Biometrika (2009), **96**, 3, *pp*. 663–676
doi: 10.1093/biomet/asp028
Advance Access publication 30 June 2009

Gaussian process emulation of dynamic computer codes

BY S. CONTI

Centre for Infections, Health Protection Agency, 61 Colindale Ave., London, NW9 5EQ, U.K.
stefano.conti@hpa.org.uk

J. P. GOSLING

Central Science Laboratory, Department for Environment, Food and Rural Affairs, Sand Hutton, York, YO41 1LZ, U.K.
jp.gosling@csl.gov.uk

J. E. OAKLEY AND A. O'HAGAN

Department of Probability and Statistics, University of Sheffield, Sheffield, S3 7RH, U.K.
j.oakley@sheffield.ac.uk a.ohagan@sheffield.ac.uk

SUMMARY

Computer codes are used in scientific research to study and predict the behaviour of complex systems. Their run times often make uncertainty and sensitivity analyses impractical because of the thousands of runs that are conventionally required, so efficient techniques have been developed based on a statistical representation of the code. The approach is less straightforward for dynamic codes, which represent time-evolving systems. We develop a novel iterative system to build a statistical model of dynamic computer codes, which is demonstrated on a rainfall-runoff simulator.

Some key words: Bayesian inference; Computer experiment; Dynamic simulator; Emulation; Gaussian process; Iterative modelling.

1. INTRODUCTION

Complex computer codes are used to make predictions about real-world systems in many fields of science and technology. We refer to such a computer code as a simulator. We can represent the simulator in the form of a function $y = f(x)$, and a run of the simulator is defined to be the process of producing one set of outputs y for one particular input configuration x. Throughout this paper, we assume that the simulator is deterministic; that is, running the simulator for the same x twice will yield the same y. The complexity of a simulator can become a problem when it is necessary to make many runs for different x. For example, the simulator user may wish to study the sensitivity of y to variations in x, which entails a large number of simulator runs. In particular, standard Monte Carlo-based methods of sensitivity analysis require thousands of simulator runs; these methods are extensively reviewed by Saltelli et al. (2000).

A two-stage approach based on emulation of the simulator's output has been developed that offers substantial efficiency gains over standard methods; see for example Sacks et al. (1989), Kennedy & O'Hagan (2001) or O'Hagan (2006). An emulator is a statistical representation of $f(\cdot)$ that is constructed using a training sample of simulator runs. Uncertainty and sensitivity

analyses can then be tackled using the emulator as shown by Oakley & O'Hagan (2002, 2004). The efficiency gains arise because it is usually possible to emulate the simulator output to a high degree of precision using only a few hundred runs of the simulator.

Many simulators are dynamic: they operate iteratively over fixed time-steps to model a system that is evolving over time. A single run of such a simulator generally consists of a simulation over many time-steps, and we can think of it in terms of a simpler, single-step simulator being run iteratively many times. The single-step simulator requires the current value of a state vector as an input, and the updated value of the state vector becomes an output. It may have other inputs that can be classified as model parameters and forcing inputs. Model parameters have fixed values for all the time-steps of a simulator run. They describe either fundamental parameters of the mathematical model or enduring characteristics of the system being simulated. Forcing inputs vary from one time-step to the next and represent external influences on the system. At time-step t, the simulator may be written in the form $Y_t = f(z_t, Y_{t-1})$, where Y_{t-1} is the state vector at the previous time-step, $z_t = (x, w_t)$ subsumes both the model parameters x and the forcing inputs w_t at time-step t, and the output of the simulator is the new state vector Y_t. We use Y_t rather than y_t to differentiate between a state vector in the series of interest and a simulator output from the training dataset.

Emulation techniques can be applied to dynamic simulators in two different ways. One approach is to use existing methods to emulate a complete multi-step run of the simulator, while the other is to emulate the simpler, single-step simulator and then to use the emulator iteratively. In this paper, we develop the second of these strategies, which requires two distinct developments of the existing methodology as described by Sacks et al. (1989) and in a yet unpublished paper by Conti & O'Hagan. Emulation is used to take account of our uncertainty about the simulator. This is usually considered secondary to the uncertainty in the simulator output caused by our uncertainty about the inputs. The iterative nature of the emulator proposed in this paper allows us to handle uncertainty in the time-varying forcing inputs that are usually taken as being known.

2. EMULATION OF COMPLEX SIMULATORS

In this section, we review the theory of emulation for multi-output simulators as presented in a paper by Conti & O'Hagan. We consider a deterministic simulator that returns outputs $y \in \mathbb{R}^q$ given inputs x from some input space $\mathcal{X} \subseteq \mathbb{R}^p$. Although, in principle, the simulator is a known function, so that $y = f(x)$ can be determined for any x, the complexity of the simulator means that before running the computer code y is unknown in practice. Therefore, we regard $f(\cdot)$ as an unknown function, and we represent it by the q-dimensional Gaussian process

$$f(\cdot) \mid B, \Sigma, R \sim \mathcal{N}_q \{m(\cdot), c(\cdot, \cdot)\Sigma\}. \tag{1}$$

This implies that, for all x, $E\{f(x) \mid B, \Sigma, R\} = m(x)$ and, for all x and x', $\mathrm{cov}\{f(x), f(x') \mid B, \Sigma, R\} = c(x, x')\Sigma$, where $c(\cdot, \cdot)$ is a correlation function having the property that $c(x, x) = 1$ for every x. We assume a stationary, separable covariance structure, with covariance between the outputs at any single input given by the matrix $\Sigma = [\sigma_{jj'}]$ and with $c(\cdot, \cdot)$ providing correlation across $\mathcal{X}$. We expect that the separable covariance function will serve well in many situations even where outputs are not of a common type. This covariance function assumes that, for a given input, all the outputs respond with a common length scale, and this is what is generally perceived to be the weakness of separability. However, if, after subtracting the mean functions $m(\cdot)$, the outputs are strongly correlated, then they must necessarily have very similar

length scales. If they are only weakly correlated, then they can have different length scales, but we can use independent emulators in this case.

We model the mean and covariance functions in terms of further unknown hyperparameters B and R by

$$m(x) = B^{\mathrm{T}}h(x), \quad c(x, x') = \exp\{-(x - x')^{\mathrm{T}}R(x - x')\}. \tag{2}$$

Here $h: \mathcal{X} \longmapsto \mathbb{R}^m$ is a known vector of m regression functions $h_1(x), \ldots, h_m(x)$ shared by each individual function $f_j(\cdot)$ $(j = 1 \ldots, q)$, $B = [\beta_1, \ldots, \beta_q] \in \mathbb{R}_{m,q}$ is a matrix of regression coefficients, and $R = \mathrm{diag}\{\theta_i^{-2}\}$ is a diagonal matrix of p positive length scale parameters. The length scales are also called ranges by Cressie (1993) and correlation lengths by Santner et al. (2003). The squared-inverses of the length scales are called roughness parameters by Kennedy & O'Hagan (2001).

The selection of the prior mean structure should be driven by both experience and simplicity; a linear specification $h(x) = (1, x)^{\mathrm{T}}$ has been found to be appropriate in most applications. The form of $c(\cdot, \cdot)$ assumed in equation (2) implies that the $f_j(\cdot)$ are smooth, infinitely differentiable functions. There are many possible correlation functions we could use here, and we choose the squared-exponential form for its analytical tractability when we develop an approximation for the dynamic emulator in § 3·4. The squared-exponential has been found to model the correlation in complex computer codes well; a recent example of this is in Kennedy et al. (2008).

We start by running the computer code on a preselected design set $\mathcal{S} = \{s_1, \ldots, s_n\} \subset \mathcal{X}$ and this yields outputs organized in the data matrix $D = [f_j(s_r)] \in \mathbb{R}_{n,q}$. The set $\mathcal{S}$ is selected in accordance with some space-filling design criterion. If p is large, then it will be expensive to have a design set $\mathcal{S}$ that spans $\mathcal{X}$ effectively. Indeed, most of $\mathcal{X}$ will be an extrapolation from $\mathcal{S}$. However, if we use a good space-filling design, then most points in $\mathcal{X}$ will lie close to a point in $\mathcal{S}$ and the emulator should provide a good representation of our uncertainty about any point. If we were interested in the function at a point far away from $\mathcal{S}$, then $h(\cdot)$ may require a more complex structure to capture our beliefs about the simulator.

From (1) and (2), the joint distribution of the code output matrix D conditional on hyperparameters B, Σ and R is the matrix-normal distribution

$$D \mid B, \Sigma, R \sim \mathcal{N}_{n,q}\left(HB,\ \Sigma \otimes A\right),$$

where $H^{\mathrm{T}} = [h(s_1), \ldots, h(s_n)]$, $A = [c(s_r, s_l)]$ and $\otimes$ denotes the Kronecker product operator. If we let $t^{\mathrm{T}}(\cdot) = \{c(\cdot, s_1), \ldots, c(\cdot, s_n)\}$, standard normal theory leads to the following conditional posterior distribution for the simulator:

$$f(\cdot) \mid B, \Sigma, R, D \sim \mathcal{N}_q\{m^*(\cdot), c^*(\cdot, \cdot)\Sigma\}, \tag{3}$$

where, for $x_1, x_2 \in \mathcal{X}$,

$$m^*(x_1) = B^{\mathrm{T}}h(x_1) + (D - HB)^{\mathrm{T}}A^{-1}t(x_1), \quad c^*(x_1, x_2) = c(x_1, x_2) - t^{\mathrm{T}}(x_1)A^{-1}t(x_2).$$

One way of obtaining the posterior process of $f(\cdot)$ conditional on R alone is through the integration of (3) with respect to the posterior distribution of B and Σ. Since any substantial information about such parameters will hardly ever be elicited from the code developers, the conventional noninformative prior $\pi^{\mathrm{J}}(B, \Sigma \mid R) \propto |\Sigma|^{-(q+1)/2}$ is selected. An alternative prior distribution, which allows for expert judgement for B and Σ, is given by Rougier (2008).

Combining the distribution in (3) and $\pi^{\mathrm{J}}(\cdot)$ using Bayes' theorem yields

$$\begin{aligned} f(\cdot) \mid \Sigma, R, D &\sim \mathcal{N}_q\{m^{\star\star}(\cdot), c^{\star\star}(\cdot,\cdot)\Sigma\}, \\ m^{\star\star}(x_1) &= \hat{B}^{\mathrm{T}} h(x_1) + (D - H\hat{B})^{\mathrm{T}} A^{-1} t(x_1), \\ c^{\star\star}(x_1, x_2) &= c^{\star}(x_1, x_2) + \{h(x_1) - H^{\mathrm{T}} A^{-1} t(x_1)\}^{\mathrm{T}} \\ &\qquad (H^{\mathrm{T}} A^{-1} H)^{-1} \{h(x_2) - H^{\mathrm{T}} A^{-1} t(x_2)\}, \end{aligned} \tag{4}$$

with $\hat{B} = (H^{\mathrm{T}} A^{-1} H)^{-1} H^{\mathrm{T}} A^{-1} D$. Provided now that $n \geqslant m + q$, so that the posterior is proper, the conditional t-process with $n - m$ degrees of freedom,

$$f(\cdot) \mid R, D \sim \mathcal{T}_q\{m^{\star\star}(\cdot), c^{\star\star}(\cdot,\cdot)\hat{\Sigma}; n - m\}, \tag{5}$$

is obtained, in which $\hat{\Sigma} = (n-m)^{-1}(D - H\hat{B})^{\mathrm{T}} A^{-1} (D - H\hat{B})$. A full Monte Carlo Markov chain strategy for removing the dependence on the unknown length scales in R is computationally expensive. An alternative is to use posterior estimates of $(\theta_1, \ldots, \theta_p)$. There are contrasting opinions about using this plug-in strategy: Kennedy & O'Hagan (2001) found uncertainty about R to be relatively unimportant, but Abt (1999) and an unpublished University of British Columbia technical report by B. Nagy, J. L. Loeppky and W. J. Welch show that prediction uncertainty can be underestimated when using a Gaussian correlation function. However, in our example of § 4, we found no evidence of overconfidence.

3. EMULATION OF DYNAMIC SIMULATORS

3·1. *Iterative use of emulators*

Dynamic simulators model the evolution of state variables over a number of time-steps. If we are interested in the state variables or some transformation thereof after a fixed number of time-steps, then ordinary emulation techniques will suffice. However, if we want to emulate the behaviour of the simulator over a number of time-steps, we need a different formulation.

A run of the simulator over the time-steps 0 to T can be expressed iteratively in terms of the single-step simulator:

$$\begin{aligned} Y_T &= f(z_T, Y_{T-1}) = f\{z_T, f(z_{T-1}, Y_{T-2})\} \\ &= \cdots = f[z_T, f\{z_{T-1}, \ldots, f(z_1, Y_0)\}] \\ &= f^{(T)}(x, z, Y_0), \end{aligned}$$

where $f^{(T)}(\cdot)$ represents the T-step simulator, which takes as its inputs the model parameters x, the whole sequence $z = (z_1, \ldots, z_T)$ of forcing inputs, and the initial state vector Y_0.

An emulator of $f^{(T)}(\cdot)$ for given T can be constructed using the theory of § 2. This approach of directly emulating the multi-step simulator to quantify our uncertainty about $(Y_0, \ldots, Y_T)$ has two main disadvantages. First, the dimension of the corresponding input space becomes very large because it must include the whole time sequence of forcing inputs. Second, the resulting emulator is specific to a particular T. We consider the iterative process explicitly: we emulate the single-step simulator $f(\cdot)$ and use this to emulate $f^{(T)}(\cdot)$ indirectly. The dimension of the input space is then more manageable, and the emulator can be used for simulator runs of any length.

3·2. *Exact emulation of dynamic simulators*

The emulation of the single-step simulator $f(\cdot)$ is straightforward. The challenge now is the construction of an emulator of $f^{(T)}(\cdot)$ from the emulator of $f(\cdot)$. This cannot be done analytically.

The joint distribution of Y_1 and Y_2 is no longer bivariate normal as in the standard emulation case because of the dependency of Y_2 on Y_1. In fact, there is potential for an emulator that is considerably different from the standard because Y_2 is a stochastic nonlinear function of Y_1.

A simple brute-force approach is to use a Monte Carlo scheme, replacing $f(\cdot)$ by its emulator in the iterative scheme. In order to train the single-step emulator, we choose a set of well-spaced points that covers the portion of input space of interest; that is, a set that covers the range of the forcing inputs over the T steps and the areas to which we expect the state variables to move in the T steps. We then update our beliefs about $f(\cdot)$ given the training data to arrive at the posterior distribution given in (5).

Suppose that we know the initial values of the state variables Y_0 and the forcing inputs for the first time-step z_1. A simulation technique is used where we draw a realization of $Y_1 = f(z_1, Y_0)$ from the multivariate t-distribution given in (5). We next draw a realization of $Y_2 = f(z_2, Y_1)$, but at this step the distribution should be conditional on $f(z_1, Y_0) = Y_1$. In effect, this adds $f(z_1, Y_0) = Y_1$ as an extra training run and imposes the condition that, if the series revisits the same set of state variables and forcing inputs, we will recover the same result we have encountered previously. We proceed in this sequential manner by successively drawing each Y_t from the emulator constructed by adding the random realization of $(Y_1, \ldots, Y_{t-1})$ to the training data D. By repeating this process many times, we draw a sample of values from the posterior distribution of $(Y_0, \ldots, Y_T)$. Each time we add a simulated point to the training dataset, we update the correlation matrix A and compute its inverse. In order to do this, we use the recursion formulae of Strassen (1969) to compute the new inverse using the previous step's inverse and some simple matrix arithmetic. Our simulation scheme for obtaining a sample of N_{MC} series is set out below.

Step 1. Create design $\mathcal{S}$ of size n to span the space of interest, $\mathcal{Z} \times \mathcal{Y}$.

Step 2. Evaluate $f(\cdot)$ at the n design points to obtain D.

Step 3. Estimate R for the posterior distribution of $f(\cdot) \mid D$.

Step 4. Set $N = 1$ and $t = 1$.

Step 5. Simulate $f(z_t, Y_{t-1})$ from the distribution of $f(\cdot) \mid D$, R and store $Y_t^{(N)}$.

Step 6. If $t = T$, set $N = N + 1$.

Step 7. If $t = T$ and $N \leqslant N_{\mathrm{MC}}$, go to Step 4 and reset $\mathcal{S}$ and D; otherwise stop.

Step 8. Add $(z_t, Y_{t-1}^{(N)})$ to $\mathcal{S}$ and $Y_t^{(N)}$ to D, set $t = t + 1$ and go to Step 5.

New points added to the training dataset can be close to an existing point in $\mathcal{S}$ when we condition during the simulation process. Hence, our uncertainty about the function at the new point may be small. Using the emulator for the single-step function, we can calculate $\mathrm{var}\{f(z_{t+1}, Y_t) \mid D, R, Y_0, \ldots, Y_{t-1}\}$. If $\mathrm{var}\{f(z_{t+1}, Y_t) \mid D, R, Y_0, \ldots, Y_{t-1}\}$ is small, we have little uncertainty about the value of $f(z_{t+1}, Y_t)$, and adding the point to the training dataset can cause the correlation matrix A to become ill-conditioned. In this case, the point can be taken as being known and will not be added to $\mathcal{S}$.

If we find that we are still very uncertain about $(Y_0, \ldots, Y_T)$, we can use our posterior distribution for $(Y_0, \ldots, Y_T)$ to select additional design points. First, we check that the predicted state variable values fall within the area specified when we created the initial design. If the series of state variables moves outside this area, we may add more training data to cover $f(\cdot)$s unexpected behaviour. We can use the posterior means for the state variables at each time-point

along with their corresponding forcing inputs to create a set of points where we want to reduce uncertainty.

3·3. *Efficiency considerations*

An alternative approach to our algorithm is reviewed by Bhattacharya (2007). It is possible to simulate the single-step function over the region of interest using methods described by Oakley & O'Hagan (2002). The idea is to draw from the posterior process given in equation (5) on a grid of points. Once we have simulated the single-step function, we can use it iteratively to determine one possible sequence $\{Y_0, \ldots, Y_T\}$. An additional assumption used by Bhattacharya (2007) is that the values of the state variables at previous time-steps are ignored; that is, $p(Y_T \mid D, D^*, Y_0, \ldots, Y_{T-1}) = p(Y_T \mid D, D^*)$, where D^* is the set of points at which we sample the function and the associated draws. If we repeatedly draw from the posterior process at D^* and D^* is dense enough in the input space, we will obtain a Monte Carlo sample that is equivalent to the sample we obtain from the method detailed in this section.

Both of these simulation methods can be thought of as drawing realizations of $f(\cdot)$ from the posterior process. Consider obtaining the complete realization $f_{(i)}(\cdot)$. We must sample from the joint distribution of $f(s_1), f(s_2), \ldots$, where $\{s_1, s_2, \ldots\}$ is the set of all possible input values. In theory, this set is uncountably infinite; however, in practice it would be finite because of limitations of computer storage. In Bhattacharya (2007), a fixed and relatively large subset of $\{s_1, s_2, \ldots\}$ is chosen. In our method, we sample each $f(s)$ sequentially. Let G_j be the jth variable simulated in obtaining the single realization $f_{(i)}(\cdot)$. If we prespecify that we will sample $f(s_1)$, then $f(s_2) \mid f(s_1)$, and so on, then the marginal distribution of G_j will be the marginal distribution of $f(s_j)$. Now suppose we randomly reorder the input values $\{s_1, s_2, \ldots\}$ in the sequential simulation to obtain $(s_{k_1}, s_{k_2}, \ldots)$. This will change the joint distribution of $G_1, G_2, \ldots$, and we may no longer be able to derive joint or marginal distributions of any G_js analytically. However, changing the order in which we do the sequential simulation has no effect on the joint distribution of $f(s_1), f(s_2), \ldots$, and we can ignore the fact that the order of the inputs has been changed when simulating $f(s_{k_1})$, then $f(s_{k_2}) \mid f(s_{k_1})$, and so on. In the exact simulation approach, we are randomly choosing the order of the inputs by setting the jth input to be the value of the $(j-1)$th simulated output. This ensures that the first T simulated outputs are precisely the T output values of the realization $f_{(i)}(\cdot)$ needed to determine $Y_1, \ldots, Y_T$.

The main computational difference between the two methods is the selection of the points at which we sample the single-step function. In Bhattacharya (2007), a grid of points at which we are going to sample must be defined, and there may be a lot of redundancy in this set. In our method, we select the points as we need them. The higher the dimensionality of the input space, the more points will be needed to cover the space for the method of Bhattacharya (2007). As we increase the size of the training dataset, we increase the effort required to construct a single-step emulator: as A becomes a larger matrix, we require more computational time to invert it.

If the single-step emulator for either method has been built from a sufficiently large training dataset, the posterior uncertainty in the series of outputs will be small and the approaches will provide an accurate emulation of $(Y_0, \ldots, Y_T)$. The question then arises of whether or not iterating the single-step emulator in this way is more efficient than directly emulating the multi-step simulator $f^{(T)}(\cdot)$. To emulate a long simulator run accurately, it will be necessary to emulate the single-step simulator to a high degree of accuracy. Thus, relatively large numbers of training runs may be needed. However, these runs will be much faster than the full simulator runs as they are over just a single step. Also, the two methods provide two alternative statistical representations of the simulator output; hence, validation of the emulator is important.

Single-step emulation will generally be more efficient than multi-step emulation for dynamic simulators, but implementing the Monte Carlo exact solution becomes a computationally intensive process. There could even be the case that the single-step simulator is quicker to run than the emulator. We develop an approximation that does not require the computational effort of a Monte Carlo simulation.

3·4. *Approximate emulation of dynamic simulators*

To avoid the repeated use of the single-step emulator in a Monte Carlo scheme, we introduce two approximations. To motivate the first approximation in the exact computation, if the training dataset is large enough, a new point added to this set at each iteration should have negligible effect. We would obtain essentially the same distribution for Y_T if we sampled each $f(z_t, Y_{t-1})$ from its posterior distribution based only on the original training dataset. Accordingly, the first approximation replaces $p(Y_t \mid D, R, Y_0, \ldots, Y_{t-1})$ with $p(Y_t \mid D, R)$.

The second approximation is to set the distribution of Y_t to be multivariate normal for all $t = 1, \ldots, T$. At $t = 1$, the distribution is multivariate t, which will be very close to normal for even a moderately large training dataset, but normality cannot hold for $t > 1$ unless $f(\cdot)$ is linear. Nevertheless, again given a large enough training sample, uncertainty in any Y_t should be small, and it is reasonable to assume approximate linearity over a small part of the input space.

Subject to this condition, first- and second-order moments uniquely identify the posterior distribution of $f(z_{t+1}, Y_t)$, given knowledge about the distribution of Y_t and the matrices Σ and R. Denote the posterior mean of Y_t by μ_t and its variance matrix by V_t. Using the assumption of approximate normality, we have $Y_t \mid \Sigma, R \sim \mathcal{N}_q(\mu_t, V_t)$, and the recursion will derive equations for μ_{t+1} and V_{t+1} in terms of their values at step t. Using these assumptions, we can show that

$$\mu_{t+1} = \hat{B}^{\mathrm{T}} E\{h(z_{t+1}, Y_t) \mid D\} + (D - H\hat{B})^{\mathrm{T}} A^{-1} E\{t(z_{t+1}, Y_t) \mid D\}, \tag{6}$$

$$V_{t+1} = \mathrm{var}\{m^{\star\star}(z_{t+1}, Y_t) \mid D\} + E[c^{\star\star}\{(z_{t+1}, Y_t), (z_{t+1}, Y_t)\} \mid D]\Sigma. \tag{7}$$

The expectation and variance in (6) and (7) are given in the Appendix. Equations (6) and (7) are conditional on Σ and R; the removal of this conditioning is also described in the Appendix.

The computational speed of the approximation is much greater than that of the exact simulation method or the method of Bhattacharya (2007). Only one set of matrix inversion calculations needs to be performed to obtain results using the proposed approximation whereas thousands are required for the Monte Carlo scheme within the exact simulation method. However, as this is not an exact representation of our beliefs, we will find cases where our uncertainty about the series is badly approximated and our posterior mean for $(Y_0, \ldots, Y_T)$ could be far from the series produced by exact simulation. Validation of the single-step emulator is therefore of great importance.

3·5. *Uncertainty analysis of dynamic simulators*

Uncertainty analysis that considers input uncertainty can be carried out using a simple Monte Carlo scheme. We draw one set of inputs required to run the simulator from the inputs' distribution. We then use the approximation to the exact emulator to find our posterior mean and variance for $(Y_1, \ldots, Y_T)$ given these input values. We repeat this thousands of times to find the mean and variance for $(Y_1, \ldots, Y_T)$ given our uncertainty about the simulator and the inputs. This is computationally expensive, but we have found that it yields results comparable to those of uncertainty analysis in the standard emulation framework of Oakley & O'Hagan (2002) for just a fraction of runs of the simulator's single-step function. In § 4, this Monte Carlo scheme is put to use and results are compared with the standard method.

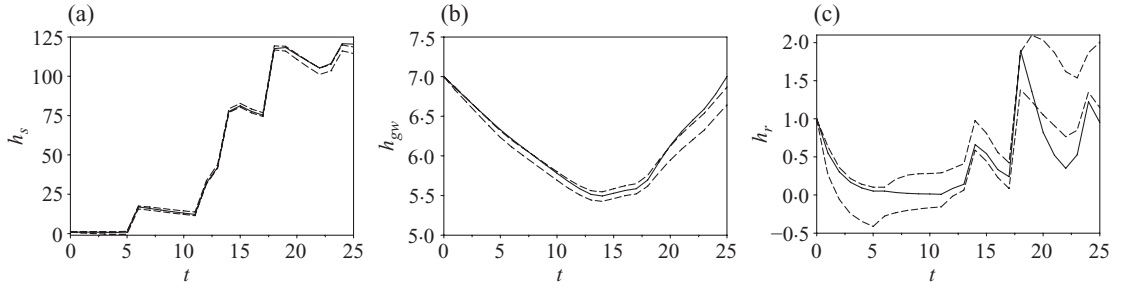

Fig. 1. Posterior 95% credible intervals (dashed) for 25 time-steps and the actual series (solid) based on 30 training runs, for (a) h_s, (b) h_{gw} and (c) h_r.

4. DYNAMIC RAINFALL-RUNOFF SIMULATOR

The simulator described by Kuczera et al. (2006) is a rainfall-runoff simulator that models the interaction between three water-bearing pools near a river. The simulator has three state variables; namely, volume of water in the soil h_s, volume of water in the ground-water pool h_{gw} and volume of water in the river h_r; two forcing inputs, namely, rainfall at time t, RAIN(t), and evapotranspiration potential at time t, PET(t); and seven other model-parameter inputs that govern the simulator's differential equations. The state variables of the simulator are all of the same type: they are all volumes of water-bearing compartments. Hence, the choice of covariance structure, as given in (2), is appropriate for this simulator.

First, we consider code uncertainty in the simulator output, the three state variables over 25 time-steps. We will take the initial values as being known: $h_s(0) = 1$, $h_{gw}(0) = 7$ and $h_r(0) = 1$. We also take the sequences of forcing inputs as being known. We begin by emulating the single-step function of the rainfall-runoff simulator using 30 training runs of the simulator. The input configurations for the initial runs are chosen using the maximin design strategy of Morris & Mitchell (1995). This design strategy uses ranges of the inputs derived from the area of the five-dimensional input space that we expect the simulator to cover. The following ranges were used for the design: $h_s \in [0, 100]$, $h_{gw} \in [5, 9]$, $h_r \in [0, 2]$, RAIN $\in [0, 50]$ and PET $\in [3, 6]$. The ranges for the state variables are selected on the basis of knowledge of the simulator and the ranges for the forcing inputs are taken from the known sequences.

We now employ the exact simulation scheme of § 3·2 to emulate the three state variables over 25 time-steps. Figure 1 shows the results of this. It can be seen from Fig. 1 that we expect two of the state variables to move outside the range specified for the initial design. Therefore, we add 20 extra training runs that target the unexplored areas of the state variable space; to be specific, $h_s \in [100, 125]$ and $h_r \in [2, 2{\cdot}5]$. The results of the exact simulation scheme are then shown in Fig. 2. The emulated series of state variables now mirror the real series very closely. However, we have used 50 training runs of the single-step simulator to emulate one run of a 25-step simulator.

The potential of the emulator is shown in Figs 3 and 4. Figure 3 is the result of employing the exact simulation scheme on nine different sets of initial values for h_s and h_r. Also, we can emulate the series over many more time-points. Figure 4 shows the results of emulating the state variables over 250 time-steps using the same 50 training runs of the single-step function and 20 additional runs that were selected to allow for departures in the state vector.

Using a multi-output emulator to deal with the whole 250-step series would be computationally more demanding: we would have an output space of 750 dimensions. By breaking the process down into single time-steps, we reduce the problem to a manageable size. However,

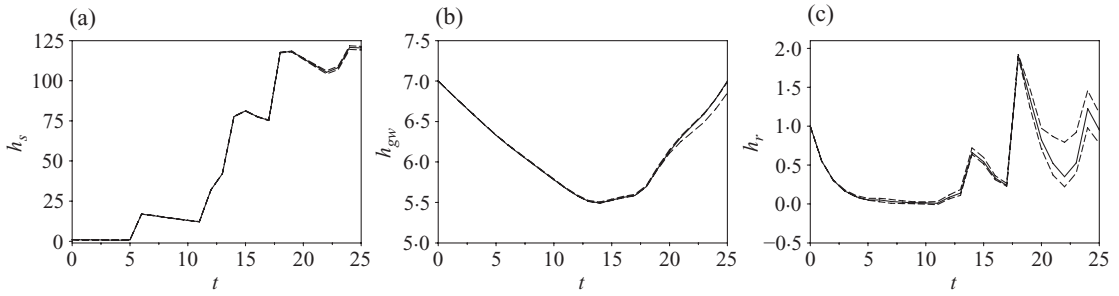

Fig. 2. Posterior 95% credible intervals (dashed) for 25 time-steps and the actual series (solid) with 20 extra training runs, for (a) h_s, (b) h_{gw} and (c) h_r.

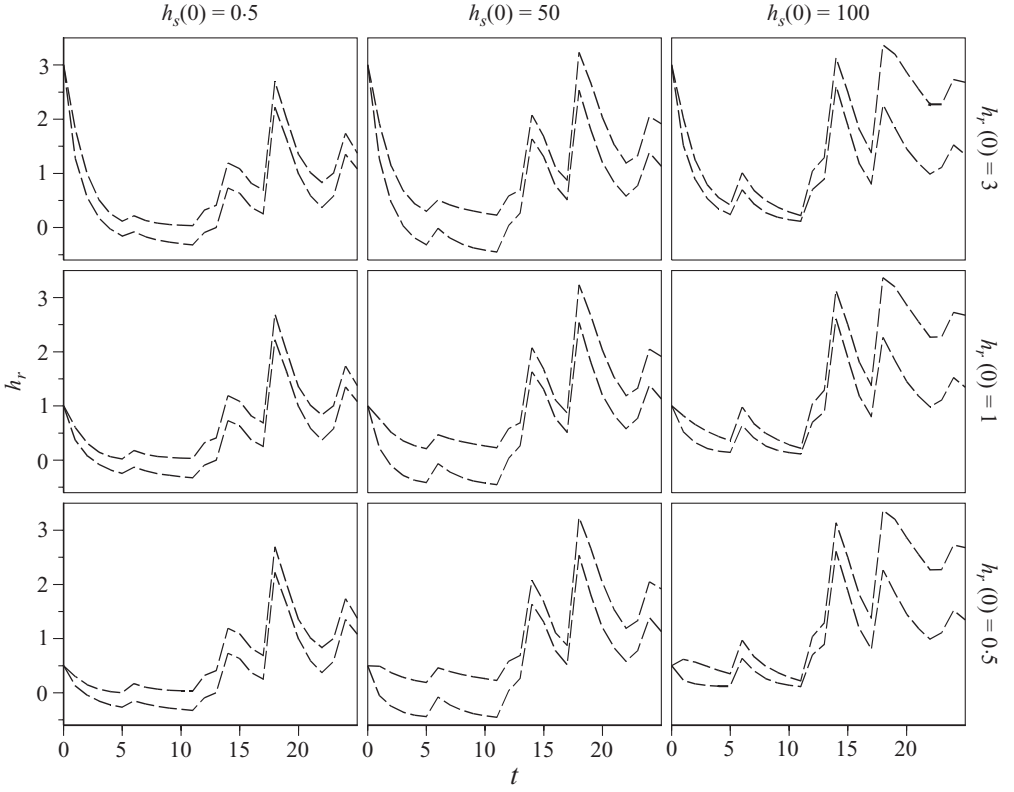

Fig. 3. Nine sets of posterior 95% credible intervals for h_r for 25 time-steps for different values of $h_s(0)$ and $h_r(0)$.

the outer-product emulator of Rougier (2008) makes a high-dimensional multi-output emulator computationally feasible.

We suppose now that we are uncertain about the initial values of the three state variables, $h_s(0)$, $h_{gw}(0)$ and $h_r(0)$, three of the most influential model parameters, x_1, x_2 and x_3 say, and the sequences of the forcing inputs. The following independent distributions were given to the uncertain state variables and model parameters:

$$h_s(0) \sim N(0{\cdot}4, 0{\cdot}01), \quad h_{gw}(0) \sim N(7{\cdot}5, 1), \quad h_r(0) \sim N(0{\cdot}145, 0{\cdot}0005),$$
$$x_1 \sim N(1{\cdot}5, 0{\cdot}4), \quad x_2 \sim N(2, 0{\cdot}36), \quad x_3 \sim N(6{\cdot}5, 0{\cdot}36).$$

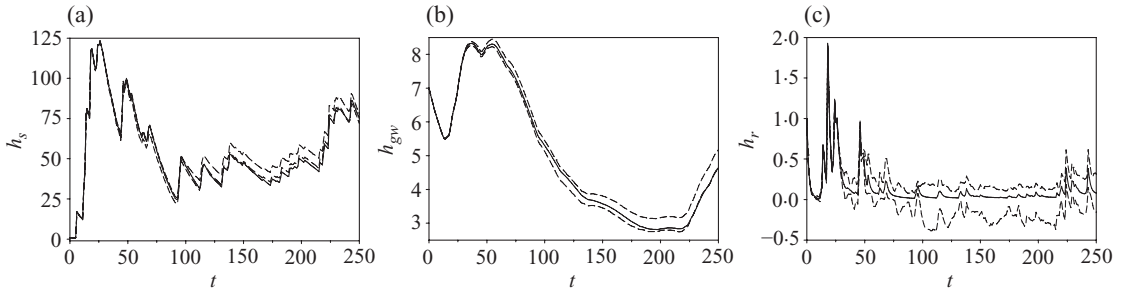

Fig. 4. Posterior 95% credible intervals (dashed) for 250 time-steps and the actual series (solid) based on 70 training runs, for (a) h_s, (b) h_{gw} and (c) h_r.

Table 1. *Comparison of uncertainty analysis results for the rainfall-runoff simulator after* 10 *time-steps*

	Standard emulation		Approximation	
	Mean	Variance	Mean	Variance
$h_s(10)$	13·92	0·60	13·91	0·65
$h_{gw}(10)$	6·23	0·81	6·23	0·80
$h_r(10)$	0·010	0·005	0·010	0·007
Number of single-step evaluations	2000		200	

For the forcing inputs, we take a known sequence and add noise; we used uniform noise over $[0, 0{\cdot}25]$ for RAIN(t) and $N(0, 0{\cdot}0625)$ for PET(t). These distributions do not represent anyone's beliefs and are simply for illustrative purposes. We are interested in our uncertainty about the state variables over 10 time-steps caused by this input uncertainty and uncertainty about the simulator. A simple uncertainty analysis can be performed through a Monte Carlo scheme: first, we draw from the input distributions, then we apply the approximation of § 3·4 conditional on the drawn values, and we repeat these two steps many times.

To emulate the single-step function well, we required 200 single-step training runs. We also carried out an uncertainty analysis using standard emulation techniques where 200 training runs over a 26-dimensional space were required to produce comparable emulator accuracy for the simulator output after 10 time-steps. The 200 ten-step training runs in the standard emulation case are equivalent to 2000 single-step training runs. The uncertainty analysis results for the two approaches are given in Table 1, where the variance represents our uncertainty in the outputs caused by our uncertainty about the inputs and the code. The two approaches yield similar results, with the approximation to the dynamic emulator using a fraction of the single-step training runs. In addition to the results given in Table 1, we also obtain the uncertainty analysis results for all the intermediate time-steps and at subsequent time-steps of interest when using the dynamic emulator; these are shown in Fig. 5.

5. DISCUSSION

In situations where running the simulator a modest number of times is so computationally expensive that emulation through standard procedures is infeasible, the emulation techniques developed in this paper can offer time savings as the single-step function does not need to be evaluated so often. However, there can be a much greater cost when building an emulator based on the single-step function. This cost is application-specific, and our methods will have the greatest efficiency gains when employed on simulators that are slow to evaluate a single time-step.

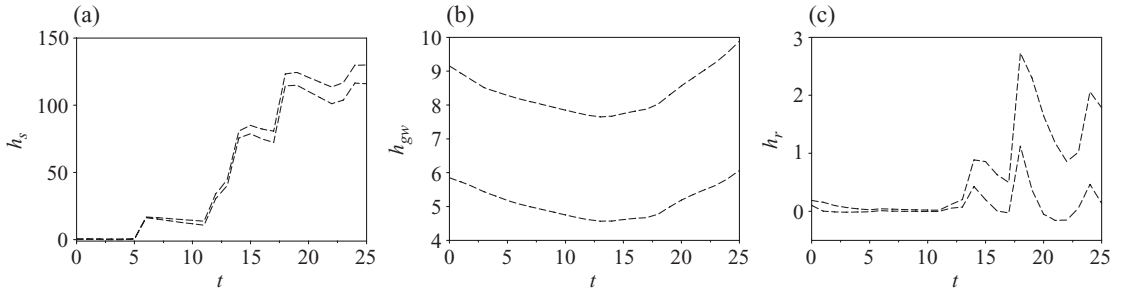

Fig. 5. Posterior 95% credible intervals (dashed) for 25 time-steps including uncertainty about the simulator inputs, for (a) h_s, (b) h_{gw} and (c) h_r.

Another benefit of using a single-step emulator is the potential to handle better any numerical error in the simulator. If the simulator involves systems of differential equations, numerical methods typically have to be used to evaluate the single-step function that can introduce numerical error. If we are only running the code over a single time-step, this gives us an opportunity to reduce or remove potential error by obtaining more accurate numerical solutions. However, this may not be practicable if we are running the simulator over many time-steps.

The method that has been presented in this paper has the potential to help a user to understand all the uncertainty surrounding a dynamic computer code. By reducing the simulator to the single-step function that dictates how the state inside the model evolves, we have an opportunity to link the model to reality through potentially simpler expert judgements and data assimilation at different time-points. This would be a shift from the current methods for dealing with the model-to-reality discrepancy, which add on the discrepancy at a fixed time-point as set out in Kennedy & O'Hagan (2001), to a scheme where the discrepancy is considered at every time-step.

Acknowledgement

The work in this paper is part of the activities of the Managing Uncertainty in Complex Models project that is funded by a Research Councils UK grant. We thank Peter Reichert for providing the details of the rainfall-runoff model analyzed in this paper. We would also like to thank Professor D. M. Titterington and the anonymous referees for their helpful and stimulating comments on earlier drafts of this paper.

Appendix

Calculation of the approximation to exact emulation

We use the theory of § 2 to emulate the single-step simulator $f(\cdot)$, but the p-dimensional argument x of this function is partitioned into the $(p-q)$-dimensional z and the q-dimensional y. We partition each of the training-set input vectors s_t into the first $p-q$ and last q components by $s_t = (s_t', s_t'')$. Similarly, $R' = \mathrm{diag}\{(\theta')^{-2}\}$ and $R'' = \mathrm{diag}\{(\theta'')^{-2}\}$ are the upper-left $(p-q)\times(p-q)$ and the lower-right $q \times q$ submatrices of the diagonal matrix R.

The following result will be invoked for appropriate $G \in \mathbb{R}_{q,q}$, $g \in \mathbb{R}^q$, $B \in \mathbb{R}_{s,q}$ and $b \in \mathbb{R}^s$:

$$E[(BY_t + b)\exp\{-(Y_t - g)^{\mathrm{T}} G (Y_t - g)\}] = |2V_t G + I_q|^{-1/2}\left\{B\left(2G + V_t^{-1}\right)^{-1}\left(2Gg + V_t^{-1}\mu_t\right) + b\right\} \times \exp\left\{-(\mu_t - g)^{\mathrm{T}}(2V_t + G^{-1})^{-1}(\mu_t - g)\right\}.$$

From (4), the equation for μ_{t+1} is given by

$$\begin{aligned}\mu_{t+1} &= E\{f(z_{t+1}, Y_t) \mid D\} = E\{m^{\star\star}(z_{t+1}, Y_t) \mid D\} \\ &= \hat{B}^{\mathrm{T}} E\{h(z_{t+1}, Y_t) \mid D\} + (D - H\hat{B})^{\mathrm{T}} A^{-1} E\{t(z_{t+1}, Y_t) \mid D\}. \end{aligned} \tag{A1}$$

The first expectation in (A1) will depend on the form of $h(\cdot)$, but, when $h(x)^{\mathrm{T}} = (1, x^{\mathrm{T}})$, we have $E\{h(z_{t+1}, Y_t) \mid D\}^{\mathrm{T}} = (1, z_{t+1}^{\mathrm{T}}, \mu_t^{\mathrm{T}})$. The second expectation is a vector that for $r = 1, \ldots, n$ is

$$\begin{aligned} E\{t_r(z_{t+1}, Y_t) \mid D, R\} &= \exp\{-(z_{t+1} - s_r')^{\mathrm{T}} R'(z_{t+1} - s_r')\} E[\exp\{-(Yt - s_r'')^{\mathrm{T}} R''(Y_t - s_r'')\} \mid D] \\ &= |2V_t R'' + I_q|^{-1/2} \exp\{-(z_{t+1} - s_r')^{\mathrm{T}} R'(z_{t+1} - s_r')\} \\ &\quad \times \exp\{-(\mu_t - s_r'')^{\mathrm{T}} (2V_t + R''^{-1})^{-1} (\mu_t - s_r'')\}. \end{aligned} \tag{A2}$$

The equation for V_{t+1} can be decomposed into two parts as

$$\begin{aligned} V_{t+1} &= \mathrm{var}\{f(z_{t+1}, Y_t) \mid D, R, \Sigma\} \\ &= \mathrm{var}\{m^{\star\star}(z_{t+1}, Y_t) \mid D, R\} + E[c^{\star\star}\{(z_{t+1}, Y_t), (z_{t+1}, Y_t)\} \mid D, R]\Sigma, \end{aligned} \tag{A3}$$

again by the law of iterated expectations. The first term in (A3) is given by

$$\begin{aligned} \mathrm{var}\{m^{\star\star}(z_{t+1}, Y_t) \mid D, R\} = {} & \hat{B}^{\mathrm{T}} \mathrm{var}\{h(z_{t+1}, Y_t) \mid D\} \hat{B} \\ & + \hat{B}^{\mathrm{T}} \mathrm{cov}\{h(z_{t+1}, Y_t), t(z_{t+1}, Y_t) \mid D, R\} A^{-1} (D - H\hat{B}) \\ & + (D - H\hat{B})^{\mathrm{T}} A^{-1} \mathrm{cov}\{t(z_{t+1}, Y_t), h(z_{t+1}, Y_t) \mid D, R\} \hat{B} \\ & + (D - H\hat{B})^{\mathrm{T}} A^{-1} \mathrm{var}\{t(z_{t+1}, Y_t) \mid D, R\} A^{-1} (D - H\hat{B}). \end{aligned} \tag{A4}$$

Now, in the case $h(x)^{\mathrm{T}} = (1, x^{\mathrm{T}})$, it follows that

$$\mathrm{var}\{h(z_{t+1}, Y_t) \mid D\} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & V_t \end{pmatrix},$$

$$\mathrm{cov}\{t(z_{t+1}, Y_t), h(z_{t+1}, Y_t) \mid D, R\} = (0\ 0\ \mathrm{cov}\{t_l(z_{t+1}, Y_t), Y_t \mid D, R\}).$$

Then the elements of the remaining terms required for the evaluation of (A4) are derived using (A2) and the following two results. First, for $r, l = 1, \ldots, n$,

$$\begin{aligned} & E\{t_r(z_{t+1}, Y_t) t_l(z_{t+1}, Y_t) \mid D, R\} \\ & \quad = \exp\{-(z_{t+1} - s_r')^{\mathrm{T}} R'(z_{t+1} - s_r') - (z_{t+1} - s_l')^{\mathrm{T}} R'(z_{t+1} - s_l')\} \\ & \qquad \times E[\exp\{-(Y_t - s_r'')^{\mathrm{T}} R''(Y_t - s_r'') - (Y_t - s_l'')^{\mathrm{T}} R''(Y_t - s_l'')\} \mid D, R] \\ & \quad = |4V_t R'' + I_q|^{-1/2} \exp\left\{-\frac{1}{2}(s_r'' - s_l'')^{\mathrm{T}} R''(s_r'' - s_l'')\right\} \\ & \qquad \times \exp\{-(z_{t+1} - s_r')^{\mathrm{T}} R'(z_{t+1} - s_r') - (z_{t+1} - s_l')^{\mathrm{T}} R'(z_{t+1} - s_l')\} \\ & \qquad \times \exp\left[-\left\{\mu_t - \frac{1}{2}(s_r'' + s_l'')\right\}^{\mathrm{T}} \left(2V_t + \frac{1}{2}R''^{-1}\right)^{-1} \left\{\mu_t - \frac{1}{2}(s_r'' + s_l'')\right\}\right]. \end{aligned}$$

Then, for $l = 1, \ldots, n$,

$$\begin{aligned} E\{t_l(z_{t+1}, Y_t)Y_t \mid D, R\} &= \exp\{-(z_{t+1} - s_l')^{\mathrm{T}} R'(z_{t+1} - s_l')\} \\ &\quad \times E[\exp\{-(Y_t - s_l'')^{\mathrm{T}} R''(Y_t - s_l'')\} \mid D, R] \\ &= |2V_t R'' + I_q|^{-1/2} \exp\{-(z_{t+1} - s_l')^{\mathrm{T}} R'(z_{t+1} - s_l')\} \\ &\quad \times \exp\{-(\mu_t - s_l'')^{\mathrm{T}}(2V_t + R''^{-1})^{-1}(\mu_t - s_l'')\} \\ &\quad \times \left(2R'' + V_t^{-1}\right)^{-1}\left(2R''s_l'' + V_t^{-1}\mu_t\right). \end{aligned}$$

Finally, some linear algebra manipulations allow us to compute the second summand in (A3) via

$$\begin{aligned} &E[c^{\star\star}\{(z_{t+1}, Y_t), (z_{t+1}, Y_t)\} \mid D, R] \\ &= 1 - \mathrm{tr}[\{A^{-1} - A^{-1}H(H^{\mathrm{T}}A^{-1}H)^{-1}H^{\mathrm{T}}A^{-1}\}E\{t(z_{t+1}, Y_t)t^{\mathrm{T}}(z_{t+1}, Y_t) \mid D, R\}] \\ &\quad + \mathrm{tr}[(H^{\mathrm{T}}A^{-1}H)^{-1}E\{h(z_{t+1}, Y_t)h^{\mathrm{T}}(z_{t+1}, Y_t) \mid D\}] \\ &\quad - 2\mathrm{tr}[A^{-1}H(H^{\mathrm{T}}A^{-1}H)^{-1}E\{h(z_{t+1}, Y_t)t^{\mathrm{T}}(z_{t+1}, Y_t) \mid D, R\}]. \end{aligned}$$

These results are conditional on the unknown parameters in Σ and R. As in § 2, we advocate simply plugging in an estimate of R. A method of marginalizing with respect to Σ is documented below.

Marginalization with respect to the dispersion matrix Σ. The complicated way in which the dispersion and length scale matrices Σ and R enter formulae (A1) to (A3) precludes any closed-form marginalization. The Student process given in (5) can be defined equivalently as follows: mix the distribution of $[f(\cdot) \mid R, D] \sim \mathcal{N}_q\{m^{\star\star}(\cdot), c^{\star\star}(\cdot)\Sigma\}$ with the density function of $[\Sigma \mid R, D] \sim \mathcal{W}_q^{-1}\{(n-m)\hat{\Sigma}; n-m\}$ or mix $[f(\cdot) \mid \xi, R, D] \sim \mathcal{N}_q\{m^{\star\star}(\cdot), \xi c^{\star\star}(\cdot)D^{\mathrm{T}}GD\}$, where $G = A^{-1} - A^{-1}H(H^{\mathrm{T}}A^{-1}H\}^{-1}H^{\mathrm{T}}A^{-1}$, with an auxiliary random variable $\xi \sim \chi^{-2}_{n-m}$.

It becomes possible to marginalize Σ out of formulae (A1) to (A3) just by applying the law of iterated expectations while conditioning on ξ rather than on Σ. In practical terms, in the above derived formulae the dispersion matrix Σ should be replaced by $\xi D^{\mathrm{T}}GD$, and thereafter the auxiliary quantity ξ can be integrated out via some one-dimensional numerical techniques. For instance, we have

$$\begin{aligned} E\{f(z_{t+1}, Y_t) \mid R, D\} &= E[E\{f(z_{t+1}, Y_t) \mid \xi, R, D\} \mid R, D] \\ &= E[\hat{B}^{\mathrm{T}}E\{h(z_{t+1}, Y_t) \mid \xi, R, D\} + (D - H\hat{B})^{\mathrm{T}}A^{-1} \\ &\quad \times E\{t(z_{t+1}, Y_t) \mid \xi, R, D\} \mid R, D], \end{aligned}$$

where, for example, $E\{t(z_{t+1}, Y_t) \mid \xi, R, D\} = E\{t(z_{t+1}, Y_t) \mid \Sigma = \xi D^{\mathrm{T}}GD, R, D\}$.

Marginalization with respect to the length scales. For simplicity, it is assumed that the length scales are a priori independent of both B and Σ. Given the prior proposed in § 2 for (B, Σ), we have

$$\pi(B, \Sigma, R) \propto \pi_{\mathrm{R}}(R)|\Sigma|^{-(q+1)/2},$$

with $\pi_{\mathrm{R}}(\cdot)$ deliberately left unspecified. Use of this prior in combination with the matrix-normal likelihood given in § 2 yields, via Bayes theorem, the full posterior for the hyperparameters,

$$\begin{aligned} \pi(B, \Sigma, R \mid D) \propto \pi_{\mathrm{R}}(R) \mid A \mid^{-q/2} \mid \Sigma \mid^{-(n-m+q+1)/2} \\ \times \exp\left(-\frac{1}{2}[\mathrm{tr}(D^{\mathrm{T}}GD\Sigma^{-1}) + \mathrm{tr}\{(B - \hat{B})^{\mathrm{T}}H^{\mathrm{T}}A^{-1}H(B - \hat{B})\Sigma^{-1}\}]\right). \end{aligned}$$

We can integrate out the matrices B and Σ, and this yields the marginal posteriors

$$\pi(\Sigma, R \mid D) \propto \pi_{\mathrm{R}}(R) \mid A \mid^{-q/2} \mid H^{\mathrm{T}} A^{-1} H \mid^{-q/2} \mid \Sigma \mid^{-(n-m+q+1)/2} \exp\left\{-\frac{1}{2}\mathrm{tr}(D^{\mathrm{T}} G D \Sigma^{-1})\right\},$$

$$\pi_R(R \mid D) \propto \pi_{\mathrm{R}}(R) \mid A \mid^{-q/2} \mid H^{\mathrm{T}} A^{-1} H \mid^{-q/2} \mid D^{\mathrm{T}} G D \mid^{-(n-m)/2}, \tag{A5}$$

the latter being of direct interest for drawing inferences on the length scales. In our example of § 4, we found the mode of the distribution in (A5) and used this estimate as our value for R. Characterizing the smoothness of the code's response surface by means of (A5) implies that input variables exhibit the same degree of smoothness throughout the whole emulation. Although for many simulators this is arguably realistic, accounting for time-dependent length scales would be computationally expensive.

REFERENCES

ABT, M. (1999). Estimating the prediction mean squared error in Gaussian stochastic processes with exponential correlation structure. *Scand. J. Statist.* **26**, 563–78.

BHATTACHARYA, S. (2007). A simulation approach to Bayesian emulation of complex dynamic computer models. *Bayesian Anal.* **2**, 783–816.

CRESSIE, N. A. (1993). *Statistics for Spatial Data,* rev. ed. New York: Wiley.

KENNEDY, M., ANDERSON, C., O'HAGAN, A., LOMAS, M., WOODWARD, F., GOSLING, J. & HEINEMEYER, A. (2008). Quantifying uncertainty in the biospheric carbon flux for England and Wales. *J. R. Statist. Soc.* A **171**, 109–35.

KENNEDY, M. & O'HAGAN, A. (2001). Bayesian calibration of computer models (with Discussion). *J. R. Statist. Soc.* B **63**, 425–64.

KUCZERA, G., KAVETSKI, D., FRANKS, S. & THYER, M. (2006). Towards a Bayesian total error analysis of conceptual rainfall-runoff models: characterising model error using storm-dependent parameters. *J. Hydrol.* **331**, 161–77.

MORRIS, M. & MITCHELL, T. (1995). Exploratory designs for computer experiments. *J. Statist. Plan. Infer.* **43**, 381–402.

OAKLEY, J. & O'HAGAN, A. (2002). Bayesian inference for the uncertainty distribution of computer model outputs. *Biometrika* **89**, 769–84.

OAKLEY, J. E. & O'HAGAN, A. (2004). Probabilistic sensitivity analysis of complex models: a Bayesian approach. *J. R. Statist. Soc.* B **66**, 751–69.

O'HAGAN, A. (2006). Bayesian analysis of computer code outputs: a tutorial. *Reliab. Eng. Syst. Safety* **91**, 1290–300.

ROUGIER, J. (2008). Efficient emulators for multivariate deterministic functions. *J. Comp. Graph. Statist.* **17**, 827–43.

SACKS, J., WELCH, W., MITCHELL, T. & WYNN, H. (1989). Design and analysis of computer experiments. *Statist. Sci.* **4**, 409–23.

SALTELLI, A., CHAN, K. & SCOTT, E. (Eds.) (2000). *Sensitivity Analysis*. New York: Wiley.

SANTNER, T., WILLIAMS, B. & NOTZ, W. (2003). *The Design and Analysis of Computer Experiments*. New York: Springer.

STRASSEN, V. (1969). Gaussian elimination is not optimal. *Numer. Math.* **13**, 354–6.

[*Received June* 2007. *Revised November* 2008]