Contents

14

Computer experiments

In a computer experiment we investigate a deterministic function $f(\boldsymbol{x})$ on what may be a high dimensional domain and where f might be very expensive to evaluate. We then have to carefully pick out which input values to use and how to interpolate/extrapolate to the uncomputed values. That extension can then be a cheap surrogate function which we can explore intensively.

The odd thing is that we are used to applying statistical methods in a setting with $Y = f(\boldsymbol{x}) + \varepsilon$. Here we are ordinarily missing the $+\varepsilon$ term which is the usual entry point for statistical methods into a problem setting. What we will do is sample $Y_i = f(\boldsymbol{x}_i)$ for strategically chosen $\boldsymbol{x}_i$, essentially turning numerical problems into statistical ones. The designs we use are ordinarily descibed informally as ***space filling***. When $f(\cdot)$ has no measurement error then we normally turn to interpolation methods to estimate $f(\boldsymbol{x})$ at points where we have not sampled. The most common choices are based on ***kriging*** which stems from modeling $f(\cdot)$ is if it were a randomly chosen continuous function.

For background on computer experiments, see the texts Santner et al. (2018) and Fang et al. (2006) as well as the articles Sacks et al. (1989), Koehler and Owen (1996) and Roustant et al. (2012).

This chapter is based on two lectures. It might have been good to have one lecture for design and another for analysis. That however is not a good fit. The designs and analyses are more naturally developed jointly. This chapter is also a survey of computer experiment ideas; readers will need to follow up in the references for more details.

14.1 Motivating problems

In class we first looked at animations of some computer experiment output. One was a finite element analysis of a car crashing into a post and another showed fuel sloshing in the back of a transport truck that was braking. Similar videos are readily found online though any specific one might disappear. The users of that code can extract quantities of interest like the amount of energy absorbed by the front of the car (and hence not delivered to the passenger compartment) in a collision or the amount of force on the sides and ends of the fuel container. The designer can then vary baffles in the truck or the configuration of the car to get the best tradeoff among numerous quantities of interest.

Codes of this type are used in the design of aircraft, semiconductors and factories. They are also used in modeling of climate and oil reservoirs. For instance we might have a model that predicts how much the arctic will warm at night in 10 years time given some assumptions about CO_2 emissions and reflectivity of clouds among other things. In settings like this we have

$$Y = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d, \quad Y \in \mathbb{R}^q.$$

The function $f(\cdot)$ is usually deterministic and expensive (e.g. 24 hours to run). The number d of input variables may be large and the output dimension q can be large too. The output might even be a curve such as a time trajectory of some scalar quantity.

The authors of these codes put knowledge from physics and chemistry into them in order to describe as faithfully as possible what Y will be like given $\boldsymbol{x}$. The users often have the opposite goal. They may want the $\boldsymbol{x}$ values that give some specific value of Y or that optimize some function $g(Y)$. It is as if they want "$f(\cdot)^{-1}$", the inverse of the physics or chemistry that was used to write f.

In computer experiments we make a strategic choice of input points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ and compute $f(\boldsymbol{x}_i)$. We then use those function values to answer questions about $f(\cdot)$. Those questions usually depend on values like $f(\boldsymbol{x}_0)$ for new points $\boldsymbol{x}_0$ that are not among those n points. We do this by finding an ***emulator*** function $\tilde{f}(\cdot)$. Ordinarily $\tilde{f}(\boldsymbol{x}_i) = f(\boldsymbol{x}_i)$ for $i = 1, \ldots, n$. That is, the emulator usually interpolates the known values exactly. We might be able to evaluate $\tilde{f}$ millions of times if it is fast enough.

There are compelling advantages to computer experiments. They are usually cheaper than physical experiments. Safer too. Design iterations can happen more quickly in computation than physically. For problems involving the future or objects deep in space, computer experiments may be the only option.

An interesting feature of computer experiments for statisticians is that we are very used to $Y = f(\boldsymbol{x}) + \varepsilon$. Now, all we have is $Y = f(\boldsymbol{x})$ without the noise term that is usually the invitation to think statistically. Experimental design has more to offer than just reducing the impact of noise. It also helps one cope with the complexity arising from interactions. In computer experiments, there may be no issue at all about causality. If we change $\boldsymbol{x}$ then $f(\boldsymbol{x})$ changes in a perfectly predictable way. There can still be issues of external validity arising from the assumptions baked into f.

As an example function, consider the ***wing weight function*** from Surjanovic and Bingham (2013). This is available online at `https://www.sfu.ca/~ssurjano` with some code. The function is

$$0.036 S_{\mathrm{w}}^{0.758} W_{\mathrm{fw}}^{0.0035} \Big(\frac{A}{\cos^2(\Lambda)}\Big)^{0.6} q^{0.006} \lambda^{0.04} \Big(\frac{100 t_{\mathrm{c}}}{\cos(\Lambda)}\Big)^{-0.3} (N_{\mathrm{x}} W_{\mathrm{dg}})^{0.49} + S_{\mathrm{w}} W_{\mathrm{p}},$$

It represents the weight of the wing of an aircraft depending on variables defined at Surjanovic and Bingham (2013).

Diaconis (1988) asks whether seeing the functional form is enough to understand a function. We might ask: which variables are most important? how do we get wing weight below some target?, which variables affect the function monotonically and what sizeable interactions are there? Even with a closed form expression like the above it is difficult to address these questions. Usually in computer experiments, we don't even have a formula, just code.

The questions above are actually trick questions too. Their answers depends on set $\mathcal{X}$ of interesting values $\boldsymbol{x}$. We may also need to introduce a distribution on $\boldsymbol{x}$ to pose the questions well. Even with a known distribution for $\boldsymbol{x}$ on $\mathcal{X}$, the point that Diaconis raises remains. It is not obvious how to interpret even a moderately complicated function from its formula. Variable importance is mentioned briefly near the end of this chapter.

14.2 Latin hypercube sampling

We seldom want to take a grid of $N = n^d$ points because the cost grows too quickly with d. We could just take $\boldsymbol{x}_i \sim \mathbb{U}[-1,1]^d$ or $\mathbb{U}[0,1]^d$ or $\mathbb{U}(\mathcal{X})$ for a set $\mathcal{X}$ of interest. We can however do better than this.

The first space filling design to consider is the ***Latin hypercube sample*** of McKay et al. (1979). We spread out $x_{1j}, \dots, x_{nj}$ to fill the range 0 to 1 for each $j = 1, \dots, d$. That is we take $\mathcal{X} = [0,1]^d$. For a Latin hypercube sample of n points in d dimensions we take

$$x_{ij} = \frac{\pi_j(i-1) + u_{ij}}{n} \quad 1 \leqslant i \leqslant n, \quad 1 \leqslant j \leqslant d$$

where π_j are uniform random permutations of $0, 1, \dots, n-1$ and $u_{ij} \sim \mathbb{U}[0,1)$. All d permutations and nd uniform random variables are mutually independent. Many computing environments include methods to make a uniform random permutation. Figure 14.1 shows a small Latin hypercube sample. We see that the n values of x_{ij} for $i = 1, \dots, n$ are equispaced (stratified) and this is true simultaneously for all $j = 1, \dots, d$. It balances nd prespecified rectangles in $[0,1]^d$ using just n points. If any one of those inputs is extremely important, we have sampled it evenly without even knowing which one it was.

A notable strength of a Latin hypercube sample is that it allows $d > n$ or even $d \gg n$. It is also easy to show that each $\boldsymbol{x}_i \sim \mathbb{U}[0,1]^d$. The name of this

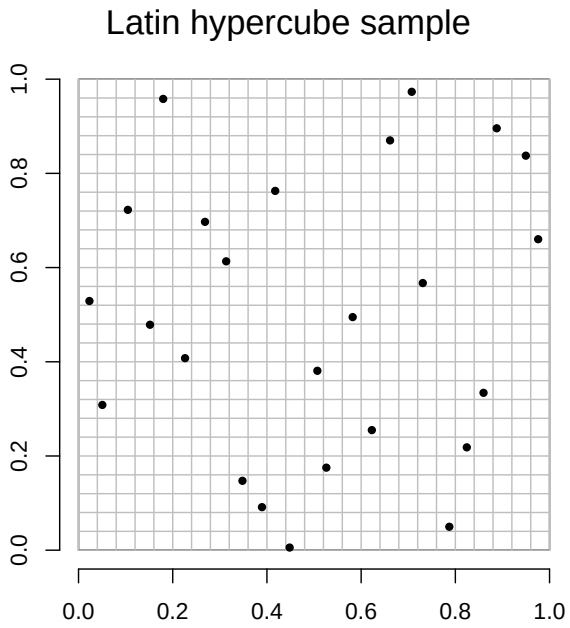

Figure 14.1: This shows a Latin hypercube sample of $n = 25$ points in $d = 2$ dimensions. From Owen (2020).

design is connected to Latin squares. For Figure 14.1, imagine placing letters A, B, C, $\cdots$, Y in the 25×25 grid with each letter appearing once per row and once per column. That would be a Latin square. In an LHS we sample within the cells occupied by just one of those 25 letters, perhaps 'A'. This design is also used in computer graphics by Shirley (1991) who gives it the name ***n rooks*** because if the sampled points were rooks on an $n \times n$ chessboard, no rook could take any other.

Figure 14.1 showed points uniformly and randomly distributed inside tiny squares. If we prefer, we could evaluate f at the centers of those squares by taking

$$x_{ij} = \frac{\pi_j(i-1) + 1/2}{n} \quad 1 \leqslant i \leqslant n, \quad 1 \leqslant j \leqslant d.$$

This centered version of Latin hypercube sampling was described by Patterson (1954) for an agricultural setting (crediting Yates).

14.3 Orthogonal arrays

Orthogonal array designs are a generalization of Latin hypercube samples that allow us to balance more than one input variable at a time in our sampling. Randomized orthogonal arrays, described below, are suitable designs for ex-

ploring deterministic functions. The definitive reference for orthogonal arrays is Hedayat et al. (1999). There is also Neil Sloane's web site `http://neilsloane.com/oadir/` which has results from after the publication of that book as well as some files containing orthogonal arrays.

For an integer base $b \geqslant 2$, a dimension $d \geqslant t \geqslant 1$, and an integer $t \geqslant 1$, the discrete matrix

$$A \in \{0, 1, \dots, b-1\}^{n \times d}$$

is an ***orthogonal array of strength t*** if each $n \times t$ submatrix of A has all b^t possible distinct rows the same number λ of times. The nomenclature for it is $\mathrm{OA}(n, d, b, t)$. It is clear that $n = \lambda b^t$. Often $\lambda = 1$. Geometrically, if we were to pick any t columns of A make a t-dimensional scatter plot, we would get a full b^t grid, with λ copies per point.

In computer experiments, odd things can happen in the "corners". Those are points where two variables are both at extremes be they maxima or minima or a mix of the two. For instance, our codes might crash there. Orthogonal arrays get points into the corners that Latin hypercube samples could miss.

There are many constructions of orthogonal arrays. We will look at one from Bose (1938). It is very useful and the construction can be understood using just one fact about prime numbers. A prime number is an integer $p \geqslant 2$ whose only divisors are 1 and itself. That is not the fact about primes; that's the definition. The fact is that if positive integers a and b are such that ab is a multiple of p, then at least one of a or b is a multiple of p. This follows from the prime factorization theorem.

The Bose construction will give us $\mathrm{OA}(p^2, p+1, p, 2)$. Therefore we can handle $p+1$ variables and every $p \times 2$ submatrix has all p^2 rows once. We will be able to use that design to get into all 4 corners of all $p(p-1)/2$ pairwise scatterplots of our data. We will be sampling in more corners than there are data points. Any two columns of the orthogonal array could be reordered to look like columns labeled x and y in Table 14.1. Exercise: explain why this gives us $p-1$ mutually orthogonal Latin squares.

It is especially convenient that $p = 11$ is prime. Using $p = 11$ we get $n = 121$ points in $[0,1]^{12}$ after rescaling. All bivariate plots are 11×11 grids. We could use round number values 0.0, 0.1, 0.2, $\cdots$, 1.0. Similarly, for $p = 101$ we get $n = 10{,}201$ points in $[0,1]^{102}$. All bivariate plots are 101×101 grids. We could use round number values 0.00, 0.01, 0.02, $\cdots$, 1.00.

The construction goes as follows. We start with the two columns in Table 14.1, and then we adjoin columns $x + y$, $x + 2y$, $x + 3y$, $\dots$, $x + (p-1)y$ all in arithmetic mod p. Note that this construction does not work if p isn't prime. There is a generalization for prime powers $b = p^r$ but when $r > 1$ the generalization does not use arithmetic modulo b.

To see why it works, let's ignore the y column for now (leaving an exercise for later). Then any column can be written $x + cy \bmod p$ for some integer c. It is enough to show that any pair of columns has all p^2 possible vectors $(a_1, a_2) \in \{0, 1, \dots, p-1\}^2$. Since there are only $n = p^2$ rows, each of those vectors must then be present exactly once.

x	y
0	0
0	1
⋮	⋮
0	$p-1$
1	0
⋮	⋮
1	$p-1$
⋮	⋮
$p-1$	0
⋮	⋮
$p-1$	$p-1$

Table 14.1: Here are the first 2 columns of a Bose OA$(p^2, p+1, p, 2)$ orthogonal array. We have labeled them x and y for future use.

Let's pick any two of these columns indexed by c_1, c_2 with $c_1 < c_2$. The (a_1, a_2) that we need satisfies

$$\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 1 & c_1 \\ 1 & c_2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \bmod p$$

where $(x, y)^{\mathsf{T}} \in \{0, 1, \ldots, p-1\}^2$ indexes the row of Table 14.1 that we want.

We can solve this formally by taking

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & c_1 \\ 1 & c_2 \end{pmatrix}^{-1} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \frac{1}{c_2 - c_1} \begin{pmatrix} c_2 & -c_1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}.$$

The matrix multiplication above is well defined in arithmetic modulo p but dividing by the determinant $d = c_2 - c_1$ has to be checked.

First, because $c_1 \neq c_2$ (for the two distinct columns) we have $d = c_2 - c_1 \neq 0 \bmod p$, so the determinant is not zero. Next we show that the value "$1/d$" is an integer $d^{-1} \in \{0, 1, \ldots, p-1\}$ such that $dd^{-1} = 1 \bmod p$, and that there is only one such integer. There are $p-1$ candidates $\{1, 2, \ldots, p-1\}$ for d^{-1}. Suppose that we multiply all of them by d getting $\{d, 2d, \ldots, (p-1)d\} \bmod p$. None of these can be zero because $0 < j, d < p$ with $jd = \bmod p$ would make jd a multiple of p contradicting the prime number fact above. This means that one of them has to be equal to $1 \bmod p$. As for uniqueness, suppose that there are two values $0 < j_1 < j_2 < p$ with $j_1 d = j_2 d = 1 \bmod p$. Then $(j_2 - j_1)d = 0 \bmod p$ which we have already ruled out.

It follows that we get all p^2 possible rows from any two columns of the Bose OA apart from cases where one of those columns is labeled 'y' above. Exercise: use a similar argument for the case of columns y and $x + cy \bmod p$ for $c \neq 0$.

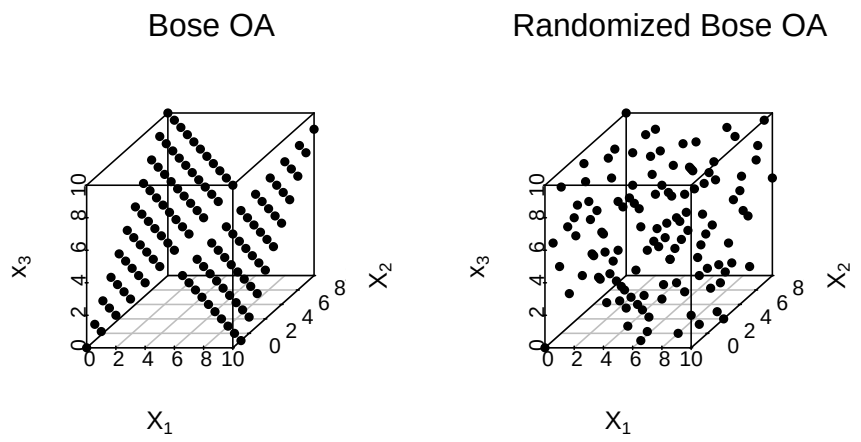

Figure 14.2: The left panel shows three columns of OA$(121, 12, 11, 2)$ unscrambled. The right panel shows a scramble of them. From Owen (2020).

The Bose OA "balances" $\binom{p}{2}p^2 = p^3(p+1)/2$ hyperrectangular subsets of $[0,1]^{p+1}$ with just p^2 points. For $p = 101$ we have balanced more than 5×10^7 strata with only about 10^5 points.

To use an orthogonal array as a space filling design it is important to randomize the levels. In a ***randomized orthogonal array*** (Owen, 1992) we begin with an OA(n, d, b, t) matrix A and take

$$x_{ij} = \frac{\pi_j(a_{ij}) + u_{ij}}{b} \quad \text{or} \quad x_{ij} = \frac{\pi_j(a_{ij}) + 1/2}{b}$$

for independent uniform random permutations π_j of $\{0, 1, \dots, b-1\}$ and $u_{ij} \sim \mathbb{U}[0,1)^d$. Random offsets u_{ij} produce $\boldsymbol{x}_i \sim \mathbb{U}[0,1)^d$ (which is the same distribution as $\mathbb{U}[0,1]^d$). These points are dependent because by construction they must avoid each other in any t-dimensional coordinate projection. The centered versions might be better for plotting contours of f in plane given by x_j and $x_{j'}$ for two variables $1 \leqslant j, j' \leqslant d$. Exercise: is a Latin hypercube sample a randomized orthogonal array?

It is important to apply the permutations π_j. Without permutation, the points will lie in or near two planes and then not fill the space well. See Figure 14.2. Figure 14.3 shows several pairwise scatterplots from a randomized orthogonal array.

Tang (1993) perturbs the points of an orthogonal array slightly, to make them also a Latin hypercube sample. Compared to a randomized orthogonal array the result is more uniform univariate marginal distributions.

There are many more orthogonal arrays to chose from. See Hedayat et al. (1999). As mentioned above, the Bose construction can be generalized to

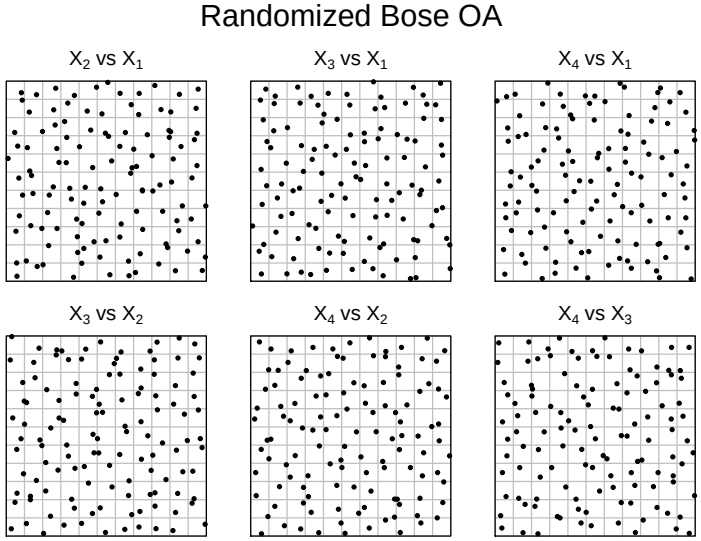

Figure 14.3: Some coordinate projections of randomized orthogonal arrays based on OA$(121, 12, 11, 2)$. From Owen (2020).

OA$(b^2, b+1, b, 2)$ for prime powers $b = p^r$. There are constructions of the form OA$(2b^2, 2b+1, b, 2)$ for $b = p^r$. These constructions from Bose & Bush or Addleman & Kempthorne nearly double the number of variables that can be explored by doubling the number n of function evaluations. Higher strength constructions due to Bush in 1952 produce OA$(b^t, b+1, b, t)$ for prime powers $b = p^r$ with $b \geqslant t-1 \geqslant 0$. There are also arrays with mixed levels. That is the number of distinct values can be different from one column to another. Some can be downloaded from `http://neilsloane.com/oadir/`.

14.4 Exploratory analysis

Given a space filling sample $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, we can compute $Y_i = f(\boldsymbol{x}_i)$ yielding data pairs $(\boldsymbol{x}_1, Y_1), \ldots, (\boldsymbol{x}_n, Y_n)$. We can explore these using statistical and graphical methods. For instance we could select the "good Y values" and see what $\boldsymbol{x}_i$ they have. If f is expensive to compute thane we could also do that for the values $\tilde{f}(\boldsymbol{x}_i)$ for $i = 1, \ldots, N$ with $N \gg n$ once we have worked out how to extend f to an emulator $\tilde{f}$.

Sobol' sequences are a kind of quasi-Monte Carlo (QMC) sampling described below. Like orthogonal arrays there are advantages to scrambling QMC points. Using 1024 scrambled Sobol' points in $[0,1]^{10}$ as inputs to the wing weight function, we obtain Figure 14.4. When taking this many points is infeasible for our original function f, we might do it for an emulator.

It is surprising to see that the range of interesting wing weights has results

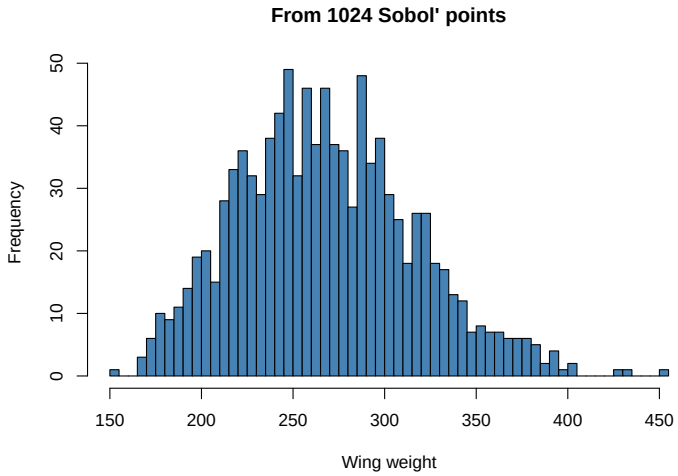

Figure 14.4: Histogram of the wing weight function evaluated at 1024 scrambled Sobol' points.

that vary by a factor of about 3–fold. Perhaps the input ranges are quite wide or there are strong effects in the corners. If we were interested in the lightest wings, then we could select out the points with $f(\boldsymbol{x}_i) < 200$ and plot their input values as in Figure 14.5. In many of the scatterplots we see very empty corners and densely sampled corners. For instance variables 3 and 8 together seem to have a strong impact on whether the wing weight is below 200. Figures like this are exploratory in nature; we might see something we did not anticipate, or we might not see anything that we can interpret.

Figure 14.6 show linear regression output for the wing weight function on the randomized Sobol' inputs. This function is nearly linear with $R^2 \doteq 0.9833$ (adjusted R^2 virtually identical at 0.9831). Because the sampling model is not linear plus noise the usual interpretation of regression output does not apply. Notwithstanding that, this function is surprisingly close to linear, even though the formula did not look linear. By Taylor's theorem, a smooth and very nonlinear function could look locally linear especially in a small region not centered at a point where the gradient vanishes. Reading the variables' described ranges online does not make them appear to be very local. Also, a small region of interest in design space would seem like it ought to restrict the wing's weight to a much narrower range than 3–fold. It appears that this nonlinear looking function is actually close to linear over a wide range of inputs. A plain regression as a quadratic polynomial with cross terms and pure quadratic terms scores an adjusted R^2 of 0.9997. The function is simpler than it looks.

This is by no means what always happens with computer experiments. It is however also quite common that potentially quite complicated functions that come in real applications are not maximally complex. There are additional

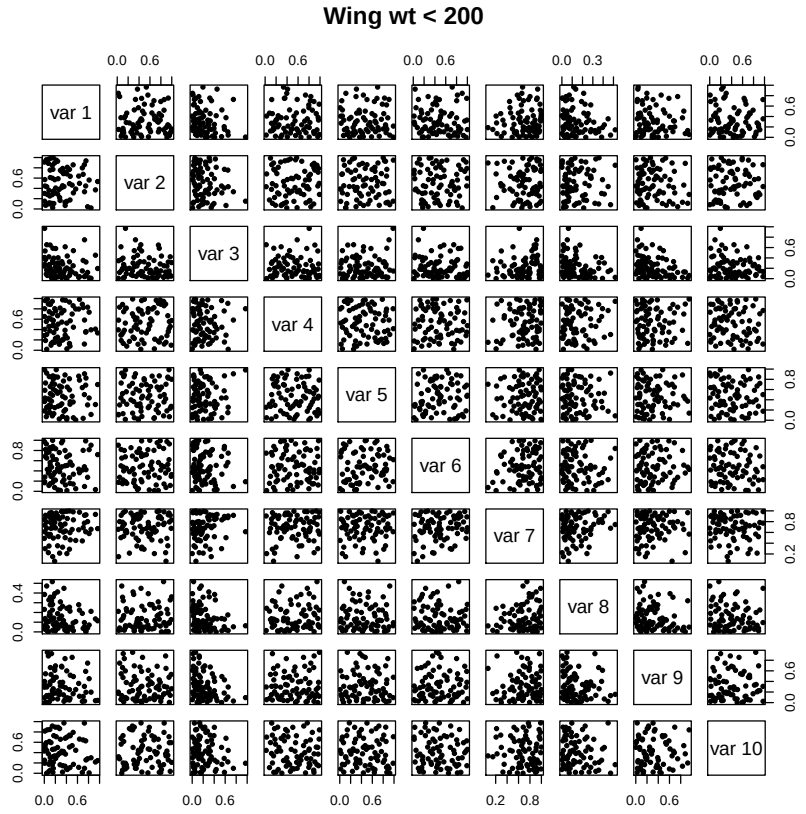

Figure 14.5: Scatterplot matrix of sample points for which the wing weight was below 200.

examples in Constantine (2015). See also Caflisch et al. (1997) who find that a complicated 360-dimensional function arising in financial valuation is almost perfectly additive.

14.5 Kriging

The usual way to predict $f(\boldsymbol{x}_0)$ at a point $\boldsymbol{x}_0 \notin \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ where we have evaluated f is based on a method called kriging from geostatistics. Kriging originated in the work of Krige (1951). Kriging is based on Gaussian process models of the unknown function f. Stein (2012) gives the theoretical background. Sacks et al. (1989) applies it to computer experiments drawing on a body of work developed by J. Sacks and D. Ylvisaker. Much of the description below is based on Roustant et al. (2012) who present software for kriging.

```
> summary(lm( wingwt1024pts ~ rsobo1024pts ))
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)    143.64941    1.08843 131.978   <2e-16 ***
rsobo1024pts1   58.76020    0.67690  86.807   <2e-16 ***
rsobo1024pts2    0.16170    0.67693   0.239   0.8112
rsobo1024pts3   78.22236    0.67691 115.557   <2e-16 ***
rsobo1024pts4   -0.00758    0.67697  -0.011   0.9911
rsobo1024pts5    1.50798    0.67699   2.227   0.0261 *
rsobo1024pts6    7.45687    0.67693  11.016   <2e-16 ***
rsobo1024pts7  -62.02511    0.67697 -91.622   <2e-16 ***
rsobo1024pts8  106.70114    0.67698 157.614   <2e-16 ***
rsobo1024pts9   48.60273    0.67693  71.798   <2e-16 ***
rsobo1024pts10   9.46825    0.67694  13.987   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 6.253 on 1013 degrees of freedom
Multiple R-squared:  0.9833,    Adjusted R-squared:  0.9831
F-statistic:  5956 on 10 and 1013 DF,  p-value: < 2.2e-16
```

Figure 14.6: Some regression output for the wing weight function.

For an intuitive understanding of kriging consider Figure 14.7 and suppose we would like to know how much gold there might be per cubic meter at the location marked '?' given the other measured values in that figure. We would like to estimate it by a weighted average of those values. Because the point labeled 1.9 is closest we should give it more more weight than the point labeled 1.7. The three values somewhat larger than 2 should get more weight than the value 1.1 because there are three of them, and the distances from the target point are similar between them and the point labeled 1.7. They should not get in triple the weight in aggregate because, by being so close together we anticipate that the three readings might be somewhat redundant.

To turn these intuitive notions into a formula for computation we introduce a Gaussian model for the gold values with a covariance function to describe how similar nearby points are to each other.

First we review properties of the ***Multivariate Gaussian*** distribution. We assume that the standard normal distribution $\mathcal{N}(0,1)$ is familiar. If $\boldsymbol{z} \in \mathbb{R}^d$ has independent $\mathcal{N}(0,1)$ components then we write this as $\boldsymbol{z} \sim \mathcal{N}(0, I)$ where here 0 is a vector of d zeroes and I is the d-dimensional identity matrix. The vector $\boldsymbol{x} = \mu + C\boldsymbol{z}$ then has the $\mathcal{N}(\mu, CC^{\mathsf{T}})$ distribution. The general case is written $\boldsymbol{x} \sim \mathcal{N}(\mu, \Sigma)$ where $\mu = \mathbb{E}(\boldsymbol{x})$ and $\Sigma = \mathrm{cov}(\boldsymbol{x}) = \mathbb{E}((\boldsymbol{x}-\mu)(\boldsymbol{x}-\mu)^{\mathsf{T}})$. If $\boldsymbol{x} \sim \mathcal{N}(\mu, \Sigma)$, then $\boldsymbol{x} = \mu + C\boldsymbol{z}$ for some matrix C satisfying $CC^{\mathsf{T}} = \Sigma$. The matrix square root C is not uniquely determined. Linear combinations of

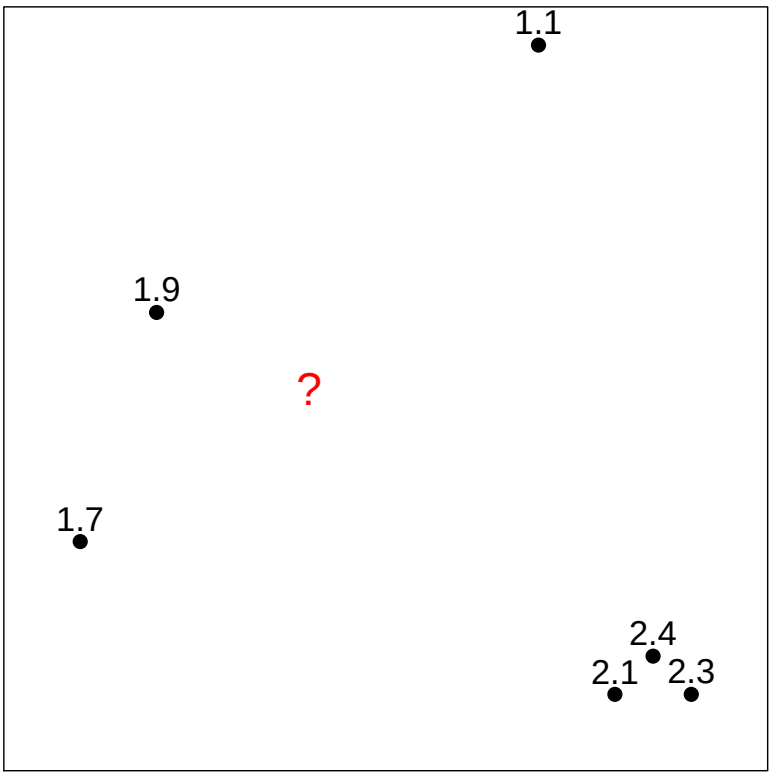

Figure 14.7: Hypothetical setting to illustrate kriging.

Gaussian vectors are also Gaussian: $A\boldsymbol{x}+b \sim \mathcal{N}(A\mu+b, A\Sigma A^{\mathsf{T}})$.

If $\boldsymbol{x}_1 = (x_1,\dots,x_r)^{\mathsf{T}}$ and $\boldsymbol{x}_2 = (x_{r+1},\dots,x_d)^{\mathsf{T}}$ and

$$\boldsymbol{x} = \begin{pmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix}\mu_1\\ \mu_2\end{pmatrix}, \begin{pmatrix}\Sigma_{11} & \Sigma_{12}\\ \Sigma_{21} & \Sigma_{22}\end{pmatrix}\right)$$

then

$$\boldsymbol{x}_1 \perp\!\!\!\perp \boldsymbol{x}_2 \iff \Sigma_{12} = 0.$$

For our present purposes, the most useful property of the multivariate Gaussian distribution is that, if Σ_{22} invertible, then

$$\mathcal{L}(\boldsymbol{x}_1 \,|\, \boldsymbol{x}_2) = \mathcal{N}\Big(\mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(\boldsymbol{x}_2-\mu_2), \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}\Big).$$

This condition is not very restrictive. If Σ_{22} is singular then some component of $\boldsymbol{x}_2$ is linearly dependent on the others. We could just condition on those others if they have a nonsingular covariance. More generally, we need only condition on a subvector of $\boldsymbol{x}_2$ that has an invertible covariance.

In a computer experiment context, we have $Y_i = f(\boldsymbol{x}_i)$, $i = 1, \ldots, n$ and we want to know about $f(\boldsymbol{x}_0)$. We adopt a multivariate Gaussian model

$$\boldsymbol{Y} = \begin{pmatrix} f(\boldsymbol{x}_0) \\ f(\boldsymbol{x}_1) \\ \vdots \\ f(\boldsymbol{x}_n) \end{pmatrix} \sim \mathcal{N}(\mu, \Sigma),$$

with μ and Σ chosen as described below. Then, to predict $f(\boldsymbol{x}_0)$ we may use

$$\tilde{f}(\boldsymbol{x}_0) = \mathbb{E}(f(\boldsymbol{x}_0) \,|\, f(\boldsymbol{x}_1), \cdots, f(\boldsymbol{x}_n)).$$

We also get $\mathrm{var}(f(\boldsymbol{x}_0) \,|\, f(\boldsymbol{x}_1), \cdots, f(\boldsymbol{x}_n))$ for uncertainty quantification, from the multivariate Gaussian distribution.

Because $\boldsymbol{x}_0$ could be anywhere in the set $\mathcal{X}$ of interest, we need a model for $f(\boldsymbol{x})$ at all $\boldsymbol{x} \in \mathcal{X}$. For this, we select functions $\mu(\boldsymbol{x}) = \mathbb{E}(f(\boldsymbol{x}))$ defined on $\mathcal{X}$ and $\Sigma(\boldsymbol{x}, \boldsymbol{x}') = \mathrm{cov}(f(\boldsymbol{x}), f(\boldsymbol{x}'))$ defined on $\mathcal{X} \times \mathcal{X}$. The covariance function $\Sigma(\cdot, \cdot)$ must satisfy

$$\mathrm{var}\left(\sum_{i=1}^{n} a_i f(\boldsymbol{x}_i)\right) = \sum_{i=1}^{n}\sum_{i'=1}^{n} a_i a_{i'} \Sigma(x_i, x_{i'}) \geqslant 0 \tag{14.1}$$

for all $\boldsymbol{a} \in \mathbb{R}^n$ and all $\boldsymbol{x}_i \in \mathcal{X}$ for all $n \geqslant 1$, or else it yields negative variances that are invalid. Interestingly, the condition (14.1) is sufficient for us to get a well defined Gaussian process.

There is an interesting question about in what precise sense is $f(\cdot)$ random? We simply treat it as if the function f were drawn at random from a set of possible functions that we could have been studying. The function is usually chosen to fit a scientific purpose, though perhaps a random function model describes our state of knowledge about $f(\cdot)$. Perhaps not. However, the kriging method is widely used because it often gives very accurate emulator functions $\tilde{f}(\cdot)$. They interpolate because for $1 \leqslant i \leqslant n$,

$$\tilde{f}(\boldsymbol{x}_i) = \mathbb{E}(f(\boldsymbol{x}_i) \,|\, f(\boldsymbol{x}_1), \cdots, f(\boldsymbol{x}_n)) = f(\boldsymbol{x}_i).$$

Knowledge about $f(\cdot)$ can be used to guide the choice of $\mu(\cdot)$ and $\Sigma(\cdot, \cdot)$.

After choosing $\Sigma(\cdot, \cdot)$, we may pick a model like

$$Y = f(\boldsymbol{x}) = \mu(x) + Z(x)$$

where $\mu(\cdot)$ is a known function and Z is a mean zero Gaussian process with covariance Σ. This method is known as ***simple kriging***. For $\mu(\cdot)$, we might choose an older/cheaper/simpler version of the function f.

A second choice, known as ***universal kriging*** takes

$$Y = f(\boldsymbol{x}) = \sum_j \beta_j f_j(\boldsymbol{x}) + Z(x)$$

where once again $Z(\cdot)$ is a mean zero Gaussian process. Here β_j are unknown coefficients, while $f_j(\cdot)$ are known predictor functions. They could for instance be polynomials or sinusoids, or as above, some prior versions of f. Sometimes β_j are given a Gaussian prior, and other times they are treated as unknown constants. That makes $\sum_j \beta_j f_j$ a fixed effect term which in combination with a random effect $Z(\cdot)$ makes this model one of mixed effects. It has greater complexity than simple kriging. Roustant et al. (2012) describe how to analyze this situation.

The third model we consider is ***ordinary kriging***. Here

$$Y = f(\boldsymbol{x}) = \mu + Z(x)$$

for an unknown $\mu \in \mathbb{R}$ and a Gaussian process $Z(\cdot)$. We can view this as Z absorbing all of the $f_j(\cdot)$ from universal kriging. Ordinary kriging is the most commonly used method for computer experiments. It has a simpler theory and estimation strategy than universal kriging.

14.6 Covariance functions for kriging

The usual choice of covariance has $\Sigma(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 R(\boldsymbol{x}, \boldsymbol{x}')$ for a correlation function

$$R(\boldsymbol{x}, \boldsymbol{x}') = \mathrm{corr}(f(\boldsymbol{x}), f(\boldsymbol{x}')).$$

It is also common to choose a ***stationary*** covariance, meaning one with

$$R(\boldsymbol{x}, \boldsymbol{x}') \equiv R(\boldsymbol{x} - \boldsymbol{x}').$$

For $\boldsymbol{x} \in \mathbb{R}^d$ we now have R depending just on a difference vector in $\mathbb{R}^d$ instead of depending on two vectors in $\mathbb{R}^d$. A ***radial*** or ***isotropic*** corelation function takes the form

$$R(\boldsymbol{x}, \boldsymbol{x}') = \rho(\|\boldsymbol{x} - \boldsymbol{x}'\|; \theta)$$

for a parameter θ. While these are commonly used in earth sciences (see Stein (2012)), computer experiments more commonly use a ***tensor product*** model

$$R(\boldsymbol{x}, \boldsymbol{x}') = \prod_{j=1}^{d} \rho_j(x_j - x_j') = \prod_{j=1}^{d} \rho(x_j - x_j'; \theta_j)$$

This model fits with factor sparsity. The value of θ_j may make some x_j very important and others unimportant. It reduces our problem to finding covariances for the $d = 1$ case. If all d correlations $\rho_j(\cdot)$ are valid then so is their tensor product.

One common choice is the ***squared exponential covariance***

$$\rho(s - t; \theta) = \exp(-\theta(s - t)^2)$$

for a parameter $\theta > 0$. This covariance resembles a Gaussian probability density function and so it is sometimes called a Gaussian correlation though that term

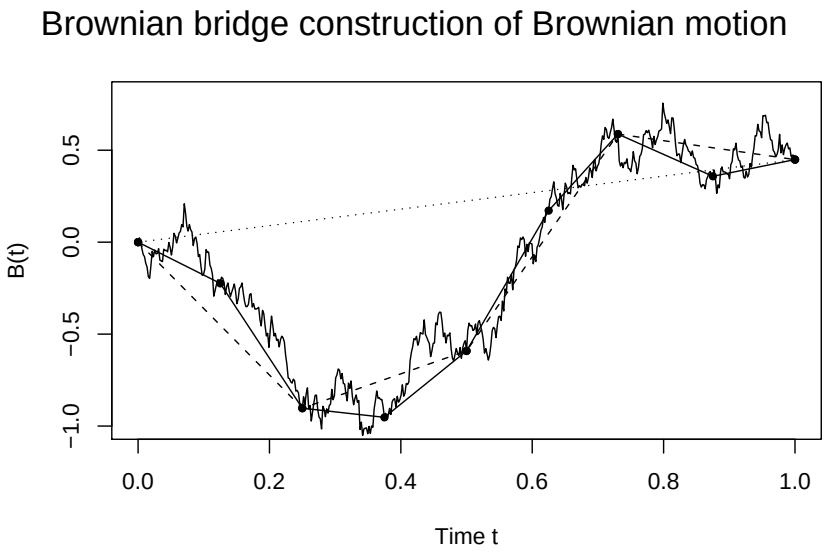

Figure 14.8: A Brownian motion path and some 'skeletons of it'. From Owen (2020).

is potentially confusing since the adjective 'Gaussian' is already being used in another sense to describe the distribution of $f(\boldsymbol{x})$. For $s - t = \epsilon$, we get $\exp(-\theta\epsilon^2) \approx 1 - \theta\epsilon^2$. Nearby points are ***very strongly*** correlated and as a consequence $f(\cdot)$ must be very smooth. Stein (1989) points out that the realizations of f are infinitely differentiable, which is unrealistically smooth for many applications.

The ***exponential covariance*** has $\rho(s-t;\theta) = \exp(-\theta|s-t|)$, for a parameter $\theta > 0$. For $s - t = \epsilon$, we get $\exp(-\theta\epsilon) \approx 1 - \theta\epsilon$. The correlation drops off much faster than for the squared exponential covariance. The realizations of $f(\cdot)$ are much less smooth. They are not even differentiable once, resembling instead Brownian motion. This of course is not smooth enough for many applications. Figure 14.8 shows a sample path of Brownian motion which has a covariance with a decay similar to that of the exponential correlation function. The piecewise linear 'skeletons' there give $\tilde{f}(\boldsymbol{x})$ based on the $(t, f(t))$ values that they connect.

The effect of θ also has implications on the smoothness of the realizations. A large θ implies that the correlation between $f(t)$ and $f(t+\delta)$ drops rapidly as $\delta > 0$ is increased. If we think of t as a time, then the process rapidly forgets $f(t)$ when θ is large. This can induce more rapid oscillations whether the sample paths of f are smooth or rough.

A compromise between exponential and square exponential correlations can be attained via the ***Matern kernels***. They are defined in terms of Bessel functions. When the paramter ν is set to $m + 1/2$, then the realizations have m derivatives and the correlation function has a closed form. Here are the first

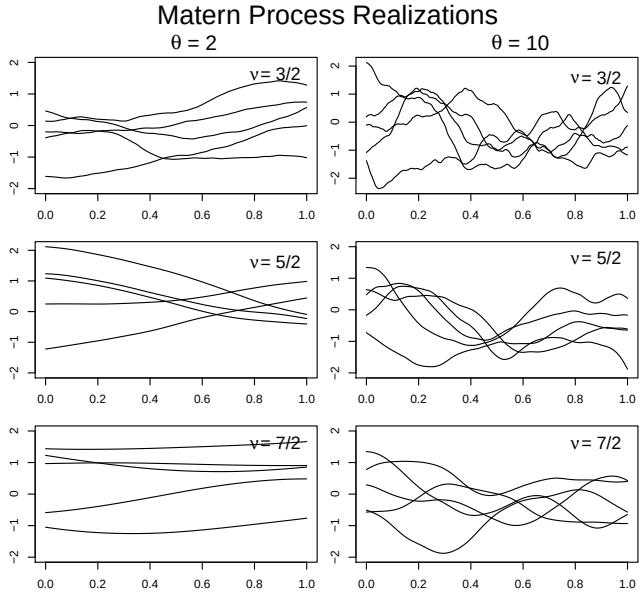

Figure 14.9: Some realizations of Gaussian processes with one dimensional Matern covariances. Larger values of the parameter ν increase their number of derivatives while larger θ value increase the speed at which they oscillate by forgetting their past. From Owen (2020).

four of them:

$$\begin{aligned}
\rho(s,t;1/2) &= \exp(-\theta|s-t|),\\
\rho(s,t;3/2) &= \exp(-\theta|s-t|)(1+\theta|s-t|),\\
\rho(s,t;5/2) &= \exp(-\theta|s-t|)\Big(1+\theta|s-t|+\frac{1}{3}\theta^2|s-t|^2\Big), \quad \text{and}\\
\rho(s,t;7/2) &= \exp(-\theta|s-t|)\Big(1+\theta|s-t|+\frac{2}{5}\theta^2|s-t|^2)+\frac{1}{15}\theta^3|s-t|^3\Big).
\end{aligned}$$

Notice that $\nu = 0$ gives the exponential covariance. Letting $\nu \to \infty$ produces the squared exponential covariance. Figure 14.9 shows some sample paths.

To understand the connection between process smoothness and the correlation function, we proceed informally. We begin with the correlation between f at one point and a divided difference of f some place else:

$$\operatorname{cov}\Big(\frac{f(x+h)-f(x)}{h}, f(\tilde{x})\Big) = \frac{1}{h}\operatorname{cov}\Big(f(x+h)-f(x), f(\tilde{x})\Big).$$

Next, if we let $h \to 0$ on both sides, and are not rigorous about limits, we find that

$$\operatorname{cov}\Big(\frac{\mathrm{d}}{\mathrm{d}x}f(x), f(\tilde{x})\Big) = \frac{\mathrm{d}}{\mathrm{d}x}\rho(x,\tilde{x})$$

A similarly informal argument gives

$$\operatorname{cov}\Big(\frac{\mathrm{d}}{\mathrm{d}x}f(x), \frac{\mathrm{d}}{\mathrm{d}\tilde{x}}f(\tilde{x})\Big) = \frac{\mathrm{d}^2}{\mathrm{d}x\mathrm{d}\tilde{x}}\rho(x,\tilde{x}).$$

For stationary correlations this becomes

$$\operatorname{cov}\Big(\frac{\mathrm{d}}{\mathrm{d}x}f(x), \frac{\mathrm{d}}{\mathrm{d}\tilde{x}}f(\tilde{x})\Big) = \frac{\mathrm{d}^2}{\mathrm{d}x\mathrm{d}\tilde{x}}\rho(x-\tilde{x}) = -\rho''(x-\tilde{x})$$

and then

$$\operatorname{var}\Big(\frac{\mathrm{d}}{\mathrm{d}x}f(x)\Big) = -\rho''(0).$$

The exponential covariance $\rho(\delta) = \exp(-\theta|\delta|)$ is not twice differentiable at the origin and that resulting process does not have a derivative. For a properly rigorous account, see Stein (2012).

14.7 Interpolation, noise and nuggets

Suppose that we obtain $f(\boldsymbol{x}_1) \cdots f(\boldsymbol{x}_n)$ and some partial derivatives like $\partial f(\boldsymbol{x}_1)/\partial x_{1j}$ and even some integrals $\int_A f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$. When f is a Gaussian process for which the derivative exist, then all of these quantities are jointly Gaussian along with any unmeasured function values, derivatives and integrals.

This is a very powerful property. It means that using the recipe

$$\mathbb{E}(\text{any unknowns}\,|\,\text{all the knowns})$$

We get a function $\tilde{f}$ that matches all known values and derivatives and integrals while also being consistently extendable to the unknown values and derivatives and integrals. It is quite common for automatic differentiation codes to deliver gradients along with function values. Kriging can make use of known gradients to better predict function values.

If there is some Monte Carlo sampling embedded in $f(\boldsymbol{x})$, then we may need to model f as having an unknown true value somewhat different from the value we computed. Let's suppose that $Y_i = f(\boldsymbol{x}_i) + \varepsilon_i$ for measurement noise ε_i that is independent of the Gaussian process distribution of f. Then

$$\begin{aligned}\operatorname{cov}(Y_i, Y_{i'}) &= \operatorname{cov}(f(\boldsymbol{x}_i) + \varepsilon_i, f(\boldsymbol{x}_{i'}) + \varepsilon_{i'})\\ &= \operatorname{cov}(f(\boldsymbol{x}_i), f(\boldsymbol{x}_{i'}) + \operatorname{cov}(\varepsilon_i, \varepsilon_{i'})\\ &= \sigma_1^2 R(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) + \sigma_0^2 1_{i=i'}\end{aligned}$$

if we assume that ε_i are IID with mean zero and variance σ_0^2. Here $R(\cdot,\cdot)$ may be any one of our prior correlation functions.

Now suppose that there is sporadic roughness in $f(\cdot)$ such as small step discontinuities that we described as numerical noise above. We can model that using what is called a ***nugget effect***. The covariance is

$$\operatorname{cov}(f(\boldsymbol{x}_i), f(\boldsymbol{x}_{i'})) = \sigma_1^2 R(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) + \sigma_0^2 1_{\boldsymbol{x}_i = \boldsymbol{x}_{i'}}.$$

This looks a lot like the way we handled measurement noise above. What changed is that $1_{i=i'}$ has become $1_{\boldsymbol{x}_i=\boldsymbol{x}_{i'}}$. These would be different if we had two observations $i \neq i'$ with $\boldsymbol{x}_i = \boldsymbol{x}_{i'}$, that is replicates. A nugget effect is a kind of "reproducible noise".

In class we looked at and discussed Figures 1, 2 and 3 from Roustant et al. (2012). Figure 1 shows $\tilde{f}(x)$ and 95% confidence bands. It is simple kriging with a quadratic curve $11x + 2x^2$. The covariance is Matern with $\nu = 5/2$, $\theta = 0.4$ and $\sigma = 5$. Figure 2 shows three different θ. The leftmost panel of Figure 3 shows a mean reversion issue. As we move $\boldsymbol{x}_0$ away from the region sampled by $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ then $\tilde{f}(\boldsymbol{x}_0)$ reverts towards (the esimated value of) μ. The dissertation Lee (2017) provides an alternative form of kriging that reverts toward the nearest neighbor.

The interpolating prediction for simple kriging is

$$\tilde{f}(\boldsymbol{x}_0) = \mu(\boldsymbol{x}_0) + c(\boldsymbol{x}_0)^{\mathsf{T}} C^{-1}(\boldsymbol{Y} - \boldsymbol{\mu})$$

where

$$\begin{aligned} \boldsymbol{Y} &= (f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n)^{\mathsf{T}}, \\ c(\boldsymbol{x}_0) &= (\Sigma(\boldsymbol{x}_1, \boldsymbol{x}_0), \ldots, \Sigma(\boldsymbol{x}_n, \boldsymbol{x}_0))^{\mathsf{T}}, \quad \text{and} \\ C &= \begin{pmatrix} \Sigma(\boldsymbol{x}_1, \boldsymbol{x}_1) & \ldots & \Sigma(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \vdots & \ddots & \vdots \\ \Sigma(\boldsymbol{x}_n, \boldsymbol{x}_1) & \ldots & \Sigma(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{pmatrix}. \end{aligned}$$

This follows from the multivariate Gaussian model. For $\Sigma(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \sigma^2 R(\boldsymbol{x}, \tilde{\boldsymbol{x}})$ we can replace $\Sigma(\cdot, \cdot)$ by $R(\cdot, \cdot)$.

One difficulty with kriging is that the cost of the linear algebra ordinarily grows proportionally to n^3. This may be ok if $f(\cdot)$ is so expensive that only a tiny value of n is possible. Otherwise we might turn to polynomial chaos or quasi-regression. A second difficulty with kriging is that C is very often nearly singular. Indeed that is perhaps very good. For instance if $f(\boldsymbol{x}_n)$ is almost identical to the linear combination of $f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_{n-1})$ that we would have used to get $\tilde{f}(\boldsymbol{x}_n)$ from the first $n-1$ points then C is nearly singular.

Ritter (1995) shows that kriging can attain prediction errors $\tilde{f}(\boldsymbol{x}_0) - f(\boldsymbol{x}_0)$ that are $O(n^{-r-1/2+\epsilon})$ for n evaluations of a function with r derivatives. Here $\epsilon > 0$ hides powers of $\log(n)$. The interpolations in kriging have some connections to classical numerical analysis methods that may explain why they work so well (Diaconis, 1988). When the process is Brownian motion $f(t) = B(t)$, then the predictions are linear splines (in one dimension). For a process that is once integrated Brownian motion $f(t) = \int_0^t B(x)\,\mathrm{d}x$, we find that $\tilde{f}$ is a cubic spline interpolator.

14.8 Optimization

One of the main uses of kriging is to approximately find the optimum of the function $f(\boldsymbol{x})$ on $\mathcal{X}$. That is, we might seek $\boldsymbol{x}_* = \arg\min_{\boldsymbol{x}} f(\boldsymbol{x})$.

Given $f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n)$, we could estimate $\boldsymbol{x}_*$ by $\tilde{\boldsymbol{x}}_* = \arg\min_{\boldsymbol{x}} \tilde{f}(\boldsymbol{x})$ where $\tilde{f}(\boldsymbol{x}) = \mathbb{E}(f(\boldsymbol{x}) \mid f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n))$. However, if we are still searching for $\boldsymbol{x}_*$ then $\tilde{\boldsymbol{x}}_*$ is not necessarily the best choice for $\boldsymbol{x}_{n+1}$. Suppose for instance that a confidence interval yields $f(\tilde{\boldsymbol{x}}_*) = 10 \pm 1$ while at a different location $\boldsymbol{x}'$ we have $f(\boldsymbol{x}') = 11 \pm 5$. Then $\boldsymbol{x}'$ could well be a better choice for the next function evaluation.

The DiceOptim package of Roustant et al. (2012) chooses $\boldsymbol{x}_{n+1}$ to be the point with the greatest ***expected improvement*** as described next. First let $f^* \equiv \min_i f(\boldsymbol{x}_i)$. Then, if $f(\boldsymbol{x}) < f^*$ we improve by $f^* - f(\boldsymbol{x})$. Otherwise, we improve by 0. The expected improvement at $\boldsymbol{x}$ is

$$\mathrm{EI}(\boldsymbol{x}) \equiv \mathbb{E}((f^* - f(\boldsymbol{x}))_+ \,|\, f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n)).$$

There is a closed form expression for EI in terms of $\varphi(\cdot)$ and $\Phi(\cdot)$. We could then take

$$\boldsymbol{x}_{n+1} = \arg\max_{\boldsymbol{x}} \mathrm{EI}(\boldsymbol{x}).$$

Like bandit methods, this approach involves some betting on optimism. The optimization of EI may be difficult because that function could be very multimodal. However, if each evaluation of $f(\cdot)$ takes many hours, then there is plenty of time available to search for the optimum of EI, since it will ordinarily be inexpensive to evaluate. Figure 22 of Roustant et al. (2012) illustrates this process. That paper also describes how to choose multiple candidates for the improvement of EI in case they can be computed in parallel. Balandat et al. (2019) use randomized QMC in their search for the best expected improvement.

14.9 Further designs for computer experiments

Here we consider some optimal designs for computer experiments. There are figures illustrating some of those designs in the dissertation of Koehler and Owen (1996) as well Koehler and Owen (1996). One criterion is to maximize entropy, choosing $\boldsymbol{x}_i$ to

$$\min_{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n} \mathbb{E}(-\log(p(\boldsymbol{Y}))).$$

Figures 8(abc) of Koehler and Owen (1996) show some of these for $d = 2$. Another criterion is

$$\min_{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n} \int_{[0,1]^d} \mathbb{E}((\widetilde{f}(\boldsymbol{x}) - f(\boldsymbol{x}))^2)\, \mathrm{d}\boldsymbol{x}$$

to minimize the integrated mean square error (MISE). Figures 9(ab) show some examples for $d = 2$. The maximum entropy designs place points on the boundary of $[0, 1]^2$ while the MISE designs are internal.

Some other design approaches presented in Johnson et al. (1990) are called ***minimax*** and ***maximin*** designs. They are related to ***packing*** and ***covering*** as described in Conway and Sloane (2013).

The minimax design chooses $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ to

$$\min_{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n} \max_{\boldsymbol{x}_0 \in [0,1]^d} \|\boldsymbol{x}_0 - \boldsymbol{x}_i\|.$$

If you were placing coffee shops at points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ you might want to minimize the maximum distance that a customer (who might be at $\boldsymbol{x}_0$) has to go to get to one of your shops. We can also think of "covering" the cube $[0,1]^d$ with n disks of small radius and centers $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.

The maximin design chooses $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ to

$$\max_{1 \leqslant i \leqslant n} \min_{i' \neq i} \|\boldsymbol{x}_i - \boldsymbol{x}_{i'}\|.$$

In coffee terms, we would not want any two coffee shops to be very close to each other. We can think of this as successfully "packing" n disks into $[0,1]^d$ without any of them overlapping.

Johnson et al. (1990) show that minimax designs are G-optimal in the $\theta \to \infty$ limit. This is the limit in which values of $f(\boldsymbol{x})$ and $f(\boldsymbol{x}')$ most quickly become independent as $\boldsymbol{x}$ and $\boldsymbol{x}'$ move away from each other.

Park (1994) looks a ways to numerically optimize criteria such as the above among Latin hypercube sample designs.

14.10 Quasi-Monte Carlo

Quasi-Monte Carlo methods are designed for numerical integration. They can attain much better convergence rates than plain Monte Carlo does. Niederreiter (1992) and Dick and Pillichshammer (2010) provide comprehensive references.

In quasi-Monte Carlo methods we choose $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ to minimize a "discrepancy". There are many different notions of discrepancy. We can think of them as distances $\big\|\mathbb{U}[0,1]^d - \mathbb{U}\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}\big\|$ between the discrete uniform distribution on our n points and the continuous uniform distribution on the unit cube. As such they have a "space filling" interpretation. See Figure 14.10.

14.11 Variable importance

Variable importance is a harder problem than causal inference. Even when there is absolutely no doubt about the causal effect of $\boldsymbol{x}$ on f we can still debate about ways to measure importance of variables and rank them. Variable importance involves the "causes of effects" that we avoided in the causal inference chapter in favor of studying "effects of causes".

Variable importance is about the extent to which $f(\boldsymbol{x})$ changes when one component x_j is changed. There are local versions called ***sensitivity analysis*** about small changes to components of $\boldsymbol{x}$. There are also global sensitivity analysis methods where components of $\boldsymbol{x}$ are change completely at random.

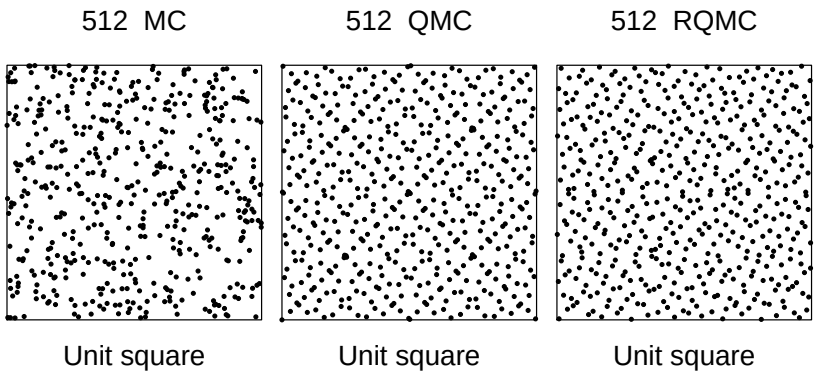

Figure 14.10: The left panel shows 1024 random points. The middle panel shows a Sobol' sequence. The right panel shows some scrambled Sobol' points.

For an introduction to global sensitivity analysis, see the book by Saltelli et al. (2008). Some of the key quantities are Sobol' indices defined in Sobol' (1993).

There is currently a lot of interest in explaining black box functions, such as those in artificial intelligence and machine learning. See Molnar (2018), Ribeiro et al. (2016) and Lundberg and Lee (2017) for an entry to that literature. For some methods that take special care about dependent input variables, see Owen and Prieur (2017) and Mase et al. (2019).

Bibliography

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. (2019). Botorch: Programmable bayesian optimization in pytorch. Technical report, arXiv:1910.06403.

Bose, R. C. (1938). On the application of the properties of galois fields to the problem of construction of hyper-graeco-latin squares. *Sankhyā: The Indian Journal of Statistics*, pages 323–338.

Caflisch, R. E., Morokoff, W., and Owen, A. B. (1997). Valuation of mortgage backed securities using Brownian bridges to reduce effective dimension. *Journal of Computational Finance*, 1(1):27–46.

Constantine, P. G. (2015). *Active subspaces: Emerging ideas for dimension reduction in parameter studies*. SIAM, Philadelphia.

Conway, J. H. and Sloane, N. J. A. (2013). *Sphere packings, lattices and groups*, volume 290. Springer Science & Business Media.

Diaconis, P. (1988). Bayesian numerical analysis. In *Statistical Decision Theory and Related Topics IV, in two volumes*, volume 1, pages 163–176.

Dick, J. and Pillichshammer, F. (2010). *Digital sequences, discrepancy and quasi-Monte Carlo integration*. Cambridge University Press, Cambridge.

Fang, K.-T., Li, R., and Sudjianto, A. (2006). *Design and modeling for computer experiments*. CRC press, Boca Raton, FL.

Hedayat, A. S., Sloane, N. J. A., and Stufken, J. (1999). *Orthogonal arrays: theory and applications*. Springer Science & Business Media, New York.

Johnson, M. E., Moore, L. M., and Ylvisaker, D. (1990). Minimax and maximin distance designs. *Journal of statistical planning and inference*, 26(2):131–148.

Koehler, J. R. and Owen, A. B. (1996). Computer experiments. *Design and analysis of experiments*, 13:261–308.

Krige, D. G. (1951). A statistical approach to some basic mine valuation problems on the witwatersrand. *Journal of the Southern African Institute of Mining and Metallurgy*, 52(6):119–139.

Lee, M. R. (2017). *Prediction and Dimension Reduction Methods in Computer Experiments*. PhD thesis, Stanford University.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774.

Mase, M., Owen, A. B., and Seiler, B. (2019). Explaining black box decisions by shapley cohort refinement. Technical report, arXiv:1911.00467.

McKay, M. D., Beckman, R. J., and Conover, W. J. (1979). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245.

Molnar, C. (2018). *Interpretable machine learning: A Guide for Making Black Box Models Explainable*. Leanpub.

Niederreiter, H. (1992). *Random Number Generation and Quasi-Monte Carlo Methods*. S.I.A.M., Philadelphia, PA.

Owen, A. B. (1992). Orthogonal arrays for computer experiments, integration and visualization. *Statistica Sinica*, 2(2):439–452.

Owen, A. B. (2020). Monte Carlo theory, methods and examples. `https://statweb.stanford.edu/~owen/mc/`.

Owen, A. B. and Prieur, C. (2017). On Shapley value for measuring importance of dependent inputs. *SIAM/ASA Journal on Uncertainty Quantification*, 5(1):986–1002.

Park, J.-S. (1994). Optimal Latin-hypercube designs for computer experiments. *Journal of statistical planning and inference*, 39(1):95–111.

Patterson, H. D. (1954). The errors of lattice sampling. *Journal of the Royal Statistical Society, Series B*, 16(1):140–149.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). Why should I trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, New York. ACM.

Ritter, K. (1995). *Average case analysis of numerical problems*.

Roustant, O., Ginsbourger, D., and Deville, Y. (2012). DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by Kriging-based metamodeling and optimization. *Journal of Statistical Software*, 51(1).

Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P. (1989). Design and analysis of computer experiments. *Statistical science*, pages 409–423.

Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., and Tarantola, S. (2008). *Global Sensitivity Analysis. The Primer*. John Wiley & Sons, Ltd, New York.

Santner, T. J., Williams, B. J., and Notz, W. I. (2018). *The design and analysis of computer experiments*. Springer, New York, second edition.

Shirley, P. S. (1991). *Physically based lighting calculations for computer graphics*. PhD thesis, University of Illinois at Urbana-Champaign.

Sobol', I. M. (1993). Sensitivity estimates for nonlinear mathematical models. *Mathematical Modeling and Computational Experiment*, 1:407–414.

Stein, M. L. (1989). Design and analysis of computer experiments: Comment. *Statistical Science*, 4(4):432–433.

Stein, M. L. (2012). *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media.

Surjanovic, S. and Bingham, D. (2013). Virtual library of simulation experiments: test functions and datasets. `https://www.sfu.ca/~ssurjano/`.

Tang, B. (1993). Orthogonal array-based latin hypercubes. *Journal of the American statistical association*, 88(424):1392–1397.