cost as the performance metric. These results suggest that the knowledge gradient algorithm consistently outperforms SKO, sometimes dramatically so. There was only one dataset where SKO outperformed KGCP in a statistically significant way (for the Six Hump Camelback, with $\lambda = 10$), but the expected opportunity cost for KGCP was 1.0264 versus .8488 for SKO, which

is a small difference, especially compared to the relative performance for some of the other problems. As a general pattern, the relative improvement of KGCP over SKO was largest for problems where the experimental noise was the smallest. We suspect that as a general rule, differences in algorithms will become less noticeable as the experimental noise becomes larger.

		KGCP		SKO	
Test Function	$\sqrt{\lambda}$	$\mathbb{E}(OC)$	$\sigma(OC)$	$\mathbb{E}(OC)$	$\sigma(OC)$
Ackley 5 ($\mathcal{X} = [-15, 30]^5$)	$\sqrt{.1}$	$\mathbf{5.7304}$	.1874	7.8130	.1802
	$\sqrt{1.0}$	$\mathbf{10.8315}$	.2413	12.6346	.2088
$p = 5, \sigma = 1.126$	$\sqrt{10.0}$	$\mathbf{17.3670}$	.1477	18.1126	.1156
Branin	$\sqrt{.1}$	$\mathbf{.0141}$	.0044	.0460	.0023
	$\sqrt{1.0}$	$\mathbf{.0462}$	.0039	.1284	.0218
$p = 2, \sigma = 51.885$	$\sqrt{10.0}$	$\mathbf{.2827}$	.0186	.4396	.0248
Hartman3	$\sqrt{.1}$	$\mathbf{.0690}$	.0063	.1079	.0075
	$\sqrt{1.0}$	.5336	.0296	.5012	.0216
$p = 3, \sigma = .938$	$\sqrt{10.0}$	1.8200	.0541	1.8370	.0510
Six Hump Camelback	$\sqrt{.1}$	$\mathbf{.0714}$	.0087	.1112	.0059
	$\sqrt{1.0}$	.3208	.0192	.3597	.0156
$p = 2, \sigma = 3.181$	$\sqrt{10.0}$	1.0264	.0391	$\mathbf{.8488}$	.0370
GP ($\alpha = .1, \beta = 100$)	$\sqrt{.1}$	.0076	.0057	.0195	.0041
	$\sqrt{1.0}$	.0454	.0243	.0888	.0226
$p = 1, \sigma = 8.417$	$\sqrt{10.0}$	.3518	.0587	.2426	.0216
GP ($\alpha = 1, \beta = 100$)	$\sqrt{.1}$	$\mathbf{.0077}$	.0022	.0765	.0311
	$\sqrt{1.0}$	$\mathbf{.0270}$	.0045	.1993	.0486
$p = 1, \sigma = 9.909$	$\sqrt{10.0}$	$\mathbf{.4605}$	.1028	.6225	.0669
GP ($\alpha = 10, \beta = 100$)	$\sqrt{.1}$	$\mathbf{.1074}$	.0259	.5302	.0799
	$\sqrt{1.0}$	$\mathbf{.1846}$	.0286	.6638	.0839
$p = 1, \sigma = 10.269$	$\sqrt{10.0}$	$\mathbf{1.0239}$	.1021	1.8273	.1450

Table 14.1 Comparison of the knowledge gradient algorithm for continuous parameters to sequential kriging optimization for noisy experiments, from experiments reported in Scott et al. (2011).

14.5 EXTENSION TO HIGHER DIMENSIONAL PROBLEMS

The algorithms we have presented in this chapter can be used for multidimensional parameter vectors, but care has to be used when optimizing over higher dimensional parameter spaces. It is not unusual to see algorithms which work in multiple dimensions being tested on problems with one or two dimensions. Transitioning to as few as five dimensions can actually be quite difficult for certain algorithms. Searching a parameter space with 10 or 20 dimensions is dramatically harder than five dimensions without making suitable approximations.

There are several strategies that can be used to search higher dimensional parameter spaces. Some of these include

- Make a random sample of the parameter space $\mathcal{X}$, producing a set of possible parameters $x_1, x_2, \ldots, x_M$. Now, use traditional optimal learning policies for discrete alternatives.
- Assume that the function $\mu(x)$ is approximately separable in x, and use this property to search each dimension separately (possibly using the techniques in this chapter for continuous parameters).
- Stitch together a solution by optimizing over small subsets of dimensions. This can be done in parallel for different subsets of dimensions, after which new, overlapping subsets can be chosen.

The challenge of dimensionality exists even with classical stochastic search algorithms, where efficient learning may not be an issue. For example, imagine that we have a stochastic function $F(x, W)$ that depends on a controllable vector x and a random vector W. Let $W(\omega)$ be a sample realization of $F(x, W)$, and assume that we are able to compute the gradient $\nabla_x F(x, W(\omega))$. We might be able to solve the optimization problem

$$\max_x \mathbb{E} F(x, W)$$

using a classical stochastic gradient algorithm of the form

$$x^n = x^{n-1} + \alpha_{n-1} \nabla_x F(x^{n-1}, W(\omega^n)). \tag{14.19}$$

Stochastic gradient algorithms such as (14.19) can handle problems with thousands of dimensions, but can exhibit very slow convergence, even when we assume that we can compute the gradient $\nabla_x F(x^{n-1}, W(\omega^n))$. These algorithms work by basically linearizing the function and dealing with each dimension individually. Although we have not seen this in the research literature, it is possible to envision the use of optimal learning techniques applied in each dimension individually. However, for high dimensional problems, we would be limited to very simple policies. Needless to say, optimal learning for high dimensional problems is an interesting area of research.

14.6 BIBLIOGRAPHIC NOTES

There is an extensive literature on stochastic search for continuous variables which we have not covered in this chapter because they do not explicitly address the issue of value of information. An excellent review of the literature as of 2003 is given by Spall (2003). This literature can be largely divided between algorithms that assume that we have access to gradient information (or at least stochastic gradient), which is our focus.

There is a number of papers tackling the problem of optimizing noisy functions without derivative information. These algorithms primarily depend on methods for

Monte Carlo sampling of the search region $\mathcal{X}$, and while everyone is looking for algorithms with fast convergence, most of the theory focuses on asymptotic convergence analysis (assuming an off-line application) while convergence rates are studied empirically. Benveniste et al. (1990), Kushner & Yin (1997) and Kushner & Yin (2003) are important references for the theory behind stochastic search algorithms. Some recent contributions include adaptive search with resampling (Andradóttir & Prudius (2010)) and model reference adaptive search (Hu et al. (2007)).

Sections 14.1 - Our belief model is based on material presented in Sacks et al. (1989) and Frazier et al. (2009). Our presentation of Gaussian process regression is based on Rasmussen & Williams (2006); the material on regression kriging in Forrester et al. (2008). The recursive updating equations for the means and variances are based on Frazier et al. (2009). The maximum likelihood estimation method for the Gaussian process regression model is based on Rasmussen & Williams (2006).

Sections 14.2 - Sequential kriging optimization was proposed by Huang et al. (2006). Stein (1999) provides a thorough introduction to the field of kriging, which evolved from the field of spatial statistics.

Sections **??** - The adaptation of the knowledge gradient for continuous alternatives was developed by Scott et al. (2011).

Sections 14.3 - Efficient global optimization was proposed by Jones et al. (1998*b*).

Sections 14.4 - We draw on a series of experiments reported in Scott et al. (2011). Our test functions are culled from Frazier et al. (2009), Huang et al. (2006) and Jones et al. (1998*b*). See also Scott et al. (2010) for additional empirical work on calibration of an airline business simulator.