Computer Experiments 20

20.1 Introduction

All of the experiments described in the previous chapters are *physical experiments* where the experimenter can work directly with the treatments and experimental material. The experimenter makes an assignment of experimental units to the levels of the treatment factors, possibly including blocks, and then measures the corresponding responses. Since it is not possible to control every single variable that influences the scientific process under consideration, all measurements necessarily include random variability. Statisticians have developed numerous methods to mitigate the effects of uncontrolled variation on the study conclusions. For example, Chap. 1 introduced replication, blocking, and randomization. These standard techniques of physical experiments allow an experimenter to increase precision and decrease bias.

In some situations, however, it is economically, ethically, or temporally not possible to run a statistically appropriate physical experiment. Instead, the following scenario might be feasible. Suppose that

(i) the physical process can be described by a mathematical model (for example, a system of differential equations),
(ii) computer code (called a *simulator*) can be written to compute the response from the mathematical model, and
(iii) computational methods exist for working with this model (for example, solving differential equations) within a reasonable timeframe.

In this scenario, a researcher can conduct a *computer experiment* by running the computer code, which serves as a proxy for the physical process, to compute a "response" at any combination of values of the treatment factors, which are now called *input combinations*, or *input points*, or sometimes shortened to *inputs*. We use $\boldsymbol{x}$ to represent an input combination.

Example 20.1.1 `A Real Experiment—Prosthetic Elbow Experiment`

Many biomedical engineers use computer experiments to help with the engineering design of prosthetic devices. For example, Hayeck (2009) studied the effects of four implant position variables on the functioning of a total elbow replacement prosthetic device. One of the responses of interest was a measure of principal compressive strain in the bone.

A. Dean et al., *Design and Analysis of Experiments*,
Springer Texts in Statistics, DOI 10.1007/978-3-319-52250-0_20

The input variables were the tip displacement (x_1), the rotation of the implant axis about the lateral axis at the tip (x_2), the rotation of the implant axis about the anterior axis at the tip (x_3), and the rotation about the implant axis (x_4). The variable x_1 was measured in millimeters, while x_2, x_3, and x_4 were measured in degrees. To conduct an experiment, one would need to observe the response y at various combinations $\boldsymbol{x} = (x_1, \ldots, x_4)$ of the input variables. It would be ethically questionable to carry out this experiment on patients, since some input combinations of interest may prove harmful. However, as described by Hayeck (2009), the influence of these four variables on the functioning of the prosthetic device could be described by a mathematical model and could be implemented in computer code, and so a computer experiment could be conducted. □

A key difference between computer experiments and physical experiments is that many computer experiments are "deterministic". In other words, if an experimenter runs the computer code twice at the same input combination $\boldsymbol{x}$, the same response y will be observed both times. A deterministic computer code has no random variability, and may exhibit an unknown, and/or highly complex, functional relationship, $y = f(\boldsymbol{x})$. The lack of variability eliminates any benefit of replication and, further, since *all* the input variables to the code are known and can be controlled, randomization and blocking are likewise unnecessary.

Some deterministic computer codes are very computationally intensive—perhaps requiring hours or days for a computer to solve the underlying mathematical model to produce each response value—so time constraints limit the number of runs or observations that can be taken. Given outputs $y_1, y_2, \ldots, y_n$ from a limited number of runs of the computer code (taken at input combinations $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, respectively), a goal of a computer experiment is to fit a model that can generate good response predictions relatively rapidly throughout the set of possible input combinations. The set of possible input combinations is called the *input space* or *design space*. If the fitted model, $\hat{f}(\mathbf{x})$ say, provides a good approximation to the computer code function $f(\boldsymbol{x})$ throughout the design space, and if $\hat{f}(\boldsymbol{x})$ can be computed quickly as compared to running the cumbersome computer code, then the fitted model $\hat{f}(\boldsymbol{x})$ can be used to investigate the behavior of the computer function $f(\boldsymbol{x})$ over the input space. The fitted model is called an *emulator* since it "emulates" the output of the computer code.

In the prosthetic elbow experiment in Example 20.1.1, it would be possible to use the fitted model $\hat{f}(\boldsymbol{x})$ to find a region of inputs $\boldsymbol{x}$ that yield relatively low estimates $\hat{f}(\boldsymbol{x})$ of principal compressive strains in the bone, and finding this good region with respect to $\hat{f}(\boldsymbol{x})$ may be the end of the study. That said, having pinned down this good region of inputs $\boldsymbol{x}$ with respect to $\hat{f}(\boldsymbol{x})$, but knowing that $\hat{f}(\boldsymbol{x})$ merely approximates the computer code function $f(\boldsymbol{x})$, one might make some additional runs of the computer code using input combinations $\boldsymbol{x}$ in or around this good region. With this additional data, a refitted model $\hat{f}(\boldsymbol{x})$ should better approximate the computer code function $f(\boldsymbol{x})$ in the previously identified good region of inputs, allowing one to re-examine the effects of the inputs in this region. Better yet, since the computer code function $y = f(\boldsymbol{x})$ is only a proxy for the true physical situation, having pinned down a good region of inputs with respect to either $f(\boldsymbol{x})$ or $\hat{f}(\boldsymbol{x})$, it may then be feasible, both morally and practically, to run a physical experiment with variable settings $\boldsymbol{x}$ in this good region to study the effects of the input variables on principal compressive strains in the bone directly. This would be done using methods of physical experiments presented in earlier chapters.

For computer experiments, statisticians have developed highly flexible statistical models to emulate simulator output. These are typically smooth "interpolators", in the sense that they provide a fitted model $\hat{f}(\boldsymbol{x})$ which is relatively smooth (no sudden jumps) and, since the observed data are deterministic, $\hat{f}(\boldsymbol{x})$ passes through, or interpolates, the n observed data points $(\boldsymbol{x}_i, y_i)$; in other words, $\hat{f}(\boldsymbol{x}_i) = f(\boldsymbol{x}_i)$ for the n observed input combinations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.

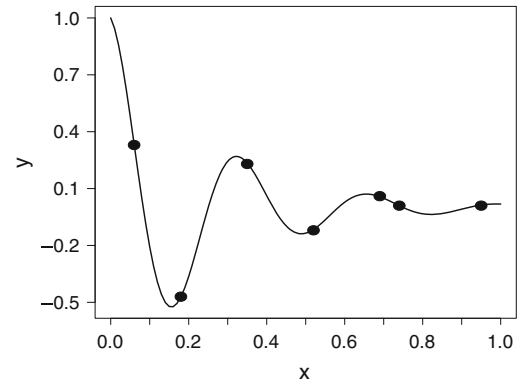


Fig. 20.1 True unknown relationship between x and y, including seven observed points

20.2 Models for Computer Experiments

Since the precise functional relationship $y = f(\boldsymbol{x})$ between the inputs and the response in a deterministic computer experiment is usually unknown a priori, and is potentially highly non-linear, we require the model for the outputs to be very flexible as well as to interpolate the data. While the complete treatment of the technical details behind the statistical models of computer experiment data is beyond the scope of this book, the intuition behind them is quite appealing and is explained in the following example.

Example 20.2.1 `One Predictor`

Suppose $n = 7$ runs of a computer code yield the following data points, written as the pairs (x_i, y_i), $i = 1, \ldots, 7$: $(0.06, 0.33)$, $(0.18, -0.47)$, $(0.35, 0.23)$, $(0.52, -0.12)$, $(0.69, 0.06)$, $(0.74, 0.01)$, $(0.95, 0.01)$. These seven points are plotted in Fig. 20.1 as dark circles. While the relationship $y = f(x)$ intrinsic in the computer code would be unknown to us, for sake of illustration, let it be $y = e^{-4x}\cos(6\pi x)$. The solid curve in Fig. 20.1 represents this true but unknown relationship between x and y, and we want to estimate it based on the seven data points. Suppose we could use the seven data points to somehow obtain a fitted model (emulator), $\hat{y} = \hat{f}(x)$, which we could compute for any value of x in the input space. Then to "estimate" this unknown relationship $y = f(x)$, we could use the fitted model to make predictions $\hat{f}(x)$ at a grid of x values, such as $x = 0, 0.01, 0.02, \ldots, 1$. Plotting these points $(x, \hat{f}(x))$ would approximate the unknown solid curve.

To fit a model with the desired characteristics, consider how we might predict y at any unobserved input point x. We want the fitted model to be an interpolator—namely, for any of the observed points (x, y), we want the model prediction $\hat{y} = \hat{f}(x)$ to match the observed value $y = f(x)$ of the computer code. We also want the fitted model to be relatively smooth. Consider obtaining a prediction $\hat{y}_a$ at, say, the unobserved input point $x_a = 0.36$. In the absence of knowing the nature of the true relationship, the model will "look around" in the x space and it will notice that $x_a = 0.36$ is very close to the observed input point $x_3 = 0.35$, and is between x_3 and $x_4 = 0.52$. Since x_a is very close to x_3, y_a should be very close to the observed value y_3 if the response is smooth and without "spikes". Hence, the model will estimate y_a by a value $\hat{y}_a$ that is close to $y_3 = 0.23$, and perhaps a little smaller to fall between y_3 and $y_4 = -0.12$.

Consider now calculating $\hat{y}$ at a different input point $x_a = 0.45$. The two input points closest to $x_a = 0.45$ for which the computer code was observed are again $x_3 = 0.35$ and $x_4 = 0.52$. Since x_a is approximately halfway between x_4 and x_3, the model could assign roughly the average of y_3 and y_4

as a value for $\hat{y}(x_a)$. Or, since x_a is not too close to either x_3 or x_4, $\bar{y}$ might be a reasonable prediction of y_a. Consider this latter notion further.

To formulate a simple predictor, we might consider $\bar{y}$ and the deviations $y_i - \bar{y}$, $i = 1, 2, \ldots, n$. For example, to make a prediction for a particular input x_a, suppose we use

$$\hat{y}(x_a) = \bar{y} + \sum_{i=1}^{n} w_i (y_i - \bar{y}),$$

where $\sum_{i=1}^{n} w_i (y_i - \bar{y})$ represents a weighted average of the deviations from the mean. The weight w_i should depend on the relative distance between x_a and x_i, with more weight associated with the inputs x_i that are closer to x_a. If only one input point, say x_p, is very close to x_a, this could result in $w_p \approx 1$ with the rest of the weights negligible, in which case one would obtain $\hat{y}_a \approx \bar{y} + (y_p - \bar{y}) = y_p$. For an input x_a sufficiently far away from all of the x_i, all weights w_i could be very small, in which case one would obtain $\hat{y}(x_a) \approx \bar{y} + 0 = \bar{y}$. The model described in Sect. 20.3 has features similar to these. □

20.3 Gaussian Stochastic Process Model

The model most commonly employed in the analysis of computer experiments is called the *Gaussian Stochastic Process model* (GaSP). The model specifies the deterministic computer code function $y = f(\boldsymbol{x})$ as the realization of a "Gaussian stochastic process". Although the code is deterministic, this statistical model provides a probabilistic framework for the response at unobserved input combinations while modeling the unknown output surface as being relatively smooth and responses at nearby inputs as being highly correlated. The GaSP model takes the form

$$Y(\boldsymbol{x}) = \beta_0 + Z(\boldsymbol{x}), \tag{20.3.1}$$

which is similar to a regression model in Chap. 8 but with a different type of error variable. Here, $Y(\boldsymbol{x})$ denotes the response at input combination $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$, and β_0 is an unknown constant. $Z(\boldsymbol{x})$ is assumed to be a *Gaussian stochastic process*, which means that, for any choice of ℓ input combinations, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_\ell$, the random variables $Z(\boldsymbol{x}_1), Z(\boldsymbol{x}_2), \ldots, Z(\boldsymbol{x}_\ell)$ have a multivariate normal distribution (so, in particular, they are generally not independent). The assumptions on the model are as follows:

(i) $Z(\boldsymbol{x})$ has mean zero and constant variance σ^2 for any input combination $\boldsymbol{x}$, so $Z(\boldsymbol{x}) \sim N(0, \sigma^2)$ and
$$Y(\boldsymbol{x}) \sim N(\beta_0, \sigma^2).$$

(ii) For any two inputs $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, the correlation between the responses $Y(\boldsymbol{x}_i)$ and $Y(\boldsymbol{x}_j)$ is denoted by $R(\boldsymbol{x}_i - \boldsymbol{x}_j|\boldsymbol{\xi})$, where $R(\boldsymbol{x}_i - \boldsymbol{x}_j|\boldsymbol{\xi})$ is a function of the distance between the inputs $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ (in d-dimensional space), and depends on a set of unknown parameters which, for simplicity, we write as $\boldsymbol{\xi}$.

Thus,

$$Cov(Y(\boldsymbol{x}_i), Y(\boldsymbol{x}_j)) = Cov(Z(\boldsymbol{x}_i), Z(\boldsymbol{x}_j)) = \sigma^2 R(\boldsymbol{x}_i - \boldsymbol{x}_j|\boldsymbol{\xi}). \tag{20.3.2}$$

Quantifying the correlation between $Y(\boldsymbol{x}_i)$ and $Y(\boldsymbol{x}_j)$ by taking into account the distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is a non-trivial problem. One possible structure that one might use is the "Gaussian correlation function", which is given by

$$R(\boldsymbol{x}_i - \boldsymbol{x}_j|\boldsymbol{\xi}) = \prod_{k=1}^{d} e^{-\theta_k(x_{ik}-x_{jk})^2} = e^{-[\theta_1(x_{i1}-x_{j1})^2+\theta_2(x_{i2}-x_{j2})^2+\cdots+\theta_d(x_{id}-x_{jd})^2]}, \tag{20.3.3}$$

where all $\theta_k > 0$, and where $\boldsymbol{\xi}$ is the set of parameters $(\theta_1, \theta_2, \ldots, \theta_d)$ that need to be estimated from the data. This correlation structure yields a relatively smooth model. Replacing the exponent 2 in $(x_{i1} - x_{j1})^2$ by some positive number p smaller than 2, would result in a less smooth model.

Consider the case of one predictor ($d = 1$). Then the notation simplifies, and the correlation function (20.3.3) reduces to $R(x_i - x_j|\theta) = e^{-\theta(x_i-x_j)^2}$. For any fixed value of $\theta > 0$, the correlation $e^{-\theta(x_i-x_j)^2}$ starts to approach zero as the absolute value $|x_i - x_j|$ of the distance between x_i and x_j gets very large, indicating that $Y(x_i)$ and $Y(x_j)$ are essentially independent so need not be similar. On the other hand, as the distance between x_i and x_j gets very small, the correlation starts to approach 1, indicating that $Y(x_i)$ and $Y(x_j)$ are strongly correlated and so should be similar. Taken to the extreme, if $x_i = x_j$, the correlation between $Y(x_i)$ and $Y(x_j)$ is 1, so $Y(x_i) = Y(x_j)$, i.e. two responses at the same input yield the same value, as desired. Exercise 1 asks the reader to investigate the effect of θ and the absolute distance $|x_i - x_j|$ on the size of the correlation between two responses while using the Gaussian correlation function. Exercises 2–4 show some alternatives to the Gaussian correlation function which are sometimes used in practice.

Under model (20.3.1) and assumption (i), the mean response is simply a constant, β_0. While β_0 could be replaced with a more complicated regression model as in Chap. 8 or 16, doing so typically does not improve the predictive performance of the model for reasonably smooth surfaces (For further reading, see Sacks et al. 1989). Using appropriate methods of prediction, the correlation structure of the Gaussian stochastic process model with constant mean apparently provides adequate model flexibility for reasonably smooth surfaces such as that of Fig. 20.1.

The parameters $(\beta_0, \sigma^2, \theta_1, \ldots, \theta_d)$ in (20.3.1)–(20.3.3) are unknown so need to be estimated. There are a few different ways to estimate these, but we shall use the most common one—maximum likelihood estimation, since such estimators have excellent statistical properties. An in-depth discussion of maximum likelihood estimation is beyond the scope of this book, but maximum likelihood estimates can be computed using appropriate statistical software, and we will rely on software for the computations (as in Sects. 20.6 and 20.7).

For the interested reader, here is the idea of maximum likelihood estimation. Under model (20.3.1), the response variables $Y(\boldsymbol{x}_1), \ldots, Y(\boldsymbol{x}_n)$ follow a multivariate normal distribution which depends on the values of the parameters $\beta_0, \sigma^2, \theta_1, \ldots, \theta_d$. For given values of these parameters, the probability density function (pdf) is a function of the possible data (output) points $y(\boldsymbol{x}_1), \ldots, y(\boldsymbol{x}_n)$. One is more likely to observe data values where the pdf is larger, and most likely to get data where the pdf is at or near its maximum. Conversely, if we have the data $\boldsymbol{y} = (y(\boldsymbol{x}_1), \ldots, y(\boldsymbol{x}_n))$ but the parameters are unknown, it seems reasonable to estimate the parameters by values that would correspond to the observed data being as likely as possible. If we estimate the parameters by values that maximize the likelihood of the observed data, these are *maximum likelihood estimates*, and this is *maximum likelihood estimation*.

Once estimates of the model parameters have been obtained, the next step is to construct a predictor $\hat{Y}(\boldsymbol{x}_a)$ of a response $Y(\boldsymbol{x}_a)$ at any new unobserved input point $\boldsymbol{x}_a$ (cf. Chap. 8). The

predictor should be easily computable but sufficiently sophisticated to utilize the observed data $\boldsymbol{y} = (y(\boldsymbol{x}_1), y(\boldsymbol{x}_2), \ldots, y(\boldsymbol{x}_n))$. The predictor $\hat{Y}(\boldsymbol{x}_a) = \hat{\beta}_0$, for example, would be too simplistic. If the correlation parameters θ_k were known, we could compute and use the *best linear unbiased predictor*—namely, the unbiased predictor that minimizes the mean squared error of prediction. Since the correlation parameters θ_k are unknown, what we can do is compute their maximum likelihood estimates, then use what would be the best linear unbiased predictor if the estimated correlations θ_k were the true values. The resulting predictor is called the *empirical best linear unbiased predictor* (eBLUP).

It can be shown that the eBLUP of $Y(\boldsymbol{x}_a)$ is $\hat{Y}(\boldsymbol{x}_a) = E[Y(\boldsymbol{x}_a)|\boldsymbol{y}]$, which is the mean of $Y(\boldsymbol{x}_a)$ conditional on the observed data $\boldsymbol{y}$ at the observed input combinations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. For model (20.3.1), the formula for the eBLUP of $Y(\boldsymbol{x}_a)$ is most easily written in terms of vectors and matrices. Readers who do not have an algebra background can jump to (20.3.6) and leave it to computer software to do the needed calculations. The formula for the eBLUP is

$$\hat{Y}(\boldsymbol{x}_a) = \hat{\beta}_0 + \hat{\boldsymbol{r}}' \hat{\boldsymbol{R}}^{-1} (\boldsymbol{y} - \mathbf{1}\hat{\beta}_0), \tag{20.3.4}$$

where $\hat{\boldsymbol{r}}$ is an $n \times 1$ vector whose ith element $\hat{r}_i = R(\boldsymbol{x}_a - \boldsymbol{x}_i | \hat{\boldsymbol{\xi}})$ is the estimated correlation between the response $Y(\boldsymbol{x}_a)$ at a new input combination $\boldsymbol{x}_a$ and the ith previous response $Y(\boldsymbol{x}_i)$, $\hat{\boldsymbol{R}}$ is a $n \times n$ matrix whose ijth element $\hat{R}_{ij} = R(\boldsymbol{x}_i - \boldsymbol{x}_j | \hat{\boldsymbol{\xi}})$ is the estimated correlation between observed responses $Y(\boldsymbol{x}_i)$ and $Y(\boldsymbol{x}_j)$, $\mathbf{1}$ is $n \times 1$ vector of 1's, and $\hat{\beta}_0 = (\mathbf{1}' \hat{\boldsymbol{R}}^{-1} \mathbf{1})^{-1} \mathbf{1}' \hat{\boldsymbol{R}}^{-1} \boldsymbol{y}$ is the generalized least squares estimate of β_0.

Under model (20.3.1), the uncertainty about the predicted value can be estimated via the estimated variance of the eBLUP given by

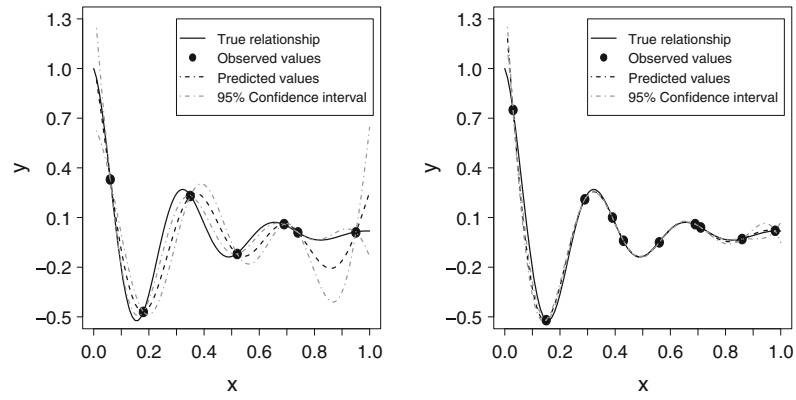
$$\hat{s}^2(\boldsymbol{x}_a) = \hat{\sigma}^2 \left[1 - \hat{\boldsymbol{r}}' \hat{\boldsymbol{R}}^{-1} \hat{\boldsymbol{r}} + \frac{(1 - \mathbf{1}' \hat{\boldsymbol{R}}^{-1} \mathbf{1})^2}{\mathbf{1}' \hat{\boldsymbol{R}}^{-1} \mathbf{1}} \right]. \tag{20.3.5}$$

It can be shown algebraically that $s^2(\boldsymbol{x}) = 0$ for any $\boldsymbol{x}$ for which the response $Y(\boldsymbol{x})$ was already observed, i.e. the model will interpolate the data.

For each possible unobserved input combination $\boldsymbol{x}_a$, the $100(1 - \alpha)\%$ prediction interval can be calculated as

$$\hat{Y}(\boldsymbol{x}_a) \pm z_{\alpha/2} \hat{s}(\boldsymbol{x}_a). \tag{20.3.6}$$

There are two key observations to be made about the predictor $\hat{Y}(\boldsymbol{x}_a)$, whose form is shown in (20.3.4). First, the predicted value at a new input site $\boldsymbol{x}_a$ is the sum of two parts. The first part is the estimated overall mean $\hat{\beta}_0$. The second part is a weighted average of the differences $y(\boldsymbol{x}_i) - \hat{\beta}_0$ between each observed response $y(\boldsymbol{x}_i)$ and the estimated overall mean $\hat{\beta}_0$. In general, the largest weight that would be calculated in (20.3.4) is associated with the term $y(\boldsymbol{x}_i) - \hat{\beta}_0$ corresponding to the $\boldsymbol{x}_i$ that is closest to $\boldsymbol{x}_a$; the second largest weight is associated with the term corresponding to $\boldsymbol{x}_i$ that is second closest to $\boldsymbol{x}_a$, etc. So, similar to Example 20.2.1, the quality of the prediction at $\boldsymbol{x}_a$ is a function of the distances between $\boldsymbol{x}_a$ and the n points $\boldsymbol{x}_i$ for which we have already observed the response. If we have not observed any responses in the vicinity of an input combination $\boldsymbol{x}_a$, then our prediction $\hat{Y}(\boldsymbol{x}_a)$ will default to a value close to $\hat{\beta}_0$ and may be a poor estimate of $Y(\boldsymbol{x}_a)$. On the other hand, if we have collected quite a few observations in a particular region of the input space, we can expect to predict the unknown surface quite well in that region.



(a) $n = 7$ (b) $n = 10$

Fig. 20.2 True unknown, observed, and predicted relationships between x and y for Example 20.3.1

The second important observation is that the predictor $\hat{Y}(\boldsymbol{x}_a)$ shown in (20.3.4) is an *interpolating* predictor. In other words, suppose that $\boldsymbol{x}_a$ is one of the inputs for which we have observed the response, say $\boldsymbol{x}_a = \boldsymbol{x}_p$. Then our predictor will return the observed value $y(\boldsymbol{x}_p)$; that is,

$$\hat{Y}(\boldsymbol{x}_p) = y(\boldsymbol{x}_p)\,.$$

This behavior from our predictor is desirable because of the deterministic nature of the computer code. If we know that we have observed a response without an error and that we will observe the same response every time we run the computer code for the same input $\boldsymbol{x}_i$, then we would like our predictor to predict the same value for the response.

Example 20.3.1 `One Predictor, continued`

As in Example 20.2.1, suppose the true but unknown input-output relationship is described by a dampened cosine curve, $y = f(x) = e^{-4x}\cos(6\pi x)$, for $0 \le x \le 1$, as shown in Fig. 20.1. This relationship is shown again as the solid line in Fig. 20.2. Suppose we try to estimate the unknown relationship based on the seven observations (x_i, y_i) that were considered in Example 20.2.1—namely, (0.06, 0.33), (0.18, −0.47), (0.35, 0.23), (0.52, −0.12), (0.69, 0.06), (0.74, 0.01), (0.95, 0.01). These seven data points are shown as the solid dots in Fig. 20.2(a). Using software (see Sects. 20.6 and 20.7) to fit the GaSP model from (20.3.1) with the Gaussian correlation function (20.3.3) and $d = 1$, we obtain the maximum likelihood estimates $\hat{\sigma} = 0.0582$, $\hat{\beta}_0 = 0.0047$, and $\hat{\theta} = 271.95$. Using (20.3.4)–(20.3.6), the predictions and 95% prediction intervals can then be calculated for $x = 0.01, 0.02, \ldots, 1$. The three dashed lines in Fig. 20.2(a) show the predicted curve (the darker middle curve) and the prediction intervals. We can see that our model does a particularly poor job of predicting the true relationship for x values roughly between 0.8 and 0.9. Since there are no observations in this area, the prediction tends to regress to (i.e. predict values closer to) the estimated mean $\hat{\beta}_0 = 0.0047$.

Figure 20.2(b) shows the same information as Fig. 20.2(a), but having fit the model using a set of $n = 10$ different observations. The maximum likelihood estimates are now $\hat{\sigma} = 0.0942$, $\hat{\beta}_0 = 0.0551$, and $\hat{\theta} = 196.17$. Figure 20.2(b) indicates a dramatic improvement in the model's prediction performance. Thus, we see that good design of the computer experiment can be crucial to good prediction. □

20.4 Design

For a computer experiment involving n runs and d input variables, a *design* consists of the n input combinations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ to be run, where $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})$, $i = 1, \ldots, n$. It is customary to collect these input combinations into an $n \times d$ array or "matrix" $\boldsymbol{X}$, where the kth row of $\boldsymbol{X}$ shows the kth input combination, and we refer to such an array as an $n \times d$ *design* $\boldsymbol{X}$, or an $n \times d$ *design matrix* $\boldsymbol{X}$.

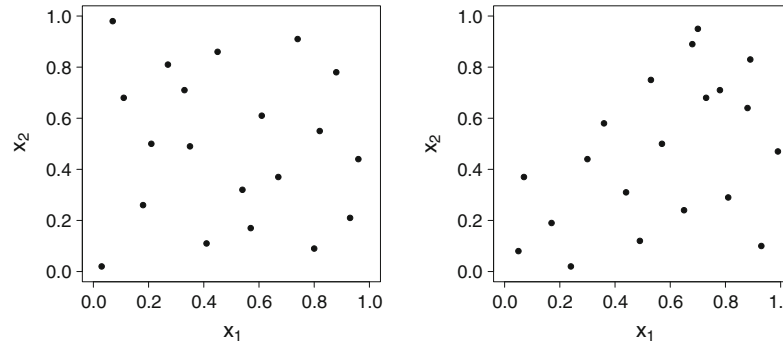
The interpretation of the magnitude of the parameters θ_k in (20.3.3) is related to the scale of the input x_k. More importantly, assessing the sensitivity of the response to the changes in inputs x_k and x_j by comparing the magnitudes of θ_k and θ_j associated with those inputs x_k and x_j can be deceiving if those inputs are defined on vastly different scales. To avoid the potential pitfalls it is common to transform the range of each input variable to become [0,1]. This is done by subtracting the minimum and then dividing by maximum−minimum. For example, if x_1 has an original range of possible values 3.5 to 9.6, we subtract 3.5 from every value of x_1 and divide by $(9.6 - 3.5) = 6.1$. From now on, we will assume that this transformation has been done and that every input variable now has values in the range [0,1].

20.4.1 Space-Filling and Non-collapsing Designs

The design of a computer experiment—namely, the choice of input combinations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ for the computer code—is guided by the deterministic nature of a computer experiment and by the particular characteristics of the predictor discussed in Sect. 20.3. We expect to predict the unknown surface well in the regions of input space where we have observed some nearby responses. Therefore, we would like our initial design to explore the input space "evenly", so we would like our design to be *space-filling*. Designs that are not space-filling will neglect one or more regions of the input space, and we can anticipate poor prediction of the response in such regions.

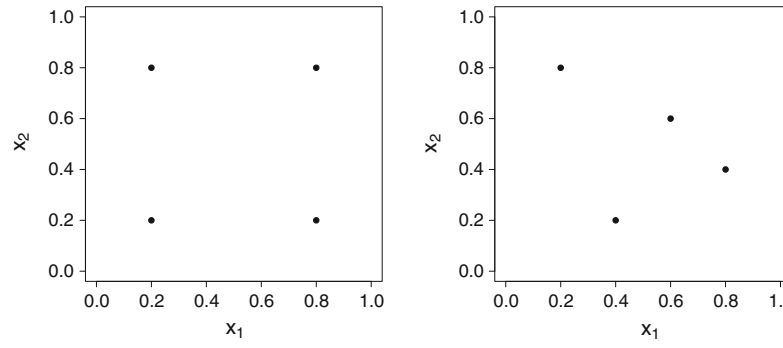
For example, consider a computer experiment involving $d = 2$ input variables and input combinations $\boldsymbol{x_i} = (x_{i1}, x_{i2})$, with $0 < x_{ik} < 1$ for $i = 1, \ldots, n; k = 1, 2$, listed as the rows of the design $\boldsymbol{X}$. Such input combinations correspond to points in a unit square, where a response can be observed. Consider the two 20-run experimental designs $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ whose design points (x_{i1}, x_{i2}) are plotted in Fig. 20.3. Although we have not yet quantified the space-filling idea, the design $\boldsymbol{X}_1$ seems to do a much better job of exploring the two-dimensional input space than $\boldsymbol{X}_2$. In particular, $\boldsymbol{X}_2$ does not contain any points (x_{i1}, x_{i2}) such that $x_{i2} > x_{i1} + 0.4$ leaving the upper left region of the input space unexplored. Hence, based on our previous discussion, the predictor based on $\boldsymbol{X}_1$ should do a better job of predicting the unknown surface in the upper left region of the input space than the predictor based on $\boldsymbol{X}_2$.

As previously noted, multiple runs of the computer code at the same input combination $\boldsymbol{x} = (x_1, x_i, \ldots, x_d)$ will yield the same output $y(\boldsymbol{x})$ due to the deterministic nature of the computer code, so replicating input points is wasteful. Moreover, when one or more input variables has no effect on the response, it is beneficial to avoid replicating any input combination for *any subset* of the input variables (as explained below). One way to avoid this replication is to ensure that, for each column (i.e. input variable) of such a design $\boldsymbol{X}$, the n input values are distinct. Such a design is called a *noncollapsing design*. To illustrate this notion, consider the following two design matrices, each for four runs (rows) with two inputs (columns):



(a) $\boldsymbol{X}_1$ (b) $\boldsymbol{X}_2$

Fig. 20.3 Two potential 20-run designs for a computer experiment with 2 input variables

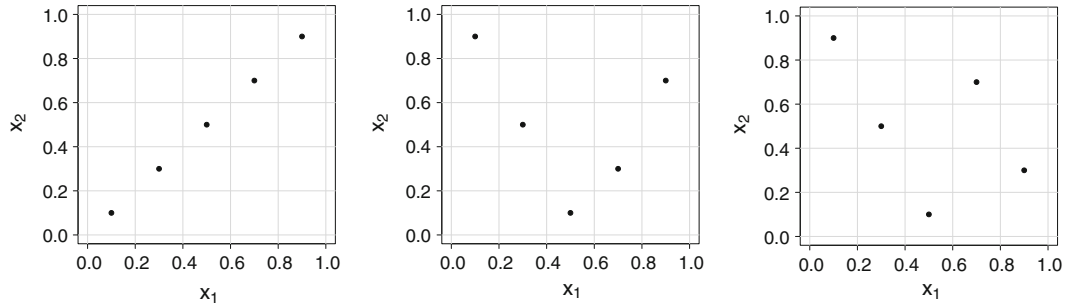


(a) $\boldsymbol{X}_3$ (b) $\boldsymbol{X}_4$

Fig. 20.4 A collapsing ($\boldsymbol{X}_3$) and a noncollapsing ($\boldsymbol{X}_4$) design

$$\boldsymbol{X}_3 = \begin{bmatrix} 0.2 & 0.2 \\ 0.2 & 0.8 \\ 0.8 & 0.2 \\ 0.8 & 0.8 \end{bmatrix} \text{ and } \boldsymbol{X}_4 = \begin{bmatrix} 0.2 & 0.8 \\ 0.4 & 0.2 \\ 0.6 & 0.6 \\ 0.8 & 0.4 \end{bmatrix}.$$

These designs are depicted in Fig. 20.4. We can see that $\boldsymbol{X}_3$ is a collapsing design since both of its columns contain replicated values, i.e. if we collapse the design points onto either the x_1- or x_2-axis, we obtain only two distinct values from the four distinct points. $\boldsymbol{X}_4$ is a noncollapsing design since neither x_1 nor x_2 values are replicated. Without any further considerations, one might wonder what possible problem a collapsing design could cause. Suppose the true, but unknown, response function is given by $y(x_1, x_2) = \exp(-2x_2)\cos(3\pi x_2)$. We see that the response does not depend on the input x_1; a fact not known to a researcher before the experiment. Based on the designs $\boldsymbol{X}_3$ and $\boldsymbol{X}_4$, the set of four corresponding observed values are given by $\boldsymbol{y}_3 = (0.21, 0.06, 0.21, 0.06)$ and $\boldsymbol{y}_4 = (0.06, 0.21, -0.24, -0.36)$, respectively. The design $\boldsymbol{X}_3$ did not contain any replicated input combinations—no two rows were the same. Still, due to replication of values of x_2 and the inactivity of x_1, we ended up with undesirable replicated responses. One might argue that half of our computational resources were wasted, since two of the four runs of the computer code did not add any new information, except perhaps to notice that x_1 may not affect the response. To prevent potential computational wastefulness, and in the light of the fact that we have other tools to identify inactive inputs, we prefer the design $\boldsymbol{X}_4$.



(a) $\boldsymbol{X}_1$ (b) $\boldsymbol{X}_2$ (c) $\boldsymbol{X}_3$

Fig. 20.5 Latin hypercube designs

There are several strategies for constructing designs for computer experiments that are noncollapsing and/or space-filling. The most commonly used approach is that of Latin Hypercube designs, which are inherently noncollapsing and many are also space-filling. We illustrate these designs below.

20.4.2 Construction of Latin Hypercube Designs

To construct an $n \times d$ Latin Hypercube design (LHD), we start by dividing the range $[0, 1]$ of each input variable into n equal-length "bins", with divisions at

$$0, 1/n, 2/n, \ldots, (n-1)/n, 1\,.$$

These bins have midpoints at $(2k-1)/2n$ for $k = 1, 2, \ldots, n$ and, for simplicity, we can use the midpoints to label the bins. For example, in each of Fig. 20.5(a)–(c), we have $n = 5$ input points, and so the bin mid-points for each of x_1 and x_2 are at

$$\frac{2-1}{10} = 0.1, \quad \frac{4-1}{10} = 0.3, \quad \frac{6-1}{10} = 0.5, \quad \frac{8-1}{10} = 0.7, \quad \frac{10-1}{10} = 0.9\,. \tag{20.4.7}$$

In total, there are 5^2 two-dimensional cells which can be labeled by the pairs of midpoints of the (x_1, x_2) bins. Similarly, in general, dividing the ranges of each of d input variables $x_1, \ldots, x_d$ into n bins leads to n^d possible d-dimensional cells; these are called *hypercubes* and can be labeled using the sets of bin midpoints for the d input variables.

Next, we want to choose n of these n^d cells (hypercubes) to contain a design point in such a way that there is only one design point in each bin for each individual input variable. The design is then non-collapsing. For example, if the 5 input points in each of the designs depicted in Figs. 20.5(a)–(c) are projected onto each axis, it can be seen that they rest on the midpoints (20.4.7) of the 5 bins for both x_1 and x_2. One way of achieving such a choice is as follows.

(a) Start with an array $\boldsymbol{X}$ with n rows and d columns where each column contains the integers $1, 2, \ldots, n$.
(b) For each column separately, randomly order the integers.
(c) Replace each integer k by the corresponding bin midpoint $(2k-1)/2n$.

(d) The ith row of of the randomly ordered $\boldsymbol{X}$ then determines the ith input combination for the experiment.

Example 20.4.1 `5 × 2 Latin hypercube designs`

Suppose that a design is required with $n = 5$ input points for a computer experiment where the simulator has $d = 2$ input variables. Following the above procedure, each of the d columns of $\boldsymbol{X}$ starts with the values 1, 2, 3, 4, 5 in order. Each column is randomly ordered and the integers k are replaced by the bin midpoints $(2k-1)/2n = 0.1, 0.3, 0.5, 0.7, 0.9$ to give the final design. Three possible designs are

$$\boldsymbol{X}_1 = \begin{bmatrix} 0.1 & 0.1 \\ 0.3 & 0.3 \\ 0.5 & 0.5 \\ 0.7 & 0.7 \\ 0.9 & 0.9 \end{bmatrix}, \quad \boldsymbol{X}_2 = \begin{bmatrix} 0.3 & 0.5 \\ 0.5 & 0.1 \\ 0.9 & 0.7 \\ 0.1 & 0.9 \\ 0.7 & 0.3 \end{bmatrix}, \text{ and } \boldsymbol{X}_3 = \begin{bmatrix} 0.5 & 0.1 \\ 0.3 & 0.5 \\ 0.1 & 0.9 \\ 0.9 & 0.3 \\ 0.7 & 0.7 \end{bmatrix},$$

where each row gives the coordinates of an input combination $\boldsymbol{x}_i = (x_{i1}, x_{i2})$. Designs $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, and $\boldsymbol{X}_3$ are the three designs depicted in Fig. 20.5a–c. Since all these designs are LHDs, they are noncollapsing.

By visual inspection, we notice that the design $\boldsymbol{X}_1$ is the least space-filling and does the poorest job of exploring the input space. Although this design does a really good job of exploring the space along the main diagonal, that is all that it does well. Since no points are placed away from the main diagonal, there is no exploration of the input space towards the upper left and lower right corners of the space. If we use this design to run our experiment, we can expect very poor predictions the further we get away from the main diagonal. Even though $\boldsymbol{X}_1$ is an LHD, it is clearly not space-filling, so is a poor design for a computer experiment. Both $\boldsymbol{X}_2$ and $\boldsymbol{X}_3$ do better at space filling, and which is the better of these two designs is less clear (although a preference will be expressed for $\boldsymbol{X}_3$ in Example 20.4.2). □

To seek a design which is both noncollapsing and space-filling, experimenters often look for a Latin hypercube design that best satisfies some space-filling property. There is a vast literature describing many attempts to define what we mean by a "best" design with respect to space-fillingness. One of the most common approaches is to choose a design which maximizes the distance between the two closest points in the design. A common measure of distance is the familiar Euclidean distance between $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{id})$ and $\boldsymbol{x}_j = (x_{j1}, \ldots, x_{jd})$; that is,

$$\sqrt{\sum_{\ell=1}^{d} (x_{i\ell} - x_{j\ell})^2}\,. \tag{20.4.8}$$

Suppose that we have two $n \times d$ designs $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$. For each design, we can determine the minimum distance between pairs of input points. The design with the larger value of this minimum distance is considered preferable, since choosing a design in this way forces the design points to be far away from each other and so tends to be more space-filling. Among all possible designs, any design that *max*imizes this *min*imum interpoint distance is called a *maximin design*.

Example 20.4.2 `5 × 2 Latin hypercube designs, continued`

One could compare the designs $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, and $\boldsymbol{X}_3$ of Fig. 20.5 based on the maximin criterion. The corresponding interpoint distances (rounded to two decimal places) are shown in Table 20.1. Notice that the smallest interpoint distance is 0.28 for both $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$, compared with the minimum interpoint distance of 0.45 for $\boldsymbol{X}_3$. Hence, among these three Latin hypercube designs, $\boldsymbol{X}_3$ is the maximin design.

Table 20.1 Interpoint distances for the designs of Fig. 20.5

	$\boldsymbol{X}_1$				$\boldsymbol{X}_2$				$\boldsymbol{X}_3$			
	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$	$\boldsymbol{x}_4$	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$	$\boldsymbol{x}_4$	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$	$\boldsymbol{x}_4$
$\boldsymbol{x}_2$	0.28				0.45				0.45			
$\boldsymbol{x}_3$	0.57	0.28			0.63	0.72			0.89	0.45		
$\boldsymbol{x}_4$	0.85	0.57	0.28		0.45	0.89	0.82		0.45	0.63	1.00	
$\boldsymbol{x}_5$	1.13	0.85	0.57	0.28	0.45	0.28	0.45	0.85	0.63	0.45	0.63	0.45

If we now look at $\boldsymbol{X}_1$ in Fig. 20.5(b) again, we can see that the two points nearest the x_1 axis are close together in the two-dimensional space, whereas all the points lie further apart in $\boldsymbol{X}_3$ in Fig. 20.5(c), leading to a slightly better coverage of the design space. □

Exercises 7 and 8 show alternative measures of space-fillingness ("minimax" and "average reciprocal distance"). There are other measures, too, in the literature. But, in this chapter, we will concentrate on the most common measure of "maximin". (For further types of designs, see Santner et al. 2003, Chaps. 5 and 6).

In practice, to construct a space-filling, noncollapsing design, one would use the aid of a computer. For example, given the number of runs n and inputs d, one could use the computer to generate lots (several hundred) Latin hypercube designs $\boldsymbol{X}$, by using many random orderings of the columns, compute the minimum interpoint distance for each design, and keep the design with the largest minimum interpoint distance. Although this may not result in *the* maximin design, it should result in one that is close to maximin and which has good space-filling properties. Sections 20.6.1 and 20.7.1 show how to generate maximin LHDs using the SAS and R software, respectively.

As a final note, design points do not necessarily need to be placed at the midpoint of the cell. One can place a design point in a randomly chosen location in the cell by adding a random number between $-(2n)^{-1}$ and $+(2n)^{-1}$ to every x_{ij} in $\boldsymbol{X}$.

20.5 A Real Experiment—Neuron Experiment

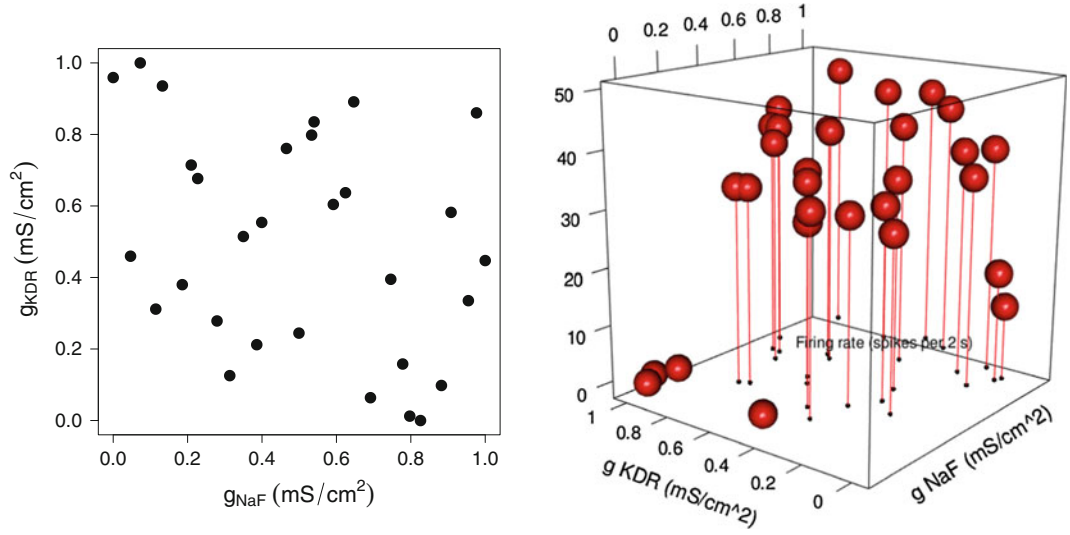
One of the authors (Danel Draguljić) was involved in a study aimed at modeling the performance of neurons. The study, which is described by Rumbell and coauthors in *Journal of Computational Neuroscience* in 2016, aimed to improve the fits of conductance-based models to in vitro whole cell recordings from pyramidal neurons of layer 3 of the prefrontal cortex from young and aged rhesus monkeys. The neuron's performance was measured by its firing rate (response, Y) which was modeled with either 4 or 8 ion channels resulting in 10 or 23 explanatory variables, respectively, to be considered in the model.

Table 20.2 shows the data from a simplified model for the firing rates of a neuron at +380 pA current injection of a young monkey. The simplified model contained two input variables; x_1 was the maximal conductance of the transient sodium, denoted g_{NaF}, and x_2 was the maximal conductance of the delayed-rectifier potassium, denoted g_{KDR}. Both of these variables affect the neuron firing rate. Both maximal conductances had original ranges between 0.05 and 0.5 mS/cm^2, but the g_{NaF} and g_{KDR} values in Table 20.2 have been scaled to the [0, 1] range, as throughout this chapter. The design chosen for the input variables was a 30×2 Latin hypercube design and is shown in Fig. 20.6(a).

Columns 1 and 2 of Table 20.2 show the first 15 values of the input variables, and columns 4 and 5 show the next 15 values. For a given value of g_{NaF} and g_{KDR} the computer simulator was run for 2

Table 20.2 The data for the neuron experiment

g_{NaF}	g_{KDR}	y_i	g_{NaF}	g_{KDR}	y_i
0.38594	0.21207	33	0.53994	0.83528	41
0.04667	0.45947	0	0.13236	0.93565	1
1.00000	0.44733	46	0.90811	0.58219	46
0.95468	0.33514	44	0.88221	0.09805	39
0.53335	0.79813	41	0.39918	0.55403	36
0.59167	0.60427	41	0.22713	0.67680	34
0.18570	0.37995	31	0.82599	0.00000	13
0.49928	0.24442	36	0.69149	0.06387	36
0.74609	0.39496	42	0.79710	0.01235	19
0.07269	1.00000	0	0.31326	0.12559	30
0.27908	0.27845	32	0.62431	0.63717	41
0.34985	0.51466	35	0.97649	0.86049	48
0.64658	0.89122	43	0.46554	0.76087	39
0.20972	0.71452	34	0.11439	0.31166	34
0.00000	0.95918	0	0.77811	0.15813	39



(a) 30×2 LHD (b) Firing rates

Fig. 20.6 The design and the data for the neuron experiment

seconds and the number of spikes, y_i, was recorded. These observed firing rates are shown in columns 3 and 6 of Table 20.2 and displayed in Fig. 20.6(b). We can observe that relatively low firing rates occur for large values of g_{KDR} and small values of g_{NaF}. For example, if we are trying to maximize neuron's firing rate, it seems that our best chance lies with values in the upper ranges of both variables.

We would like to use our data to construct an estimator $\hat{y}$ of $Y = f(g_{\text{NaF}}, g_{\text{KDR}})$. The estimator $\hat{y}$ would then enable us to identify the regions in the $(g_{\text{NaF}}, g_{\text{KDR}})$ space that are associated with either low or high neuron activity. After fitting the GaSP model (20.3.1) with Gaussian correlation function (20.3.3), we obtain the maximum likelihood estimates $\hat{\sigma} = 15.87$, $\hat{\theta}_1 = \hat{\theta}_{\text{NaF}} = 5.03$, and

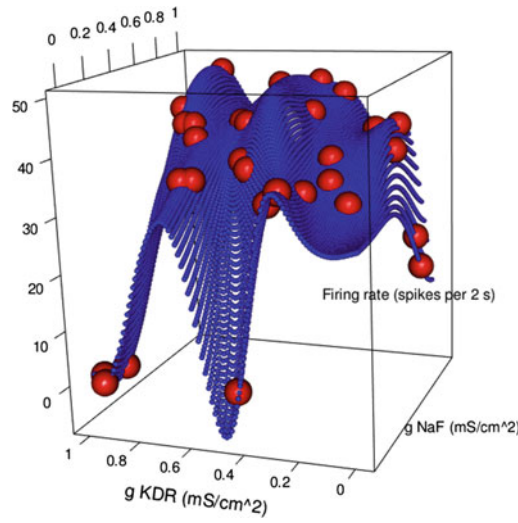


Fig. 20.7 Observed values and predicted surface for the neuron experiment

$\hat{\theta}_2 = \hat{\theta}_{\text{KDR}} = 50.22$, giving $\hat{\beta}_0 = 27.61$. Using the predictor in (20.3.4), we calculated the values of $\hat{Y}$ at a dense grid of input points over a $[0, 1] \times [0, 1]$ square. The predicted outputs are shown as the blue points in Fig. 20.7, together with the original data from the simulator (red points). The predicted maximum firing rate is 49.11 spikes per 2 seconds and it occurs at the point $(g_{\text{NaF}}, g_{\text{KDR}}) = (0.86, 0.86)$. The predicted minimum firing rate is -7.52 spikes per 2 seconds and it occurs at the point $(g_{\text{NaF}}, g_{\text{KDR}}) = (0, 0.49)$. We know that firing rate cannot be negative, but there is no intrinsic mechanism in the GaSP model to prevent it from making negative predictions. At this point, we could run the simulator at $(g_{\text{NaF}}, g_{\text{KDR}}) = (0, 0.49)$ and obtain $y(0, 0.49)$, then refit our model using all $30 + 1$ observations, and look for the minimum again. Alternatively, if we had been concerned with avoiding negative predictions, we could have considered transforming our data, say to $\ln(y)$, to fit our model and then exponentiate the predicted values.

In this example, suppose we take two input points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ such that $\boldsymbol{x}_1 = (x_{11}, x_{12})$ and $\boldsymbol{x}_2 = (x_{11} + h, x_{12} + h)$ for some $h > 0$. Then, the estimated correlation between $Y(\boldsymbol{x}_1)$ and $Y(\boldsymbol{x}_2)$ is given by

$$\hat{R}(\boldsymbol{x}_1 - \boldsymbol{x}_2 | \hat{\theta}_1, \hat{\theta}_2) = e^{-5.03h^2 - 50.22h^2} = e^{-5.03h^2} e^{-50.22h^2} = \hat{R}_1(h)\hat{R}_2(h).$$

$\hat{R}_1(h)$ and $\hat{R}_2(h)$ are shown in Fig. 20.8. Since $\hat{R}(\boldsymbol{x}_1 - \boldsymbol{x}_2 | \hat{\theta}_1, \hat{\theta}_2)$ is the product of $\hat{R}_1(h)$ and $\hat{R}_2(h)$, Fig. 20.8 indicates that, for all h larger than 0.1, $\hat{R}(\boldsymbol{x}_1 - \boldsymbol{x}_2 | \hat{\theta}_1, \hat{\theta}_2) \approx 0$ because $\hat{R}_2(h) \approx 0$. In other words, suppose that two points are more than 0.1 mS/cm^2 apart in the g_{KDR} space then, even if they are very close in the g_{NaF} space, the observations at those points are essentially treated as uncorrelated.

Here, and in general, we conclude that changes in an input variable x_k associated with a large value of θ_k will tend to have little impact on the size of the correlation. In other words, as seen by the dotted line in Fig. 20.8, if θ_k is large, the two design points have to be extremely close to each other for their corresponding observations to have sizable correlation.

20.6 Using SAS Software

In this section, we illustrate use of the SAS software for creating Latin hypercube designs (LHDs) and for analyzing data obtained from computer experiments.

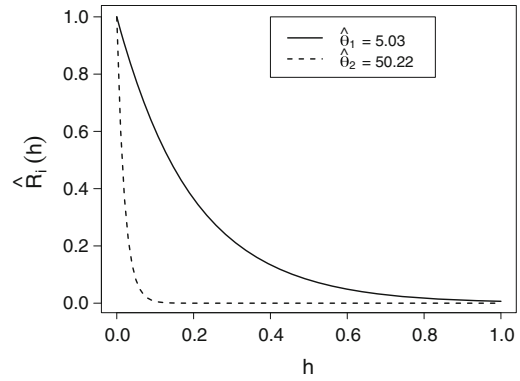


Fig. 20.8 Correlation as a function of distance $\boldsymbol{h}$ between two input points

20.6.1 Maximin Latin Hypercube Designs

This section shows how to use SAS software to get an LHD that is approximately maximin. The idea behind constructing a maximin LHD (Mm LHD) is described in Sect. 20.4.2. First, we construct an $n \times d$ LHD. Second, for the newly constructed LHD we calculate all $\binom{n}{2}$ interpoint Euclidean distances and then find the smallest one, i.e. we identify the distance between two closest points. And third, we iterate this process many times and, among many created LHDs, we identify the one with largest minimum interpoint distance. It is worth noting that the "best" design obtained after running the SAS code from this section contains the maximum minimum interpoint distance among the number of designs examined by the code. Another run of the same code with the same inputs might produce a design with even larger maximum minimum interpoint distance.

The SAS programs shown here are in the form of SAS "macros". A macro is a piece of SAS code that can be stored separately and is extremely useful when SAS software has to repeat the same calculations many times, as it does in the search for the best LHD. Table 20.3 shows a SAS macro. A macro begins and ends with SAS statements `%macro` and `%mend`, respectively. The macro in Table 20.3 is called `createLHD` and it has two inputs n and d, which will be provided when the macro is "called" (implemented). The body of a macro contains SAS statements that can be run many times by calling the macro which eliminates copying and pasting the statements themselves. Before being able to use a macro, we have to run the whole macro. Once this is done, the SAS software has this macro available for use. To then execute the SAS statements within the macro `createLHD`, we run for example `%createLHD(n = 20, d = 2)` where, here, values 20 and 2 have been selected for n and d. These values can be changed to suit the user's needs.

Within this macro, the `%` symbol indicates to the SAS software that this is a segment of the program; here we have two (nested) loops starting at `%do` and ending at `%end`. The `&` symbol in front of the variables `n, d, i` and `j` indicates that, within the macro, the variables have been assigned specific values (the last two change as we proceed around the loops). Writing macros is beyond the scope of this book, but we note that SAS code that is inside of a macro can be taken out and be used with some necessary clean-up, such as removing the `%` and `&` symbols.

The SAS code to find an approximately maximin LHD is shown in the form of three macros in Tables 20.3, 20.4, and 20.5, respectively. The macro `createLHD`, shown in Table 20.3, constructs an LHD with n rows and d columns, where d is the number of inputs to the computer simulator and n is the required number of outputs. The LHD is constructed by filling each of the d columns with $(2k-1)/2n$ for $k = 1, 2, \ldots, n$ and then the rows of each column are randomly rearranged using the `ranuni(0)` command described in Sect. 3.8.1. To then construct, say, a 20×2 LHD, one simply

Table 20.3 A SAS program for construction of an LHD

```
%macro createLHD(n=, d=);
  %do i = 1 %to &d;
    data temp&i;
      %do j = 1 %to &n;
        x&i = (2*&j - 1)/(2*&n) + (1/&n)*ranuni(-1) - 1/(2*&n);
        ranno&i = ranuni(0);
        output;
      %end;
    run;
    proc sort data = temp&i; by ranno&i; run;
    data temp&i; set temp&i; drop ranno&i; run;
  %end;
  data lhd; merge temp1 - temp&d; run;
  proc print data = Lhd; run;
%mend createLHD;
```

runs `%createLHD(n = 20, d = 2)` line. A 20×2 LHD will then be created, printed out to the SAS *Results Viewer* window, and stored in the SAS data set called `Lhd`.

The macro shown in Table 20.4 is `mipd`. This macro calculates minimum interpoint distance for a user-provided LHD. For example, if we created an LHD using `createLHD` macro, we can then calculate this LHD's minimum interpoint distance by running `%mipd(x = Lhd, d = 2)` where input x is the SAS data set that contains the LHD and d is the number of columns in this LHD. The minimum interpoint distance will be printed to the *Results Viewer* window and it will be saved in the SAS data set `Mindist`.

In order to create a maximin LHD, we do not have to call the `createLHD` and `mipd` macros explicitly. Macro `MmLHD` shown in Table 20.5 does all the work for us. In addition to other SAS statements, `MmLHD` calls macros `createLHD` and `mipd` with the appropriate inputs for each. Even though we do not call `createLHD` and `mipd` directly, we do have to run them before we run and call `MmLHD`. The macro `MmLHD` requires three inputs itself: `nrow` and `ncol` define the size of the LHD (n and d) and the integer `iter` tells the SAS software how many LHDs we want the SAS software to create from which to choose the one with largest minimum interpoint distance. If we run `%MmLHD(nrow = 30, ncol = 3, iter = 100)` the SAS software will create one hundred 30×3 LHDs and it will find the one with the largest minimum interpoint distance. This design with largest minimum interpoint distance is stored in the SAS data set `Best`. To see the design `Best`'s smallest interpoint distance, we can run `%mipd(x = Best, d = 3)`.

All designs created by `MmLHD` have input points placed at random within the selected LHD cells. If we want points placed at the midpoints of the selected cells, we have to remove `+ (1/&n)*ranuni(-1) - 1/(2*&n)` from the fifth line in the macro `createLHD`.

After creating the approximate maximin LHD, the simulator can be run with input combinations $\mathbf{x}$ defined by the n rows of the design stored in `Best`.

Table 20.4 A SAS program for calculation of minimum interpoint distance

```
%macro mipd(x=, d=);
  proc distance data = &x out = dist method = Euclid nostd;
    var interval(x1 - x&d); run;
  data dist;
    set dist;
    array var _numeric_;
    do over var;
      if var = 0 then var = .;
    end;
  run;
  proc means nolabels data = dist min noprint;
    output out = colmins; run;
  data colmins;
    set colmins;
    where _STAT_ = "MIN";
    drop _TYPE_ _FREQ_ _STAT_;
  run;
  proc transpose data = colmins out = colminslong; run;
  proc means nolabels data = colminslong min;
    var COL1;
    output out = Mindist min = minimum; run;
%mend mipd;
```

20.6.2 Fitting the GaSP Model

To use the SAS software, we view GaSP model as a mixed model where all observations are taken on the same subject and are therefore correlated. This approach allows us to fit the GaSP model using `PROC MIXED` (cf. Sect. 17.10).

Table 20.6 shows a SAS program for the analysis of the neuron computer experiment data of Table 20.2, p. 777. The goal of the experiment was to quantify the effect of g_{NaF} and g_{KDR} on a neuron's firing rate.

The 30 sets of data values are read in line by line via the first set of `DATA` statements in Table 20.6. In the second set of `DATA` statements a $101^2 \times 3$ data set `PREDS` is constructed, whose first two columns contain the values of g_{NaF} and g_{KDR} for which we would like to predict the firing rate once the model is fitted. These values consist of a 2-dimensional grid with "mesh size" of 0.01. The mesh size defines the spacing on the grid, so a grid with mesh size 0.01 has points (0, 0), (0, 0.01), (0, 0.02), ..., (1, 0.99), (1, 1). The third column in `PREDS` is y and it has all values set to missing. The third `DATA` statement constructs the new data set `FORANALYSIS` by appending `PREDS` to the original data, so that the data set `FORANALYSIS` has $30 + 101^2$ rows of data. The GaSP model (20.3.1) is fitted with `PROC MIXED`. When fitting the model, the SAS software will use only the observations whose y value is available. If the value for y is missing, the SAS software will use the fitted model to calculate $\hat{y}$ for the observations with missing y values.

There are several options in `PROC MIXED` that require some clarification. By default, SAS software will choose the restricted maximum likelihood as the method of estimation (see Sect. 19.8.3). The `METHOD = ML` option instructs the SAS software to use maximum likelihood estimation (p. 769). By leaving the right side of the `MODEL` equation empty, we instruct the SAS software to fit an intercept only model, i.e. estimate only β_0. If we assumed a richer mean structure, say $E[Y] = \beta_0 + \beta_1 g_{\text{NaF}}$, the

Table 20.5 A SAS program for construction of maximin LHD

```
%macro MmLHD(nrow=, ncol=, iter=);
  %createLHD(n = &nrow, d = &ncol)
  %let design = lhd;
  %mipd(x = &design, d = &ncol)
  data Best; set &design; run; quit;
  data _null_;
    set mindist;
    if _N_ = 1 then call symput("mipd", minimum);
    else stop;
  run; quit;
  %do k = 2 %to &iter;
    %createLHD(n = &nrow, d = &ncol)
    %mipd(x = &design, d = &ncol)
    data _null_;
      set mindist;
      if _N_ = 1 then call symput("mipdnew", minimum);
      else stop;
    run; quit;
    %if &mipdnew > &mipd %then %do;
      %let mipd = &mipdnew;
      data Best; set &design; run; quit;
    %end;
  %end;
  proc datasets library = work noprint;
    delete temp1 - temp&ncol mindist colmins colminslong dist lhd;
  run; quit;
  proc print data = Best noobs; run;
%mend MmLHD;
```

statement would have been `MODEL = g_NaF`. The option `OUTP = PREDICTIONS` will calculate predictions and save them in a data set called `PREDICTIONS`.

The `REPEATED` statement is used to specify the covariance structure. The `SP(EXPA)` choice in the `TYPE` option specifies the "anisotropic exponential spatial" covariance structure which has the structure of (20.3.2) with $R(\boldsymbol{x}_i - \boldsymbol{x}_j|\boldsymbol{\xi})$ being the Power exponential correlation function introduced in Exercise 2. The Gaussian correlation function from (20.3.3) is a special case of the Power exponential correlation function where $p_k = 2$ for $k = 1, 2, \ldots, d$, and will be specified below. The `SP(EXPA)` statement is followed by the list of the input variables ($g_{NaF}\ g_{KDR}$). Then `SUBJECT = INTERCEPT` identifies all data as coming from one subject and forces the SAS software to model the covariance between any two observations, so that no covariance is set to zero.

The `PARMS` statement specifies the initial values for the covariance parameters. We can request a grid of values for each parameter to initiate the optimization algorithm which then searches for the parameter values that maximize the likelihood function. For the Power exponential correlation function, the SAS software expects $2d + 1$ initial values ($\theta_1, \ldots, \theta_d$, $p_1, \ldots, p_d$, and σ^2). In this example, the covariance parameters to be estimated are $\theta_1 (= \theta_{\text{NaF}})$, $\theta_2 (= \theta_{\text{KDR}})$, p_1, p_2, and σ^2. To select the Gaussian correlation function, we need to set p_1 and p_2 equal to 2. Following the `PARMS` statement are the initial values. We must specify the values in the order in which they appear in the SAS output Covariance Parameter Estimates table (see Fig. 20.9). The first set of initial values is a set for θ_1 and, here, we provided a grid of 10 initial values, $\{1, 2, \ldots, 10\}$. For θ_2 we provided a grid of 41 initial

Table 20.6 A SAS program for analysis of the neuron experiment

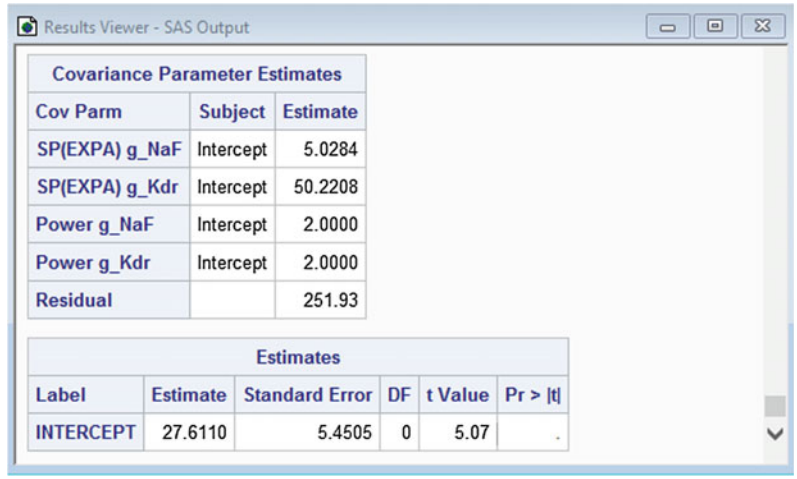
```
DATA NEURON;
  INPUT g_NaF g_KDR y;
  LINES;
  0.38594 0.21207 33
  0.04667 0.45947 0
   :    : :  :     :
  0.77811 0.15813 39
;
* Create a grid of values for prediction and append to the data set;
DATA PREDS;
   DO i = 0 TO 100;
     DO j = 0 TO 100;
       g_NaF = i/100; g_KDR = j/100; y = .;
       OUTPUT;
     END;
   END;
   KEEP g_NaF g_KDR y;

DATA FORANALYSIS; SET NEURON PREDS;

PROC MIXED METHOD = ML;
   MODEL y = / OUTP = PREDICTIONS;
   REPEATED / TYPE = SP(EXPA)(g_NaF g_KDR) SUBJECT = INTERCEPT;
   PARMS (1 TO 10 BY 1) (48 TO 52 BY 0.1) (2) (2) (60) / HOLD = 3, 4
      LOWERB = 1, 45, 2, 2, . UPPERB = 10, 55, 2, 2, .;
   ESTIMATE 'INTERCEPT' INTERCEPT 1;

DATA FORPLOT; SET PREDICTIONS; IF Y = .;
PROC G3D; PLOT g_NaF*g_KDR = Pred; RUN;
```

values, $\{48, 48.1, 48.2, \ldots 52\}$. Next, single starting values of 2, 2, and 60 were provided for p_1, p_2, and σ^2, respectively. The `HOLD=3,4` option tells the SAS software to keep values for the third and fourth parameter fixed, in this case p_1 and p_2, thereby insuring that we are using the Gaussian correlation function. The `LOWERB` and `UPPERB` options set the lower and upper bounds for each parameter. If a bound is missing, then the values for that parameter are unconstrained in that direction. In Table 20.6, the SAS software will run its optimization algorithm for $10 \times 41 \times 1^3$ different starting sets of values with lower bounds 1 and 45, and upper bounds 10 and 55, for θ_1 and θ_2, respectively.

Choosing the starting values is somewhat of a trial and error process. We should choose a relatively coarse grid of starting values that covers the space for each parameter and let the SAS software provide the estimates based on this coarse grid. This initial coarse search will hopefully prevent the algorithm from returning a "local maximum" (e.g. the top of a small hill but missing the top of the nearby mountain). Once these initial estimates are provided, we can construct a fine grid around these estimates to run the search again in this region to obtain the final estimates.

By default the SAS software will not show $\hat{\beta}_0$ in its output window. The `ESTIMATE` statement requires it to do so. The partial output from `PROC MIXED` displaying the parameter estimates for the GaSP model is shown in Fig. 20.9. It shows the maximum likelihood estimates of θ_1 and θ_2, the fixed p_1 and p_2, and the maximum likelihood estimate of σ^2 (labeled "Residual").

The predictions for the data in `PREDS` are calculated and stored in data set `PREDICTIONS`. The last two lines in Table 20.6 are used to make a plot of the predicted response surface plot similar to the


Results Viewer - SAS Output

Covariance Parameter Estimates

Cov Parm	Subject	Estimate
SP(EXPA) g_NaF	Intercept	5.0284
SP(EXPA) g_Kdr	Intercept	50.2208
Power g_NaF	Intercept	2.0000
Power g_Kdr	Intercept	2.0000
Residual		251.93

Estimates

Label	Estimate	Standard Error	DF	t Value	Pr > |t|
INTERCEPT	27.6110	5.4505	0	5.07	.

Fig. 20.9 GaSP model estimates—the neuron experiment

one in Fig. 20.7. The `DATA` statement creates a data set to be plotted by keeping only the predictions. The `PROC G3D` creates the 3D plot (the actual plot is not shown here).

20.7 Using R Software

In this section, we illustrate use of the R software for creating Latin hypercube designs (LHDs) and for analyzing data obtained from computer experiments.

20.7.1 Maximin Latin Hypercube Designs

This section shows how to use R to get an LHD that is approximately maximin (denoted Mm LHD). The idea behind constructing an Mm LHD is described in Sects. 20.4.2 and 20.6.1. While it would have been fairly straight-forward to write our own R code similar to the SAS code shown in Sect. 20.6.1 to construct an approximate Mm LHD, we can use R's package `lhs` for this purpose. To construct an appropriate Mm LHD, we use `lhs`'s function `maximinLHS` which has three inputs. These are n, the planned number of outputs of the computer simulator, k, the number of inputs to the computer simulator, and `dup`, an integer tuning parameter that determines the number of candidate points considered by `maximinLHS` while constructing an Mm LHD. Larger values of `dup` lead to more points considered but also increase the time needed to construct a design. `dup` $= 5$ is suggested as a reasonable choice.

For example, to construct an approximate 30×3 Mm LHD named `Best`, we run the R command `Best <- maximinLHS(30, 3, 5)`. After creating the approximate Mm LHD, the simulator can be run with input combinations $\boldsymbol{x}$ defined by the $n = 30$ rows of the design stored in `Best`. Currently, `maximinLHS` does not support construction of designs whose points are at the midpoint of the cells, so the points in `Best` will be randomly placed within the selected cells.

Table 20.7 An R program and selected output for analysis of the neuron experiment

```
> neuron <- read.table("data/neuron.txt", header = T)
> head(neuron, 3)

       g_NaF   g_KDR  y
[1,] 0.38594 0.21207 33
[2,] 0.04667 0.45947  0
[3,] 1.00000 0.44733 46

> # install.packages("mlegp")
> library(mlegp)
> gasp <- mlegp(neuron[, 1:2], neuron[, 3])
> summary(gasp)

Total observations = 30
Dimensions = 2
mu = 27.6111
sig2: 251.9121

Correlation parameters:
       beta a
1  5.027256 2
2 50.233487 2

Log likelihood = -104.4487

> predictedX <- expand.grid(g_NaF = seq(0, 1, 0.01),
+                           g_KDR = seq(0, 1, 0.01))
> yhats <- predict(gasp, predictedX)
> # install.packages("rgl")
> library(rgl)
> plot3d(neuron[, 1], neuron[, 2], neuron[, 3],
+       col = "red", size = 3, type = "s",
+       xlab = "g NaF (mS/cm^2)", ylab = "g KDR (mS/cm^2)",
+       zlab = "Firing rate (spikes per 2 s)")
> plot3d(predictedX[, 1], predictedX[, 2], yhats, col = "blue",
>       size = 0.5, type = "s", add = TRUE)
```

20.7.2 Fitting the GaSP Model

In this section, we illustrate the analysis of computer experiment data with R software, using the neuron experiment data in Table 20.2, p. 777. The goal of the experiment was to quantify the effect of g_{NaF} and g_{KDR} on a neuron's firing rate.

Table 20.7 contains the R program and selected output illustrating the fitting of the GaSP model to the neuron experiment data. The first lines of Table 20.7 read the data from the file `neuron.txt` and print out the first three rows of the data. The model is then fit using the `mlegp` function from the R package `mlegp`, which computes maximum likelihood estimates (mle) for Gaussian (stochastic) processes (gp). To calculate the maximum likelihood estimates of the GaSP parameters, the `mlegp` function requires two inputs. The first one is the design matrix $\boldsymbol{X}$ defined here by the first two columns of `neuron` data. The second one is the vector of the observed responses $\boldsymbol{y}$ given here in the third column of the `neuron` data set. The `mlegp` function has the capability to fit noisy data, data with

multiple responses, etc. For more information on the `mlegp` function see R's help file by typing `?mlegp`. The model details are stored in `gasp`.

The maximum likelihood parameter estimates can be viewed by invoking the `summary` command. In the resulting output, $\hat{\beta}_0 \approx 27.61$ is labeled as `mu`, and $\hat{\sigma}^2 \approx 251.91$ is labeled as `sig2`. The correlation parameters are given in a $d \times 2$ array where the first and second columns correspond to $\hat{\theta}_k$ and $\hat{p}_k$ for $k = 1, 2, \ldots, d$ but are labeled `beta` and `a`, respectively. $\hat{\theta}_1 \approx 5.03$ and $\hat{\theta}_2 \approx 50.23$. Since we fitted a Gaussian correlation function, $p_1 = p_2 = 2$.

Having fitted the model, we would like to construct the estimated response surface by predicting the response at a dense grid over the $[0, 1] \times [0, 1]$ experimental region. The function `expand.grid` takes two sequences $\{0, 0.01, 0.02, \ldots, 1\}$ of length 101 as inputs and creates a "data frame" whose 101^2 rows represent all combinations of the members of the two sequences (e.g. all combinations of two sequences {.3, .6} would be (.3, .3), (.3, .6), (.6, .3), (.6, .6)). The resulting grid is stored in `predictedX`. To make the predictions `yhats`, we use R's `predict` function, first specifying `gasp` as the object that holds the estimated parameters and then specifying `predictedX` as the object that holds the new prediction sites. Depending upon the computer being used, calculating the `yhats` could take some time. The `plot3d` function from the R graphics library package `rgl` is used to produce the plot shown in Fig. 20.7, p. 778.

Exercises

1. **Gaussian correlation function**

   The Gaussian correlation function $R(\boldsymbol{x}_i - \boldsymbol{x}_j|\boldsymbol{\xi})$ introduced in Sect. 20.3, p. 768, quantifies the correlation between outputs at two points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ based on the distance between them. For parts (a) and (b) below, consider the case of $d = 1$ input variable.

   (a) To investigate the effect of the value of parameter θ on the correlation between outputs at two points, calculate the twenty five correlations $R(x_i - x_j|\theta)$ for $\theta \in \{0.5, 2, 5, 20, 100\}$ and $|x_i - x_j| \in \{0.1, 0.2, 0.4, 0.6, 0.7\}$, where $|x_i - x_j|$ is the absolute value of the distance between x_i and x_j.
   (b) Construct a plot with $|x_i - x_j|$ on the x-axis and $R(x_i - x_j|\theta)$ on the y-axis, plot $R(x_i - x_j|\theta)$ for each value of θ on the same plot, and comment on the relationship between θ and $R(x_i - x_j|\theta)$.

2. **Power exponential correlation function**

   The Gaussian correlation function (20.3.3) is a special case of a Power exponential correlation function. The latter is given by

$$R(\boldsymbol{x}_i - \boldsymbol{x}_j|\boldsymbol{\xi}) = \prod_{k=1}^{d} \exp(-\theta_k |x_{ik} - x_{jk}|^{p_k})$$

   where $\boldsymbol{\xi}$ represents the parameters $(\theta_1, \ldots, \theta_d, p_1, \ldots, p_d)$, with all $\theta_k \geq 0$, and $0 < p_k \leq 2$.

   (a) Suppose there is $d = 1$ input variable. To investigate the effect of θ on the correlation between outputs at two points x_i and x_j for the Power exponential correlation function, calculate the nine

correlations $R(x_i - x_j|\theta)$ for $\theta \in \{0.5, 5, 100\}$ and $|x_i - x_j| \in \{0.1, 0.4, 0.7\}$, for each value of $p \in \{0.5, 1, 1.5, 2\}$.

(b) Construct four plots, one for each value of p, with $|x_i - x_j|$ on the x-axis and $R(x_i - x_j|\theta)$ on the y-axis. Plot $R(x_i - x_j|\theta)$ for each value of θ on the same plot, and comment on the relationship between θ and $R(x_i - x_j|\theta)$ for each value of p.

3. **Cubic correlation function**

The cubic correlation function is given by

$$R(\boldsymbol{x}_i - \boldsymbol{x}_j|\boldsymbol{\xi}) = \prod_{k=1}^{d} R(x_{ik} - x_{jk}|\boldsymbol{\xi})$$

where

$$R(x_{ik} - x_{jk}|\boldsymbol{\xi}) = \begin{cases} 1 - 6\left(\frac{x_{ik}-x_{jk}}{\theta_k}\right)^2 + 6\left(\frac{|x_{ik}-x_{jk}|}{\theta_k}\right)^3 & \text{if } |x_{ik} - x_{jk}| \le \frac{\theta_k}{2} \\ 2\left[1 - \left(\frac{|x_{ik}-x_{jk}|}{\theta_k}\right)^3\right] & \text{if } \frac{\theta_k}{2} < |x_{ik} - x_{jk}| \le \theta_k \\ 0 & \text{if } \theta_k < |x_{ik} - x_{jk}| \end{cases},$$

where $\boldsymbol{\xi}$ represents the parameters $(\theta_1, \theta_2 \ldots, \theta_d)$, and $\theta_k > 0$. Repeat Exercise 1 with the cubic correlation function and $d = 1$.

4. **Bohman correlation function**

The Bohman correlation function is given by

$$R(\boldsymbol{x}_i - \boldsymbol{x}_j|\boldsymbol{\xi}) = \prod_{k=1}^{d} R(x_{ik} - x_{jk}|\boldsymbol{\xi})$$

where $R(x_{ik} - x_{jk}|\boldsymbol{\xi})$ is given by

$$\begin{cases} 1 - \frac{x_{ik}-x_{jk}}{\theta_k} \cos\left(\frac{\pi(x_{ik}-x_{jk})}{\theta_k}\right) + \frac{1}{\pi} \sin\left(\frac{\pi(x_{ik}-x_{jk})}{\theta_k}\right) & \text{if } |x_{ik} - x_{jk}| < \theta_k \\ 0 & \text{if } \theta_k \le |x_{ik} - x_{jk}| \end{cases},$$

and $\boldsymbol{\xi}$ represents the parameters $(\theta_1, \theta_2 \ldots, \theta_d)$, and $\theta_k > 0$. Repeat Exercise 1 with the Bohman correlation function and $d = 1$.

5. **Euclidean interpoint distance**

For two points $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})$ and $\boldsymbol{x}_j = (x_{j1}, x_{j2}, \ldots, x_{jd})$ in a d-dimensional space, the Euclidean distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is defined by (20.4.8), p. 775. For the questions below assume that the range of each input variable is [0, 1].

(a) What is the largest possible distance between two points in (i) a 1-dimensional space? (ii) a 2-dimensional space? (iii) a d-dimensional space?

(b) What is the smallest possible distance between two points in a d-dimensional space? Why would it be undesirable to have two input points with this minimum distance between them in a computer experiment?

6. **Maximin Latin hypercube designs**

Consider the three LHDs shown below. Note that the location of the points is not in the center of each cell but is randomly chosen.

$$\boldsymbol{X}_1 = \begin{bmatrix} 0.66 & 0.65 \\ 0.23 & 0.38 \\ 0.78 & 0.21 \\ 0.38 & 0.98 \end{bmatrix}, \boldsymbol{X}_2 = \begin{bmatrix} 0.50 & 0.68 \\ 0.24 & 0.39 \\ 0.89 & 0.19 \\ 0.35 & 0.92 \end{bmatrix}, \text{ and } \boldsymbol{X}_3 = \begin{bmatrix} 0.98 & 0.10 \\ 0.39 & 0.54 \\ 0.18 & 0.76 \\ 0.57 & 0.39 \end{bmatrix}$$

We can think of *maximin designs* in the following way. Suppose we need to place n grocery stores (design points, corresponding to rows in $\boldsymbol{X}$) in a particular county, where the county is the experimental region, taken as a rectangle determined by the ranges of the input variables x_i and then scaled to $[0, 1]^2$. We would like to choose the store locations in a way that prevents any two stores from being close together. In other words, we are maximizing the minimum distance between the stores and are constructing a maximin design.

(a) For each design, calculate all $\binom{4}{2}$ Euclidean interpoint distances (20.4.8), p. 775, i.e. calculate the distances between each pair of proposed store locations.
(b) For each design, identify the two closest points and their distance.
(c) Between $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, and $\boldsymbol{X}_3$ identify the design that maximizes the minimum interpoint distance; i.e. identify the maximin design.

7. **Minimax designs**

Referring back to the intuitive explanation of the maximin designs in Exercise 6, consider now the distance between each customer and the location of the stores in a county. A reasonable placement of the stores could be such that no customer is too far from the closest store. In other words, we are minimizing the maximum distance of each customer to the closest store. When we are minimizing the maximum distance of any point in the experimental region (input space) from the closest design point, we are constructing a *minimax design.*
Minimax designs are notoriously difficult to construct. When building a maximin design we have to calculate only the $\binom{n}{2}$ distances among the design points but, when constructing a minimax design, we have to consider infinitely many distances (since there are infinitely many points in the experimental region).
Consider the three LHDs from Exercise 6 (i.e. the potential grocery store locations) and suppose that we are interested in the points in the experimental region (i.e. the customer locations) given by

$$\boldsymbol{C} = \begin{bmatrix} 0.59 & 0.10 \\ 0.89 & 0.55 \\ 0.14 & 0.35 \\ 0.38 & 0.90 \\ 0.72 & 0.63 \end{bmatrix}.$$

(a) For each of five points (customer locations) in $\boldsymbol{C}$, determine the closest of the four design points in $\boldsymbol{X}_1$ by calculating four appropriate Euclidean distances (20.4.8), p. 775. Identify a point (customer) in $\boldsymbol{C}$ with the largest distance to its closest point in $\boldsymbol{X}_1$.
(b) Repeat part (a) for $\boldsymbol{X}_2$ and $\boldsymbol{X}_3$.
(c) Identify which of the three designs is minimax for this scenario.

8. **Minimum average reciprocal distance designs**

An alternative measure of space-fillingness is that of *minimum average reciprocal distance*. If design points are spaced out, then the distance between any pair of points will not be small, and so their reciprocal distance will not be large. Thus an alternative to constructing a maximin design is to construct a design with smallest possible average reciprocal distance between pairs of design points. The Euclidean distances between the points are calculated as in (20.4.8), p. 775, and the average of their reciprocals is the measure of goodness of the design.

(a) Of the three designs $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, $\boldsymbol{X}_3$ in Exercise 6, which has the minimum average reciprocal distance? Does this coincide with the maximin design?
(b) Of the three designs $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, $\boldsymbol{X}_3$ in Example 20.4.2, p. 775, which has the minimum average reciprocal distance? Does this coincide with the maximin design?

9. **Prediction**

Suppose that a computer simulator with one input variable was run at 3 input points and the GaSP model with Gaussian correlation function (20.3.3) was fitted to the data:

x	y
0.20	−0.3635
0.50	−0.1353
0.80	−0.0330

leading to parameter estimates $\hat{\beta}_0 = -0.2104$, $\hat{\sigma}^2 = 0.0264$, and $\hat{\theta} = 4.9003$. Based on this information, we would like to predict Y at $x_a = 0.20$ using the predictor in (20.3.4), p. 770.

(a) Calculate $\hat{\boldsymbol{R}}$ (defined below (20.3.4)).
(b) Calculate $\hat{\boldsymbol{r}}$ (defined below (20.3.4)).
(c) Calculate $\hat{Y}(x_a)$ in (20.3.4) at $x_a = 0.20$. Is this the answer you expected? Explain.
(d) Calculate the estimated variance $\hat{s}^2(x_a)$ (20.3.5) of your prediction. Is this the answer you expected? Explain.
(e) Repeat parts (b), (c), and (d) for $x_a = 0.49$ and $x_a = 0.65$.

10. **Tool coating experiment**

D. Draguljić, S. Nekkanty, T. J. Santner, A. M. Dean, and R. Shivpuri, in *Quality Engineering* in 2015, described a computer experiment used to develop multilayer coatings which are used to protect tools, drills, cutting blades, bearings, etc. The experiment consisted of modeling the effect of the number of coating layers and the thicknesses of those layers on the normalized measures of maximum normal radial stress, Y_1, and the maximum shear stress, Y_2. The two responses were modeled independently. Large values of either stress would lead to coating failures (either peeling of of the coating or occurrence of cracks in the coating). Here we will focus on coatings with only two layers. Therefore we have input variables x_1 and x_2 (both in μm), the thicknesses of the top and the bottom layer, respectively. The data for this experiment are given in Table 20.8. Note that x_1 and x_2, as shown in Table 20.8, are scaled from their original (0, 6) μm scale to (0, 1) μm scale.

(a) Estimate θ_1 and θ_2 from the GaSP model that relates x_1 and x_2 to y_1 using the data from Table 20.8.

Table 20.8 Data for the tool coating experiment

x_1	x_2	y_1	y_2
0.6250	0.0833	0.87771	0.13567
0.8750	0.1250	0.84920	0.12590
0.2083	0.2083	0.53123	0.13394
0.0417	0.4583	0.26112	0.15829
0.3750	0.0833	0.75124	0.13708
0.0417	0.8333	2.56179	0.15464
0.1667	0.5417	1.09043	0.22275
0.3750	0.5000	1.75458	0.19860
0.6250	0.3750	1.40447	0.12491
0.4167	0.2917	1.19420	0.13259

(b) For this experiment the total thickness of the coating was required to be between 1/3 and 1, i.e. $1/3 \leq x_1 + x_2 \leq 1$ while the thickness of any individual layer needed to be a multiple of 1/24. Construct a grid with mesh size of 1/24 that satisfies this constraint and predict the values of Y_1 for this grid.
(c) Based on your predictions, what pair of coating thicknesses (x_1, x_2) seems to minimize Y_1?
(d) Repeat parts (a)–(c) for Y_2.
(e) Based on your predictions for Y_1 and Y_2, what single pair of coating thicknesses (x_1, x_2) would you suggest to try to minimize both Y_1 and Y_2 simultaneously (as well as you can)? It may help to make a plot of the predicted values of Y_1 and Y_2.

11. **Borehole function**

A borehole is a narrow tunnel drilled in the ground. Boreholes serve numerous purposes, including extraction of water or gases, mineral exploration, temperature measurement, etc. The *borehole* function (see Surjanovic and Bingham 2013) models water flow through a borehole and is given by

$$y = \frac{2\pi T_u (H_u - H_l)}{a\left(1 + + \frac{2LT_u}{ar_w^2 K_w} + \frac{T_u}{T_l}\right)} \tag{20.7.9}$$

where $a = \ln(r/r_w)$. The output y measures the water flow rate in m^3/year. There are eight inputs to the borehole function. Their names and ranges are given in Table 20.9.

(a) Construct an 80×8 maximin LHD, $\boldsymbol{X}$, where each element of $\boldsymbol{X}$ is in [0, 1]. Let the elements x_{i1} in the first column of $\boldsymbol{X}$ represent the scaled values of the first variable r_w for which we will "observe" (calculate) y from the simulator, the elements x_{i2} in the second column of $\boldsymbol{X}$ represent the scaled values of the second variable r, and so on for all 8 columns.
(b) To be able to calculate the simulator data $y(\boldsymbol{x}_i)$ using (20.7.9), where $\boldsymbol{x}_i$ is the i^{th} row (input combination) of $\boldsymbol{X}$, we need to transform the value of each input variable in $\boldsymbol{X}$ (which has range [0, 1]) to the variables and matching ranges given in Table 20.9. This will allow the appropriate values to be entered into (20.7.9). For transforming x_1 to r_w with range in Table 20.9, the scaling is done via

$$r_w^{[0.05,0.15]} = (0.15 - 0.05)x_1^{[0,1]} + 0.05\,.$$

Table 20.9 Input variables for the borehole function

Variable	Variable name	Range
r_w	Radius of the borehole (m)	[0.05, 0.15]
r	Radius of influence (m)	[100, 50000]
T_u	Transmissivity of the upper aquifer (m^2/yr)	[63070, 115600]
H_u	Potentiometric head of the upper aquifer (m)	[990, 1110]
T_ℓ	Transmissivity of the lower aquifer (m^2/yr)	[63.1, 116]
H_ℓ	Potentiometric head of the lower aquifer (m)	[700, 820]
L	Length of the borehole (m)	[1120, 1680]
K_w	Hydraulic conductivity of the borehole (m/yr)	[9855, 12045]

The superscripts represent the ranges for the input variable x_1 and transformed variable r_w. The scaling for other variables is done in a similar fashion. Scale each value in each row $\boldsymbol{x}_i$ of $\boldsymbol{X}$ to the appropriate range. Calculate the output $y(\boldsymbol{x}_i)$, $i = 1, 2, \ldots, 80$ using (20.7.9).

(c) One of the simple ways to assess how sensitive Y is to the changes in a particular input x_k is to plot y versus the x_{ik}'s and examine the plot for any obvious patterns. Construct eight scatterplots, one for each input variable x_1–x_8, and identify the variables that seem to influence the response the most.

(d) Using $\boldsymbol{X}$ and $y(\boldsymbol{x}_i)$, fit the GaSP model. What are the maximum likelihood estimates of the parameters? Do the values of $\hat{\theta}_k$, $k = 1, 2, \ldots, 8$, support your conclusion from part (c)?

(e) Construct a grid of mesh size roughly equal to 0.5 (i.e. $x_{ik} \in \{0, 0.5, 1\}$) and predict the outputs $y(\boldsymbol{x}_i)$ over the grid. Based on your predictions, what are the values of $x_1, \ldots, x_8$ that result in the predicted maximal water flow? How about the minimal water flow? Using the ranges in Table 20.9, again, scale these values to show the values of the original variables that result in predicted maximal and minimal water flow.