Gaussian process emulation of dynamic computer codes

By Stefano Conti,

Centre for Health Economics, University of York, York, YO10 5DD, UK.

sc536@york.ac.uk

John Paul Gosling, Jeremy Oakley and Anthony O'Hagan.

Department of Probability & Statistics, University of Sheffield, Sheffield, S3 7RH, UK.

j.p.gosling@sheffield.ac.uk j.oakley@sheffield.ac.uk a.ohagan@sheffield.ac.uk

Abstract

Computer codes are used widely in scientific research to study and predict the behaviour of complex systems. The run times of computationally-intensive computer codes are often such that they are impractical to run the thousands of times as is conventionally required for uncertainty analysis, sensitivity analysis or calibration. In response to this problem, efficient techniques have been developed based on a statistical representation of the computer code. The approach, however, is less straightforward for dynamic computer 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 experiments; Dynamic simulators; Emulation; Gaussian process; Recursive 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 size and complexity of a simulator

can become a problem when it is necessary to make very 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 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; for example, see Sacks et al. (1989), Kennedy & O'Hagan (2001) or O'Hagan (2006). An emulator is a statistical representation of $f(.)$ that is constructed using a training sample of simulator runs. Uncertainty and sensitivity analyses can then be tackled using the emulator as shown in 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 model a system that is evolving over time and operate iteratively over fixed time steps. 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 specific system being simulated by that run. 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 simulator outputs 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 data set.

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 run the emulator iteratively. In this paper, we develop the second of these strategies, which entails two distinct developments of the existing methodology as described in Sacks et al. (1989) and in Conti & O'Hagan (2007). Emulation models our uncertainty about the simulator; however, this is secondary to the uncertainty in the simulator output caused by our uncertainty about the model inputs. Due to the iterative nature of the emulator proposed in this paper, we are able to handle uncertainty in the time-varying forcing inputs that are usually taken as being known. In § 2, we review the theory of emulation for multi-output simulators. We develop the theory so that we can emulate dynamic simulators in § 3. To illustrate these ideas, a rainfall-runoff simulator is considered in § 4 in which three state variables evolve over a number of days.

2 Emulation of complex simulators

In this section, we review the theory of emulation for multi-output simulators as presented in Conti & O'Hagan (2007). 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. We regard $f(.)$ as an unknown function, and, in line with O'Hagan (1992), we represent it by the q-dimensional Gaussian process:

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

The notation in (1) means that for all x, $E\left\{f(x) \,|\, B, \Sigma, R\right\} = m(x)$ and for all x and x', $\operatorname{cov}\left\{f(x), f(x') \,|\, B, \Sigma, R\right\} = c(x, x')\Sigma$, 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 model the mean and covariance functions in terms of further unknown hyperparameters B and R by

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

Here $h\colon \mathcal{X} \longmapsto \mathbb{R}^m$ is a known vector of m regression functions $h_1(x), \ldots, h_m(x)$ shared by each individual function $f_j(\cdot),\ j = 1 \ldots, q$; $B = [\beta_1 \cdots \beta_q] \in \mathbb{R}_{m,q}$ is a matrix of regression coefficients; and $R = \text{diag}\{\theta_i^{-2}\}$ is a diagonal matrix of p positive length scale parameters. The length scales are also called ranges in Cressie (1993) and correlation lengths in Santner et al. (2003); and the squared-inverse of the length scales are called roughness parameters in 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. When we develop the emulation technique for dynamic simulators in § 3, the linear term is important as we expect each state variable to be strongly influenced by its previous value. The form of $c(\cdot, \cdot)$ assumed in equation (2) implies that the $f_j(\cdot)$ are smooth, infinitely differentiable functions. We expect that the separable covariance function will serve well in many situations even where outputs are not of a common type. The reason for this is as follows. This covariance function assumes, 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 outputs are strongly correlated, then they must necessarily have very similar length scales. If they are only weakly correlated, then of course they can have different length scales, but we can use independent emulators in this case.

We start by running the computer code on a pre-selected 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 $h(.)$ may require a more complex structure for us to confident about our emulator's performance.

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

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

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

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

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

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

$$c^*(x_1, x_2) = c(x_1, x_2) - t^{\mathrm{T}}(x_1) A^{-1} t(x_2) \ .$$

One possible way to obtain the posterior process of $f(\cdot)$ conditional on R alone is by 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 non-informative prior $\pi^{\mathrm{J}}(B, \Sigma \mid R) \propto |\Sigma|^{-\frac{q+1}{2}}$ is selected. An alternative prior distribution, which allows for expert judgement for B and Σ, is given in Rougier (2007b). Combining the distribution in (3) and $\pi^{\mathrm{J}}(\cdot)$ using Bayes' theorem yields

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

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

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

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 Nagy et al. (2007) 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 of them 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:

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

where $f^{(T)}(\cdot)$ represents the T step simulator, which takes as its inputs the model-parameters x, the whole sequence $z = \{z_t : t = 1, \ldots, T\}$ of forcing inputs and the initial state vector Y_0.

A Gaussian process emulator of $f^{(T)}(\cdot)$ for given T can be constructed using the theory of § 2. Although theoretically straightforward, 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 input space $\mathcal{X}$ 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 building of the emulator of the single-step simulator $f(\cdot)$ is straightforward, but the technical challenge now is how to build 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 due to the conditioning of Y_2 on 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, to cover the area we expect the state variables to move to over the T steps and the range of the forcing inputs over the T steps. We then update our beliefs about $f(\cdot)$ given the training data to arrive at the posterior distribution given in (5).

Our exact simulation approach can be understood in terms of generating complete realizations from the posterior distribution of $f(.)$. Suppose we have obtained one such realization, $f_{(i)}(.)$. Conditional on $f(.) = f_{(i)}(.)$, there is no uncertainty about $Y_1, \ldots, Y_T$, as we just evaluate $Y_t = f_{(i)}(Y_{t-1}, z_t)$.

Given that we know the initial values of the state variables Y_0 and the forcing inputs for the first time-step z_1, we can use our emulator to predict Y_1. A simulation technique is used where we draw a realization of $Y_1 = f(z_1, Y_0)$ from the multivariate t-distribution implied by (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, then we will recover the same result we have encountered previously. This is how the simulator behaves, as it is deterministic. To draw a

random realization of $Y_2, Y_3, \ldots, Y_T$, we proceed in this sequential manner, successively drawing each Y_t from the emulator constructed by adding the random realization of $\{Y_1, Y_2, \ldots, Y_{t-1}\}$ to the training data D. By repeating this process many times, we draw a sample of emulated values from the posterior distribution of $\{Y_0, \ldots, Y_T\}$. Each time we add a simulated point to the training data set, 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 last inverse and some simple matrix arithmetic. Our simulation scheme is set out below.

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

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

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

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

Step 5. Derive posterior distribution of $f(\cdot)|D, R$ using (5).

Step 6. Simulate $f(z_t, Y_{t-1})$ from the distribution of $f(\cdot)|D, R$ and store $Y_t^{(N)}$.

Step 7. If $t = T$, set $N = N + 1$.

Step 8. If $t = T$ and $N \leq N_{\mathrm{MC}}$, goto *Step* 4 and reset $\mathcal{S}$ and D, else end.

Step 9. Add $\left(z_t, Y_{t-1}^{(N)}\right)$ to $\mathcal{S}$ and $Y_t^{(N)}$ to D, set $t = t + 1$ and goto *Step* 5.

Consider obtaining the complete realization $f_{(i)}(.)$. 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 due to limitations of computer storage. We sample each $f(s)$

sequentially, and define G_i to be the ith simulated variable. If we prespecify that we will sample $f(s_1)$, then $f(s_2)|f(s_1)$, etc., then the marginal distribution of G_i will be the marginal distribution of $f(s_i)$. Now suppose we randomize the order of the input values $\{s_1, s_2, \ldots\}$ in the sequential simulation to get $\{s_{i_1}, s_{i_2}, \ldots\}$. 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_is analytically. However, randomising 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 randomized when simulating $f(s_{i_1})$, $f(s_{i_2})|f(s_{i_1})$ etc.

In the exact simulation approach, we are randomising the order of the inputs by setting the ith input to be the value of the $(i-1)$th simulated output. This ensures that the first T simulated outputs are precisely the T output values of the realization $f_{(i)}(.)$ needed to determine $Y_1, \ldots, Y_T$.

New points added to the training data set when we condition during the simulation process can be relatively close to an existing point in $\mathcal{S}$. Hence, our uncertainty about the function at the new point may be small. Using the emulator for the single-step function, we can calculate $\text{var}(f(z_{t+1}, Y_t)|D, R, Y_0, \ldots, Y_{t-1})$. If $\text{var}(f(z_{t+1}, Y_t)|D, R, Y_0, \ldots, Y_{t-1})$ is small, we have little uncertainty about the value of $f(z_{t+1}, Y_t)$ and adding the point to the training data set 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 still have a great amount of uncertainty 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 creating the initial design. If the series of state variables strays outside this area, we may want to add more training data to cover $f(\cdot)$'s behaviour in regions that the initial design did not cover. We can also 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.

An alternative approach to this algorithm is reviewed in Bhattacharya (2007). It is possible to simulate the single-step function over the whole of the region of interest using methods described in 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 the values of the state variables at previous time-steps are ignored; that is, $p(Y_T|D, D^*, Y_0, \ldots, Y_{T-1}) = p(Y_T|D, D^*)$ 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 get a Monte Carlo sample that is equivalent to the sample we get from the method detailed in this section. Both of these simulation methods can be thought of as ways of sampling a multivariate normal vector.

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), the points are chosen before $\{Y_0, \ldots, Y_T\}$ is sampled. In order to cover all possibilities, a grid of points where we are going to sample at 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.

If the single-step emulator for either method has been built from a sufficiently large training set, the posterior uncertainty in the series of outputs will be small and this approach will provide an accurate emulation of $\{Y_0, \ldots, Y_T\}$. The question then arises whether iterating the single-step emulator in this way is more efficient than directly emulating the multi-step simulator $f^{(T)}(\cdot)$. To emulate accurately a long simulator run, 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 higher dimensionality of the input space of the full simulator will mean that a relatively large number of training runs is required for its emulation. A second consideration is the time taken to apply the Monte Carlo approach to compute the posterior distribution of $\{Y_0, \ldots, Y_T\}$ by numerically iterating the single-step emulator. In contrast, direct emulation of $f^{(T)}(\cdot)$ provides the posterior distribution of Y_T analytically through the analogue of the distribution given in (5), conditional on R. 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 actually quicker to run than the emulator. We develop an approximation that does not require the computational effort of Monte Carlo.

3.3 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, note that in the exact computation, if the training set is large enough, then the fact that a new point is added to this set at each iteration should have negligible effect. We would obtain essentially the same distribution for Y_T if we ignored this refinement and sampled each $f(z_t, Y_{t-1})$ from its posterior distribution based only on the original training set. Accordingly, the first approximation is to replace $p(Y_t|D, R, Y_0, \ldots, Y_{t-1})$ by $p(Y_t|D, R)$.

The second approximation is to assume that the distribution of Y_t is multivariate normal, for all $t = 1, 2, \ldots, T$. At $t = 1$, the distribution is multivariate t, which will be very close to normal for even a moderately large training set, but normality cannot hold for $t > 1$ except in the case that $f(\cdot)$ is linear. Nevertheless, again assuming a large enough training sample, uncertainty in any Y_t should be small, and it is reasonable to assume approximate linearity over a small enough 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 around 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,

$$Y_t|\Sigma, R \sim \mathcal{N}_q(\mu_t, V_t) \ ,$$

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, it can be shown that

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

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

The expectations and variance in equations (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 detailed in the appendix.

The computational speed of the approximation is much quicker than that of the exact simulation method or the method of Bhattacharya (2007). Only one set of matrix inversion calculations need to be performed to obtain results using the 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 being predicted is badly approximated and our posterior mean for $\{Y_0, \ldots, Y_T\}$ could be far away from the series produced by the exact simulation method. Validation

of the single-step emulator is therefore of great importance.

3.4 Uncertainty analysis of dynamic simulators

Throughout this section, we have only considered code uncertainty. Our uncertainty about the model-parameters, the forcing inputs and the initial state variables has been ignored. Nevertheless, the uncertainty in the model output due to input uncertainty is a key aspect of analysing complex computer codes.

Uncertainty analysis 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 it yields results comparable to those of uncertainty analysis in the standard emulation framework of O'Hagan et al. (1999) 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.

4 Dynamic rainfall-runoff simulator

We now give an example of emulation involving a dynamic rainfall-runoff simulator. The simulator described in Kuczera et al. (2006) is a rainfall-runoff

simulator that models the interaction between three water-bearing pools near a river. Hence, the simulator has three state variables: 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: 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 only 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 utilizes ranges of the inputs that cover 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 using 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 Figure 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; specifically, $h_s \in [100, 125]$ and $h_r \in [2, 2.5]$. The results of the exact simulation scheme are then shown in Figure 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 usefulness of the emulator is shown in Figures 3 and 4. Figure 3 is the result of employing the exact simulation scheme on eight 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 only 70 training runs of the single-step function. The approximation gives very similar results for the rainfall-runoff simulator. The uncertainty bounds are slightly smaller for the approximation.

Using a multi-output emulator to deal with the whole 250 step series would be computationally more demanding: we would have a output space of 750 dimensions. By breaking the process down into single time-steps, we reduce the problem to a manageable size. However, the outer-product emulator of Rougier (2007a) makes a high-dimensional multi-output emulator computationally feasible.

As stated in § 3.4, we are typically interested in the uncertainty in the simulator outputs due to our uncertainty in its inputs. To demonstrate this for a dynamic simulator, we now say 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:

$$h_s(0) \sim N(0.4, 0.01), \quad h_{gw}(0) \sim N(7.5, 1), \quad h_r(0) \sim N(0.145, 0.0005),$$
$$x_1 \sim N(1.5, 0.4), \qquad x_2 \sim N(2, 0.36), \qquad x_3 \sim N(6.5, 0.36).$$

For the forcing inputs, we take a known sequence and add noise; we used uniform noise over $[0, 0.25]$ for $rain(t)$ and $N(0, 0.0625)$ for $PET(t)$. Note that 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 due to 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 § 3.3 conditional on the drawn values, and we repeat these two steps many times.

We are interested in the value of the state variables over 10 time-steps. In order 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. Note that the 200 10-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 due to our uncertainty about the inputs and the code. It can be seen that the two approaches yield similar results and that the approximation to the dynamic emulator uses a fraction of the single-step training runs. In addition to the results given in Table 1, we also get the uncertainty analysis results for all the intermediate time-steps when using the dynamic emulator, and these are shown in Figure 5. To obtain these results, we used a simple Monte Carlo scheme, and, for a small set of 200 single-step training runs, it requires a lot of computational effort to obtain the uncertainty analysis results.

5 Summary

The methods presented in this paper offer solutions to the problem of emulating dynamic simulators. If we consider the simulator's single-step action on the state variables, we can emulate any series by iteratively using an emulator of the single-step function. We have also found that massive savings in computation time can be made by using a simple, yet often accurate, approximation.

In situations where running the model a modest amount of times is so computationally expensive that emulation through standard procedures is infeasible, the emulation techniques detailed in this paper can offer time savings as the single-step function does not need to be evaluated as often. In § 4, we saw that

we needed a tenth of the single-step evaluations using our new method to get almost the same results as in the standard analysis. 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.

The effectiveness of the emulation of dynamic simulators, either by exact simulation or approximation, depends on the level of uncertainty about the single-step function over the range of input values that will be visited by the simulator in producing the true series. If the single-step function is appreciably non-linear, then it will be difficult to keep the number of training runs down to an acceptable level. However, as in the rainfall-runoff simulator described in § 4, the single-step functions of dynamic simulators tend to be almost linear and system variables can come close to repeating themselves.

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. This may not be practical if we are running the simulator over many time-steps.

The theory of emulating dynamic computer codes that has been presented in this paper has the potential to help us understand the uncertainties in computa-

tionally expensive simulators. By reducing the simulator down 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 that add on the discrepancy at a fixed time point as set out in Kennedy & O'Hagan (2001) to a scheme where the model-to-reality discrepancy is considered at every time-step.

Acknowledgements

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 that is analysed in this paper. We would also like to thank the editor and the anonymous referee 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, $R' = \text{diag}\left\{(\theta')^{-2}\right\}$ and $R'' = \text{diag}\left\{(\theta'')^{-2}\right\}$ 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 frequently invoked for appropriate $G \in \mathbb{R}_{q,q},\ g \in \mathbb{R}^q,\ B \in \mathbb{R}_{s,q}$ and $b \in \mathbb{R}^s$:

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

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

$$
\begin{aligned}
\mu_{t+1} &= E\left\{f(z_{t+1}, Y_t) \,|\, D\right\} = E\left\{m^{\star\star}(z_{t+1}, Y_t) \,|\, D\right\} \\
&= \hat{B}^{\mathrm{T}} E\left\{h(z_{t+1}, Y_t) \,|\, D\right\} + (D - H\hat{B})^{\mathrm{T}} A^{-1} E\left\{t(z_{t+1}, Y_t) \,|\, D\right\}, \qquad (8)
\end{aligned}
$$

where we have used the first approximation to justify independence of Y_t and the $f(\cdot)$ because the latter is treated as a new realization of the emulator. Two expectations need to be evaluated. The first will depend on the form of $h(\cdot)$, but, in the widely used case $h(x)^{\mathrm{T}} = (1, x^{\mathrm{T}})$, we have $E\left\{h(z_{t+1}, Y_t) \,|\, D\right\}^{\mathrm{T}} =$

$(1, z_{t+1}^{\mathrm{T}}, \mu_t^{\mathrm{T}})$. The second expectation is a vector that for $r = 1, \ldots, n$ is

$$\begin{aligned} E\left\{t_r(z_{t+1}, Y_t) \,|\, D, R\right\} &= \exp\left\{-(z_{t+1} - s_r')^{\mathrm{T}} R' (z_{t+1} - s_r')\right\} \\ &\quad \cdot E\left[\exp\left\{-(Y_t - s_r'')^{\mathrm{T}} R'' (Y_t - s_r'')\right\} \,|\, D\right] \\ &= |2V_t R'' + I_q|^{-\frac{1}{2}} \exp\left\{-(z_{t+1} - s_r')^{\mathrm{T}} R' (z_{t+1} - s_r')\right\} \\ &\quad \cdot \exp\left\{-(\mu_t - s_r'')^{\mathrm{T}} \left(2V_t + R''^{-1}\right)^{-1} (\mu_t - s_r'')\right\}. \end{aligned} \tag{9}$$

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

$$\begin{aligned} V_{t+1} &= \mathrm{var}\left\{f(z_{t+1}, Y_t) \,|\, D, R, \Sigma\right\} \\ &= \mathrm{var}\left\{m^{\star\star}(z_{t+1}, Y_t) \,|\, D, R\right\} + E\left[c^{\star\star}\left\{(z_{t+1}, Y_t), (z_{t+1}, Y_t)\right\} \,|\, D, R\right] \Sigma, \end{aligned} \tag{10}$$

again by the Law of Iterated Expectations. The first term in (10) is given by

$$\begin{aligned} \mathrm{var}\left\{m^{\star\star}(z_{t+1}, Y_t) \,|\, D, R\right\} &= \hat{B}^{\mathrm{T}} \mathrm{var}\left\{h(z_{t+1}, Y_t) \,|\, D\right\} \hat{B} \\ &\quad + \hat{B}^{\mathrm{T}} \mathrm{cov}\left\{h(z_{t+1}, Y_t), t(z_{t+1}, Y_t) \,|\, D, R\right\} A^{-1} (D - H\hat{B}) \\ &\quad + (D - H\hat{B})^{\mathrm{T}} A^{-1} \mathrm{cov}\left\{t(z_{t+1}, Y_t), h(z_{t+1}, Y_t) \,|\, D, R\right\} \hat{B} \\ &\quad + (D - H\hat{B})^{\mathrm{T}} A^{-1} \mathrm{var}\left\{t(z_{t+1}, Y_t) \,|\, D, R\right\} A^{-1} (D - H\hat{B}). \end{aligned} \tag{11}$$

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

$$\operatorname{var}\{h(z_{t+1}, Y_t) \mid D\} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & V_t \end{pmatrix}, \tag{12}$$

$$\operatorname{cov}\{t(z_{t+1}, Y_t), h(z_{t+1}, Y_t) \mid D, R\} = (0 \;\; 0 \;\; \operatorname{cov}\{t_l(z_{t+1}, Y_t), Y_t \mid D, R\}) \;\; . \tag{13}$$

Then the elements of the remaining terms required for the evaluation of (11) are derived using (9) and the following two results. First, for $r, l = 1, \ldots, n$,

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

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

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

Finally, letting $\mathrm{Tr}(\cdot)$ denote the matrix trace operator, some linear algebra manipulations allow to compute the second summand in (10) via

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

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 marginalising with respect to Σ is documented below.

Marginalization w.r.t. the dispersion matrix Σ

The complex way in which the dispersion and length scale matrices Σ and R enter formulae (8) to (10) precludes any closed-form marginalization. Focusing upon Σ, it is apparent that brute-force integration based on sampling from an inverse-Wishart population is computationally too inefficient to be pursued. A reasonable alternative was sought in integrating over a more tractable space, in the hope of significantly simplifying the current $(q\frac{q+1}{2})$-dimensional problem. This was attained by recalling from multivariate probability theory that the Student's process (5) can be defined equivalently as follows: by mixing the distribution of $[f(\cdot)\,|\,R, D] \sim \mathcal{N}_q\left(m^{\star\star}(\cdot), c^{\star\star}(\cdot)\Sigma\right)$ with the p.d.f. of $[\Sigma\,|\,R, D] \sim \mathcal{W}_q^{-1}\Big((n-m)\hat{\Sigma}; n-m\Big)$ or by mixing $[f(\cdot)\,|\,\xi, R, D] \sim \mathcal{N}_q\left(m^{\star\star}(\cdot), \xi c^{\star\star}(\cdot) D^{\mathrm{T}} G D\right)$, where $G = A^{-1} - A^{-1}H\left(H^{\mathrm{T}}A^{-1}H\right)^{-1}H^{\mathrm{T}}A^{-1}$, with an auxiliary r.v. $\xi \sim \chi^{-2}_{n-m}$.

It becomes possible to efficiently marginalize Σ out of formulae (8) to (10) 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 $\xi D^{\mathrm{T}} G D$, and thereafter the auxiliary quantity ξ be integrated out via some one-dimensional numerical technique. For instance,

we have

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

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

Marginalization w.r.t. the length scales

For simplicity, it is assumed that the length scales are a priori independent of both B and Σ: recalling the prior proposed in § 2 for (B, Σ), this leads to the full prior

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

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

$$
\begin{aligned}
\pi(B, \Sigma, R \,|\, D) \propto \pi_{\mathrm{R}}(R) |A|^{-\frac{q}{2}} |\Sigma|^{-\frac{n-m+q+1}{2}} \exp\Bigg\{ -\frac{1}{2} \Big[&\mathrm{Tr}\left(D^{\mathrm{T}} G D \Sigma^{-1}\right) \\
&+ \mathrm{Tr}\left\{(B - \hat{B})^{\mathrm{T}} H^{\mathrm{T}} A^{-1} H (B - \hat{B}) \Sigma^{-1}\right\} \Big] \Bigg\}
\end{aligned}
$$

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

$$\pi(\Sigma, R \,|\, D) \propto \pi_{\mathrm{R}}(R)|A|^{-\frac{q}{2}}|H^{\mathrm{T}}A^{-1}H|^{-\frac{q}{2}}|\Sigma|^{-\frac{n-m+q+1}{2}}$$

$$\times \exp\left\{-\tfrac{1}{2}\mathrm{Tr}\left(D^{\mathrm{T}}GD\Sigma^{-1}\right)\right\},$$

$$\pi_R(R \,|\, D) \propto \pi_{\mathrm{R}}(R)|A|^{-\frac{q}{2}}|H^{\mathrm{T}}A^{-1}H|^{-\frac{q}{2}}|D^{\mathrm{T}}GD|^{-\frac{n-m}{2}}, \qquad (14)$$

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 (14) and used this estimate as our value for R. As a concluding remark, characterising the smoothness of the code's response surface by means of (14) 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 is computationally expensive and often unnecessary.

References

Abt, M. (1999). Estimating the prediction mean squared error in gaussian stochastic processes with exponential correlation structure. *Scandinavian Journal of Statisitcs* 26 563–578.

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

Conti, S. & O'Hagan, A. (2007). Bayesian emulation of complex multi-output and dynamic computer models. Tech. rep., *Research report 569/07*, Department of Probability and Statistics, University of Sheffield. Submitted to *Journal of Statistical Planning and Inference*.

Cressie, N. (1993). *Statistics for spatial data (rev.ed.)*. New York: Wiley.

Kennedy, M. & O'Hagan, A. (2001). Bayesian calibration of computer models (with discussion). *J. R. Statist. Soc. Ser. B* 63 425–464.

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. *Journal of Hydrology* 331 161–177.

Morris, M. & Mitchell, T. (1995). Exploratory designs for computer experiments. *J. Statist. Planning and Inference* 43 381–402.

Nagy, B., Loeppky, J. L. & Welch, W. J. (2007). Fast bayesian inference for gaussian process models. Tech. rep., 230, Department of Statistics, The University of British Columbia.

Oakley, J. & O'Hagan, A. (2002). Bayesian inference for the uncertainty distribution of computer model outputs. *Biometrika* 89 769–784.

Oakley, J. & O'Hagan, A. (2004). Probabilistic sensitivity analysis of complex models: a bayesian approach. *J. R. Statist. Soc. Ser. B* 66 751–769.

O'Hagan, A. (1992). Some bayesian numerical analysis. In *Bayesian Statistics 4* (eds. Bernado, J.M. *et al.*). Oxford: Oxford University Press, 345–363.

O'Hagan, A. (2006). Bayesian analysis of computer code outputs: a tutorial. *Reliability Engineering and System Safety* 91 1290–1300.

O'Hagan, A., Kennedy, M. & Oakley, J. (1999). Uncertainty analysis and other inference tools for complex computer codes (with discussion). In *Bayesian Statistics 6* (eds. Bernado, J.M. et al.). Oxford: Oxford University Press, 503–524.

Rougier, J. (2007a). Efficient emulators for multivariate deterministic functions. Tech. rep., Department of Mathematics, Bristol University.

Rougier, J. (2007b). Lightweight emulators for complex multivariate functions. Tech. rep., Department of Mathematics, Bristol University.

Sacks, J., Welch, W., Mitchell, T. & Wynn, H. (1989). Design and analysis of computer experiments. *Statistical Science* 4 409–423.

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. *Numerical Mathematics* 13 354–356.

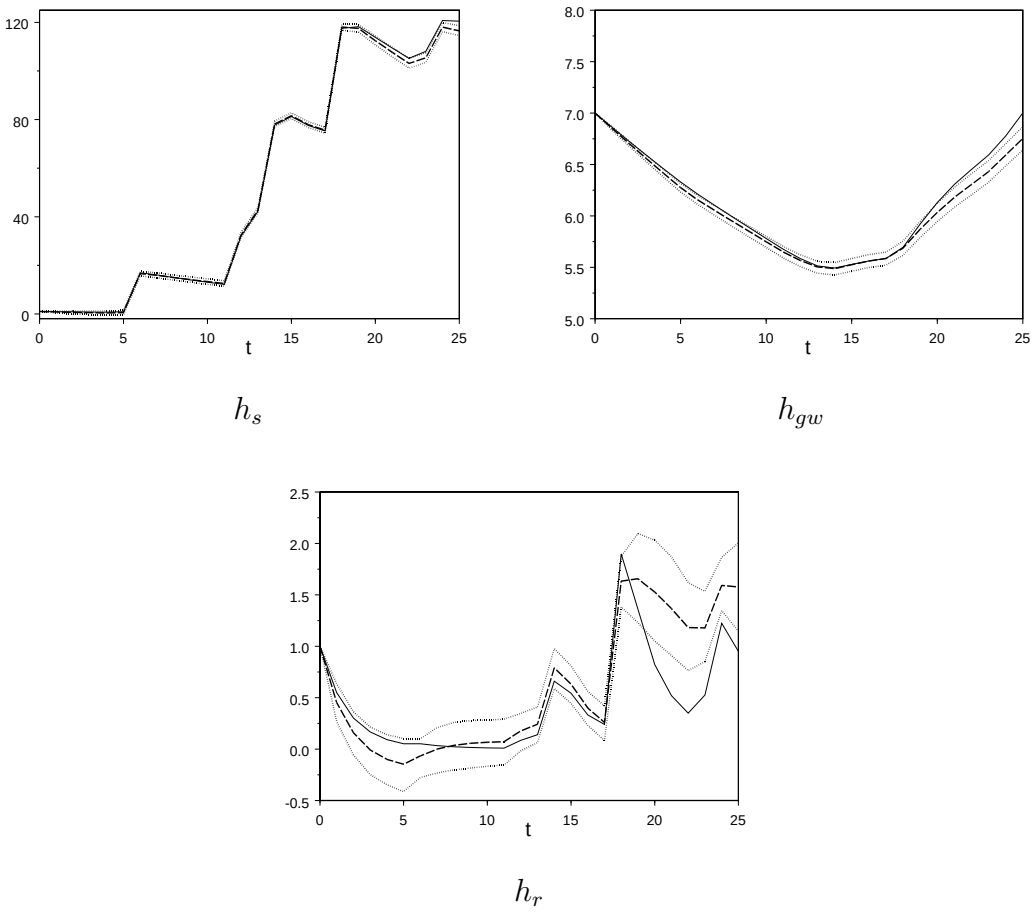

Figure 1: Posterior medians (dashed) and 95% credible intervals (dotted) for twenty-five time-steps and the actual series (solid).

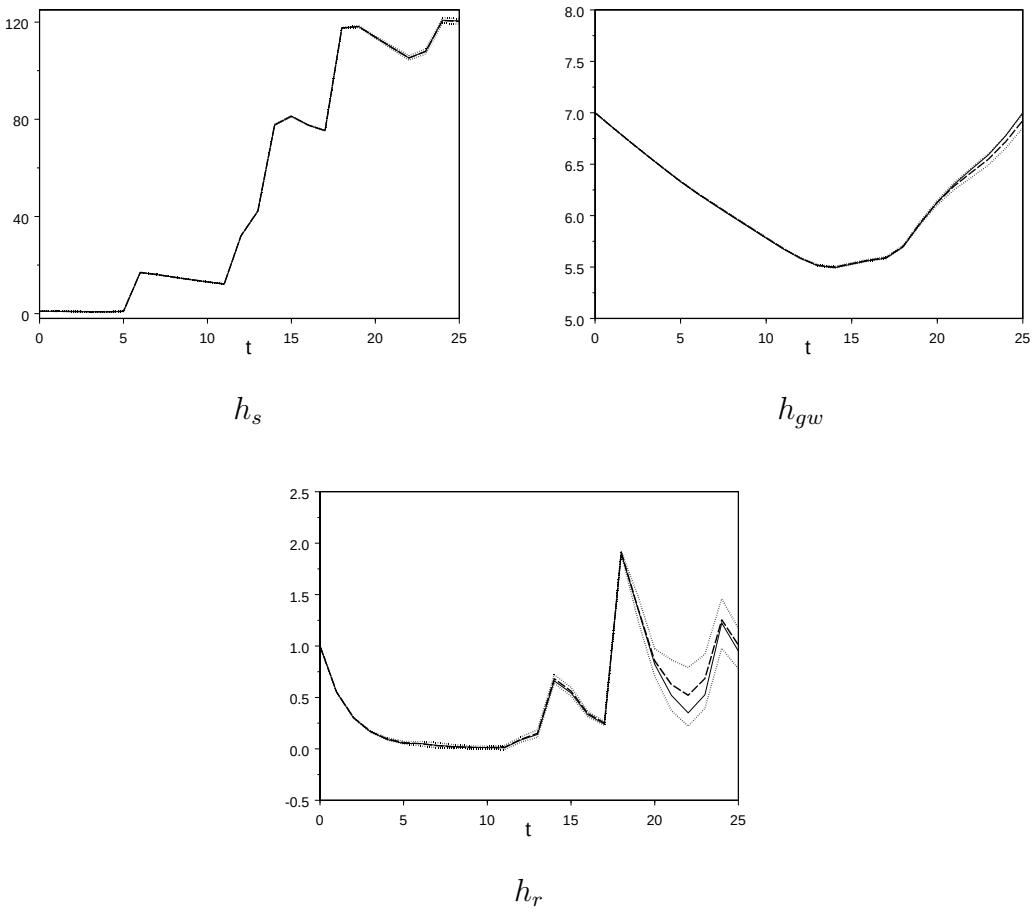

Figure 2: Posterior medians (dashed) and 95% credible intervals (dotted) for twenty-five time-steps and the actual series (solid) with twenty extra training runs.

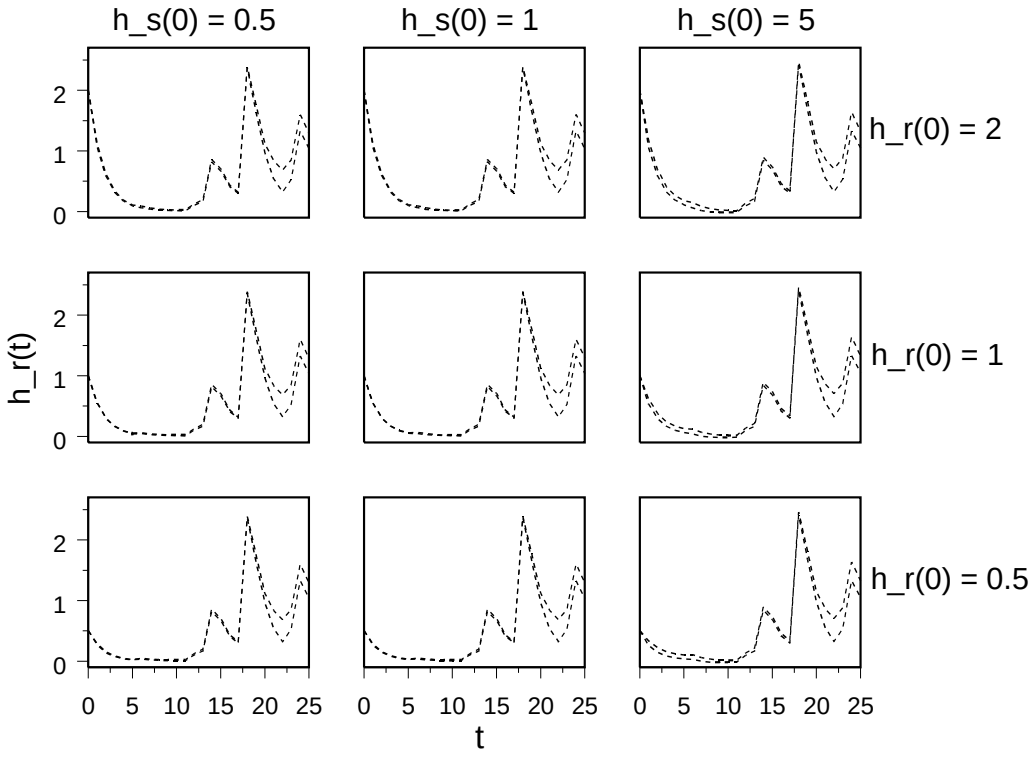

Figure 3: Nine sets of 95% posterior credible intervals for twenty-five time-steps for different values of $h_s(0)$ and $h_r(0)$.

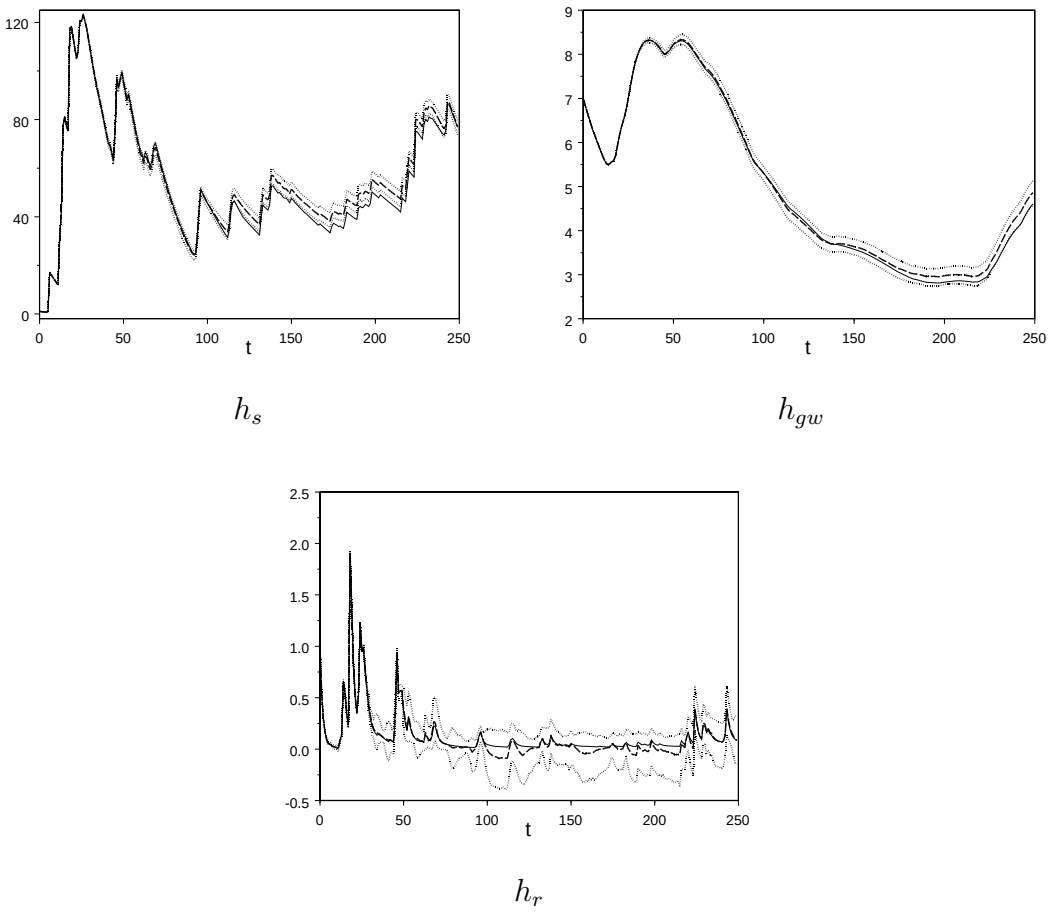

Figure 4: Posterior medians (dashed) and 95% credible intervals (dotted) for two-hundred-and-fifty time-steps and the actual series (solid).

	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	

Table 1: Uncertainty analysis results for the rainfall-runoff simulator after 10 time-steps.

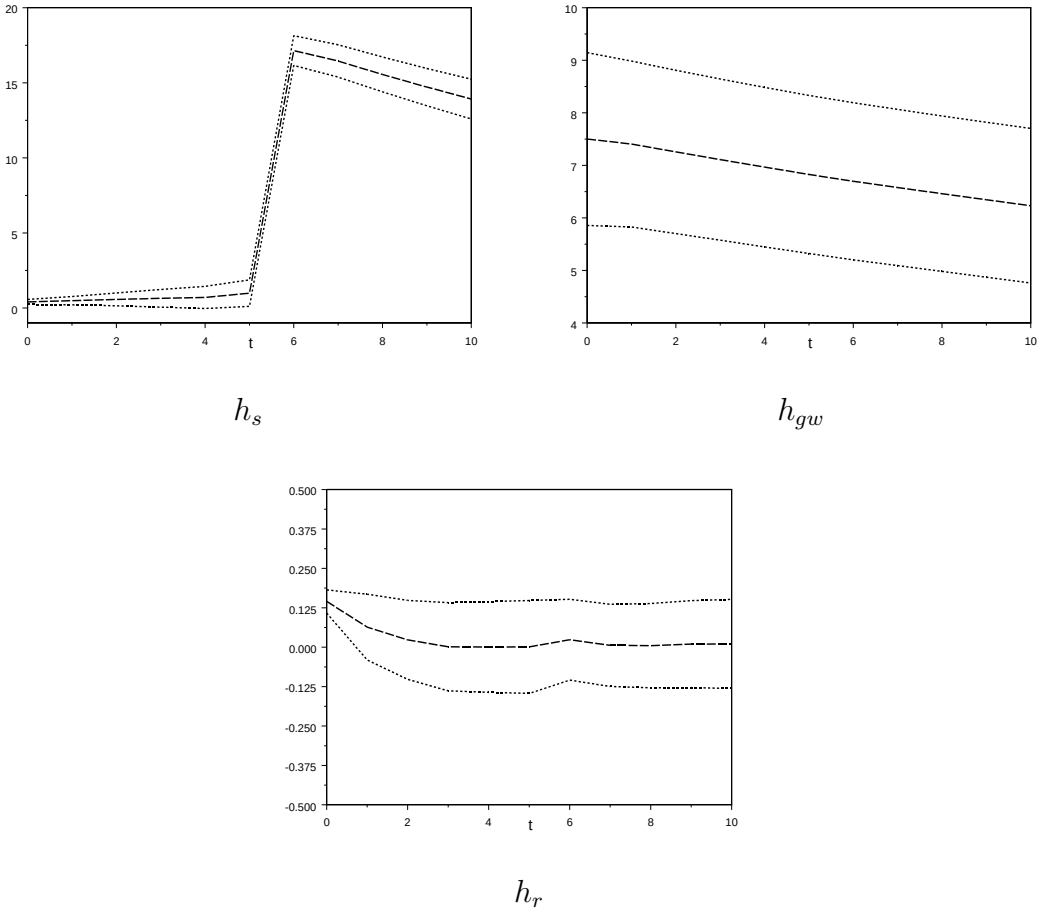

Figure 5: Posterior medians (dashed) and 95% credible intervals (dotted) for ten time-steps including uncertainty about the simulator inputs.