SIAM/ASA J. Uncertainty Quantification
Vol. 4, pp. 495–519

Hierarchical Emulation: A Method for Modeling and Comparing Nested Simulators*

Rachel H. Oughton† and Peter S. Craig†

Abstract. Computer simulators often contain options to include extensions, leading to different versions of a particular simulator with slightly different input spaces. We develop *hierarchical emulation*, a method for emulating such simulators and for learning about the differences between versions of a simulator. In an example using data from an ocean carbon cycle model, hierarchical emulators outperformed standard emulators both in their predictive accuracy and their coherence with the emulation model. The hierarchical emulator performed particularly well when a comparatively small amount of training data came from the extended simulator. This benefit of hierarchical emulation is advantageous when the extended simulator is costly to run compared to the simpler version.

Key words. Bayesian emulation, computer models, nested simulators, ecosystem models

AMS subject classification. 62

DOI. 10.1137/15M1007914

1. Introduction.

Computer simulators are used in many fields to better understand the behavior of complex systems. Although a simulator is built using experts' understanding of a system, the combination of many subprocesses means that in terms of going from input to output variables, it is often a black box. The number of input variables can be large, and the complexity of the model can make the simulator slow to run. Because of this, it is usually impossible to fully explore the behavior of a simulator (denoted $s(\cdot)$), over its input space x.

Bayesian emulation [3, 7, 13, 8], summarized in section 2, allows us to approximate the simulator's behavior. An emulator is usually much faster to run than the simulator and enables us to make probability statements about the output anywhere in the input space, given some simulator data we have seen and the probability model we specify. Emulators are powerful tools for dealing with continuous input variables.

Two simulators, $s_0(\cdot)$ and $s_1(\cdot)$, of the same system may be *nested* in the sense that the input variables for s_0 are a subset of the input variables for s_1. Alternatively, a single simulator can have one or more "switch" input variables. A switch turns a particular process on or off, or adds additional parameters to a process when on. A switch may be binary or continuous. A continuous switch takes effect through one or more continuous input variables for which there is a *null value* which corresponds to turning the process off or removing parameters. The simulator version with all switches off, s_0, is nested in the version s_1 with all switches on. When there is more than one switch, there is a more complicated nesting structure involving

*Received by the editors February 10, 2015; accepted for publication (in revised form) January 19, 2016; published electronically May 3, 2016.
http://www.siam.org/journals/juq/4/M100791.html

†Department of Mathematical Sciences, Durham University, Durham DH1 3LE, UK (r.h.oughton@durham.ac.uk, p.s.craig@durham.ac.uk). The work of the first author was supported by a UK Natural Environment Research Council doctoral training grant.

simulator versions intermediate between s_0 and s_1 for which a subset of the switches is on.

If turning on switches makes the simulator much more computationally intensive, there may be particular value in seeking to build an emulator of s_1 using more data from s_0, or from intermediate versions, than from s_1. Alternatively, for example if the effect of an added process is unknown or poorly understood, there may be particular interest in understanding the difference between simulator output from s_0 and s_1 when the input variables to s_0 are set to the same values in s_1.

Section 3 presents *hierarchical emulation*, a method for emulating nested simulators, with particular emphasis on situations involving continuous switches. The relationship between s_0 and s_1, and any intermediate simulators, is captured in the emulation structure. The framework of hierarchical emulation presents issues for the choice of prior distribution and for the design of training data, and these are discussed in sections 3.2.1 and 3.2.2.

Any task that a hierarchical emulator can perform can also be done by a "standard" emulator. In section 4 we discuss how to compare the performance of hierarchical and standard emulators. In sections 5–7 we demonstrate hierarchical emulation using HadOCC, an ocean carbon cycle simulator, and compare the performance of hierarchical and standard emulators. Section 7 shows a substantial improvement over standard methods when only a small proportion of the data comes from the more complex version of the simulator. Overall, we find that the hierarchical emulators predict the simulator's behavior more accurately than a standard emulator.

An alternative hierarchical structure is provided in [6] but is restricted to situations where all switches, also known as branching factors, are binary. In [6], simulators need not be nested, and the two values of a switch can add different sets of input variables to the set of input variables common to the two simulator versions. No comparison is made in [6] of the performance of different approaches to emulation.

2. Emulation.

An emulator is a statistical representation of beliefs about a simulator. Instead of giving precise output values for a set $\mathbf{x}$ of input points (as the simulator does), an emulator returns a joint probability distribution, or some other measure of uncertainty, for $s(\mathbf{x})$ (the simulator output) conditional on some known simulator data. This gives an approximation of the simulator's output, and a measure of confidence in that approximation.

Usually an emulator is much faster to run than the simulator. Consequently, the number of input points at which approximate output values can be obtained using the emulator is higher than the number for which it would be feasible to obtain true values from the simulator. If the emulator's approximation is good enough, then it may be used for subsequent analyses.

Begin by representing the simulator, which is assumed here to be scalar-valued, as

$$s(x) = \sum_{j=1}^{q} \beta_j \xi_j(x) + \epsilon(x), \tag{2.1}$$

where $\xi_j(x)$ are q specified regression basis functions intended to account for a substantial part of the variation in simulator output, $\boldsymbol{\beta}$ is a vector of uncertain coefficients, and $\epsilon(x)$ is a correlated error term modeled by a stationary Gaussian process with specified autocorrelation structure and uncertain variance σ^2. We use the Gaussian correlation function, but the hierarchical emulation method would work with other choices. The strength of the correlation

of $\epsilon(x)$ is controlled by some parameters commonly known as "correlation lengths" which determine how rough or smooth the function is. For a collection of input points $\mathbf{x}$ this becomes a linear model $s(\mathbf{x}) = \mathbf{X}\boldsymbol{\beta} + \epsilon(\mathbf{x})$. Here, $s(\mathbf{x})$ is the vector of simulator output values, $\mathbf{X}$ is the model matrix built from the regression basis functions, and $\epsilon(\mathbf{x})$ is the vector of values of $\epsilon(.)$.

Bayesian emulator construction requires *training data*, a collection of n input values $\mathbf{x} = [x_1, \ldots, x_n]$ and a vector of corresponding output values $s(\mathbf{x}) = (s(x_1), \ldots, s(x_n))'$ obtained from the simulator. It also requires a prior distribution for $\boldsymbol{\beta}$ and σ^2. The commonly used normal inverse-gamma prior is conjugate. In the absence of expert judgments with which to choose a prior distribution, we use its weak form $p(\boldsymbol{\beta}, \sigma^2) \propto 1/\sigma^2$.

The aim is to model the simulator's behavior at any collection $\tilde{\mathbf{x}} = [\tilde{x}_1, \ldots, \tilde{x}_m]$ of input points, using the training data. This is equivalent to finding the posterior predictive distribution $s(\tilde{\mathbf{x}}) \mid s(\mathbf{x})$. Using the conjugate prior and treating correlation parameters for ϵ as fixed, $s(\tilde{\mathbf{x}}) \mid s(\mathbf{x})$ is available analytically and is a location-scale multivariate t-distribution with $n-q$ degrees of freedom. This distribution represents both uncertainty about the smooth error term at unobserved values of x and uncertainty about the regression coefficients. The procedure can be extended to emulate multivariate output, and the posterior predictive distribution is derived in [2] for the case of a stationary and separable covariance structure. For the fixed correlation parameters, we use their marginal maximum a posteriori estimates based on a uniform prior, which is equivalent to maximizing the marginal likelihood. The uncertainty lost by using these estimates could be restored, if merited in a particular application, by carrying out some form of Markov chain Monte Carlo (MCMC) sampling for the correlation length parameters. Dropping the explicit dependence on $\tilde{\mathbf{x}}$ (and the training inputs $\mathbf{x}$), we will write $\hat{\mathbf{s}}$ for the emulator's expected value for simulator output $\mathrm{E}\left(s(\tilde{\mathbf{x}}) \mid s(\mathbf{x}) \right)$, and $\hat{\mathrm{V}}$ for the emulator's predictive covariance matrix. Derivation of these quantities is shown in [9].

Often $s(x)$ will be some function of the simulator's raw output, as the latter can be high-dimensional both in time and space. We may choose an average or a collection of time points. It may also be that the emulator will perform better using some transformation of these quantities. To find such a transformation we can use methods such as the Box–Cox model selection procedure.

For example, some emulation choices have been made in this description of emulation. A different correlation function family would easily be accommodated. Others, for example, the authors of [6], might dispense with regression functions and compensate by increasing the complexity of the stationary autocorrelation model for ϵ. This could be accommodated in the procedure described above but at the price of increasing the number of parameters for which posterior uncertainty is neglected. On the other hand, the choice to be Bayesian is fairly fundamental for three reasons: (i) there is no repeatable random experiment being modeled here, and so probability is being used to quantify uncertainty rather than variability; (ii) it accounts for uncertainty about β and σ; (iii) the need in section 3.3 to combine uncertain predictions from multiple emulators requires that emulator predictions and uncertainties be expressed using multivariate probability distributions. Avoiding MCMC by fixing correlation parameters means that the emulation procedure is easy to implement and does not involve a heavy computational burden. As a consequence, the emulator can be used for a wider range of tasks, for example in design calculations for choosing input points at which to run a computationally expensive simulator.

2.1. Validating an emulator. For an input point x whose output we do not know, the emulator's approximation should be plausible in view of our beliefs and the data we have, and the probability distribution should reflect our uncertainty about what the simulator may do at this point. Verifying this requires some care, and here we describe some validation techniques for univariate emulators. More detail is given in [1].

If we observe the actual simulator output $\tilde{\mathbf{s}} = s(\tilde{\mathbf{x}})$, a measure for comparing the predictions with the true values is the *root mean squared error*

$$\mathrm{RMSE}(\tilde{\mathbf{s}}) = \sqrt{\frac{1}{m}\|\tilde{\mathbf{s}} - \hat{\mathbf{s}}\|^2} = \sqrt{\frac{1}{m}\sum_{1}^{m}\Big(s(\tilde{x}_i) - \hat{s}(\tilde{x}_i)\Big)^2},$$

which should be as small as possible and is a measure of the accuracy of the emulator's prediction.

To include the effects of variance, one can find the individual *standardized prediction errors,*

$$\mathrm{SPE}(\hat{s}(\tilde{x}_i)) = (\hat{s}(\tilde{x}_i) - s(\tilde{x}_i)) \Big/ \left(\sqrt{\hat{\mathrm{V}}_{ii}}\right).$$

These are not independent but, for input points which are well spaced relative to the correlation lengths being used, should approximately follow a normal distribution, and, bearing in mind that σ will not be known precisely from the training data, the standard deviation should be close to 1 provided that $n - q$ is large. There should be no observable trends with respect to x. Emulating the SPE itself as a function of x using a more complex regression surface than that of the emulator of the output can expose problems or show whether the regression surface in the output's emulator is appropriate.

In order to account for the covariances between emulator predictions, which the individual prediction errors do not incorporate, the authors of [1] suggest looking at the Mahalanobis distance,

$$\mathrm{MD}(\hat{\mathbf{s}}) = (\tilde{\mathbf{s}} - \hat{\mathbf{s}})' \hat{\mathrm{V}}^{-1}(\tilde{\mathbf{s}} - \hat{\mathbf{s}}).$$

The posterior predictive distribution of $\mathrm{MD}(\hat{\mathbf{s}})$ is a scaled F-distribution with m and $n - q$ degrees of freedom,

$$[(n-q)\,/\,(m(n-q-2))]\,\mathrm{MD}(\tilde{\mathbf{s}}) \mid s(\mathbf{x}) \sim F_{m,n-q}.$$

Any set of new inputs $\tilde{\mathbf{x}}$ will give one Mahalanobis distance, and the outcomes for different sets will be predictively correlated unless they are well separated. Consequently, fit to this distribution cannot easily be checked, but the value for each set can be compared to the distribution to give some diagnostic for how well the emulator fits the simulator. More detail on emulator validation is given in [1].

3. Hierarchical emulation for nested simulators. In the simplest case, there is a single binary switch variable v which can be either on or off. If it is off, the simulator is a function $s_0(x)$, where x is the collection of input variables. If the switch is on, extra input variables w are introduced and the simulator becomes $s_1(x, w)$. If, instead, the single switch is continuous,

the more complex version of the simulator is $s_1(x, v, w)$ and has the property that $s_1(x, v^*, w) = s_0(x)$ when the variables v that control the switch are at the null value v^*.

These two situations and those involving more switches are covered by the following general framework: there is a simulator $s_0(x)$ which can be extended to $s_1(x, v, w)$ and where s_1 reproduces s_0 if the parameters v are set to a particular *null value* v^*. Thus,

$$s_1(x, v^*, w) \equiv s_0(x) \tag{3.1}$$

for all valid values of x and w. The variables w belong only to the extended simulator $s_1(\cdot)$ and do not affect the matching of s_1 with s_0 at $v = v^*$. From here on, v are described as *hierarchical variables* and w as *extra variables*.

This structure is considered, albeit with a different application in mind, in [5] (see especially sections 4.3 and 6.2). They form a relationship between two simulators, one which actually exists, and the other, a "reified" idealization, which is a hypothetical step in making the simulator a better representation of the real system. Below, that structure is used to develop a method to jointly emulate two simulators.

In the standard emulation framework, as described in section 2, the two versions would essentially be emulated as two different functions, making it difficult to compare their behavior closely, and also losing the relationship between the functions at $v = v^*$. We now introduce *hierarchical emulation*, which enables emulation of s_1 in a way that incorporates the information in (3.1) and the structure of multiple switches. In section 5 we give an example using HadOCC, an ocean carbon cycle model containing switch variables.

3.1. A hierarchical emulation structure for multiple switches. Suppose first that there is a single switch, and assume that a suitable *transformation function* $g(\cdot)$, satisfying $g(v^*) = 0 \iff v = v^*$, can be chosen. The two simulator versions are then linked by $s_1(x, v, w) = s_0(x) + g(v)\psi(x, v, w)$, where the new *hierarchical basis function* $\psi(\cdot)$ is determined, for $v \neq v^*$, by $s_1(.)$, $s_0(.)$, and $g(.)$.

For k switches, with hierarchical variables $v_1, \ldots, v_k$ and corresponding null values $v_1^*, \ldots, v_k^*$, the structure generalizes to

$$\begin{aligned} s_1(x, v, w) &= s_0(x) + \sum_{i=1}^{k} g_i(v_i)\psi_i(x, v_i, w) + \sum_{\substack{i=1 \\ j>i}}^{k} g_{ij}(v_i, v_j)\psi_{ij}(x, v_i, v_j, w) \\ &\quad + \cdots + g_{1\ldots k}(v_1, \ldots, v_k)\psi_{1\ldots k}(x, v_1, \ldots, v_k, w) \\ &= s_0(x) + \sum_{[i]} g_{[i]}(v_{[i]})\psi_{[i]}(x, v_{[i]}, w), \end{aligned} \tag{3.2}$$

where the last sum is over all possible nonempty subsets $[i]$ of $\{1, \ldots, k\}$. For each $[i]$, $v_{[i]}$ is the corresponding collection of hierarchical variables; $g_{[i]}(.)$ is the corresponding transformation function; and the corresponding hierarchical basis function $\psi_{[i]}(.)$ is a function of x, $v_{[i]}$, and the extra variables w. Equation (3.2) allows for all possible interactions involving the hierarchical and extra variables. In what follows, we extend the notation for subsets of hierarchical variables to include the case where $[i]$ is empty by taking $\psi(x, w) = s_0(x)$, and

$g() \equiv 1$, so that (3.2) can be rewritten as

$$s_1(x, v, w) = \sum_{[i]} g_{[i]}(v_{[i]})\psi_{[i]}(x, v_{[i]}, w), \tag{3.3}$$

where the sum is now over all subsets $[i]$ of $\{1, \ldots, k\}$.

We require for each subset $[i]$ that

$$g_{[i]}(v_{[i]}) \neq 0 \iff v_j \neq v_j^* \quad \forall j \in [i]. \tag{3.4}$$

The consequence is that, once the transformation functions are specified, the hierarchical basis functions $\psi_{[i]}(.)$ are uniquely defined in terms of $s_1(.)$. When all switches are off, $s_1(x, v, w) = s_0(x) = \psi(x, w)$, which determines $\psi(.)$. When only switch i is on, $g_i(v_i) \neq 0$, and all hierarchical variables other than v_i are set to their null values. Then $s_1(x, v, w) = \psi(x, w) + g_i(v_i)\psi_i(x, v_i, w)$ so that $\psi_i(x, v_i, w) = (s_1(x, v, w) - \psi(x, w))/g_i(v_i)$. Note that $\psi_i(x, v_i^*, w)$ makes no contribution in (3.2) and is therefore effectively irrelevant.

When switches 1 and 2 are on and others are off, we have

$$s_1(x, v, w) = \psi_0(x, w) + g_1(v_1)\psi_1(x, v_1, w) + g_2(v_2)\psi_2(x, v_2, w) + g_{12}(v_1, v_2)\psi_{12}(x, v_1, v_2, w), \tag{3.5}$$

which determines $\psi_{12}(x, v_1, v_2, w)$, since all other quantities are already determined and $g_{12}(v_1, v_2) \neq 0$ when the switches are on. The same process works for any other pair of switches. The hierarchical basis function for any three switches can now be determined by turning on only those switches, and all remaining $\psi_{[i]}$ can be determined by proceeding hierarchically.

A simple way to construct the transformation functions would be first to choose a suitable function $g_i(v_i)$ for each individual switch, satisfying $g_i(v_i) = 0 \iff v_i = v_i^*$. For a binary switch, $v_i^* = 0$, and one can simply take $g_i(v_i) = v_i$. Guidance on choosing a transformation function for a continuous switch is provided in section 3.2.2. Transformation functions for interactions could then be obtained as $g_{[i]}(v_{[i]}) = \prod_{j \in [i]} g_j(v_j)$, although other choices could be made if needed for a particular application. In order to be able to emulate the hierarchical basis functions, they must be continuous and finite, and so the transformation function for a continuous switch should at least be continuous.

For each hierarchical basis function, it is possible that some of the extra variables will be inactive in the sense that changing those inputs does not change the output of the function. This might be the case if, for example, two switches, or groups of switches, switch on different additional processes in the full simulator. This should not cause difficulties when constructing emulators provided that we know which extra variables are inactive for each hierarchical basis function. For consistency, we would also expect that when an extra variable is active in $\psi_{[i]}$ it is also active in $\psi_{[i']}$ if $[i] \subset [i']$.

3.2. Building a hierarchical emulator. Having decomposed $s_1(x, v, w)$ using (3.3), a hierarchical emulator can be built using a collection of standard emulators constructed as in section 2. The hierarchical emulation strategy is to emulate $s_1(x, v, w)$ by emulating each

of the hierarchical basis functions $\psi_{[i]}(x, v_{[i]}, w)$ separately, so that (2.1) generalizes to

$$\psi_{[i]}(x, v_{[i]}, w) = \sum_{j=1}^{q_{[i]}} \beta_{[i],j}\xi_{[i],j}(x, v_{[i]}, w) + \epsilon_{[i]}(x, v_{[i]}, w) \quad \text{for each } [i], \tag{3.6}$$

where the regression basis functions and parameters for $\epsilon_{[i]}$ can be different in each emulator.

Suppose that some observations $\boldsymbol{\psi}_{[i]}$ of the corresponding hierarchical basis function at known inputs are available for each $[i]$. Then each hierarchical basis function emulator can be built separately using the framework in section 2, so that for m new input points $(\tilde{\mathbf{x}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}})$ the posterior distribution

$$\psi_{[i]}(\tilde{\mathbf{x}}, \tilde{\mathbf{v}}_{[i]}, \tilde{\mathbf{w}}) \mid \boldsymbol{\psi}_{[i]} \tag{3.7}$$

can readily be found for each $[i]$. However, some issues need to be addressed: (i) why, and under what conditions, is it legitimate to separate into multiple emulators in this way; and (ii) how are we to combine the separate emulators to find the posterior distribution of $s_1(\tilde{\mathbf{x}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}})$ given the combined data?

It would clearly seem incorrect to do the Bayesian updating separately for each emulator if there was prior dependence between the emulators. If, for a particular application, prior knowledge was available which linked, for example, values of s_0 and ψ_1, that should be taken into account. However, such prior knowledge is likely to be rare, and so we consider situations where the uncertain components $(\beta_{[i]}, \sigma_{[i]}, \epsilon_{[i]})$ for the emulator of $\psi_{[i]}$ are independent a priori of the components for each of the other hierarchical basis function emulators. This makes all the hierarchical basis functions $\psi_{[i]}(\mathbf{x}, \mathbf{v}_{[i]}, \mathbf{w})$ independent a priori for any set of inputs. A consequence is that the prior variance of $s_1(x, v, w)$ increases as v moves away from v^*. Whether or not this is undesirable is not obvious. It is evident in the examples in later sections, but does not appear problematic.

However, even with a priori independence, were we simply to observe $s_1(x, v, w)$ at a single point where each switch was on, it is clear from (3.3) that this would induce a posteriori dependence between all the emulators. On the other hand, if it were possible to make direct observations of hierarchical basis functions and those were the only data, the emulators would remain independent a posteriori if they were independent a priori. We shall see that certain designs for obtaining training data from s_1 are equivalent to making only direct observations of hierarchical basis functions.

We do not claim that the only way to avoid posterior dependence is prior independence combined with the training data design proposed below, but we believe that there is little, if any, useful possibility of doing so in another way. Some further discussion, based on the separability ideas in the supplementary material of [7], may be found in [10]. Maintaining dependence a posteriori is not fundamentally necessary but is advantageous: if observations are made which lead to two of the emulators ceasing to be independent, the prior is no longer conjugate and the joint posterior predictive distribution will involve two uncertain scale parameters and will no longer be a multivariate t-distribution. As a result, MCMC would be needed in order to address uncertainty about the σ parameters for the two emulators.

3.2.1. Design for training data. For training data, $s_1(x, v, w)$ must be observed at specified points. In principle, $s_0(x)$ can also be observed, but this is equivalent to observing $s_1(x, v^*, w)$ for any w, and so the design question is simply about where to evaluate s_1. Two problems are intertwined here: (1) how to avoid posterior dependence between emulators and (2) how to observe values of the hierarchical basis functions. The latter are not simulators and therefore cannot be evaluated directly.

Consider a point (x, v, w) where all switches are on. For each $[i]$, there is a natural corresponding point where only the switches in $[i]$ are on. The corresponding point is obtained by moving the hierarchical variables for switches not in $[i]$ to their null values. At the corresponding point, all terms in (3.3) which involve hierarchical variables for switches not in $[i]$ are zero. More generally, for any point where only switches in $[i]$ are on, there is a corresponding point for each $[i'] \subset [i]$ at which only switches in $[i']$ are on.

Suppose that s_1 is observed at a single point where all switches are on and that, for each $[i]$, it is also observed at the corresponding point. The observations of s_1 at all of those points are then sufficient to determine the value of each hierarchical basis function appearing in (3.3), the expansion of $s_1(x, v, w)$ at the original point. On the other hand, if one knew all those values of the hierarchical basis functions, one would be able to calculate s_1 at the original point and at each of the corresponding points. Because observing s_1 at the original and at all corresponding points is logically equivalent to making a number of observations of hierarchical basis functions, observing those data does not induce a posteriori dependence between the emulators of the hierarchical basis functions.

More generally, consider observing s_1 at some point where only some subset $[i]$ of switches is on. The only hierarchical basis functions $\psi_{[i']}$ which then contribute to the expansion in (3.3) are those where $[i'] \subseteq [i]$. Suppose that s_1 is observed at that point and at all corresponding points where only a subset of the switches in $[i]$ is on. This collection of values of s_1 is equivalent to observing the value of each $\psi_{[i']}$. Consequently, observing such a cluster of points would again not induce dependence between emulators.

Therefore, the fundamental principle of the training design is that if the design includes a point where the switches in $[i]$ are on and all others are off, it must also include, for each $[i'] \subset [i]$, the corresponding point where only the switches in $[i']$ are on. The resulting training dataset (TD) has two equivalent representations: (i) as a collection of values of s_1 at specified inputs and (ii) as a union of training datasets $\mathrm{TD}_{[i]}$ where each $\mathrm{TD}_{[i]}$ is a collection of values of $\psi_{[i]}$ at specified inputs. Updating beliefs about s_1 based on the TD is equivalent to updating beliefs about each $\psi_{[i]}$ using the data in $\mathrm{TD}_{[i]}$.

This principle might seem to lead to prohibitively large designs if there are even a few switches. However, the number of points can be decreased as the order of interaction increases, thereby reducing the total number. For example, s_1 might be observed at a number of points, where all switches are on, and also at all corresponding points. The design could then be augmented by any number of additional observations of s_0. We will see in the example in section 7 that this can produce an emulator that outperforms a standard emulator built with the same data. The design could also be augmented by any number of observations at points where any single switch was on, provided that observations on the corresponding points for s_0 were also included. For situations with more than two switches, augmentations involving groups of switches would be possible.

3.2.2. Choosing the transformation function for a continuous switch. The transformation functions determine the values and distributions of the $\boldsymbol{\psi}_{[i]}$ used to train the emulator of $\psi_{[i]}(.)$. A poor choice could lead to a poor emulator, with especially poor variance estimates for predictions. As indicated earlier, we propose to construct the transformation functions for interactions of switches by multiplying transformation functions chosen for individual switches. Here we discuss how to choose a transformation function for a continuous switch.

A common approach to finding transformations of the independent variable for a regression type model is the Box–Cox family of transformations. This is a power-law family scaled so that it includes a logarithmic transform as the power tends to zero. The power can be estimated using maximum likelihood. However, in finding the transformation functions $g_i(\cdot)$, the plan is not to transform the simulator output but rather the hierarchical variables. The transformation will then affect the definition of the hierarchical basis functions since they are determined by s_1, s_0, and the transformation functions.

Consider first the situation where there is a single switch and the switch is continuous. For clarity, we omit the subscripts for subsets of switches from the notation. Suppose that some parametric family $g(v;\theta)$ of transformation functions has been chosen. We propose using maximum likelihood to estimate θ, taking as data the differences $d(x,v,w) = s_1(x,v,w) - s_0(x) = g(v;\theta)\psi(x,v,w)$ for points in the training design, where the switch is on, and ignoring the correlation structure of $\epsilon(x)$ so that values at distinct points are independent. For any choice of regression functions for the emulator of ψ, the approximate maximum likelihood procedure leads to a computationally straightforward estimation of θ. This is because, given θ, the estimates of β and σ are available in closed form and the profile log-likelihood for θ is easily computed. This approach has a number of advantages: (i) it would be easy to embed within a more complex procedure for choosing regression functions, (ii) the likelihood maximization is more computationally demanding if the autocorrelation of $\epsilon(x)$ is not ignored, and (iii) estimation of correlation parameters in generalized least squares in conjunction with other parameters has been reported to be problematic in geostatistics (see [4]).

The log-likelihood for θ, β, and σ is $-\sum_\ell \log g(v_\ell;\theta) - n\log\sigma - \frac{1}{2\sigma^2}\|\boldsymbol{\psi} - X\beta\|^2$, where ℓ indexes observations, $\psi_\ell = d_\ell/g(v_\ell;\theta)$, and X is the model matrix for the emulator of ψ. Given θ, this is maximized when β is the usual least squares estimate $\hat{\beta} = (X^TX)^{-1}X^T\boldsymbol{\psi}$ and σ^2 is the usual error variance maximum likelihood estimate $\hat{\sigma}^2 = \|\psi - X\hat{\beta}\|^2/n$. Substituting these yields the profile log-likelihood.

Turning to situations with multiple switches, we see that each continuous switch may have its own parameterized transformation function. The overall profile log-likelihood for transformation function parameters is a sum of individual profile log-likelihood terms. Each term is associated with a particular subset of switches. The term for subset $[i]$ is a function of the transformation function parameters for each switch in $[i]$ and takes as data $d_{[i]}(x, v_{[i]}, w) = g_{[i]}(v_{[i]})\psi_{[i]}(x, v_{[i]}, w)$ for points in the training design at which $[i]$ is the set of switches that are on. It will then be necessary to maximize numerically the overall profile log-likelihood. The dimension of the optimization problem will be the total number of transformation function parameters, and this is likely to be roughly proportional to the number of hierarchical inputs.

3.3. The resulting emulator. Suppose that a training dataset has been obtained, the regression functions have been chosen for each emulator, and the transformation functions

have been chosen/estimated. Let $(\tilde{\mathbf{x}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}})$ be a set of new input points. Denoting the diagonal matrix with diagonal $g_{[i]}(\tilde{\mathbf{v}}_{[i]})$ by $D_{[i]}$, we have

$$\mathrm{E}[s_1(\tilde{\mathbf{x}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}}) \mid \mathrm{TD}] = \sum_{[i]} D_{[i]} \hat{\boldsymbol{\psi}}_{[i]}, \tag{3.8}$$

$$\mathrm{cov}[s_1(\tilde{\mathbf{x}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}}) \mid \mathrm{TD}] = \sum_{[i]} D_{[i]} \hat{V}_{[i]} D_{[i]}, \tag{3.9}$$

where $\hat{\boldsymbol{\psi}}_{[i]} = \mathrm{E}[\psi_{[i]}(\tilde{\mathbf{x}}, \tilde{\mathbf{v}}_{[i]}, \tilde{\mathbf{w}}) \mid \mathrm{TD}_{[i]}]$ and $\hat{V}_{[i]} = \mathrm{cov}[\psi_{[i]}(\tilde{\mathbf{x}}, \tilde{\mathbf{v}}_{[i]}, \tilde{\mathbf{w}}) \mid \mathrm{TD}_{[i]}]$ are the predictive mean vector and covariance matrix for the emulator of $\psi_{[i]}$ at the corresponding input points. If more predictive detail is needed than provided by first and second moments, a sample can be drawn from $s_1(\tilde{\mathbf{x}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}})|\mathrm{TD}$ by first sampling from each of the independent hierarchical basis function emulators and then combining those samples using (3.3).

One feature of the hierarchical emulator is that it includes within it an emulator of the difference between the two simulator versions, since $s_1(x, v, w) - s_0(x) = \sum_{[i]} g_{[i]}(\tilde{v}_{[i]}) \psi_{[i]}(\tilde{x}, \tilde{v}_{[i]}, \tilde{w})$, where the sum is over nonempty $[i]$. Having built emulators of the functions $\psi_{[i]}(\cdot)$, we can easily find the predictive mean and variance of the simulator difference for new input points, or sample from the posterior distribution, and thus explore the effect of the extension from $s_0(x)$ to $s_1(x, v, w)$. For simulators having more than one switch, similar possibilities exist for exploring intermediate versions of the simulator.

Building a hierarchical emulator requires many calculations and a lot of sorting and storing of data. To do this effectively requires a careful framework, and so we developed an object-oriented structure for hierarchical emulation in *R* [12].

3.4. Summary of method for building hierarchical emulator.

1. Determine the common, hierarchical, and extra input variables, and for each switch determine v_i^*. It may be necessary to reparameterize the input space, as in the example in section 5.
2. Design the simulator inputs for training data, as in section 3.2.1, and run the simulator at these points to produce the TD.
3. For each continuous switch, use the TD to choose a suitable transformation function $g_i(v_i)$. This will depend on the regression surface being used in each emulator. See section 3.2.2.
4. Use the transformation functions and the TD to find the relevant values of hierarchical basis functions to produce $\mathrm{TD}_{[i]}$.
5. Build emulators for each of the 2^k hierarchical basis functions $\psi_{[i]}(x, v_{[i]}, w)$.
6. Validate the emulators using techniques from section 2.1. If these expose modeling flaws, revisit the previous steps, focusing particularly on the structures of the regression surfaces and correlated errors.
7. Use these emulators, combined with the $g_{[i]}(\mathbf{v}_{[i]})$ vectors, to emulate $s_1(x, v, w)$ and $s_1(x, v, w) - s_0(x)$, as described in section 3.3.

4. Comparing hierarchical emulation with the standard method.

To uncover any benefits deriving from hierarchical emulation, we should compare it to the status quo. Questions we must therefore ask are the following:

1. What tasks are we asking the hierarchical emulator to perform?
2. What are the "standard" emulators against which we will compare the hierarchical emulator?

We discuss these issues below in the general context of a simulator having a single switch, before comparing different emulation strategies using two versions of HadOCC in subsequent sections.

4.1. Tasks for comparison. When the hierarchical emulator is used to emulate $s_0(x)$, all terms in (3.3) apart from the first are "switched off," and we are left with the emulator of $s_0(x)$. Therefore, the standard and hierarchical emulators of $s_0(x)$ will perform identically. Predicting $s_1(x, v, w)$ output is useful in its own right, and in a situation where $s_0(x)$ is much cheaper to run than $s_1(x, v, w)$, building a hierarchical emulator by using many runs from $s_0(x)$ and fewer from $s_1(x, v, w)$ could be an attractive option. Consequently, the relative performance of different emulators of s_1 is of interest.

In comparing the two versions of the simulator, being able to reliably predict the difference between them may be useful. This may make it possible to discern not only the circumstances in which the two versions are very different but also when they behave similarly. Relative performance of different emulators of $s_1(x, v, w) - s_0(x)$ is therefore of interest.

When the switch is continuous, a particular interest lies in how prediction accuracy of different emulators depends on the values of the hierarchical inputs v. This may reveal inappropriate features of some emulators. In particular, it may be interesting to compare predictions of the more complex version's behavior when it is very close to the simpler simulator, that is, when the v are very close to v^*.

4.2. Standard emulators. By "standard" emulators we refer to those built using the methods in section 2, where the emulator is the sum of one regression surface and one stationary autocorrelated error term. In these terms, a hierarchical emulator is a linear combination of standard emulators. In either setting there are choices to be made about prior distribution, regression surface, and correlation function, and, while these are important, they are not the focus of this paper. In building any emulator, these choices should be made to best suit the simulator at hand. Here, we focus on the choice of an independent variable and the use of training data. The set of "standard" emulators, with which hierarchical emulators might be compared, should include the choices that intuitively best suit the tasks we have determined.

First, one could build a standard emulator of s_1. It could be used to predict both $s_1(\tilde{\mathbf{x}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}})$ and $s_0(\tilde{\mathbf{x}}) = s_1(\tilde{\mathbf{x}}, \tilde{\mathbf{v}}^*, \tilde{\mathbf{w}})$. If the predictions were made for both sets of inputs simultaneously, the predictive covariance matrix would enable calculation of the predictive covariance matrix for the difference $s_1(\tilde{\mathbf{x}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}}) - s_0(\tilde{\mathbf{x}})$. This emulator could therefore be used for each of the tasks discussed. An emulator of s_1 could be built either using only the training data, where the switch is on, or using all of the simulator data available. In the examples in section 5, both approaches are applied.

Instead of using an emulator of s_1 only, one could build separate emulators of s_1 and s_0, and then use these to predict $s_1(\tilde{\mathbf{x}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}})$ and $s_0(\tilde{\mathbf{x}})$. The two emulators could be combined to find the expected difference. Bounds on the variance of the difference for each input point could be calculated using the Cauchy–Schwarz inequality, but the exact value $\mathrm{var}[s_1(\tilde{x}, \tilde{v}, \tilde{w}) - s_0(\tilde{x})]$ could not.

Finally, as long as the training data are set up in the correct way, the difference $s_1(x, v, w) - s_0(x)$ can be calculated exactly, and then emulated using standard methods. Intuitively, this might be expected to produce the best emulator of the difference. It would not, however, provide a way to see the difference relative to the values of each output, and so it would be useful only when combined with an emulator of s_0 or s_1.

A truly comparable standard emulator? In comparing the performance of the hierarchical emulator with that of the standard method, it seems appropriate to include, as far as possible, the same information in both emulators. In terms of input data, this can be achieved by using the same training data. However, the very structure of the hierarchical emulator includes the information in (3.1) because of the transformation functions which switch off terms as necessary. A fair question to ask then is, can this same information be included in a standard emulator?

A crucial aspect of this information is that when $v = v^*$, the value of w does not affect the value of $s_1(\cdot)$. This information could be incorporated into a regression function simply by making sure that all terms involving w also involve v in such a way that this is true. However, for (3.1) to hold in the correlated error term, the correlation structure would have to be drastically changed. One method would be to have correlation lengths for w that depend on v such that when $v = v^*$ the extra inputs w add no variance. This seems a sufficiently serious deviation from standard emulation for it to be fair to compare the hierarchical emulator with those in section 4.2 and to consider the capacity to include the information in (3.1) as a benefit, rather than an unfair advantage, of hierarchical emulation.

4.3. A small example. Suppose that $s_0(x) = \sin(8\sqrt{x})$ and that $s_1(x, y, z) = s_0(x) + y^2[\cos(12x) + \sin(9z + 3x)]$, where x, y, and z all lie in [0,1]. Then there is a continuous switch with hierarchical variable y, for which the null value $y^* = 0$, and z is an extra variable.

Emulation of s_0 is straightforward since both function and derivative vary smoothly. Any decent method should perform well.

Emulation of s_1 is much more challenging. The amount of variability of s_1 with respect to changing z (and to some extent x also) is strongly dependent on y. An emulator which does not incorporate this fact is likely to have too high a predictive variance for y near 0, especially for points where only z differs from an observation point, and too low a predictive variance for y near 1. Moreover, when the predictive variance is unnecessarily high, the predictions themselves are likely to be unnecessarily inaccurate.

A standard emulator, with or without basis functions, has a stationary autocorrelated error and so cannot incorporate the heterogeneity described in the previous paragraph. Basis functions can help capture the right mean behavior but do not address the heterogeneity of variability.

A hierarchical emulator can perform much better, provided that a suitable transformation function $g(y)$ is chosen. It is not necessary that $g(y)$ be exactly some multiple of y^2, but it does need to capture that behavior for small y and to be a reasonable approximation to y^2 throughout $[0, 1]$. Then the emulator will automatically build in the heterogeneity referred to previously.

Note that emulation of $s_1(x, y, z) - s_0(x)$ is more challenging: the heterogeneity issue now applies to the variability due to changing either x or z. Both cases are strongly influenced by the value of y.

5. An example: HadOCC. HadOCC [11] is a compartmental ocean carbon cycle model that models the concentrations of four tracers: nutrient (N), phytoplankton (P), zooplankton (Z), and detritus (D). The output variables are given as time series over a year and include quantities such as particulate organic nitrogen, total carbon, and concentrations of the four tracers.

HadOCC has several switch parameters, which are listed in the supplementary material. One that is relevant to this method is `rcchlopt`, which enables us to change the carbon:chlorophyll (C:Chl) ratio. The C:Chl ratio can be made constant (by setting `rcchlopt = 0`) or varying (by setting `rcchlopt = 1`). In the case where C:Chl varies, two new parameters `rcchlmin` and `rcchlmax` are introduced, setting the minimum and maximum of the ratio. HadOCC's continuously varying inputs are given in the supplementary material.

In either case, a parameter `rcchl` sets the initial value of C:Chl (this remains the same if C:Chl is constant). If C:Chl is varying and we set `rcchlmin = rcchl = rcchlmax`, then HadOCC behaves exactly as with C:Chl constant at the same rate. So, with some reparameterization, we can have a v^* such that $s_0(x) = s_1(x, v^*, w)$. Therefore, these versions of HadOCC can be used to build a hierarchical emulator. We choose `iz.chl`, the chlorophyll concentration, to be the output because it is strongly affected by the C:Chl ratio, and therefore by the switch variable `rcchlopt`. We hoped that this would lead to nonnegligible values for the $\psi_{[i]}()$ functions.

Although the two versions of HadOCC described above are identical for certain input points, we cannot immediately discern the hierarchical variables v and extra variables w. For this, the inputs must be reparameterized. Two valid parameterizations using `rcchlmin`, `rcchl`, and `rcchlmax` are introduced here, either of which could be used to build a hierarchical emulator.

Cuboid parameterization. One difference between the possible parameterizations is the shape of the new input space. This parameterization takes the three inputs related to C:Chl (`rcchlmin`, `rcchl`, and `rcchlmax`) and produces three more, R, m_1, and m_2, whose ranges form a cuboid. These are defined by

$$\begin{aligned} R &= \texttt{rcchlmax} - \texttt{rcchlmin} \in [0, M_+ - M_-), \\ m_1 &= \frac{\texttt{rcchlmin} - M_-}{M_+ - M_- - R} \in [0, 1], \\ m_2 &= \frac{\texttt{rcchlmax} - \texttt{rcchl}}{R} \in [0, 1], \end{aligned}$$

where M_- and M_+ are the minimum and maximum, respectively, for `rcchlmin`, `rcchl`, and `rcchlmax`, given that each has the same range.

R is the only hierarchical variable and $R^* = 0$, since the two versions of HadOCC are the same at $R = 0$. When $R = 0$, `rcchlmin = rcchl`, and $m_1 = (\texttt{rcchl} - M_-)/(M_+ - M_-)$, and so m_1 replaces `rcchl` in the common inputs. Finally, m_2 is an extra variable and exists only when $R \neq 0$.

Noncuboid parameterization. In this parameterization, we keep `rcchl` and introduce two new variables,

$$d_{min} = \texttt{rcchl} - \texttt{rcchlmin} \in [0,\, \texttt{rcchl} - M_-]\,,$$
$$d_{max} = \texttt{rcchlmax} - \texttt{rcchl} \in [0,\, M_+ - \texttt{rcchl}]\,.$$

These are both hierarchical variables, since we must have $d_{min} = d_{max} = 0$ to achieve

$$\texttt{rcchlmin} = \texttt{rcchl} = \texttt{rcchlmax}.$$

This parameterization creates a more complicated input space, whose shape depends on `rcchl`.

By contrast, generating input Latin hypercubes in the cuboid parameterization is simple. In fact, because of the constraint in the original input space that $\texttt{rcchlmin} \le \texttt{rcchl} \le \texttt{rcchlmax}$, generating designs is more straightforward in the cuboid input space than in the original input space. That said, the cuboid parameterization may create scaling issues. When R is small, the effect of m_1 could potentially be large, since the small range can be anywhere within $[M_-, M_+]$, but the effect of m_2 is limited, because there is only a small range in which `rcchl` can sit. When R is large, this is reversed. Whether or not this is a problem is not immediately obvious. Another advantage of this parameterization is that having only one hierarchical variable, R, means that the training data design criteria do not force us to have too great a number of points.

6. Equal data from both versions of the simulator. Here we show some hierarchical emulators of the two versions of HadOCC built using the cuboid parameterization, and we compare them to some standard emulators.

Training data. The training data for this example were formed using a 1,000 point Latin hypercube over the reparameterized input space of $s_1(x, R, m_2)$, where x denotes 26 continuously varying parameters which are common to both versions of HadOCC. HadOCC was run at each point, and the annual mean `iz.chl` calculated. To satisfy the design requirements, the annual mean of `iz.chl` was also found for $s_0(x)$ at each point. This gave an input design of 2,000 points, which we refer to as lhd1. The subdesign containing points at which $R = 0$ will be lhd1_0, and the subset of points at which $R \neq 0$ will be lhd1_1, and each of these contains 1,000 points. Note that the corresponding points in lhd1_0 and lhd1_1 share common m_1 values rather than common `rcchl` values. This means we must be careful and make sure to use the reparameterized input variables when building standard emulators of the difference.

Emulation choices. This study requires many emulators, and for them to be as comparable as possible we decided to fix the type of emulator. Therefore, in this section the regression terms are all first order, with all input variables included, and the correlated error terms are isotropic, with the correlation length found using maximum marginal likelihood. To make the isotropic correlation more appropriate, the inputs are all first rescaled to the same interval. None of these choices is necessary for the hierarchical emulator to work, and indeed, one should usually make such choices carefully for the problem at hand. These are also general issues in emulation about which there is not a consensus. We are not suggesting that our specifications are somehow correct; rather, they are a way of avoiding tuning which might unintentionally favor one emulation method over another.

The relationship between the annual mean `iz.chl` and some of the more influential input variables suggested that the logarithm of the annual mean would be an appropriate choice of output variable, and this was confirmed by the Box–Cox procedure. Therefore, from now on $s_0(\cdot)$ and $s_1(\cdot)$ denote the logarithm of the annual mean of `iz.chl`.

For the transformation function for R, we chose the family $g(R) = [\log(1 + \kappa(R))/\kappa]^\lambda$ which has two positive parameters κ and λ. This $g(v)$ behaves like the power law R^λ over the range of R when κ is small enough. For larger κ, it has the same power law behavior for R close to 0 but grows less rapidly than the power law for higher R. The damping effect for higher R increases as κ increases. One might also like the possibility for more extreme growth at larger R, but we felt it was less likely and not needed for the example.

Using the profile log-likelihood described in section 3.2.2, we found that the optimization headed toward the boundary where $\kappa = 0$ with λ tending at the same time to 0.7781, and so we set $g(R) = R^{0.7781}$.

Validation data. HadOCC is relatively quick to run, and so we produced a large dataset with which to validate our emulators. The design lhd6 was formed using a one million point Latin hypercube built using the staggered Latin hypercube design (LHD) method introduced by [10], with $c = 1{,}000$ and $m = 1{,}000$. This means that it breaks down into 1,000 sub-LHDs, each containing 1,000 points. Each of these points was matched by a corresponding point with $R = 0$. HadOCC was then run at all two million points to produce both s_1 and s_0 data. For each emulator we can therefore produce 1,000 sets of prediction summaries for each version of HadOCC, that is, one for each sub-LHD.

Standard emulator performance. Performance summaries of emulators of $s_0(x)$, $s_1(x, R, m_2)$, and $(s_1(x, R, m_2) - s_0(x))$ are given in the supplementary material. The "best standard emulator" choices are summarized in Table 1. The best standard emulator of $s_0(x)$ uses the cuboid parameterization and only the data lhd1_0 from $s_0(x)$, and therefore is exactly the emulator used for the first term of the hierarchical emulator. Diagnostics for s_0 are therefore omitted from the hierarchical emulator tables.

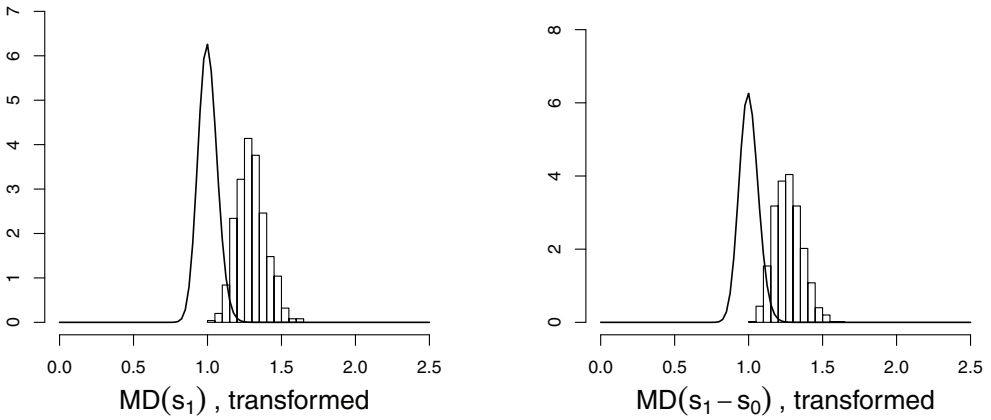
Diagnostics are summarized by their minima, maxima, mean, and standard deviation. Plots are also given and show interesting results. Because the Mahalanobis distance combines so much information, it will not be used to choose between methods, although the values will be shown for some emulators.

For the first three tasks, Tables 1 and 2 give a clear indication of the best strategy. The emulator built using lhd1_0 ($s_0(x)$ data only) and the cuboid parameterization outperforms each of the other standard emulators at predicting $s_0(x)$.

Table 1

Summaries for the best standard emulators of s_0, s_1, and $s_1 - s_0$ used over each of the 1,000 sub-LHDs in lhd6.

	Min.	Max.	Mean	SD
Cuboid inputs, lhd1_0				
RMSE(s_0)	0.113	0.137	0.125	0.00437
Mean SPE(s_0)	-0.0502	0.103	0.0261	0.0268
Variance SPE(s_0)	0.940	1.43	1.18	0.0804
Cuboid inputs, lhd1_1				
RMSE(s_1)	0.126	0.157	0.140	0.00510
Mean SPE(s_1)	-0.0705	0.120	0.0268	0.0301
Variance SPE(s_1)	0.940	1.44	1.15	0.0817
Difference data				
RMSE($s_1 - s_0$)	0.0941	0.115	0.103	0.00357
Mean SPE($s_1 - s_0$)	-0.114	0.0979	0.00878	0.0304
Variance SPE($s_1 - s_0$)	0.948	1.39	1.12	0.0706



(a) MD(s_1), standard emulator of s_1 built from lhd1_1.

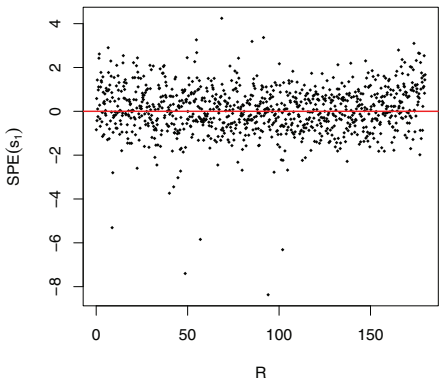
(b) MD($s_1 - s_0$), standard emulator of $s_1(\cdot) - s_0(\cdot)$ built from difference data.

Figure 1. *Transformed Mahalanobis distances for the predictions of s_1 (left) and $s_1(\cdot) - s_0(\cdot)$ (right) for the best standard emulator. The corresponding F-distribution densities are shown by solid lines.*

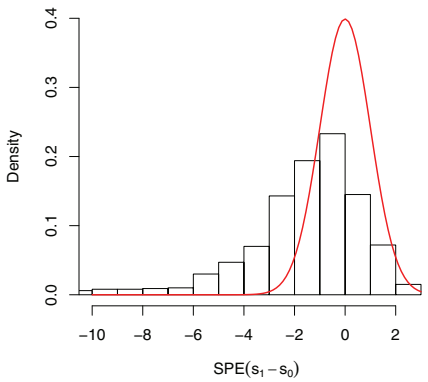
The best method for predicting $s_1(\cdot)$ is the cuboid parameterization built with the training data lhd1_1. The difference is best predicted by the direct emulator of the difference between logs calculated from the data lhd1. In most cases, standard emulators that perform well at one of the tasks perform poorly at the others, where they can achieve them at all.

Although the predictions from the lhd1 emulator are quite accurate, the SPE and MD summaries show that the emulator's variance is not as it should be according to the model. Figures 1a and 1b show the Mahalanobis distances, transformed to fit the F-distribution as described in section 2.1, compared with the true F-distribution, and the correspondence is poor.

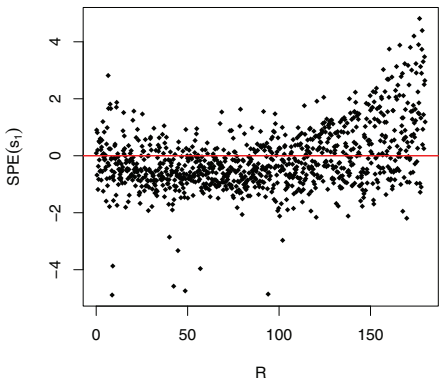
The plots in Figure 2 illustrate the standard emulators' performance as the hierarchical input R changes. Panels (a) and (b) show the SPE for the standard emulator of $s_1(\cdot)$ built using lhd1_1. The SPE for s_1 and the SPE for $s_1 - s_0$ (not shown) both have a slight trend with R, and the distribution of the SPE values for the difference is biased. Panels (c) and (d) show SPE versus R, for s_1 and $s_1 - s_0$, for the emulator built from lhd1. This time, both show a considerable trend to SPE against R, particularly the difference prediction. Other plots not included here showed a similar pattern for several of the other standard emulators, especially the emulator built from the difference data, with SPE values becoming more variable with increased R.



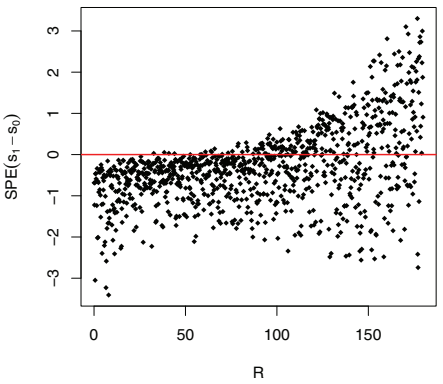
(a) Standard $s_1(\cdot)$ emulator, built using lhd1_1, used to predict s_1.



(b) Standard $s_1(\cdot)$ emulator, built using lhd1_1, used to predict $s_1(\cdot) - s_0(\cdot)$, with the $N(0, 1)$ density plotted as a line.



(c) Standard $s_1(\cdot)$ emulator, built using lhd1, used to predict s_1.



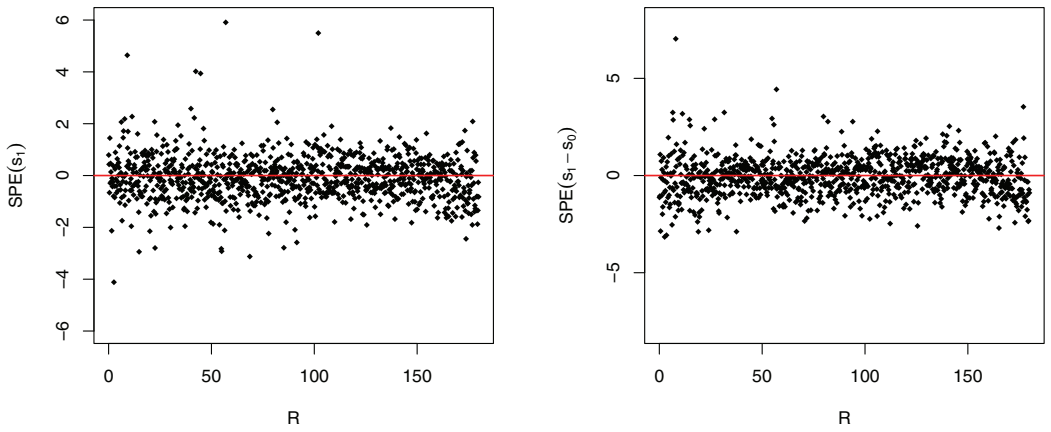
(d) Standard $s_1(\cdot)$ emulator, built using lhd1, used to predict $s_1(\cdot) - s_0(\cdot)$.

Figure 2. *Examples of poor behavior in the standardized prediction errors (SPEs) for predictions of* s_1 *(plots* (a) *and* (c)*) and* $s_1(\cdot) - s_0(\cdot)$ *(plots* (b) and (d)*) for some standard emulators. The prediction dataset used for these plots is a pair of* 1,000 *point sub-LHDs from the lhd*6 *data.*

Table 2

Diagnostics summaries for the hierarchical emulator with $g(R) = R^{0.7781}$ used over lhd6.

	Min.	Max.	Mean	SD
RMSE(s_1)	0.113	0.142	0.125	0.00471
Mean error s_1	-0.0159	0.00332	-0.00694	0.00356
Mean SPE(s_1)	-0.0343	0.110	0.0424	0.0247
Variance SPE(s_1)	0.621	1.05	0.768	0.0598
RMSE($s_1 - s_0$)	0.0792	0.103	0.0902	0.00400
Mean error $s_1 - s_0$	-0.0127	0.00396	-0.00402	0.00250
Mean SPE($s_1 - s_0$)	-0.0334	0.144	0.0497	0.0271
Variance SPE($s_1 - s_0$)	0.858	1.37	1.06	0.0782



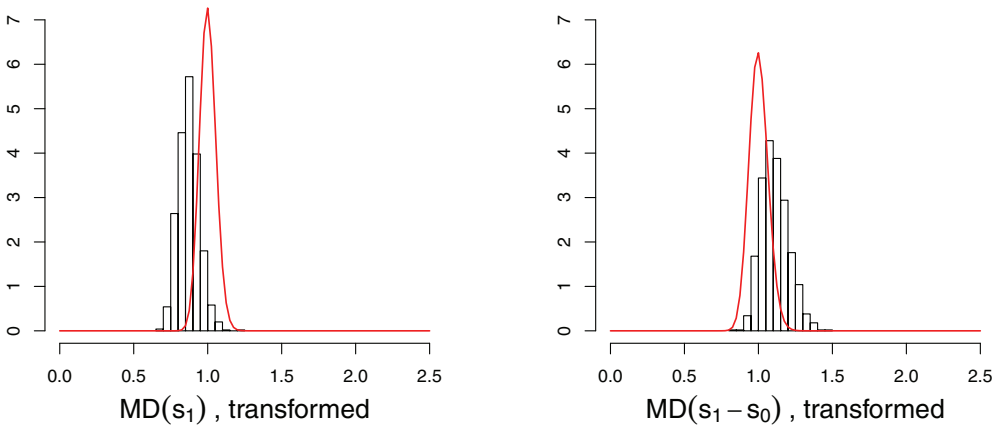
(a) Predicting s_1. **(b)** Predicting $s_1(\cdot) - s_0(\cdot)$.

Figure 3. *SPE for predictions made using the hierarchical emulator with $g(R) = R^{0.7781}$ over one of the sub-LHDs from lhd6.*

Hierarchical emulators. The diagnostic summaries for the hierarchical emulator are shown in Table 2. The predictions are more accurate using a hierarchical emulator than with the best standard alternatives listed in Table 1, shown by the smaller RMSE values for both s_1 and $s_1 - s_0$. Recall that the summaries for the hierarchical emulator's predictions of s_0 will be identical to that of the best standard emulator shown in Table 1.

Although the improvement is modest, showing roughly a 10% reduction in RMSE, it is achieved by a single emulator. When compared to any single standard emulator, the improvement is more striking.

The SPE values for the hierarchical emulator or $s_1(\cdot)$ are promising, with mean and variance consistently close to 0 and 1, respectively, although the variance tends to be somewhat less than 1. Figure 3 shows the behavior of the SPE with R for the hierarchical emulator. Mahalanobis distances are shown in Figure 4 and indicate a much better fit than that for



(a) MD(s_1), hierarchical emulator. **(b)** MD($s_1 - s_0$), hierarchical emulator.

Figure 4. *Transformed Mahalanobis distances for the predictions of* s_1 (a) *and* $s_1(\cdot) - s_0(\cdot)$ (b) *for the hierarchical emulator with* $g(R) = R^{0.7781}$. *The corresponding* F*-distribution densities are shown by solid lines.*

the corresponding standard emulators. Strictly speaking, the F-distribution shown in panel (a) is not appropriate, as the hierarchical emulator is effectively the sum of two independent emulators, each of which has an uncertain error variance; however, we show the F-distribution for a single emulator to give some feel for the variability expected.

A smaller training dataset. In order to see if the conclusions would be similar for a smaller training set, a set of 100 points was selected from the original 1,000 s_1 input points. The 100 points were chosen to have an acceptable minimum distance and provide a close-to-orthogonal design for the main effects. Tables of emulator performance are provided in the supplementary material. The hierarchical emulator is still the best for emulating s_1 in terms of RMSE, although the variance of standardized prediction errors is a bit low. It is also the best emulator of the difference $s_1 - s_0$, and there the SPE variance is satisfactory. The standard emulator of s_0 inside the hierarchical emulator is still the best emulator of s_0. Overall, the hierarchical emulator remains the best.

7. Working with reduced $s_1(\cdot)$ data. It is possible to build a hierarchical emulator using training data containing many runs of $s_0(x)$ and comparatively few of $s_1(x, v, w)$. This is particularly helpful when $s_1(x, v, w)$ is more costly to run than $s_0(x)$.

We built a hierarchical emulator using 1,000 input points for the simpler version of HadOCC, and only 100 for the extended version. These 100 runs from s_1 are all matched in their x and m_1 values by a point in the s_0 data, and therefore the design satisfies the criteria in section 3.2.1. They were taken from a 2,000 point design, so that other emulators could be built with 1,000 points each in the s_0 and s_1 input spaces.

The emulator was constructed using the same choices as the previous ones in this example, namely, a first order regression surface involving all terms and a single estimated correlation length for each level of the hierarchy. The transformation function $g(R) = R^{0.7781}$ was used.

Three other emulators were also built for comparison with this reduced s_1 emulator. A standard emulator was built using the 100 s_1 and 1,000 s_0 points (i.e., the same reduced s_1 data as the hierarchical emulator), and another using just the 1,000 s_1 points.[1] A second hierarchical emulator was built from the full 2,000 point design.

We suspected that when a small number of s_1 data were used, the emulator would perform considerably better "closer to s_0," i.e., for smaller values of R. This feature has not manifested in the emulators with equal numbers of points from both versions. The errors (emulator prediction minus simulator output) for each emulator are plotted against R in Figure 5 (for emulators of s_1) and Figure 6 (for emulators of $s_1 - s_0$).

The standard emulators summarized in Figures 5a and 6a use the same data as the hierarchical emulators in Figures 5c and 6c. For predictions of $s_1(\cdot)$, this is the 1,100 point design. For predictions of the difference, only those points evaluated by both $s_0(\cdot)$ and $s_1(\cdot)$ can be used, so the design is reduced to 200 points.

There is a clear reduction in the error for the hierarchical emulator, particularly for small R. Improvement is greatest in the hierarchical emulators of the difference, where the hierarchical structure enforces the relationship between the simulator versions, so that the difference when $R = 0$ is always zero. The error gradually increases with R (see Figure 6c) until its accuracy appears roughly the same as the standard alternative.

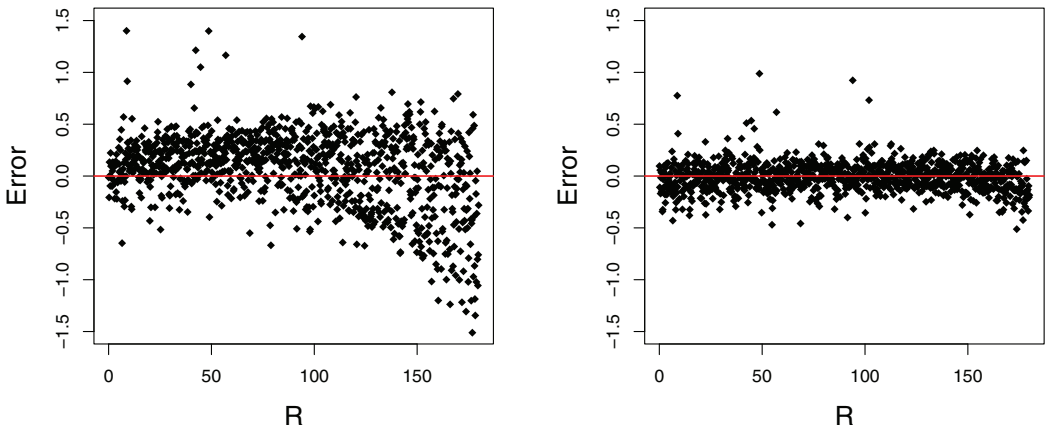
Figure 7 shows how the RMSE changes as the range of R in the data changes. Because the number of points used to find the RMSE changes along the x axis, we should be careful when comparing these values. In particular, the values with small maximum R contain far fewer points, and therefore involve more sampling errors.

Figure 7a shows four emulators used to predict s_1. The hierarchical emulator built with reduced s_1 data (solid line) shows an increase in RMSE as R increases, performing well for small R. For most of the range of R this emulator outperforms the standard emulator built with all 1,000 s_1 input points. The hierarchical emulator built with all 2,000 data points also shows increasing RMSE as R increases, but less noticeably.

Figure 7b shows emulators of the difference, $s_1 - s_0$. The standard emulators are emulators of the difference calculated from the training data, which was shown to be the best standard method in section 5. The reduced s_1 standard emulator is therefore built from the difference at each of the 100 points included with $R \neq 0$. The hierarchical emulator of the difference is built from the points for which a difference (or $\boldsymbol{\psi}$ data) is available, and therefore the hierarchical emulator built from reduced s_1 data is also built only from these 100 points.

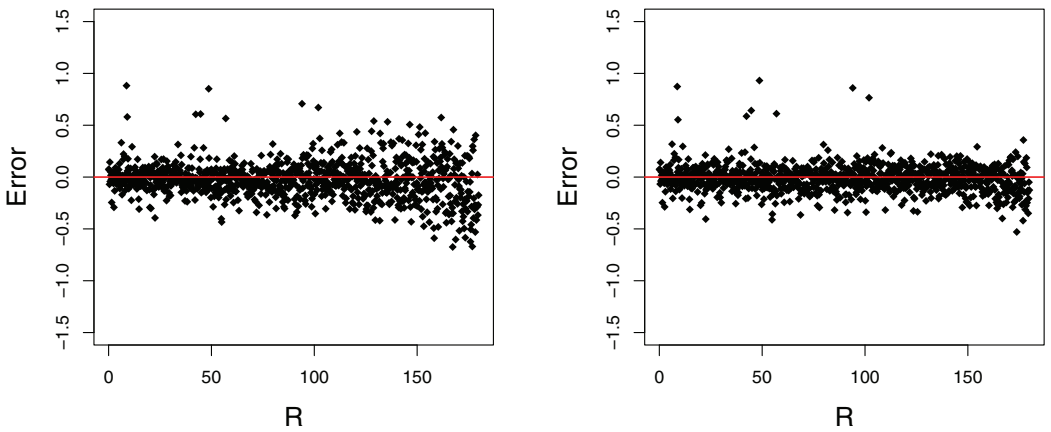
The advantage of the hierarchical emulation structure is much more apparent with reduced s_1 data. The only extra information contained in the hierarchical emulator, compared to the standard emulator with reduced s_1 data (dot-dashed line), is the inclusion of the relationship $s_1(x, R, m_2) = s_0(x) + g(R)\psi(x, R, m_2)$, and it appears from Figure 7 that this is a valuable addition to the emulator. This supports the idea that when the more complex version is costly to run compared to the simpler version, hierarchical emulation is a very effective strategy.

[1] From the previous examples of standard emulators, this appears to be a better strategy than using all 2,000 points.



(a) Standard emulator, 1,100 point design.

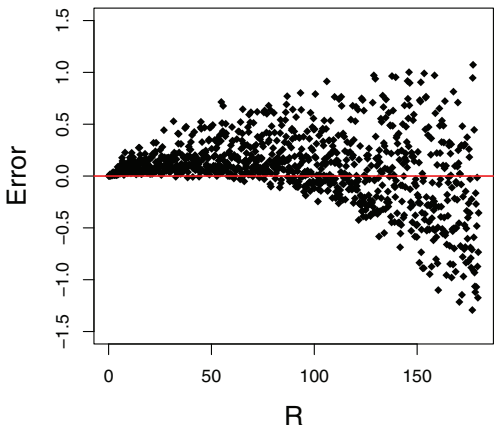
(b) Standard emulator, full 1,000 point design over s_1.



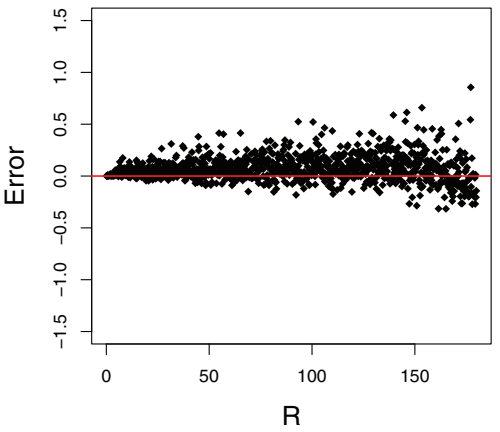
(c) Hierarchical emulator, 1,100 point design.

(d) Hierarchical emulator, full 2,000 point design.

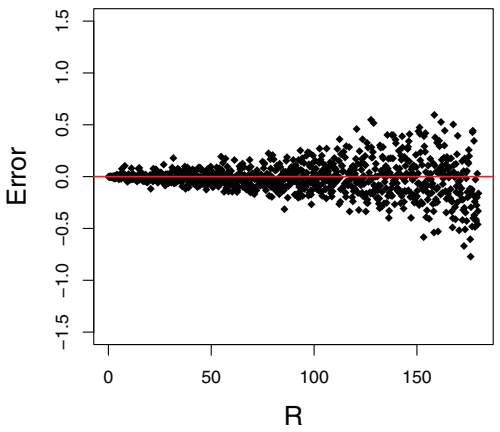
Figure 5. *Errors for four emulators of s_1, plotted against R.*



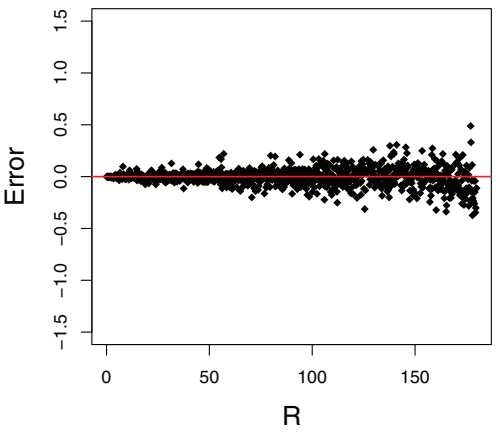
(a) Standard emulator, 200 point design.



(b) Standard emulator, 2,000 point design.

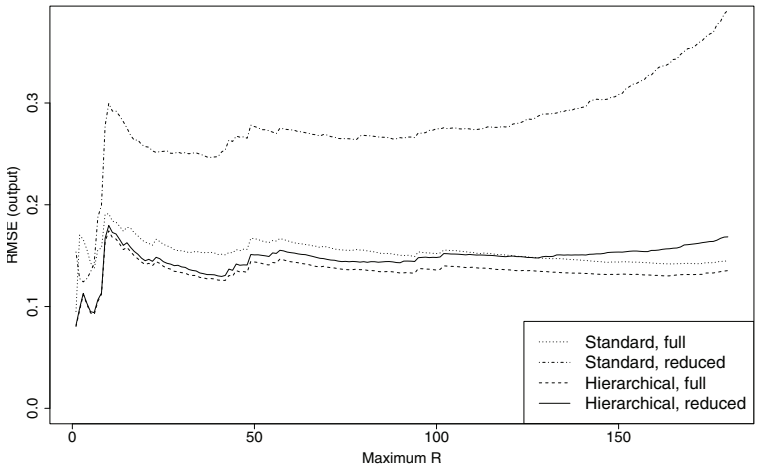


(c) Hierarchical emulator, 200 point design.

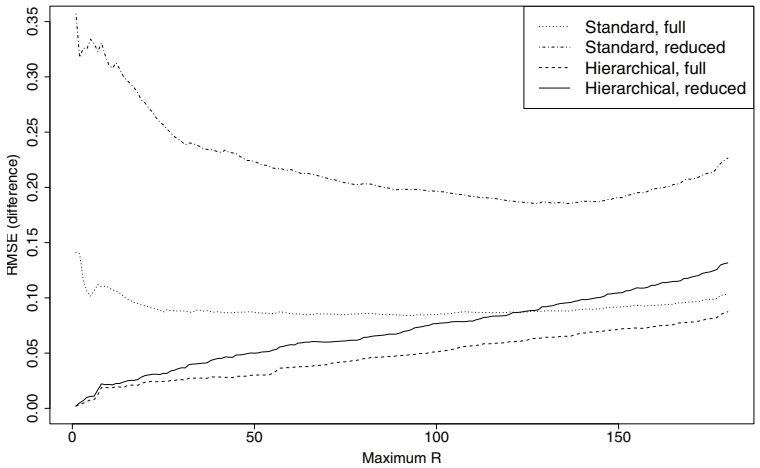


(d) Hierarchical emulator, full 2,000 point design.

Figure 6. *Errors for four emulators of $s_1 - s_0$, plotted against R.*



(a) Emulators of s_1.



(b) Emulators of $s_1 - s_0$.

Figure 7. *RMSE for four emulators, changing as the subset of data used changes. The prediction data were restricted to R less than "maximum R" for values from* 1 *to* 180 *(the maximum value R takes in our experiment). The number of input points considered ranges from* 5 *(when $R \leq 1$) to* 1,000 *(when $R \leq 180$).*

8. Summary. In this paper we have introduced hierarchical emulation, a method for emulating a simulator that can be extended by turning on one or more switches. The simulator versions must have the property that when a small subset of the inputs, the *hierarchical inputs*, are set to particular values, the two versions behave identically. Hierarchical emulation makes use of this relationship to emulate the more complicated simulator using a combination of other emulators, one of which is an emulator of the simpler simulator.

We have established a prior structure for the emulator that ensures separability between terms. The implications of this method for the design of experiments have been explored, and some criteria have been established for the structure of the training data. A hierarchical emulator requires some *transformation functions* $g(\cdot)$, and desirable properties for these have been explored.

In order to assess the performance of hierarchical emulation, a validation study was conducted using two versions of HadOCC. For this, a 1 million point staggered LHD was created so that hierarchical and standard emulators could be compared with respect to various tasks. Overall, this showed that hierarchical emulation outperforms the standard method, both in its predictive accuracy and its coherence with the emulation model.

A further experiment to assess the performance of a hierarchical emulator built with a reduced amount of data from the more complicated version of the simulator showed very promising results compared to the standard emulator, and this reinforced our beliefs that including the relationship between the two versions is beneficial.

One natural generalization of the model might be to consider situations where a switch cannot be turned on unless some other switch(es) has/have already been turned on. Another might be to consider situations where switches are mutually exclusive, in other words, they could not be turned on simultaneously. Both would have implications for the training dataset design.

Acknowledgments. The authors would like to thank the associate editor and two referees for their suggestions for improvements to the paper.

REFERENCES

[1] L. S. Bastos and A. O'Hagan, *Diagnostics for Gaussian process emulators*, Technometrics, 54 (2009), pp. 425–438.

[2] S. Conti and A. O'Hagan, *Bayesian emulation of complex multi-output and dynamic computer models*, J. Stat. Plann. Inference, 140 (2010), pp. 640–651.

[3] P. S. Craig, M. Goldstein, A. H. Seheult, and J. A. Smith, *Pressure matching for hydrocarbon reservoirs: A case study in the use of Bayes linear strategies for large computer experiments*, in Case Studies in Bayesian Statistics, Vol. II, Lecture Notes in Statistics 121, C. Gatsonis et al., eds., Springer-Verlag, New York, 1997, pp. 37–93.

[4] N. A. C. Cressie, *Statistics for Spatial Data*, revised reprint of the 1991 edition, Wiley Ser. Probab. Math. Statist. Appl. Probab. Statist., John Wiley & Sons, Inc., New York, 1993.

[5] M. Goldstein and J. Rougier, *Reified Bayesian Modelling and Inference for Physical Systems*, J. Stat. Plann. Inference, 139 (2009), pp. 1221–1239.

[6] Y. Hung, V. R. Joseph, and S. N. Melkote, *Design and analysis of computer experiments with branching and nested factors*, Technometrics, 51 (2009), pp. 354–365.

[7] M. C. Kennedy and A. O'Hagan, *Bayesian calibration of computer models*, J. R. Stat. Soc. Ser. B Stat. Methodol., 63 (2001), pp. 425–464.

[8] A. O'Hagan, *Bayesian analysis of computer code outputs: A tutorial*, Reliability Engineering and System Safety, 91 (2006), pp. 1290–1300.
[9] A. O'Hagan and J. Forster, *Kendall's Advanced Theory of Statistics. Vol. 2B. Bayesian Inference,* 2nd ed., John Wiley & Sons, Ltd., Chichester, 2004.
[10] R. H. Oxlade, *Comparing Multiple Simulators Using Bayesian Emulators*, Ph.D. thesis, Department of Mathematical Sciences, University of Durham, Durham, UK, 2012.
[11] J. R. Palmer and I. J. Totterdell, *Production and export in a global ocean ecosystem model*, Deep-Sea Res., 48 (2001), pp. 1169–1198.
[12] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2011. Available online at http://www.R-project.org/.
[13] T. J. Santner, B. J. Williams, and W. Notz, *The Design and Analysis of Computer Experiments*, Springer Ser. Statist., Springer-Verlag, New York, 2003.