Efficient Global Optimization of Expensive Black-Box Functions

Article *in* Journal of Global Optimization · December 1998

DOI: 10.1023/A:1008306431147

3 authors:

Donald R. Jones

22 PUBLICATIONS **8,906** CITATIONS

SEE PROFILE

Matthias Schonlau

University of Waterloo

84 PUBLICATIONS **10,413** CITATIONS

SEE PROFILE

William J. Welch

University of British Columbia - Vancouver

73 PUBLICATIONS **14,644** CITATIONS

SEE PROFILE

Efficient Global Optimization of Expensive Black-Box Functions

DONALD R. JONES (djones@gmr.com)
Operations Research Department, General Motors R&D Operations, 30500 Mound Road, Warren, Michigan 48090, USA

MATTHIAS SCHONLAU (schonlau@niss.org)*
National Institute of Statistical Sciences, P.O. Box 14006, Research Triangle Park, North Carolina 27709-4006, USA

WILLIAM J. WELCH (wjwelch@uwaterloo.ca)†
Department of Statistics and Actuarial Science and The Institute for Improvement in Quality and Productivity, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada

(Received: .....; Accepted: .....)

Abstract. In many engineering optimization problems, the number of function evaluations is severely limited by time or cost. These problems pose a special challenge to the field of global optimization, since existing methods often require more function evaluations than can be comfortably afforded. One way to address this challenge is to fit response surfaces to data collected by evaluating the objective and constraint functions at a few points. These surfaces can then be used for visualization, tradeoff analysis, and optimization. In this paper, we introduce the reader to a response surface methodology that is especially good at modeling the nonlinear, multimodal functions that often occur in engineering. We then show how these approximating functions can be used to construct an efficient global optimization algorithm with a credible stopping rule. The key to using response surfaces for global optimization lies in balancing the need to exploit the approximating surface (by sampling where it is minimized) with the need to improve the approximation (by sampling where prediction error may be high). Striking this balance requires solving certain auxiliary problems which have previously been considered intractable, but we show how these computational obstacles can be overcome.

Keywords: Bayesian global optimization, kriging, random function, response surface, stochastic process, visualization.

* Matthias Schonlau was a graduate student in the Department of Statistics and Actuarial Science and The Institute for Improvement in Quality and Productivity when this research was conducted.

† Research supported by the Natural Sciences and Engineering Research Council of Canada.

1. Introduction

In the automotive and semiconductor industries, as well as many others, there is a growing emphasis on designing products using math/computer models. Computer models facilitate the exploration of alternative designs and reduce the need for expensive hardware prototypes. This math-based approach is often made difficult, however, by the long running times of the computer codes involved. For example, an automotive crash simulation may take twenty hours. Designing optimization algorithms that can deal with such expensive functions is a great challenge to the optimization community. In this paper, we explore an approach based on fitting response surfaces to data collected by evaluating the objective and constraint functions at a few points. These response surfaces are then used to visualize input-output relationships, estimate the location of the optimum, and suggest points where additional function evaluations may help improve this estimate.

The response surface methodology we use is based on modeling the objective and constraint functions with stochastic processes—an approach that to many people seems complex and unnatural. The basic idea behind this approach, however, is quite intuitive. When we fit a stochastic process to data, we are essentially calibrating a model that summarizes how the function typically behaves, properties like how much the function tends to change as we move by different amounts in each coordinate direction. When predicting at a new point, we are essentially computing the function value that is most consistent with this estimated typical behavior. As we show later, this methodology is especially good at modeling the nonlinear, multimodal functions that often occur in engineering.

The stochastic process approach to approximating functions has a long history in at least three literatures: mathematical geology, global optimization, and statistics. In the mathematical geology literature, the approach is called "kriging" and dates back to the early 1960's (see [10], [11], [12], and [21]). Here the data often consist of core samples taken at different locations, and the goal is to find a function that approximates the underground concentration of a valuable mineral.

In global optimization, the use of stochastic processes is called "Bayesian global optimization" or the "random function approach." It dates back to a seminal article by Harold Kushner in 1964 [19] and has since been pursued by many authors (e.g., [4], [5], [9], [16], [23], [24], [27], [31], [33], and [37]). The focus here is on using the stochastic process to develop "figures of merit" for where to take search points. These figures of merit balance local and global search in an attractive fashion.

In statistics, the approach began in the early 1970's out of a general interest in approximating integrals and other hard-to-compute "functionals" of functions [30]. Most recently, the focus has been on developing accurate approximations to expensive computer codes and then using these approximations for visualization and optimization ([13],[18],[29], [36]).

While all the above literatures use stochastic processes, there are differences in emphasis, in the specific types of stochastic processes, and in parameter estimation. In mathematical geology, for example, some of the model fitting techniques are specifically designed for two or three dimensions and it is usually assumed that the functions are noisy—all quite reasonable when one is modeling ore grade as a function of location in a mineralized zone. In the statistics literature, on the other hand, the functions are usually deterministic and have more than two variables.

In our work, we take the stochastic process model commonly used in the statistics literature and apply it to global optimization. Two things set us apart from previous work in Bayesian global optimization.

First, we emphasize the need to validate the stochastic process model before using it to guide an optimization procedure. For this purpose we have developed several diagnostic tests based on cross validation. When the model fails to validate, it is sometimes possible to transform the function or otherwise modify the model so that it does validate.

Second, the figure of merit we use to select search points is rigorously based on a statistical model, and we optimize this figure of merit exactly using a special branch-and-bound algorithm. In contrast, Mockus [23] and Zilinskas [37] use simplified statistical models and optimize their figures of merit heuristically using multistart and Monte Carlo methods. The algorithms of Elder [16], Perttunen [27], and Stuckman [33] are heuristic extensions of Kushner's one-dimensional algorithm [19] and, as such, do not use any statistical model at all. Cox and John [9] use a statistical model, but they optimize the figure of merit by complete enumeration over a fine grid—a procedure obviously limited to one or two dimensions. Booker et al. [6], working on applications at Boeing Corporation, develop a procedure called BLGS based on a statistical model but do not use a figure of merit to select search points. Instead, at each iteration they sample some points where the value of the response surface is very good, as well as other points where the estimated error in the surface is high. Booker et al. [7] explore hybrid approaches in which the stochastic process model is combined with other optimization techniques.

The response surface approach to global optimization has three major advantages. First, the technique often requires the fewest function

evaluations of all competing methods. This is possible because, with typical engineering functions, one can often interpolate and extrapolate quite accurately over large distances in the design space. Intuitively, the method is able to "see" obvious trends or patterns in the data and "jump to conclusions" instead of having to move step-by-step along some trajectory.

Second, the response surface approach provides a credible stopping rule based on the expected improvement from further searching. Such a stopping rule is possible because the statistical model provides confidence intervals on the function's value at unsampled points—and the "reasonableness" of these confidence intervals can be checked by model validation techniques.

Third, the response surface approach provides a fast approximation to the computer model that can be used to identify important variables, visualize the nature of the input-output relationships, and quantify tradeoffs between multiple objectives. In short, the approach not only provides an estimate of the optimal point, but also facilitates the development of intuition and understanding about what is going on in the model.

In Section 2 we introduce the reader to the stochastic process model. Rather than giving a formal presentation, we motivate the stochastic process model as a modification to linear regression that addresses some of regression's major shortcomings. We show how the stochastic process model is used to construct response surfaces and comment on how this approach is related to splines and traditional design-of-experiment methods. Section 3 describes our techniques for validating a stochastic process model, and Section 4 shows how these models are used for global optimization. The topic of visualization is explored in Section 5 using examples from real problems. Finally, Section 6 discusses some remaining challenges and opportunities.

2. The Stochastic Process Model

Suppose we have evaluated a deterministic function of k variables at n points. Denote sampled point i by $\mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_k^{(i)})$ and the associated function value by $y^{(i)} = y(\mathbf{x}^{(i)})$, for $i = 1, \ldots, n$. Perhaps the simplest and most familiar way to fit a response surface to such data is linear regression. In this technique, the observations are treated as if they were generated from the following model:

$$y(\mathbf{x}^{(i)}) = \sum_h \beta_h f_h(\mathbf{x}^{(i)}) + \epsilon^{(i)} \qquad (i = 1, \ldots, n).$$

In this equation, each $f_h(\mathbf{x})$ is a linear or nonlinear function of $\mathbf{x}$; the β_h's are unknown coefficients to be estimated; and the $\epsilon^{(i)}$'s are normally distributed, independent error terms with mean zero and variance σ^2.

Applying linear regression to a computer code has major practical and conceptual problems. The practical problem is that we usually do not know what functional form to specify for the regression terms. After all, if we knew a form that fit well, why would we have developed a complex computer code in the first place? Of course, one can always use a flexible functional form that assumes different shapes via different parameter settings. But flexible functional forms, by their very nature, have many parameters, and so we would need many function evaluations to estimate these parameters.

The conceptual problem with regression is that the assumption of independent errors is blatantly false when modeling a deterministic computer code. Because the code is deterministic, any lack of fit will be entirely modeling error (incomplete set of regression terms), not measurement error or noise. This means that the error terms are really collections of left-out terms in $\mathbf{x}$, so that we may write $\epsilon^{(i)}$ as $\epsilon(\mathbf{x}^{(i)})$. Moreover, if $y(\mathbf{x})$ is continuous, then $\epsilon(\mathbf{x})$ is also continuous, because it is the difference between $y(\mathbf{x})$ and the continuous regression terms. It follows that, if $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are two points that are close together, then the errors $\epsilon(\mathbf{x}^{(i)})$ and $\epsilon(\mathbf{x}^{(j)})$ should also be close. In short, it makes no sense to assume that $\epsilon(\mathbf{x}^{(i)})$ and $\epsilon(\mathbf{x}^{(j)})$ are independent. Instead, it is more reasonable to assume that these error terms are related or "correlated," and that this correlation is high when $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are close and low when the points are far apart.

In the stochastic process approach, we do not assume that the errors are independent, but rather assume, as indicated above, that the correlation between errors is related to the distance between the corresponding points. We do not use the Euclidean distance, however, since this distance weights all the variables equally. Rather, we use the special weighted distance formula shown below:

$$d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sum_{h=1}^{k} \theta_h |x_h^{(i)} - x_h^{(j)}|^{p_h} \qquad (\theta_h \geq 0, p_h \in [1, 2]). \tag{1}$$

(We will discuss the roles played by the parameters θ_h and p_h shortly). Using this distance function, the correlation between the errors at $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ is

$$\text{Corr}\left[\epsilon(\mathbf{x}^{(i)}), \epsilon(\mathbf{x}^{(j)})\right] = \exp\left[-d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})\right]. \tag{2}$$

The correlation function defined in (1) and (2) has all the intuitive properties one would like it to have. In particular, when the distance

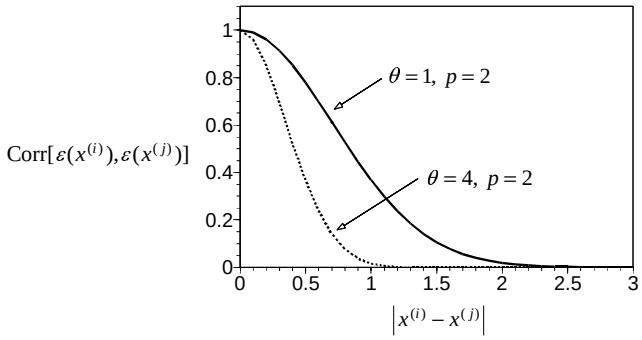

Figure 1. Example correlation functions used in the stochastic process model.

between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ is small, the correlation is near one. Similarly, when the distance between the points is large, the correlation will approach zero. The parameter θ_h in the distance formula (1) can be interpreted as measuring the importance or "activity" of the variable x_h. To see this, note that saying "variable h is active" means that even small values of $|x_h^{(i)} - x_h^{(j)}|$ may lead to large differences in the function values at $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$. Thinking in statistical terms, this means that even small values of $|x_h^{(i)} - x_h^{(j)}|$ should imply a low correlation between the errors $\epsilon(\mathbf{x}^{(i)})$ and $\epsilon(\mathbf{x}^{(j)})$. Looking at equations (1) and (2), we see that, if θ_h is very large, then it will indeed be true that small values of $|x_h^{(i)} - x_h^{(j)}|$ translate to large "distances" and hence low correlation. This is illustrated in Figure 1 for the case of only one input variable, x. Two correlation functions are shown, corresponding to $\theta = 1$ and $\theta = 4$. The curve for $\theta = 4$ relates to a more active variable, as correlation drops off more rapidly with the change in x. The exponent p_h is related to the smoothness of the function in coordinate direction h, with $p_h = 2$ corresponding to smooth functions and values near 1 corresponding to less smoothness [26].

It turns out that modeling the correlation in this way is so powerful that we can afford to dispense with the regression terms, replacing them with a simple constant term. This gives us the model we use in the stochastic process approach:

$$y(\mathbf{x}^{(i)}) = \mu + \epsilon(\mathbf{x}^{(i)}) \qquad (i = 1, \ldots, n), \tag{3}$$

where μ is the mean of the stochastic process, $\epsilon(\mathbf{x}^{(i)})$ is Normal$(0, \sigma^2)$, and, as just discussed, the correlation between errors is not zero but rather is given by equations (1) and (2). The estimates of the parameters μ and σ^2 have little direct interpretation, as they must be combined

with the estimates of the correlation parameters (the θ_h's and p_h's) in order to make predictions.

We call this model a "stochastic process model" because the error term $\epsilon(\mathbf{x})$ *is* a stochastic process, that is, it is a set of correlated random variables indexed by space (here, the k-dimensional space of $\mathbf{x}$). It has become common to call the stochastic process model in equations (1)–(3) the "DACE stochastic process model," where "DACE" is an acronym for "Design and Analysis of Computer Experiments," the title of the paper that popularized the approach [29].

The DACE model has $2k + 2$ parameters: μ, σ^2, $\theta_1, \ldots, \theta_k$, and $p_1, \ldots, p_k$. We estimate these parameters by choosing them to maximize the likelihood of the sample. Let $\mathbf{y} = (y^{(1)}, \ldots, y^{(n)})'$ denote the n-vector of observed function values, $\mathbf{R}$ denote the $n \times n$ matrix whose (i, j) entry is Corr $\left[\epsilon(\mathbf{x}^{(i)}), \epsilon(\mathbf{x}^{(j)})\right]$, and $\mathbf{1}$ denote an n-vector of ones. Then the likelihood function is:

$$\frac{1}{(2\pi)^{n/2}(\sigma^2)^{n/2}|\mathbf{R}|^{\frac{1}{2}}} \exp\left[-\frac{(\mathbf{y} - \mathbf{1}\mu)'\mathbf{R}^{-1}(\mathbf{y} - \mathbf{1}\mu)}{2\sigma^2}\right]. \tag{4}$$

Note that the dependence on the parameters θ_h and p_h for $h = 1, \ldots, k$ is via the correlation matrix $\mathbf{R}$ [see equations (1) and (2)].

Given the correlation parameters θ_h and p_h for $h = 1, \ldots, k$, we can solve for the values of μ and σ^2 that maximize the likelihood function in closed form:

$$\hat{\mu} = \frac{\mathbf{1}'\mathbf{R}^{-1}\mathbf{y}}{\mathbf{1}'\mathbf{R}^{-1}\mathbf{1}} \tag{5}$$

and

$$\hat{\sigma}^2 = \frac{(\mathbf{y} - \mathbf{1}\hat{\mu})'\mathbf{R}^{-1}(\mathbf{y} - \mathbf{1}\hat{\mu})}{n}. \tag{6}$$

Substituting equations (5) and (6) into the likelihood function, we get the so-called "concentrated likelihood function," which depends only upon the parameters θ_h and p_h for $h = 1, \ldots, k$. This is the function that we maximize in practice to give us the estimates $\hat{\theta}_h$ and $\hat{p}_h$, and hence an estimate of the correlation matrix $\mathbf{R}$. We then use formulas (5) and (6) to get the estimates $\hat{\mu}$ and $\hat{\sigma}^2$.

Now the stochastic process model in equations (1)–(3) is essentially a generalized least squares (GLS) model with a simple set of regressors (just a constant term) and a special correlation matrix that has unknown parameters and depends upon distances between the sampled points. It is well known that, when the errors are correlated, the GLS estimates of μ and σ^2 are more efficient than the ordinary least squares estimates, and our maximum likelihood estimates in (5) and (6) agree with the GLS estimates (except for the usual n versus

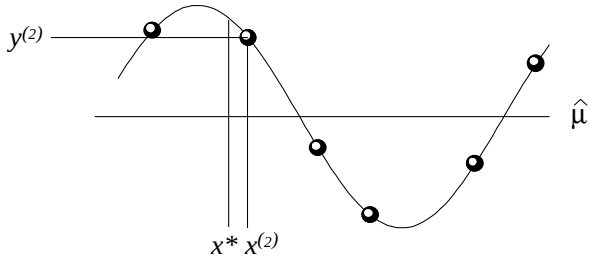

Figure 2. A simple illustration of how correlation should affect prediction. The prediction at x^*, being close to the data point $x^{(2)}$, should be adjusted from the regression line to take into account the positive residual at $x^{(2)}$.

$n-1$ in the denominator of $\hat{\sigma}^2$). But the impact of correlated errors on *prediction* is usually only discussed in more advanced statistics books (e.g., [34], p. 280) and in the literature on kriging [12] and computer experiments [29].

In order to get an intuitive understanding for how correlated errors should affect prediction, consider the illustration in Figure 2 where there is only one input variable, x. The point at which we are predicting, x^*, is close to the second data point, $x^{(2)}$. Moreover, let us assume that the value $y(x^{(2)})$ lies significantly above the estimated mean $\hat{\mu}$ as shown in Figure 2. The fact that $y(x^{(2)})$ lies above the regression line implies that the left-out terms in x, which constitute the error, have a large positive value at $x^{(2)}$. Since x^* is close to $x^{(2)}$, it makes intuitive sense that these left-out terms will also be positive (though not identical) at x^*. Thus our prediction at x^* should *not* be constructed merely by plugging x^* into the regression equation (which would just give $\hat{\mu}$). Instead, it should be equal to the value of the regression equation ($\hat{\mu}$) adjusted upward to take into account the correlation with the error at the nearby point $x^{(2)}$ where the residual is large and positive.

Formally, let $\mathbf{r}$ denote the n-vector of correlations between the error term at $\mathbf{x}^*$ and the error terms at the previously sampled points. That is, element i of $\mathbf{r}$ is $r_i(\mathbf{x}^*) \equiv \mathrm{Corr}[\epsilon(\mathbf{x}^*), \epsilon(\mathbf{x}^{(i)})]$, computed using the formula for the correlation function in (1) and (2). It then turns out that the best linear unbiased predictor of $y(\mathbf{x}^*)$ is

$$\hat{y}(\mathbf{x}^*) = \hat{\mu} + \mathbf{r}'\mathbf{R}^{-1}(\mathbf{y} - \mathbf{1}\hat{\mu}). \tag{7}$$

The derivation of this predictor can be found in [29]. On the right-hand side of equation (7), the first term, $\hat{\mu}$, is the result of simply plugging $\mathbf{x}^*$ into the regression equation, and the second term represents the adjustment to this prediction based on the correlation of $\epsilon(\mathbf{x}^*)$ with the error terms at the sampled points. Note that if there is no correlation

($\mathbf{r} = \mathbf{0}$), then we just predict $\hat{y}(\mathbf{x}^*) = \hat{\mu}$. To see that the DACE predictor interpolates the data, let us suppose we are making a prediction at the ith sampled point, so that $\mathbf{x}^* = \mathbf{x}^{(i)}$. In this case, $\mathbf{r}$ will be equivalent to the ith column of $\mathbf{R}$, which we may denote by $\mathbf{R}_i$. Hence

$$\mathbf{r}'\mathbf{R}^{-1} = (\mathbf{R}^{-1}\mathbf{r})' = (\mathbf{R}^{-1}\mathbf{R}_i)' = \mathbf{e}_i', \tag{8}$$

where $\mathbf{e}_i$ is the ith unit vector. Equation (7) then reduces to

$$\hat{y}(\mathbf{x}^{(i)}) = \hat{\mu} + \mathbf{e}_i'(\mathbf{y} - \mathbf{1}\hat{\mu}) = \hat{\mu} + (y^{(i)} - \hat{\mu}) = y^{(i)}.$$

Thus, the prediction at $\mathbf{x}^{(i)}$ is the observed value $y^{(i)}$, and the DACE predictor interpolates the data.

The correlation of the errors should also affect our estimate of prediction accuracy. Going back to Figure 2, it makes intuitive sense that, since x^* is very close to $x^{(2)}$, we should be much more confident in our prediction of $y(x^*)$ than we would be if x^* were far away from all the sampled points. This intuition is reflected in the general formula for the mean squared error of the predictor, which we denote by $s^2(\mathbf{x}^*)$:

$$s^2(\mathbf{x}^*) = \sigma^2 \left[1 - \mathbf{r}'\mathbf{R}^{-1}\mathbf{r} + \frac{\left(1 - \mathbf{1}'\mathbf{R}^{-1}\mathbf{r}\right)^2}{\mathbf{1}'\mathbf{R}^{-1}\mathbf{1}} \right]. \tag{9}$$

(A full derivation of this formula can also be found in [29].) In the expression for $s^2(\mathbf{x})$, the term $-\mathbf{r}'\mathbf{R}^{-1}\mathbf{r}$ represents the reduction in prediction error due to the fact that $\mathbf{x}^*$ is correlated with the sampled points. With no correlation, that is, if $\mathbf{r} = \mathbf{0}$, then this adjustment would be zero. The term $(1-\mathbf{1}'\mathbf{R}^{-1}\mathbf{r})^2/\mathbf{1}'\mathbf{R}^{-1}\mathbf{1}$ reflects the uncertainty that stems from our not knowing μ exactly, but rather having to estimate it from the data. Finally, let us again suppose that we are making a prediction at the ith sampled point, so that $\mathbf{x}^* = \mathbf{x}^{(i)}$. As shown in (8) we would then have $\mathbf{R}^{-1}\mathbf{r} = \mathbf{e}_i$, so that

$$\mathbf{r}'\mathbf{R}^{-1}\mathbf{r} = \mathbf{r}'\mathbf{e}_i = r_i(\mathbf{x}^*) \equiv \mathrm{Corr}(\mathbf{x}^*, \mathbf{x}^{(i)}) = \mathrm{Corr}(\mathbf{x}^{(i)}, \mathbf{x}^{(i)}) = 1 \tag{10}$$

and

$$\mathbf{1}'\mathbf{R}^{-1}\mathbf{r} = \mathbf{1}'\mathbf{e}_i = 1. \tag{11}$$

Substituting (10) and (11) into equation (9), it follows that $s^2(\mathbf{x}^{(i)}) = 0$. This is as it should be: with a deterministic function, once we have sampled a point, we know its value there. Thus, our uncertainty, as measured by mean squared error, should be zero.

It will often be convenient for us to work with the square root of the mean squared error, $s = \sqrt{s^2(\mathbf{x})}$. This provides a root mean squared error (RMSE) for measuring uncertainty in our predictions.

In summary, we have the following. The RMSE at a sampled point is zero. The RMSE at a point very far away from the data (where $\mathbf{r} \approx \mathbf{0}$) is about σ. And the RMSE in between these extremes is σ reduced by an amount that depends on how close, and therefore how correlated, the point in question is to the sampled points. Similar statements can be made about the DACE predictor $\hat{y}(\mathbf{x})$. At a sampled point, $\hat{y}(\mathbf{x})$ agrees with the data. At a point very far from the data (where $\mathbf{r} \approx \mathbf{0}$), $\hat{y}(\mathbf{x})$ is about $\hat{\mu}$. In between these extremes, $\hat{y}(\mathbf{x})$ is based on a smooth interpolation of the data according to equation (7).

The derivation of the predictor $\hat{y}(\mathbf{x})$ as the "best linear unbiased predictor" is not especially intuitive or enlightening and, for that reason, we have not repeated it here (interested readers can find it in [29]). Fortunately, however, there is a more intuitive way of deriving the predictor. As mentioned earlier, the parameters of the model—μ, σ^2, $\theta_1, \ldots, \theta_k$, and $p_1, \ldots, p_k$—describe how the objective function *typically behaves*. In particular, the parameters $\theta_1, \ldots, \theta_k$ describe how sensitive the function is to each input variable and the parameters $p_1, \ldots, p_k$ capture how smoothly the function varies in response to each variable. When we estimate these parameters by maximum likelihood, we are essentially finding values of the parameters that best describe the behavior of the function as evidenced in our sample. When we predict the function's value at some new point $\mathbf{x}^*$, we are—in a very precise sense—computing the value of the function that is most consistent with this typical behavior. Let us explain.

Suppose we just guessed the value of the function at $\mathbf{x}^*$ to be some number y^* and then added this "pseudo observation" as point $n+1$. Having added this point, we could compute the likelihood of the augmented sample. This likelihood would measure how well the pseudo observation "fits" with the original data, that is, the likelihood that they were all generated from the same model. Of course, we could guess many different values of y^*, and for each guess we would get a different likelihood value. It turns out that the value of y^* that maximizes this augmented likelihood, and in this sense is most consistent with the function's typical behavior, is precisely the predictor in equation (7). (See Appendix 1 for details.)

One weakness of DACE theory is that the predictor and its mean squared error are derived under the assumption that the parameters σ^2, $\theta_1, \ldots, \theta_k$, and $p_1, \ldots, p_k$ are *known*. In practice, the true values of these parameters are not known, and the DACE formulas are used substituting the *estimated* parameters. This theoretical sleight of hand appears to have no serious consequences, although it probably leads to a slight underestimation of prediction error in small samples. In what follows, we will write σ^2, $\theta_1, \ldots, \theta_k$, and $p_1, \ldots, p_k$ without "hats" ($\hat{}$),

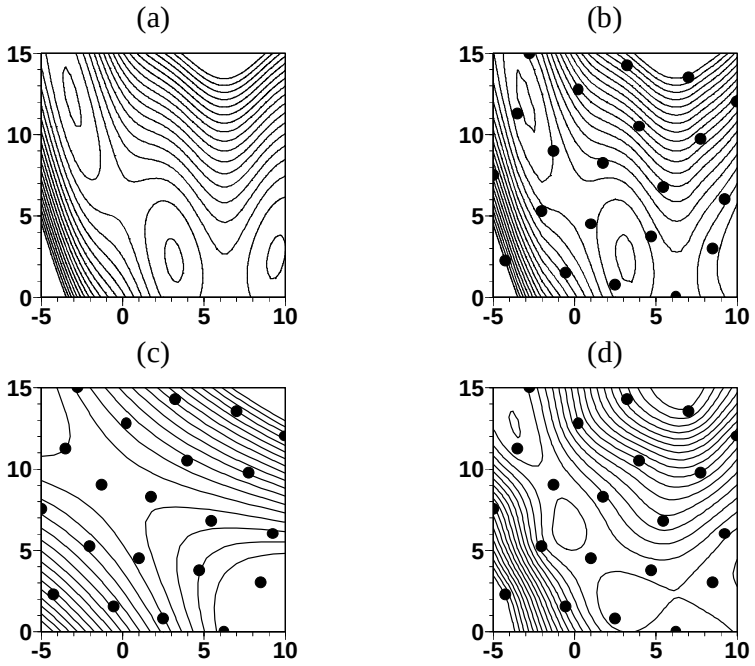

Figure 3. (a) Contours of the Branin test function; (b) contours of a DACE response surface based on the 21 sampled points shown as dots; (c) a quadratic surface fit to the 21 points; (d) a thin-plate spline fit to the 21 points.

as if they are the true values, since this is necessary for the formulas to be theoretically correct. However, the reader should understand that, for all calculations, we use the maximum likelihood estimates of these parameters. In particular, the root mean squared error, $s(\mathbf{x})$, from (9) is also an estimate in practice, and we shall refer to it as the standard error of prediction.

While the above discussion has hopefully made the DACE model conceptually appealing, nothing is as convincing as seeing it in action. In Figure 3(a) we show the contours of the Branin function, a well-known test function for global optimization [14]. In Figure 3(b), we show the contours of the predictor from a DACE model based only on the 21 points shown as dots. This response surface was created by estimating the parameters μ, σ^2, θ_1, θ_2, p_1, and p_2 via maximum likelihood and then calculating the predictor in (7) over a fine grid for the purpose of generating the contour plot. The DACE predictor is so accurate that some people do not even notice the differences between the contours in Figures 3(a) and 3(b). It is clear that we should be able to locate the optimum quite accurately with only a few more function evaluations.

In contrast, Figure 3(c) shows the result of fitting a general quadratic surface and Figure 3(d) shows a thin-plate spline. The quadratic function completely misses the minimum in the lower center. The thin-plate spline, while better than the quadratic, still does not capture the shape of the function as well as the DACE interpolator.

There is an interesting relationship between the DACE predictor and the thin-plate spline. To see this, let us write the DACE predictor in (7) as

$$\hat{y}(\mathbf{x}) = \hat{\mu} + \mathbf{c}'\mathbf{r} = \hat{\mu} + \sum_{i=1}^{n} c_i r_i(\mathbf{x}),$$

where $\mathbf{c} = \mathbf{R}^{-1}(\mathbf{y} - \mathbf{1}\hat{\mu})$ is a vector of constants and where $r_i(\mathbf{x}) = \mathrm{Corr}[\epsilon(\mathbf{x}), \epsilon(\mathbf{x}^{(i)})]$ for $i = 1, \ldots, n$. Thus, we see that the DACE predictor is a linear combination of "basis functions" $r_i(\mathbf{x})$ for $i = 1, \ldots, n$ that interpolates the data.

Now the thin-plate spline predictor shown in Figure 3(d) is also a linear combination of basis functions. These functions include polynomial terms and terms of the form

$$\varphi(\| \mathbf{x} - \mathbf{x}^{(i)} \|) \qquad (i = 1, \ldots, n), \tag{12}$$

where $\| \cdot \|$ is the Euclidean distance norm and

$$\varphi(t) = t^2 \log(t). \tag{13}$$

Given that both the DACE predictor and the thin-plate spline can be thought of as basis-function methods, it follows that the differences in their predictive accuracy, as seen in Figure 3, must be due to the difference in these basis functions. And that is exactly the case. The basis functions for the thin-plate spline are fixed in advance, being specified by equations (12) and (13). The basis functions in the DACE predictor, however, depend upon the correlation parameters θ_h and p_h for $h = 1, \ldots, k$, and these are "tuned" to the data during the maximum likelihood estimation. This is one explanation for the superior accuracy of the DACE predictor in Figure 3.

In fact, one can get much better results from the thin-plate spline if one first scales the data in a way suggested by the estimated DACE parameters. This is shown in Figure 4 where part (a) reproduces the Branin function, part (b) shows the spline with no scaling, and part (c) shows the spline with scaling. The scaled spline, though not as "smooth," is nevertheless more accurate, though still not as accurate as the DACE surface.

Readers familiar with traditional design of experiments (e.g., the textbook by Box, Hunter, and Hunter [8]) may wonder how these techniques are related to DACE. In both methods, the motivation is to

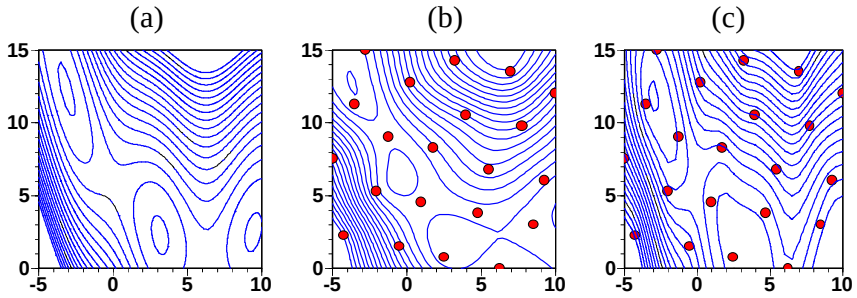

Figure 4. (a) Contours of the Branin test function; (b) a thin-plate spline fit to the 21 points; (c) a thin-plane spline fit using scaling suggested by the estimated DACE parameters.

assess how the performance of an engineering design depends upon certain factors of interest. In standard design of experiment (DOE) methods, it is assumed that the data come from physical observations subject to random error. This error is due to the presence of other factors, in addition to those under study, that cannot be easily controlled or measured. As a result, much of the effort in standard DOE is on minimizing the effect of uncontrolled factors by careful blocking and randomization of run order. Because substantial noise is assumed to be present, it is unrealistic to model complex nonlinear behavior, and so simple linear or quadratic regression models are usually employed (they are implied if not always explicitly written down). To fit these simple regression models, designs with two or three levels for each factor are sufficient.

In a computer experiment, the situation is quite different. All factors are controllable and run order is irrelevant. Moreover, with computer models, it is common to allow the factors to vary over wider ranges than in physical experiments, and over such large ranges the function will be more nonlinear. To capture this nonlinearity, we use the stochastic process model discussed above in conjunction with designs that vary each factor over many levels. This motivates the use of space-filling designs such as Latin hypercubes [22], an example of which is the 21-point design in Figure 3(b).

We now summarize the similarities and differences between the standard linear regression model and the DACE model. The two models share a common mathematical framework consisting of regressors and errors, but the emphasis is quite different. Linear regression focuses entirely on the regressors and their coefficient estimates, and makes simplistic assumptions about the errors (independence). In contrast, DACE makes simplistic assumptions about the regressors (just a con-

stant term) and focuses entirely on the correlation structure of the errors. Thus, regression and DACE are probably best thought of as diametric opposites. Regression is about estimating regression coefficients that (together with the assumed functional form) completely describe *what the function is*. DACE is about estimating correlation parameters that describe *how the function typically behaves*. As we have seen, DACE makes predictions by interpolating and extrapolating from the data in a way most consistent with this estimated typical behavior.

3. Model Validation

We saw in the previous section that the DACE model worked well for the Branin test function, giving good prediction accuracy with just 21 evaluations. We assessed this accuracy, however, by comparing the contour plot from the DACE model with the *true* contours (Figure 3). Obviously, this is not a practical procedure. After all, if it were feasible to compute the true contours, we would not need any approximation in the first place. It is practical, however, to select a few additional points as a "validation" or "test" sample and compare actual and predicted values on this small sample. But there is an even better procedure, called "cross validation," that allows us to assess the accuracy of the model without sampling any points beyond those used to fit the model.

The basic idea of cross validation is to leave out one observation, say $y(\mathbf{x}^{(i)})$, and then predict it back *based only on the* $n-1$ *remaining points*. We call this prediction the "cross-validated" prediction of $y(\mathbf{x}^{(i)})$ and denote it by $\hat{y}_{-i}(\mathbf{x}^{(i)})$. The subscript $-i$ emphasizes that observation i is *not* used in making the prediction. In principle, when making these cross-validated predictions, one should re-estimate all the DACE parameters using the reduced sample. However, unless there are very few observations or major outliers, dropping a single observation usually has a negligible effect on the maximum likelihood estimates. What we do in practice, therefore, is to use the DACE parameters estimated using all the observations, but only use the remaining $n-1$ points in computing the correlation matrix $\mathbf{R}$ and the vectors $\mathbf{r}$ and $\mathbf{y}$ in the predictor (7).

In addition to the cross-validated prediction at $\mathbf{x}^{(i)}$, we also get a cross-validated standard error of prediction analogous to (9), which we denote $s_{-i}(\mathbf{x}^{(i)})$. Together, these two quantities can be used to compute a confidence interval for $y(\mathbf{x}^{(i)})$ using the mean prediction plus or minus three standard errors (see Figure 5). Since the DACE model is approximately 99.7% confident that $y(\mathbf{x}^{(i)})$ lies in this interval, an attractive model-validation test is to see if the observed $y(\mathbf{x}^{(i)})$ does

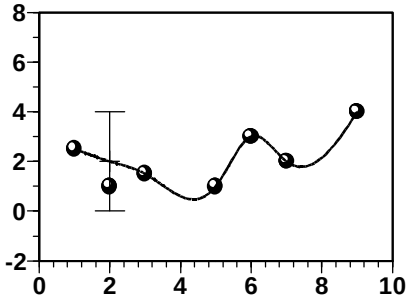

Figure 5. In ordinary cross validation, one observation (here the second) is left out and predicted back using the remaining $n-1$ observations. The cross-validated confidence interval is the cross-validated prediction plus or minus three standard errors.

indeed lie in this interval. Since each of the n points can be left out in this procedure, we can do this test n times.

Instead of actually drawing confidence intervals, it is more convenient to compute the number of standard errors that the actual value is above or below the predicted value:

$$\frac{y(\mathbf{x}^{(i)}) - \hat{y}_{-i}(\mathbf{x}^{(i)})}{s_{-i}(\mathbf{x}^{(i)})}.$$

We will refer to this quantity as the "standardized cross-validated residual." If the model is valid, the value should be roughly in the interval $[-3, +3]$.

To illustrate these ideas, Figure 6 shows the diagnostic tests for a DACE model fit to the Goldstein-Price (GP) test function [14]. The surface was based on the same 21-point experimental design we used for the Branin function [see Figure 3(b)]. In Figure 6(a) we plot the actual function value versus the cross-validated prediction. If the model were good, the points should lie on a 45° line; in this case, the relationship is very weak. Figure 6(b) plots the standardized cross-validated residuals versus the cross-validated predictions. Two problems are apparent. First, there is the one extreme standardized residual with a value around 4. Second, the remaining residuals tend to decrease with the predicted function values, suggesting a systematic bias in prediction. Finally, Figure 6(c) plots the standardized cross-validated residuals versus the values that would be expected from a random sample of n independent standard normal variables (the so-called Q-Q plot). If the standardized residuals act like normal deviates, these points should lie close to the 45° line. The extreme residual is again apparent.

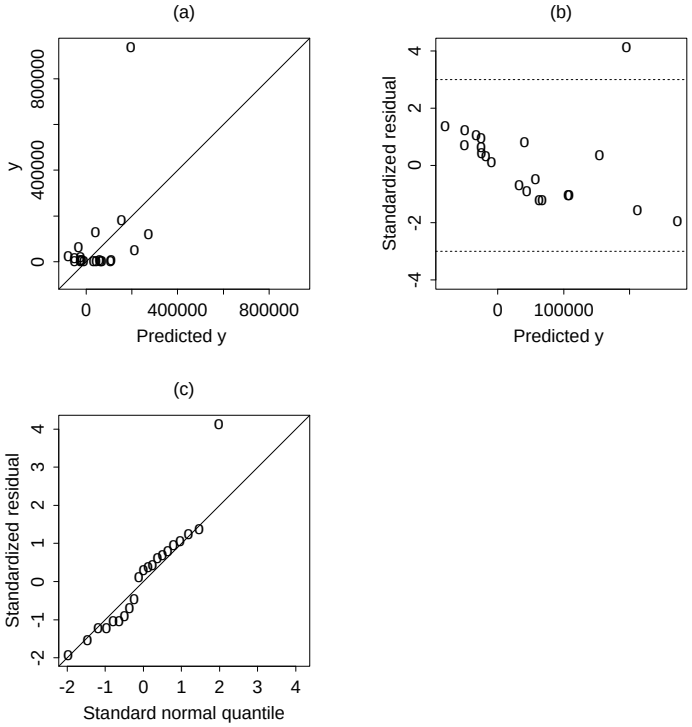

Figure 6. Diagnostic tests for the Goldstein-Price function: (a) actual function values versus cross-validated predictions; (b) standardized cross-validated residuals versus cross-validated predictions; (c) ordered standardized residuals versus standard normal quantiles.

When the diagnostic plots fail to show a good fit, as here, it is sometimes possible to improve the fit of the DACE model by transforming the function. We typically try the log transformation, $\ln(y)$, or the inverse transformation, $-1/y$. For the GP function, the log transformation works well. Figure 7 shows the same diagnostic plots after replacing all the function values by their logs. As can be seen in Figure 7(a), the prediction accuracy is better, but still not great. The important thing, however, is that the model correctly anticipates the magnitude of the prediction errors. This is made clear in Figure 7(b) where the standardized cross-validated residuals are all in the interval $[-3, +3]$. Moreover, the Q-Q plot in Figure 7(c) shows much better agreement than before. The diagnostic plots suggest that, even on the logarithmic scale, the GP function is difficult to model accurately with just 21 points; nevertheless, the model captures these difficulties through the standard error.

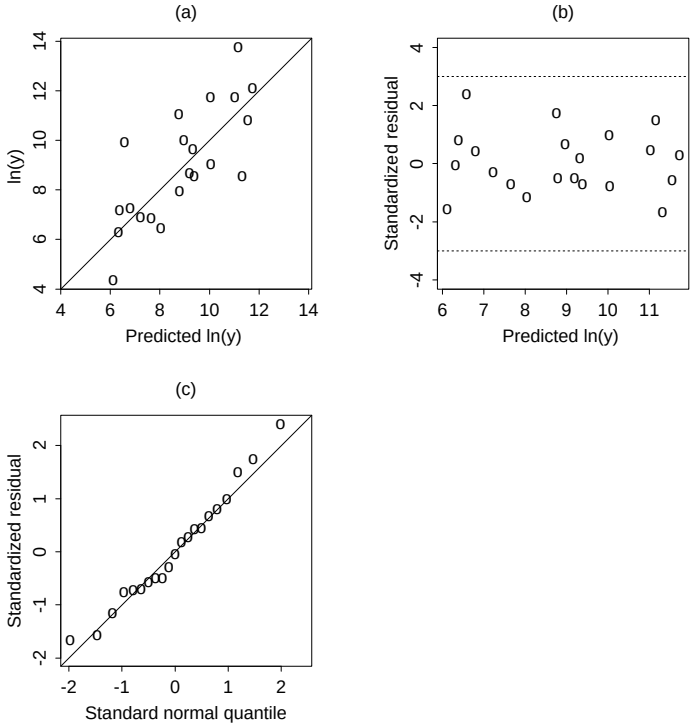

Figure 7. Diagnostic tests for the log-transformed Goldstein-Price function: (a) actual function values versus cross-validated predictions; (b) standardized cross-validated residuals versus cross-validated predictions; (c) ordered standardized residuals versus standard normal quantiles.

4. Global Optimization

4.1. Using Response Surfaces for Global Search

The simplest way to use response surfaces for optimization is to fit a surface and then find the minimum of the surface. But as shown in Figure 8 this process, even if iterated, can easily lead to a local minimum. In Figure 8, the (unknown) objective function is shown as the solid line, and we assume that we have evaluated this function at the five points shown as dots. The dotted line is the DACE predictor fit to these points. The DACE predictor is minimized almost exactly at the local minimum ($x = 2.8$). If we sample the function at the minimum of the DACE surface, update the surface, and iterate, we are clearly only going to get a highly accurate estimate of this local minimum.

The problem with simply finding the minimum of the DACE surface is that this procedure does not acknowledge our uncertainty about that

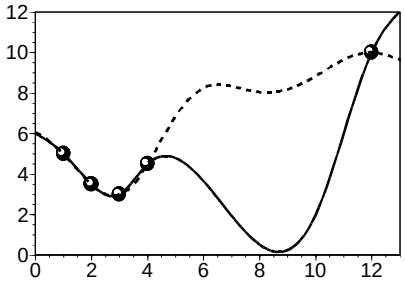

Figure 8. The solid line represents an objective function that has been sampled at the five points shown as dots. The dotted line is a DACE predictor fit to these points.

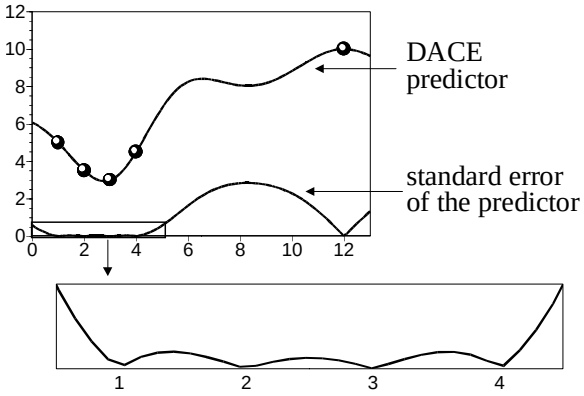

Figure 9. The DACE predictor and its standard error for a simple five-point data set.

surface. It puts too much emphasis on exploiting the predictor and no emphasis on exploring points where we are uncertain. To eliminate this problem, we must put some emphasis on sampling where we are uncertain, as measured by the standard error of the predictor.

Figure 9 shows the standard error of the predictor. Because of the large number of sampled points around $x = 2$, our uncertainty, and hence the standard error of the predictor, is very low in that region. In fact, the standard error in this region is so low that, in order for the reader to see it, we have had to magnify it in an inset. Notice that the standard error does indeed go to zero at all the sampled points, as it should. In rises up in between, but sometimes only by a little. The standard error is maximized around $x = 8.3$, suggesting that this might be a good place to search from the point of view of global search. But sampling there would be equivalent to putting all our emphasis on

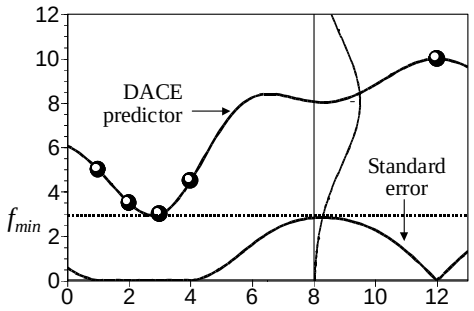

Figure 10. Our uncertainty about the function's value at a point (such as $x = 8$ above) can be treated as if the value there were a realization of a normal random variable with mean and standard deviation given by the DACE predictor and its standard error.

global search, and this is just as bad (if not worse) than putting all our emphasis on local search. What we need is a figure of merit that *balances* local and global search.

A highly attractive figure of merit that balances local and global search is "expected improvement." This concept can be found in the literature as early as 1978 (e.g., Mockus et al. [24]). The expected improvement criterion is computed as follows. Let $f_{\min} = \min(y^{(1)}, \ldots, y^{(n)})$ be the current best function value. Before we sample at some point $\mathbf{x}$, we are uncertain about the value $y(\mathbf{x})$. Of course, there is nothing random about $y(\mathbf{x})$; we simply do not know what it is. Let us model our uncertainty at $y(\mathbf{x})$ by treating it as the realization of a normally distributed random variable Y with mean and standard deviation given by the DACE predictor and its standard error. This idea is illustrated in Figure 10 where, at the point $x = 8$, we have drawn a normal density function with the mean and standard deviation suggested by the DACE model. If we treat the function's value at $x = 8$ as a realization of the random variable Y with the density function shown in Figure 10, then there is some probability that the function's value at $x = 8$ will be better than (or "improve upon") our current best function value $f_{\min}$. This is true because the tail of the density function shown in Figure 10 extends below the line $y = f_{\min}$. Different amounts of improvement, or different distances below the line $y = f_{\min}$, are associated with different density values. If we weight all these possible improvements by the associated density value, we get what we call "expected improvement."

Formally, the improvement at the point $\mathbf{x}$ is $I = \max(f_{\min} - Y, 0)$. This expression is a random variable because Y is a random variable (it models our uncertainty about the function's value at $\mathbf{x}$). To obtain

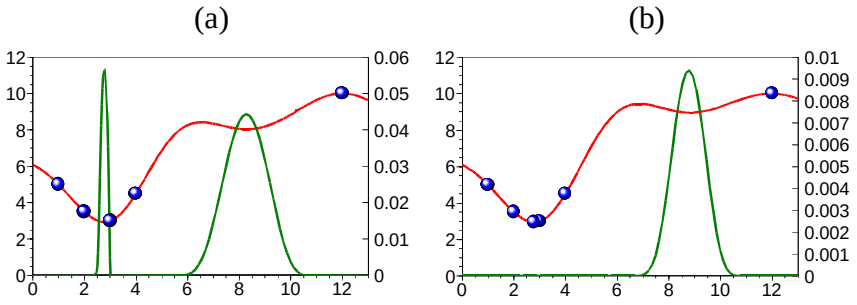

Figure 11. (a) The expected improvement function when only five points have been sampled; (b) the expected improvement function after adding a point at $x = 2.8$. In both (a) and (b) the left scale is for the objective function and the right scale is for the expected improvement.

the expected improvement we simply take the expected value:

$$\mathrm{E}[I(\mathbf{x})] \equiv \mathrm{E}\left[\max(f_{\min} - Y, 0)\right]. \tag{14}$$

To compute this expectation, let us introduce the compact notation $\hat{y}$ and s to denote the DACE predictor and its standard error at $\mathbf{x}$. In this notation, Y is Normal$(\hat{y}, s^2)$. By expressing the right-hand side of (14) as an integral, and applying some tedious integration by parts, one can express the expected improvement in closed form:

$$\mathrm{E}[I(\mathbf{x})] = (f_{\min} - \hat{y})\Phi\left(\frac{f_{\min} - \hat{y}}{s}\right) + s\phi\left(\frac{f_{\min} - \hat{y}}{s}\right). \tag{15}$$

In the above, $\phi(\cdot)$ and $\Phi(\cdot)$ are the standard normal density and distribution function. Note that it is s, not s^2 , that appears in equation (15).

Figure 11(a) shows the expected improvement function for our simple one-dimensional example (the value of the expected improvement is shown on the right-hand scale). Surprisingly, it has two peaks, one at $x = 2.8$ and another at $x = 8.3$. The peak at $x = 2.8$ is higher, so we would sample there. But on the next iteration, as shown in Figure 11(b), the expected improvement is maximized at $x = 8.8$, and thus we are driven to search globally.

As this example illustrates, the expected improvement function is highly multimodal. In fact, it is easy to show that expected improvement is zero at the sampled points and is positive in between (though perhaps very small). As in Figure 11, it is also common for there to be large areas where the expected improvement is essentially zero and so appears quite "flat." Both of these features make optimizing the expected improvement function with standard multistart approaches difficult and potentially unreliable.

Although multimodal, the expected improvement function is in closed form, and so we can hope to exploit its structure when trying to find the maximum. As we will soon show, it turns out that we can maximize $\mathrm{E}[I(\mathbf{x})]$ to guaranteed optimality using a branch-and-bound algorithm. To use branch and bound, we need some way to compute an upper bound on $\mathrm{E}[I(\mathbf{x})]$ over any rectangular subregion defined by $\ell_h \le x_h \le u_h$ for $h = 1, \ldots, k$. The computation of these bounds is greatly facilitated by the fact that $\mathrm{E}(I)$ is monotonic in $\hat{y}$ and in s. In fact, if one computes the derivative of $\mathrm{E}(I)$ as given in (15) with respect to $\hat{y}$ or s, one gets several terms that cancel, resulting in the surprisingly simple expressions:

$$\frac{\partial \mathrm{E}(I)}{\partial \hat{y}} = -\Phi\left(\frac{f_{\min} - \hat{y}}{s}\right) < 0$$

and

$$\frac{\partial \mathrm{E}(I)}{\partial s} = \phi\left(\frac{f_{\min} - \hat{y}}{s}\right) > 0.$$

Thus we see that the expected improvement is larger the lower is $\hat{y}$ and the higher is s. Because of this monotonicity, to find an upper bound on $\mathrm{E}[I(\mathbf{x})]$ over a box for $\mathbf{x}$ it suffices to find a lower bound on $\hat{y}$ and an upper bound on s over the box—call them y^L and s^U —and then compute $\mathrm{E}(I)$ using equation (15) substituting $\hat{y} = y^L$ and $s = s^U$.

Since some readers may not be interested in the details of how these bounds y^L and s^U are constructed, we will defer our discussion of them to Sections 4.3 and 4.4. In the next subsection, we will move ahead and discuss how the exact maximization of expected improvement is used to construct an optimization algorithm called EGO (for Efficient Global Optimization). We will also show results of applying EGO to a few standard test problems.

4.2. The EGO Algorithm

The EGO algorithm begins by fitting a DACE model to a set of initial points specified by what we call a "space-filling" experimental design. The 21 dots in Figure 3(b), for example, are a space-filling design in two dimensions. These space-filling designs are constructed using a special code developed by Welch that searches for a Latin hypercube design in k dimensions that has the property that all the one- and two-dimensional projections are nearly uniformly covered. Based on past experience, we find that about $n = 10k$ points are needed in the initial design. However, in order to have a convenient, finite-decimal value for the spacing between points, we usually deviate slightly from this "$10k$"

rule. For example, in two dimensions we use 21 points so that the inter-point spacing is $1/(n-1) = 1/20 = 0.05$ of the range of each variable. Similarly, in three dimensions we use 33 points and in six dimensions we use 65 points.

After evaluating the function on the initial design, we fit the parameters of a DACE model using maximum likelihood estimation. We then apply the diagnostic tests of Section 3. If the plots look satisfactory (the cross-validated standardized residuals are less than 3 in magnitude, etc.), we say the DACE model is satisfactory. Otherwise, we re-fit the DACE model after applying a log or inverse transformation ($-1/y$) to the dependent variable. If one of these transformations gives satisfactory diagnostic plots, then we use the transformed function in the rest of the analysis. (If no satisfactory transformation can be found, we would not continue with our method, but this does not occur in the examples given below.)

Once the initial DACE surface is fit and any transformation made, we proceeded iteratively. First, we maximize the expected improvement using the branch-and-bound algorithm. If the expected improvement is less than 1% of the best current function value (on the untransformed scale), we stop. Otherwise we sample the function where expected improvement is maximized, re-estimate the DACE parameters with maximum likelihood, and iterate.

We now apply EGO to four test functions from [14]: the Branin function, the Goldstein-Price function, the Hartman 3 function, and the Hartman 6 function. The dimensions of these problems are 2, 2, 3, and 6, respectively.

For all these problems, we fix $p_h = 2$ in the correlation function given in equations (1) and (2). We do this because one of our best bounding methods, discussed later in Section 4.4, is currently limited to this special case. While this bounding technique can be extended to handle the case $p_h < 2$, the extension is more complicated and we have not yet implemented it.

For the Goldstein-Price and Hartman 6 functions, the diagnostic tests suggest that the functions should be log-transformed: $\ln(y)$ for the Goldstein Price function and $-\ln(-y)$ for the Hartman 6 function.

Table I shows the results of applying EGO to the test functions. For each function, we report the number of evaluations when expected improvement is less than 1% of the current best function value, that is, when the stopping criterion is satisfied. We also report the actual relative error on convergence. Note that the algorithm stops when the *estimated* relative accuracy (based on the expected improvement criterion) is 1% or less. The *true* relative accuracy at this point, however, may differ from 1%. To facilitate comparisons with other algorithms,

Table I. Test function results for the EGO algorithm.

Test problem	Evaluations to meet stopping criterion	Actual error when stopped	Evaluations required for 1% accuracy
Branin	28	0.2%	28
Goldstein-Price	32	0.1%	32
Hartman 3	34	1.7%	35
Hartman 6	84	1.9%	121

the last column of Table I shows the number of evaluations that are required to achieve an actual relative error of 1% or better. Note also that all relative errors shown are with respect to the original, untransformed scale. When using a natural log transformation, we stop when expected improvement on the *log scale* is less that 0.01 in *absolute* terms, because this is approximately equal to a 1% relative change on the untransformed scale.

For the Hartman 3 and Hartman 6 functions, the actual relative error on convergence is slightly greater than the target of 1%. There are two reasons why this might happen. First, the standard error of the predictor does not take into account the fact that the correlation parameters (the θ_h's, p_h's, and σ^2) are not known with certainty but rather are estimated. As a result, the standard error may be a bit too small, causing us to underestimate expected improvement.

Second, our expected improvement criterion estimates the gain from sampling *one* additional point. We then find the maximum, over all points, of this expected improvement quantity. What we really want, however, is not the expected improvement at any one point, but rather the expected improvement from sampling all the remaining points, which is obviously larger. Thus we see, once again, that our expected improvement criterion understates the true potential gain from further search.

To compensate for this inevitable underestimation of the potential gain from further search, one can use a lower tolerance than actually desired (e.g., 0.1% instead of 1%) and/or require the convergence tolerance to be met for several iterations in a row. For example, if one insists that the 1% criterion be met two times in a row, then the actual relative errors for the Hartman 3 and Hartman 6 functions become 0.5% and 0.6%, respectively.

To give the reader a feeling for the computation times, on a PC with a Pentium II 266 processor, the first iterate (both maximum likelihood

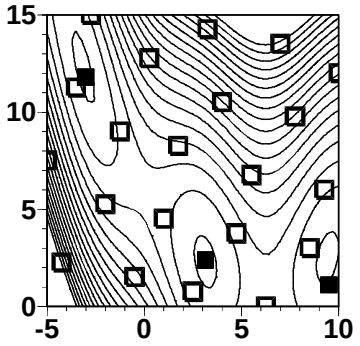

Figure 12. Contours of the Branin function with the initial sample (open squares) and the first three points selected by the expected improvement maximization (filled squares).

and branch and bound) requires 139 seconds for the Branin function, 6 seconds for the Goldstein Price, 40 seconds for the Hartman 3, and 135 seconds for the Hartman 6. In every case, the iterations take longer as the search progresses. For example, computing the last iterate for the Hartman 6 function takes 262 seconds.

For the Hartman 6 problem, the branch-and-bound algorithm is too slow to run to full convergence. Instead, the results shown in Table I are obtained using a "limited memory" branch-and-bound algorithm that only keeps the best 50 nodes in the tree and uses a maximum of 500 iterations. The branch-and-bound solution is then fine-tuned using local search to get our final estimate of where expected improvement is maximized.

Although the limited-memory branch and bound speeds up the search, it also runs the risk of not finding the true maximum of expected improvement. To check the accuracy of this limited memory version, we also ran it on the two- and three-dimensional problems where we could maximize expected improvement to full optimality. In every case we obtained the same results as in the full branch and bound (but with less time). Thus, we have some reason to believe that the limited memory version is an effective expedient for speeding up the EGO algorithm.

Figure 12 shows the first three iterations of EGO on the Branin function. The open squares are the initial sample of 21 points and the solid squares are the first three iterates from the maximization of expected improvement. The contours are those of the true Branin function. With these first three iterates, the EGO algorithm essentially

finds all three global minima. Only four more iterates are required to meet the stopping criterion.

We have not focused on proving convergence theorems for EGO, though we believe that such theorems should be possible. For example, it should be possible to prove that EGO must converge in the limit as the number of function evaluations goes to infinity. Specifically, we believe that the following conjecture is true:

CONJECTURE 1. *Assume that: (i) the parameters θ_h are strictly bounded away from zero, and (ii) the objective function is continuous and defined over a box with finite lower and upper bounds. Then the search points generated by EGO (if run without stopping) will form a dense subset of the feasible region.*

Assumption (i) is needed because, if $\theta_h = 0$, then variable x_h is irrelevant to the predictor, standard error, and expected improvement. Thus, there is no reason for the search to be dense with respect to this variable. Although we have not proved this conjecture, interested readers might consult the work of Marco Locatelli [20] who has proved a similar theorem for a related one-dimensional Bayesian algorithm.

4.3. Bounding the Mean Squared Error via Convex Relaxation

As mentioned earlier, to implement the branch-and-bound algorithm we need a lower bound on $\hat{y}(\mathbf{x})$ and an upper bound on $s(\mathbf{x})$ or, equivalently, on $s^2(\mathbf{x})$. We have explored two ways to compute these bounds. The first is fairly standard in the literature and is based on convex relaxation. The second is unusual because it is based on a nonconvex relaxation. It turns out that the convex relaxation provides the tightest bound for $s^2(\mathbf{x})$, and we will discuss this bound first. The nonconvex relaxation turns out to be better at bounding $\hat{y}(\mathbf{x})$; it is discussed in Section 4.4.

To find an upper bound on $s^2(\mathbf{x})$ over a box defined by $\ell_h \le x_h \le u_h$ for $h = 1, \ldots, k$, we need to find an upper bound on the solution to the following problem:

PROBLEM 1. *Choose* $\mathbf{r} = (r_1, \ldots, r_n)$ *and* $\mathbf{x} = (x_1, \ldots, x_k)$ *to*

$$\begin{aligned} \textit{Maximize} \quad & \sigma^2 \left[1 - \mathbf{r}'\mathbf{R}^{-1}\mathbf{r} + \frac{(1 - \mathbf{1}'\mathbf{R}^{-1}\mathbf{r})^2}{\mathbf{1}'\mathbf{R}^{-1}\mathbf{1}} \right] \\ \textit{subject to} \quad & \ln(r_i) = -\textstyle\sum_{h=1}^{k} \theta_h |x_h - x_h^{(i)}|^{p_h} \qquad (i = 1, \ldots, n) \\ & \ell_h \le x_h \le u_h \qquad (h = 1, \ldots, k). \end{aligned}$$

The reader will recognize the objective function as the formula given in equation (9) for the mean squared error, $s^2(\mathbf{x})$, of the predictor. The first set of constraints forces $\mathbf{r}$ to take on the values defined by the correlation function in equations (1) and (2).

Problem 1 is difficult both because the objective function is potentially nonconvex and because the equality constraints are nonconvex. Before we show how we handle this, it will be convenient to write Problem 1 in an equivalent but more standard form. First, note that the lower and upper bounds on $x_1, \ldots, x_k$ imply, through an interval-analysis calculation, lower and upper bounds on the variables $r_1, \ldots, r_n$. This gives us simple bounds on all the variables. Since we desire an inequality-constraint formulation, we can replace each equality constraint with two inequalities of opposite sense (both $\leq$ and $\geq$). Finally, we can convert the problem to minimization by multiplying the objective function by -1. All this gives the following problem, which is equivalent to Problem 1:

PROBLEM 2. *Choose* $\mathbf{r} = (r_1, \ldots, r_n)$ *and* $\mathbf{x} = (x_1, \ldots, x_k)$ *to*

$$
\begin{aligned}
\textit{Minimize} \quad & -\sigma^2 \left[1 - \mathbf{r}'\mathbf{R}^{-1}\mathbf{r} + \frac{(1 - \mathbf{1}'\mathbf{R}^{-1}\mathbf{r})^2}{\mathbf{1}'\mathbf{R}^{-1}\mathbf{1}} \right] \\
\textit{subject to} \quad & \ln(r_i) + \sum_{h=1}^{k} \theta_h |x_h - x_h^{(i)}|^{p_h} \leq 0 && (i = 1, \ldots, n) \\
& -\ln(r_i) + \sum_{h=1}^{k} \theta_h \left(-|x_h - x_h^{(i)}|^{p_h} \right) \leq 0 && (i = 1, \ldots, n) \\
& \ell_h \leq x_h \leq u_h && (h = 1, \ldots, k) \\
& r_i^L \leq r_i \leq r_i^U && (i = 1, \ldots, n).
\end{aligned}
$$

To deal with the nonconvexity of the objective function, we use the method proposed by Androulakis, Maranas, and Floudas in their αBB algorithm [2]. This method is based on finding the Hessian of the objective function and examining its eigenvalues. Now the Hessian of the objective function in Problem 2 is an $(n+k) \times (n+k)$ matrix since there are $n + k$ variables: $r_1, \ldots, r_n$ and $x_1, \ldots, x_k$. However, since $\mathbf{x}$ does not appear in the objective function, only the upper left $n \times n$ corner of the Hessian is non-zero. This upper-left portion is

$$
2\sigma^2 \left[\mathbf{R}^{-1} - \frac{(\mathbf{R}^{-1}\mathbf{1})(\mathbf{R}^{-1}\mathbf{1})'}{\mathbf{1}'\mathbf{R}^{-1}\mathbf{1}} \right]. \tag{16}
$$

If all the eigenvalues of this upper-left portion of the Hessian were nonnegative, then the objective function would be positive semi-definite and we would have no problem. But what if there is a negative eigenvalue? The key insight of the αBB algorithm is that we can overpower

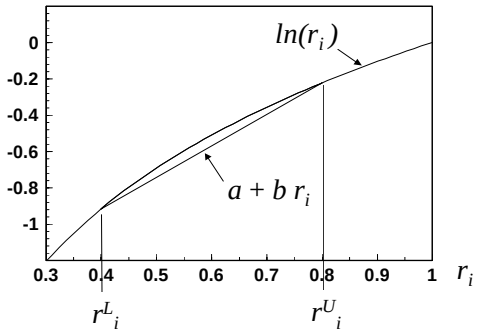

Figure 13. Linear underestimator for the nonlinear term $\ln(r_i)$.

this negative eigenvalue and simultaneously obtain a valid underestimator by adding a specially constructed term to the objective function. In our case, we would modify the objective function to be

$$-\sigma^2 \left[1 - \mathbf{r}'\mathbf{R}^{-1}\mathbf{r} + \frac{(1 - \mathbf{1}'\mathbf{R}^{-1}\mathbf{r})^2}{\mathbf{1}'\mathbf{R}^{-1}\mathbf{1}}\right] + \alpha \sum_i (r_i - r_i^L)(r_i - r_i^U). \quad (17)$$

The additional "α term" adds a constant to the diagonal of the upper-left $n \times n$ corner of the Hessian. Thus, if we set

$$\alpha = \max\left\{0, -\frac{\lambda_{\min}}{2}\right\},$$

where $\lambda_{\min}$ is the minimum eigenvalue of the matrix in equation (16), then the Hessian of the modified objective function will be positive semi-definite. Moreover, since each term $(r_i - r_i^L)(r_i - r_i^U)$ is negative, the modified objective function must be less than or equal to the original one. In short, by replacing the original objective function in Problem 2 by the modified one in (17), we will have relaxed the problem and made the objective function convex.

Unfortunately, this is not enough, because the constraints in Problem 2 are still nonconvex. We eliminate this problem by replacing the nonlinear terms in the constraints with linear underestimators. For example, the nonlinear and nonconvex term $\ln(r_i)$ in the first set of constraints in Problem 2 is replaced by a linear function $a+br_i$ that underestimates it over the interval $[r_i^L, r_i^U]$. This is illustrated in Figure 13.

Similar linear underestimators are used to replace the other nonlinear terms in Problem 2: $|x_h - x_h^{(i)}|^{p_h}$ in the first set of constraints and $-\ln(r_i)$ and $-|x_h - x_h^{(i)}|^{p_h}$ in the second set. Some of these underestimators are chords, like that shown in Figure 13, and some are tangent

lines (we compute the tangents at the midpoint of the interval for x_h or r_i). Because we are underestimating the nonlinear terms, we are making the constraints easier to satisfy, and therefore relaxing the problem. Note that it is not strictly necessary to underestimate all the nonlinear terms, since some of them, such as $-\ln(r_i)$ are actually convex. We use linear underestimators for all the nonlinear terms because this leads to a *linearly* constrained problem that can be solved *much* faster than the relaxation that retains some nonlinearities in the constraints.

When all this is done, we have a relaxation of the original problem with convex objective function, linear constraints, and simple bounds. This relaxed problem can be solved with any good local optimizer. The solution to this relaxed problem, with a sign reversal (to go from minimization back to maximization), is an upper bound to Problem 1 and hence an upper bound on $s^2(\mathbf{x})$.

Having obtained a convex-relaxation bound on $s^2(\mathbf{x})$, the corresponding convex-relaxation bound on $\hat{y}(\mathbf{x})$ is almost trivial. This is true because $\hat{y}(\mathbf{x})$ is a linear function of $\mathbf{r}$:

$$\hat{y}(\mathbf{x}) = \hat{\mu} + \mathbf{c}'\mathbf{r},$$

where

$$\mathbf{c} = \mathbf{R}^{-1}(\mathbf{y} - \mathbf{1}\hat{\mu}).$$

Thus, to find a lower bound on $\hat{y}(\mathbf{x})$, we need merely minimize this linear function subject to the linear constraints and simple bounds just discussed for bounding $s^2(\mathbf{x})$—that is, we need merely solve a LP! In practice, however, this lower bound on $\hat{y}(\mathbf{x})$ is less tight than the bound, based on a nonconvex relaxation, discussed in the next subsection.

4.4. Bounding the Predictor via Nonconvex Relaxation

In the special case when the parameters p_h in the correlation function are equal to 2, it is possible to relax the expression for $\hat{y}(\mathbf{x})$ to a sum of nonconvex, one-dimensional functions that can be easily optimized. When every $p_h = 2$, the correlation function simplifies to:

$$r_i(\mathbf{x}) \equiv \mathrm{Corr}[\epsilon(\mathbf{x}), \epsilon(\mathbf{x}^{(i)})] = \exp\left[-\sum_{h=1}^{k} \theta_h \left(x_h - x_h^{(i)}\right)^2\right].$$

The predictor $\hat{y}(\mathbf{x})$ is just a linear combination of these $r_i(\mathbf{x})$ functions using the constants $\mathbf{c}$ mentioned at the end of the last subsection:

$$\hat{y}(\mathbf{x}) = \hat{\mu} + \sum_{i=1}^{n} c_i r_i(\mathbf{x}).$$

Now let $z_i(\mathbf{x})$ be the quadratic function

$$z_i(\mathbf{x}) \equiv \sum_{h=1}^{k} \theta_h \left(x_h - x_h^{(i)} \right)^2. \tag{18}$$

If $\mathbf{x}$ is restricted to a subregion defined by $\ell_h \leq x_h \leq u_h$, then these bounds on $\mathbf{x}$ will naturally induce lower and upper bounds on $z_i(\mathbf{x})$ via an interval analysis of equation (18). Let us call these bounds z_i^L and z_i^U. Using these $z_i(\mathbf{x})$ functions, let us rewrite the predictor as

$$\hat{y}(\mathbf{x}) = \hat{\mu} + \sum_{i=1}^{n} c_i \exp[-z_i(\mathbf{x})].$$

We now find linear functions $a_i + b_i z_i$ that underestimate the terms $c_i \exp(-z_i)$ over the interval $[z_i^L, z_i^U]$. If $c_i \geq 0$ the linear underestimator will be a tangent to $c_i \exp(-z_i)$ at the midpoint of the interval $[z_i^L, z_i^U]$. If $c_i < 0$ the underestimator will be a chord. If we substitute these linear underestimators into the predictor, we obtain an expression that must always be less than or equal to $\hat{y}(\mathbf{x})$:

$$\begin{aligned}
\hat{y}(\mathbf{x}) &= \hat{\mu} + \sum_{i=1}^{n} c_i r_i(\mathbf{x}) = \hat{\mu} + \sum_{i=1}^{n} c_i \exp\left[-z_i(\mathbf{x})\right] \\
&\geq \hat{\mu} + \sum_{i=1}^{n} [a_i + b_i z_i(\mathbf{x})] \\
&= \hat{\mu} + \sum_{i=1}^{n} \left[a_i + b_i \sum_{h=1}^{k} \theta_h \left(x_h - x_h^{(i)} \right)^2 \right] \\
&= \hat{\mu} + \sum_{i=1}^{n} a_i + \sum_{h=1}^{k} \theta_h \sum_{i=1}^{n} b_i \left(x_h - x_h^{(i)} \right)^2.
\end{aligned} \tag{19}$$

Now note that the underestimator in the final expression is a linear combination of terms

$$\sum_{i=1}^{n} b_i \left(x_h - x_h^{(i)} \right)^2.$$

Furthermore, each term is a quadratic in *one* variable, x_h. A quadratic in one variable can easily be minimized over the interval $[\ell_h, u_h]$, regardless of whether the quadratic is convex or concave. Plugging the x_h's that minimize these quadratics into the last line of (19) gives us a valid lower bound on $\hat{y}(\mathbf{x})$. This bound is usually (but not always) stronger than the bound on $\hat{y}(\mathbf{x})$ based on convex relaxation described

at the end of the previous subsection. Moreover, it is incredibly fast to compute. To save time, therefore, we only use this bound in EGO.

It turns out that we can compute an upper bound on $s^2(\mathbf{x})$ using a similar nonconvex relaxation. However, for reasons we do not fully understand, this "nonconvex" bound on $s^2(\mathbf{x})$ is much weaker than the "convex" one described in Section 4.3, and hence we do not use it in EGO.

5. Visualization

One of the advantages of using response surfaces for optimization is that one not only gets an estimate of the optimum, but also gets a fast approximation to the computer code. This fast approximation (the predictor from the DACE model) can then be used to help identify important factors, visualize the nature of the input-output relationships, and quantify tradeoffs between multiple objectives.

We identify key factors by decomposing the total variation of the predictor into contributions from individual variables and from pairs of variables (interactions). To explain how this is done, we will consider the simplest case in which there are just two variables, both of which are restricted to the unit interval $[0, 1]$. The extension to more than two variables and general bounds will be obvious, but writing down the general case requires long, messy equations that obscure the simplicity of the ideas.

To assess how variable x_1 affects the output, we simply integrate out all the other variables (just x_2 in our simple two-variable case). This gives a function solely of x_1 which we denote by $a_1(x_1)$:

$$a_1(x_1) = \int_0^1 \hat{y}(x_1, x_2)\, dx_2.$$

Intuitively, the average effect $a_1(x_1)$ tells us the "typical value" of the objective function for a given value of x_1, averaging out the variation generated by the different possible values of x_2. It is also known as the "main" effect of x_1. The average or main effect of x_2, denoted $a_2(x_2)$, can be defined analogously.

The average effect of a *pair* of variables is defined by integrating out the remaining $k - 2$ variables. In the simple two-variable case we are considering, there are no other variables to integrate out. Thus, in this simple case, the average effect of variables x_1 and x_2, denoted $a_{12}(x_1, x_2)$, is the same as the predictor $\hat{y}(x_1, x_2)$. We will still write it as $a_{12}(x_1, x_2)$, however, in order to make it easier to see how the formulas generalize.

The final quantity of interest is the overall average of the predictor, denoted a_0. This is found by integrating out *all* variables (here just two):

$$a_0 = \int_0^1 \int_0^1 \hat{y}(x_1, x_2)\, dx_1\, dx_2.$$

Now consider the identity

$$\begin{aligned} \hat{y}(x_1, x_2) &= a_0 \\ &+ \quad [a_1(x_1) - a_0] + [a_2(x_2) - a_0] \\ &+ \quad \{a_{12}(x_1, x_2) - a_0 - [a_1(x_1) - a_0] - [a_2(x_2) - a_0]\}. \end{aligned} \tag{20}$$

This is seen to be true by collecting the coefficients of a_0, a_1, and a_2, all of which turn out to be zero. The only term remaining is $a_{12}(x_1, x_2)$ which, as mentioned above, is equal to $\hat{y}(x_1, x_2)$.

The three lines on the right-hand side of (20) involve terms of increasing complexity. The first line is the part of $\hat{y}(x_1, x_2)$ explained by the overall mean. Having taken this into account, we are left trying to explain $\hat{y}(x_1, x_2) - a_0$. The second line can be interpreted as the portion of this *remainder* explained by the average effect of x_1 and the average effect of x_2. Having taken these further components into account, we are left trying to explain the remainder $\hat{y}(x_1, x_2) - a_0 - [a_1(x_1) - a_0] - [a_2(x_2) - a_0]$, or the third line in (20). This component represents the non-additivity or interaction effect.

Now let us rewrite equation (20) as

$$\begin{aligned} \hat{y}(x_1, x_2) - a_0 = & \\ & [a_1(x_1) - a_0] + [a_2(x_2) - a_0] \\ + \quad & \{a_{12}(x_1, x_2) - a_0 - [a_1(x_1) - a_0] - [a_2(x_2) - a_0]\}. \end{aligned} \tag{21}$$

If we square and integrate the left-hand side, we get, by definition, the total variance of the predictor:

$$\text{Total variance} = \int_0^1 \int_0^1 [\hat{y}(x_1, x_2) - a_0]^2\, dx_1\, dx_2.$$

Squaring and integrating the right-hand side generates a number of terms. One of them will be

$$\int_0^1 \int_0^1 [a_1(x_1) - a_0]^2\, dx_1\, dx_2 = \int_0^1 [a_1(x_1) - a_0]^2\, dx_1.$$

This term can be interpreted as the component of the the total variance of the predictor explained just by variable x_1. Similarly, we have the term

$$\int_0^1 \int_0^1 [a_2(x_2) - a_0]^2\, dx_1\, dx_2 = \int_0^1 [a_2(x_2) - a_0]^2\, dx_2,$$

which is the component of variance explained just by variable x_2. Finally, we have the term

$$\int_0^1 \int_0^1 \{a_{12}(x_1, x_2) - a_0 - [a_1(x_1) - a_0] - [a_2(x_2) - a_0]\}^2 \, dx_1 \, dx_2. \tag{22}$$

This can be interpreted as the variance explained by the interaction of x_1 and x_2.

The reader will probably have noticed that we have ignored the "cross-product terms" that are created when we square and integrate the right-hand side of (21). All of these cross-product terms, however, reduce to zero. For example,

$$\begin{aligned} & \int_0^1 \int_0^1 [a_1(x_1) - a_0][a_2(x_2) - a_0] \, dx_1 \, dx_2 \\ = & \int_0^1 [a_1(x_1) - a_0] \, dx_1 \int_0^1 [a_2(x_2) - a_0] \, dx_2 \\ = & [a_0 - a_0][a_0 - a_0] = 0. \end{aligned}$$

Thus we see that the total variance in the predictor can be decomposed into components due to main effects and interactions. By dividing by the total variance, we can express these components as percentages. When there are only two variables, the percentages for the main effects and two-variable interactions will add to 100%. When there are more than two variables, these percentages will generally add up to less than 100%, the remainder being due to interactions of higher degree.

Computing this variance decomposition requires us to compute many integrals, some of which seem, on the surface, to be multidimensional integrals [e.g., equation (22)]. However, if all the variables have simple upper and lower bounds, so that the feasible region is a hyperbox, and if we use the correlation function shown in equations (1) and (2), then it turns out that all these multidimensional integrals reduce to one-dimensional integrals. The reason is that all the integrands are sums of products of one-dimensional functions in the x_h's, and hence the integral is a sum of products of the one-dimensional integrals (as long as the limits of integration do not depend upon the variables). Thus, computing the variance decomposition is usually not a problem.

To illustrate these methods, we will discuss an application in which DACE models were used to investigate how the performance of an integrated circuit depended upon 36 input variables [3]. The integrated circuit was modeled with a computer program that had a running time of a few minutes. There were several performance measures for the circuit, but we will consider just one called VCH, a voltage spike that

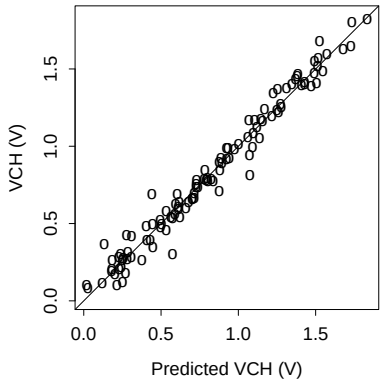

Figure 14. Actual VCH values versus cross-validated predictions in the integrated-circuit application.

was to be minimized. (We use the same variable names as in the detailed paper [3] to facilitate cross referencing.)

The initial experimental design had 120 runs. This was less than would have been suggested by our "10 times the number of variables" rule, but 120 runs seemed sufficient because it was thought that many variables would be relatively unimportant. Figure 14 plots the actual VCH values versus their cross-validated predictions and shows good agreement. The other diagnostic plots (not shown) suggested that the standard error of prediction was also realistic.

The variance decomposition showed that the main effects of two variables, called T2 and N2, accounted for 19.1% and 42.8% of the total variability of the VCH predictor. The interaction between T2 and N2 accounted for another 4.5% of the variation. Thus, these two variables alone contributed $19.1 + 42.8 + 4.5 = 66.4\%$ of the total variation in the VCH predictor from all 36 input variables. Further examination of the analysis of variance showed that only about five input variables were important in predicting VCH. This application illustrates that, even with a high-dimensional engineering problem, it may be feasible to focus attention on a fairly small number of variables in optimization.

Figure 15 shows the main effects of variables T2 and N2 [i.e., the analogs of the functions $a_h(x_h)$ in our previous notation]. In addition to the main effects, Figure 15 also shows pointwise 95% confidence intervals. It would take us too long to discuss the derivation of these confidence intervals. Intuitively, they reflect the fact that the predictor we are averaging when we compute a main effect is only an estimate of the actual function's value. The true function could be a little higher or lower, so if we averaged over the true function we might get a slightly

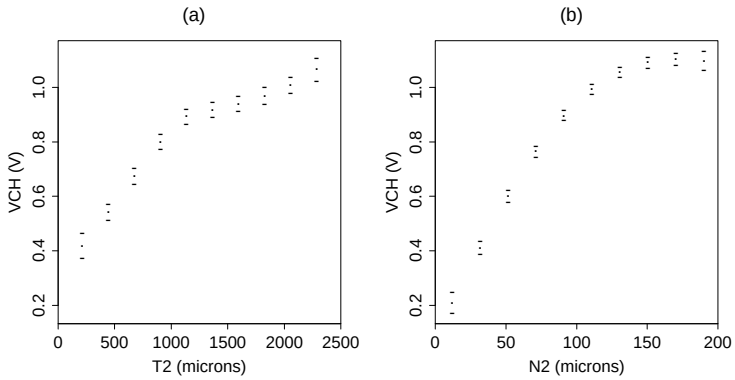

Figure 15. Estimated main effects of variables T2 and N2 on VCH in the integrated-circuit application.

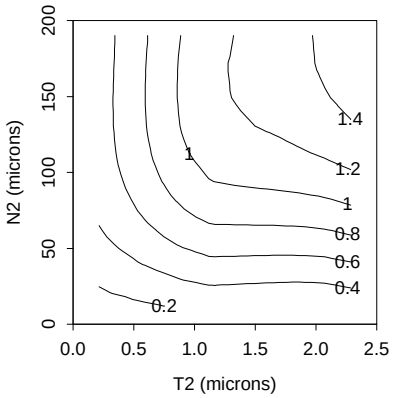

Figure 16. Estimated joint effect of N2 and T2 on VCH in the integrated-circuit application.

different result. If the model is working well then, at any given point, the "true" average effect—that is, the one computed with the true function and not the predictor—would lie in the interval with 95% confidence. The main effect plots show the nonlinearity of the relationship between VCH and variables T2 and N2.

Figure 16 shows a contour plot of the estimated joint effect of T2 and N2 on VCH [the analog of $a_{12}(x_1, x_2)$ in our earlier notation]. Note that, for low values of N2, the VCH contours are almost horizontal. This means that, when the value of N2 is small, increasing T2 has little effect on VCH. On the other hand, for high values of N2, the VCH contours are steeper. This implies that increasing T2 has a major effect on VCH. Thus we see that the effect of T2 depends upon the level of N2; that is, the variables interact.

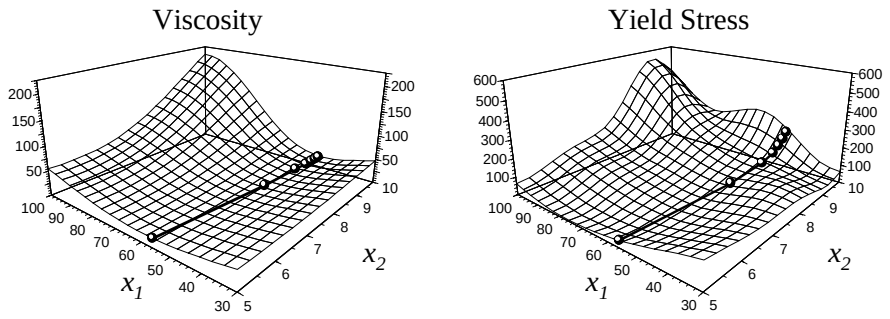

Figure 17. DACE surfaces for viscosity and yield stress in the automotive example.

As an aside, we might mention that one can also compute the standard error of the estimated joint effect. In this case, the standard error turns out to be small, suggesting that the estimated interaction in Figure 16 is indeed real.

One final example will illustrate the use of DACE models to identify and quantify tradeoffs. This example is based on an automotive application which is largely proprietary. Suffice it to say that we were interested in optimizing the properties of a material by adjusting the quantities of two key ingredients, x_1 and x_2. Two performance measures were of interest: viscosity and yield stress. Everything else equal, it was desirable to have lower viscosity and higher yield stress. Unfortunately, there was a tradeoff between these two performance measures. That is, material compositions that did well on viscosity were different than those that did well on yield stress. The goal of the analysis was to find the material compositions that were "efficient" in the sense that one could not get lower viscosity without sacrificing (i.e., lowering) yield stress.

DACE models were fit to data based on the same 21-point design used earlier on the two-dimensional test functions. The resulting surfaces for viscosity and yield stress are shown in Figure 17 . Using these surfaces as surrogates for the true functions, we could then solve the problem "choose x_1 and x_2 to minimize viscosity subject to yield stress being greater than or equal to C." The solution to this minimization problem is necessarily an efficient point, since it is not possible to do better on viscosity without relaxing the constraint, that is, without allowing worse yield strength values. By solving this optimization problem for many values of C, we were able to locate the efficient combinations of x_1 and x_2. This efficient set is the solid line in Figure 17. Furthermore, we can directly visualize the tradeoff by plotting the combinations of viscosity and yield stress that we obtain from the

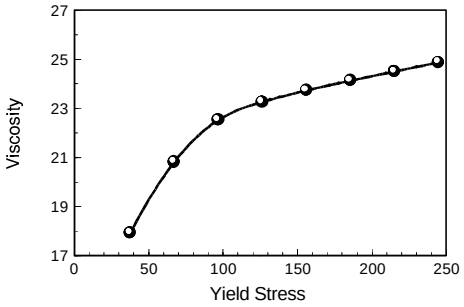

Figure 18. The tradeoff between viscosity and yield stress in the automotive example.

minimization problems. This is shown in Figure 18. As a result of this tradeoff analysis, the engineers discovered an attractive region in the space of x_1 and x_2 that had not previously been known. This allowed new materials to be made that outperformed the previous best material on both viscosity and yield stress.

6. Discussion

One of the challenges we had to face in implementing EGO was the ill-conditioning of the correlation matrix $\mathbf{R}$. Theoretically, the correlation matrix $\mathbf{R}$ is guaranteed to be positive semi-definite because the DACE correlation function is "completely monotone" (see Cressie [12], p. 86). There are two reasons, however, why the correlation matrix can be nearly singular. First, if the function is very smooth and predictable, the points in the sample will be highly correlated, which implies that each column in the correlation matrix will be almost a column of ones. As a result, the columns will be highly collinear. Second, towards the end of an optimization run, the algorithm will tend to sample points near previously sampled points. Unfortunately, when two sampled points are close together, the corresponding two columns in $\mathbf{R}$ are nearly identical, making $\mathbf{R}$ nearly singular.

This ill-conditioning is usually manageable if one is only computing the predictor and its standard error, but it creates difficulties in the bounding calculations. To date, we have addressed these difficulties by using the singular value decomposition of $\mathbf{R}$ and zeroing small singular values, as suggested in the book *Numerical Recipes* [28].

Another challenge is to speed up the branch-and-bound algorithm. The time required by the algorithm depends crucially on the tightness of the upper bound for the mean squared error of the predictor, $s^2(\mathbf{x})$.

Fortunately, we have an idea that might lead to a better bound. This idea is based on computing an upper bound on $s^2(\mathbf{x})$ via a max-min problem. We describe this idea further in Appendix 2 in hopes that some reader might be able to help us work out the details. Another desirable extension of EGO would be to handle nonlinear constraints. We have already implemented one way to do this with some success [32], but there are other approaches that might be better and need to be tried.

Since some finite-element codes naturally and cheaply produce gradients in addition to function values, it would also be desirable to extend EGO to be able to use such gradients. As it turns out, the DACE predictor is easily extended to handle derivatives [25]. When this is done, the predictor not only interpolates the data but also agrees with the gradient at each sampled point. As a result, it is much more accurate.

Although we have not discussed it here, the DACE methodology can also handle data collected using physical experiments where there is both signal and noise [17]. With noisy data, the use of DACE for visualization and what-if analyses is virtually unchanged from the deterministic case. Unfortunately, it is not immediately clear how to extend our *optimization algorithm* to the case of noisy functions. With noisy data, we really want to find the point where the *signal* is optimized. Similarly, our expected improvement criterion should be defined in terms of the signal component. Since we can not observe the signal directly (it is always masked by noise), it is less obvious how to proceed. Nevertheless, we believe that an extension to noisy data is possible, and this will be a focus for us in future work.

Most engineering systems can be approximated with models of varying degrees of accuracy or "fidelity." Everything else equal, it would be desirable to work with the most accurate model. But highly accurate models may take days to compute, making optimization difficult for almost any algorithm. On the other hand, switching to a very fast, low-fidelity model may entail an intolerable loss of accuracy. Rather than searching for the optimal intermediate degree of accuracy, the best approach is probably to use *both* low- and high-fidelity models. We believe that this is an extremely promising area for further research, and so we will digress for a moment to describe some of our ideas on this subject as well as those of other researchers.

In Figure 19(a) we show (hypothetical) low- and high-fidelity models of how system performance, y, depends on a design parameter, x. As shown in Figure 19(b), the high-fidelity model has been run at just four points (dots). In contrast, the low-fidelity model has been run not only at these four points but also at many others (squares). This reflects

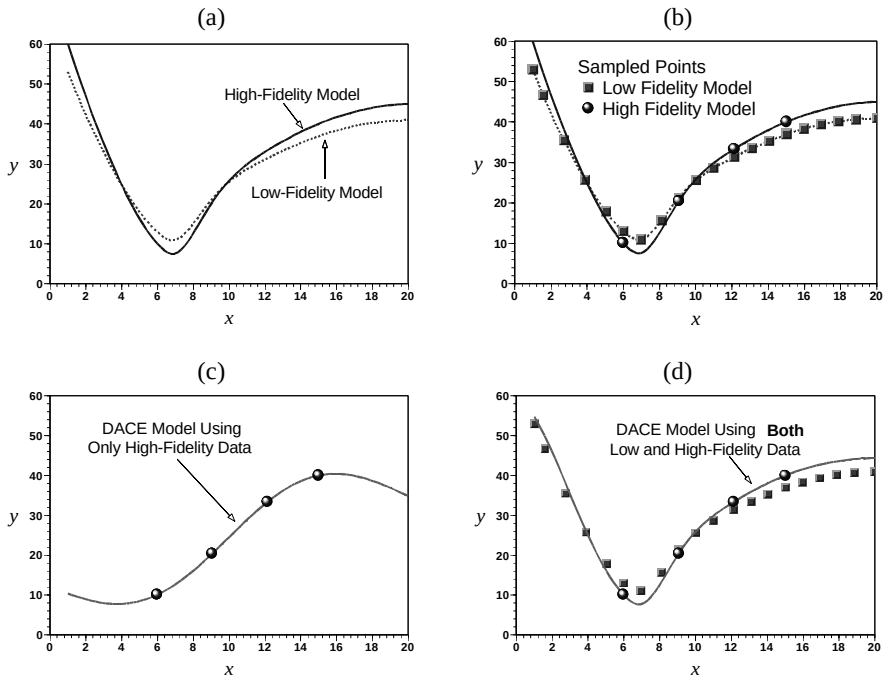

Figure 19. Exploiting engineering models of both high and low fidelity.

our assumption that the low-fidelity model is much cheaper to run. If we fit a DACE model using only the four high-fidelity observations, we would obtain the predictor shown in Figure 19(c). This predictor is fairly inaccurate, since it entirely misses the sharp increase in y as x approaches zero. This is understandable, because there are no high-fidelity observations for low values of x.

In contrast, Figure 19(d) shows a DACE predictor computed using both the high and low-fidelity observations. Intuitively, this predictor uses the low-fidelity observations to suggest the best way to interpolate the high-fidelity observations. Formally, the predictor in Figure 19(d) was constructed by using the output of the low-fidelity model as if it were another input variable. There are still only four observations on the high-fidelity model, but now we have two explanatory variables (x and the output of the low-fidelity model). To make a prediction at a new point, we first run the low-fidelity model to obtain the second "input variable" and then compute the DACE predictor as usual.

The approach illustrated in Figure 19 is just one way to combine low- and high-fidelity models. An even more appealing approach, from the conceptual point of view, would be to treat the outputs of the low- and high-fidelity models as correlated dependent variables in a multivariate

DACE model. In the mathematical geology literature, this approach is known as "co-kriging" ([12], [35]).

As mentioned earlier, other researchers are working on ways to do optimization using both low- and high-fidelity models. The ideas behind these other approaches, however, are quite different from those just discussed. For example, Eby et al. [15] run several genetic algorithms on the low-fidelity model and use the resulting solutions to "seed" the populations of other genetic algorithms run on the high-fidelity model. In essence, they are using the low-fidelity model to suggest promising areas of the search space for further analysis with the high-fidelity model. Alexandrov et al. [1] developed a variation of the classical trust region method in which search steps are based on the low-fidelity model instead of the usual local quadratic approximation. Because the low-fidelity model is presumably more accurate than the local quadratic model, their method should be able to take larger steps and converge in fewer iterations.

To sum up, we have discussed the value of using a special kind of response surface—the DACE model—for optimizing expensive black-box functions and for visualizing what is going on in a complicated model. The use of DACE for visualization is fairly mature. In fact, it has been used on quite large problems, such as the 36-variable integrated-circuit problem in Section 5. However, in the past, the use of DACE models for optimization has been ad hoc, based on manual direction of the search. We have nothing against looking at the data and working interactively. Quite the contrary, we have spent quite a few years promoting the value of gaining insight into a model through visualization. But while engineers appreciate visualization for generating understanding, at the same time they would welcome a fairly automatic method for optimization. The results here suggest that such an automatic procedure is already in hand for many unconstrained, low-dimensional problems. The challenge is to push this approach further.

Appendix 1

This appendix provides an alternative derivation of the best linear unbiased predictor given in equation (7). Suppose we want to predict the function's value at some point $\mathbf{x}^*$. Let y^* denote some candidate prediction. To assess the goodness of this candidate prediction, we add the "pseudo observation" $(\mathbf{x}^*, y^*)$ to the data as point $n+1$ and compute the likelihood of the augmented sample. This likelihood measures how well the pseudo observation "fits" with the original data; that is, the likelihood that they were all generated from the same model. We will

now show that, if one computes the value of y^* that maximizes the likelihood, then one gets the best linear unbiased predictor in equation (7).

Let $\tilde{\mathbf{y}} = (\mathbf{y}'\, y^*)'$ denote the vector of observations when augmented by the new pseudo observation and let $\tilde{\mathbf{R}}$ denote the correlation matrix with the pseudo observation. If we let $\mathbf{r}$ denote the vector of correlations of the error at $\mathbf{x}^*$ with the errors at the original n data points, then the correlation matrix is:

$$\tilde{\mathbf{R}} = \begin{pmatrix} \mathbf{R} & \mathbf{r} \\ \mathbf{r}' & 1 \end{pmatrix}.$$

Recalling the formula for the likelihood function in equation (4), it is clear that the only part of the augmented likelihood function that depends upon y^* is

$$(\tilde{\mathbf{y}} - \mathbf{1}\hat{\mu})'\tilde{\mathbf{R}}^{-1}(\tilde{\mathbf{y}} - \mathbf{1}\hat{\mu}) = \begin{pmatrix} \mathbf{y} - \mathbf{1}\hat{\mu} \\ y^* - \hat{\mu} \end{pmatrix}' \begin{pmatrix} \mathbf{R} & \mathbf{r} \\ \mathbf{r}' & 1 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{y} - \mathbf{1}\hat{\mu} \\ y^* - \hat{\mu} \end{pmatrix}.$$

By using the partitioned inverse formula (e.g., [34], p. 18), we can get an explicit expression for $\tilde{\mathbf{R}}^{-1}$, which then allows us to write the right-hand side of the above as:

$$\frac{1}{1 - \mathbf{r}'\mathbf{R}^{-1}\mathbf{r}}(y^* - \hat{\mu})^2 - \frac{2\mathbf{r}'\mathbf{R}^{-1}(\mathbf{y} - \mathbf{1}\hat{\mu})}{1 - \mathbf{r}'\mathbf{R}^{-1}\mathbf{r}}(y^* - \hat{\mu}) + \text{terms without } y^*.$$

The y^* that maximizes the augmented likelihood is then found by taking the derivative of the above expression and setting it equal to zero:

$$\frac{2}{1 - \mathbf{r}'\mathbf{R}^{-1}\mathbf{r}}(y^* - \hat{\mu}) + \frac{2\mathbf{r}'\mathbf{R}^{-1}(\mathbf{y} - \mathbf{1}\hat{\mu})}{1 - \mathbf{r}'\mathbf{R}^{-1}\mathbf{r}} = 0.$$

Solving for y^* then gives us the standard formula for the best linear unbiased predictor in (7).

Appendix 2

In the standard derivation of the best linear unbiased predictor (which we have not shown), it turns out that $s^2(\mathbf{x})$ is the optimal objective value from the following minimization problem:

$$\begin{array}{ll} \text{Choose } \mathbf{c} \text{ to minimize:} & \sigma^2(\mathbf{c}'\mathbf{R}\mathbf{c} - 2\mathbf{r}'\mathbf{c} + 1) \\ \text{subject to:} & \mathbf{c}'\mathbf{1} = 1. \end{array}$$

It follows that the maximum of $s^2(\mathbf{x})$ over a box defined by $\ell_h \leq x_h \leq u_h$ for $h = 1, \ldots, k$, is the solution to the following max-min

problem:

$$\begin{array}{ll} \max\limits_{\mathbf{x},\mathbf{r}} \min\limits_{\mathbf{c}} & \sigma^2(\mathbf{c}'\mathbf{R}\mathbf{c} - 2\mathbf{r}'\mathbf{c} + 1) \\ \text{subject to:} & \mathbf{c}'\mathbf{1} = 1 \\ & \ln(r_i) = -\sum\limits_{h=1}^{k} \theta_h |x_h - x_h^{(i)}|^{p_h} \quad (i = 1, \ldots, n) \\ & \ell_h \le x_h \le u_h \qquad\qquad\qquad\qquad (h = 1, \ldots, k). \end{array}$$

The second constraint ensures that $\mathbf{r}$ is related to $\mathbf{x}$ according to the DACE correlation function. It is conceivable that one could find a way to relax the max-min problem to a form that can be easily solved, and thereby compute a valid upper bound on $s^2(\mathbf{x})$. The appeal of this approach is that the inverse of $\mathbf{R}$—and all the associated numerical difficulties—does not appear anywhere in the formulation.

Acknowledgements

This program of research was stimulated by discussions with Jerry Sacks. Michael Powell provided us with a great deal of quality software. We used his nonsmooth optimization algorithm COBYLA to maximize the likelihood function in the EGO examples and used his gradient-based TOLMIN algorithm to solve the nonlinear lower bounding problem of Section 4.3. We also used Powell's code for computing thin-plate splines to produce the surfaces shown in Figures 3 and 4. Chris Floudas showed us how to form the convex relaxation of Problem 1 in Section 4.3. Natalia Alexandrov, Chris Floudas, and Andrew Booker all made helpful comments on an earlier draft of this paper.

References

1. N.M. Alexandrov, J.E. Dennis, Jr., R.M. Lewis, and V. Torczon. A trust-region framework for managing the use of approximation models in optimization. *Structural Optimization*, 15, 16–23, 1998.
2. I.P. Androulakis, C.D. Maranas, and C.A. Floudas. αBB: a global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7, 337–363, 1995.
3. R. Aslett, R.J. Buck, S.G. Duvall, J. Sacks, and W.J. Welch. Circuit optimization via sequential computer experiments: design of an output buffer. *Applied Statistics*, 47, 31-48, 1998.
4. B. Betro. Bayesian methods in global optimization. *Journal of Global Optimization*, 1, 1–14, 1991.
5. C.G.E. Boender and H.E. Romeijn. Stochastic methods. In R. Horst and P.M. Pardalos, editors, *Handbook of Global Optimization*, 829–869. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1995.

6. A.J. Booker, A.R. Conn, J.E. Dennis, P.D. Frank, M. Trossett, and V. Torczon. Global modeling for optimization. Boeing Information and Support Services, Technical Report ISSTECH-95-032, December 1995.
7. A.J. Booker, J.E. Dennis, P.D. Frank, D.B. Serafini, and V. Torczon. Optimization using surrogate objectives on a helicopter test example. Boeing Shared Services Group, Technical Report SSGTECH-97-027, December 1997.
8. G.E.P. Box, W.G. Hunter, and J.S. Hunter. *Statistics for Experimenters*. John Wiley, New York, 1978.
9. D.D. Cox and S. John. SDO: A statistical method for global optimization. In N. Alexandrov, and M.Y. Hussaini, editors, *Multidisciplinary Design Optimization: State of the Art*, 315–329. SIAM, Philadelphia, 1997.
10. N. Cressie. Geostatistics. *The American Statistician*, 43, 197–202, 1989. See also the comment on this article by G. Wahba, and the reply by N. Cressie, in *The American Statistician*, 44, 255–258, 1990.
11. N. Cressie. The origins of kriging. *Mathematical Geology*, 22, 239–252, 1990.
12. N. Cressie. *Statistics for Spatial Data*. John Wiley, New York, 1993.
13. C. Currin, T. Mitchell, M. Morris, and D. Ylvisaker. Bayesian prediction of deterministic functions, with applications to the design and analysis of computer experiments. *Journal of the American Statistical Association*, 86, 953–963, 1991.
14. L.C.W. Dixon and G.P. Szego. The global optimisation problem: an introduction. In L.C.W. Dixon and G.P. Szego, editors, *Towards Global Optimisation 2*, 1–15. North Holland, New York, 1978.
15. D. Eby, R.C. Averill, W.F. Punch III, and E.D. Goodman. Evaluation of injection island GA performance on flywheel design optimization. In I.C. Parmee, editor, *Adaptive Computing in Design and Manufacture*, Springer Verlag, 1998.
16. J.F. Elder IV. Global R^d optimization when probes are expensive: the GROPE algorithm. *Proceedings of the 1992 IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 1, 577–582, Chicago, 1992.
17. F. Gao, J. Sacks, and W.J. Welch. Predicting urban ozone levels and trends with semiparametric modeling. *Journal of Agricultural, Biological, and Environmental Statistics*, 1, 404–425, 1996.
18. J. Koehler and A. Owen. Computer experiments. In S. Ghosh and C.R. Rao, editors, *Handbook of Statistics, 13: Design and Analysis of Experiments*, 261–308. North Holland, New York, 1996.
19. H.J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86, 97–106, 1964.
20. M. Locatelli. Bayesian algorithms for one-dimensional global optimization. *Journal of Global Optimization*, 10, 57–76, 1997.
21. G. Matheron. Principles of geostatistics. *Economic Geology*, 58, 1246–1266, 1963.
22. M.D. McKay, W.J. Conover, and R.J. Beckman. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21, 239–245, 1979.
23. J. Mockus. Application of Bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization*, 4, 347–365, 1994.
24. J. Mockus, V. Tiesis, and A. Zilinskas. The application of Bayesian methods for seeking the extremum. In L.C.W. Dixon and G.P. Szego, editors, *Towards Global Optimisation 2*, 117–129. North Holland, New York, 1978.

25. M.D. Morris, T.J. Mitchell, and D. Ylvisaker. Bayesian design and analysis of computer experiments: use of derivatives in surface prediction. *Technometrics*, 35, 243–255, 1993.
26. E. Parzen. A new approach to the synthesis of optimal smoothing and prediction systems. In R. Bellman, editor, *Mathematical Optimization Techniques*, 75–108. University of California Press, Berkley, 1963.
27. C. Perttunen. A computational geometric approach to feasible region division in constrained global optimization. *Proceedings of the 1991 IEEE Conference on Systems, Man, and Cybernetics*, 1991.
28. W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in Fortran 77*. Cambridge University Press, 1992.
29. J. Sacks, W. J. Welch, T.J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments (with discussion). *Statistical Science*, 4, 409–435, 1989.
30. J. Sacks and D. Ylvisaker. Statistical designs and integral approximation. In R. Pyke, editor, *Proceedings of the Twelfth Biennial Seminar of the Canadian Mathematical Congress*, 115–136. Canadian Mathematical Congress, Montreal, 1970.
31. M. Schonlau. Computer experiments and global optimization. Ph.D. Thesis, University of Waterloo, Waterloo, Ontario, Canada, 1997.
32. M. Schonlau, W.J. Welch, and D.R. Jones. Global versus local search in constrained optimization of computer models. To appear in N. Flournoy, W.F. Rosenberger, and W.K. Wong, editors, *New Developments and Applications in Experimental Design*, Institute of Mathematical Statistics. Also available as Technical Report RR-97-11, Institute for Improvement in Quality and Productivity, University of Waterloo, Waterloo, Ontario, CANADA, December 1997.
33. B.E. Stuckman. A global search method for optimizing nonlinear systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 18, 965–977, 1988.
34. H. Theil. *Principles of Econometrics*. John Wiley, New York, 1971.
35. J.M. Ver Hoef and N. Cressie. Multivariate spatial prediction. *Mathematical Geology*, 25, 219–240, 1993.
36. W.J. Welch, R.J. Buck, J. Sacks, H.P. Wynn, T.J. Mitchell, and M.D. Morris. Screening, predicting, and computer experiments. *Technometrics*, 34, 15–25, 1992.
37. A. Zilinskas. A review of statistical models for global optimization. *Journal of Global Optimization*, 2, 145–153, 1992.