Some primary musings on uncertainty and sensitivity analysis for dynamic computer codes

John Paul Gosling

Note that this document will only make sense (maybe) if the reader is familiar with the terminology used in Conti et al. (2007).

1 Uncertainty and sensitivity analysis for complex computer codes

Uncertainty analysis (UA) is the quantification of uncertainty about the output of some computer code or simulator given that the inputs to that computer code are unknown. Uncertainty analysis can be posed as the following question: if, for some simulator represented by a function $f(.)$, the true input X is unknown and has distribution $G(X)$, what is the distribution of $Y = f(X)$?

Sensitivity analysis (SA) is concerned with investigating how an output of a simulator responds to changes in the simulator's inputs. In (global) sensitivity analysis, the interest is in quantifying the effect on an output of varying an input parameter over some range. If there is uncertainty regarding the values of various input parameters in the simulator, a sensitivity analysis would consider the contribution of individual uncertain inputs (or groups of inputs) to the uncertainty in the outputs.

An extensive overview of traditional approaches to UA and SA can be found in Saltelli et al. (2000). These traditional techniques can become computationally expensive if the simulator takes more than a few seconds to run. Emulation techniques have been developed that can efficiently perform both uncertainty and sensitivity analyses; these methods are detailed in Oakley and O'Hagan (2002) and Oakley and O'Hagan (2004). The rest of this report questions how far we

can take the emulation of dynamic simulators as given in Conti et al. (2007) in terms of achieving results similar to what is possible with standard emulation techniques.

2 UA for dynamic computer codes

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 in the context of a many-step run of the simulator can be classified as *model-parameters* and *forcing inputs*. Model-parameters have fixed values for all the time steps of a simulator run, and so they have the same values for each cycle of the single-step simulator. 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.

We are typically interested in the uncertainty in the simulator outputs due to our uncertainty in its inputs. To demonstrate this for a rainfall-runoff model, which is represented diagrammatically in Figure 1, we now say that we are uncertain about the initial values of the three state variables ($h_s(0)$, $h_{gw}(0)$ and $h_r(0)$) and three of the most influential model-parameters (x_1, x_2 and x_3 say). The following distributions were given to the uncertain inputs:

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

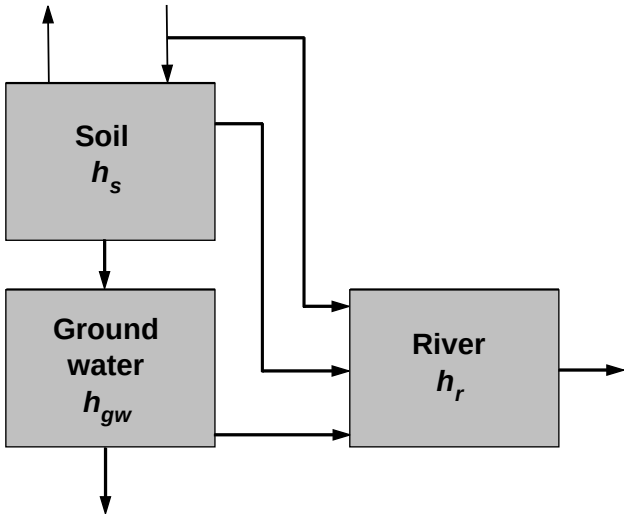

Figure 1: A simple three-compartment rainfall-runoff simulator (the arrows show the flow of water through the system).

A simple uncertainty analysis can be performed through a Monte Carlo scheme: first, we draw from the input distributions, then we apply the approximation conditional on the drawn values and we repeat these two steps many times.

We are interested in the value of the state variables after 25 time-steps. In order to emulate the single-step function well, we required 200 single-step training runs. We also carried out a uncertainty analysis using standard emulation techniques where 50 training runs were required to produce comparable emulator accuracy. Note that the 50 training runs in the standard emulation case are equivalent to 1250 single-step training runs. The results are given in Table 1 and Figures 2, 3 and 6 (the figures plot the posterior mean and 4 standard deviation range for time points 20 to 25 and the final range is the result of the standard emulation). It can be seen that the two approaches yield similar results and that the approximation to the dynamic emulator uses a fraction of the training runs. However, in order to obtain these results, we used a simple Monte Carlo scheme. For a small set of 200 single-step training runs, it requires a lot of computational

	Standard emulation		Approximation	
	Mean	Variance	Mean	Variance
$h_s(25)$	121.62	5.57	121.91	5.93
$h_{gw}(25)$	7.40	1.01	7.50	1.00
$h_r(25)$	0.91	0.01	0.91	0.01
Number of single-step evaluations	1250		200	

Table 1: Uncertainty analysis results for the rainfall-runoff simulator.

effort to get accurate results.

In this Monte Carlo scheme, the type of estimates that we produce are of interest. The approximation to the exact simulation approach gives us the following:

$$E_{\mathbf{f}(.)}\left(\mathbf{Y}_t|\mathbf{X}\right) \text{ and } Var_{\mathbf{f}(.)}\left(\mathbf{Y}_t|\mathbf{X}\right). \tag{1}$$

When we employ a Monte Carlo scheme to this with respect to the inputs $\mathbf{X}$, we can get approximations of the following:

$$E_{\mathbf{X}}\left[E_{\mathbf{f}(.)}\left(\mathbf{Y}_t|\mathbf{X}\right)\right], \tag{2}$$

$$Var_{\mathbf{X}}\left[E_{\mathbf{f}(.)}\left(\mathbf{Y}_t|\mathbf{X}\right)\right], \tag{3}$$

$$E_{\mathbf{X}}\left[Var_{\mathbf{f}(.)}\left(\mathbf{Y}_t|\mathbf{X}\right)\right]. \tag{4}$$

The means reported in Table 1 are calculated using (2). The variances are calculated by combining (3) and (4) by using the law of total variance:

$$Var_A(A) = E_B\left[Var_A\left(A|B\right)\right] + Var_B\left[E_A\left(A|B\right)\right].$$

We can compare these to the expectations and variances we get out of UA using

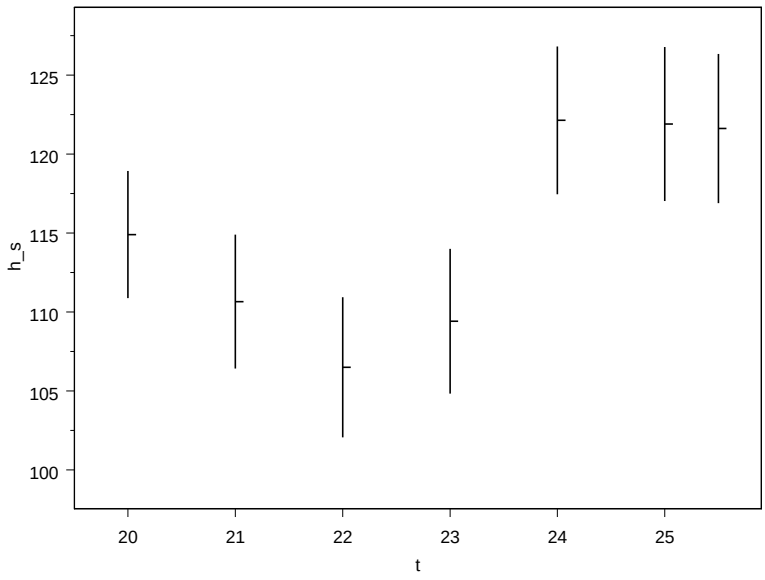

Figure 2: UA results for h_s.

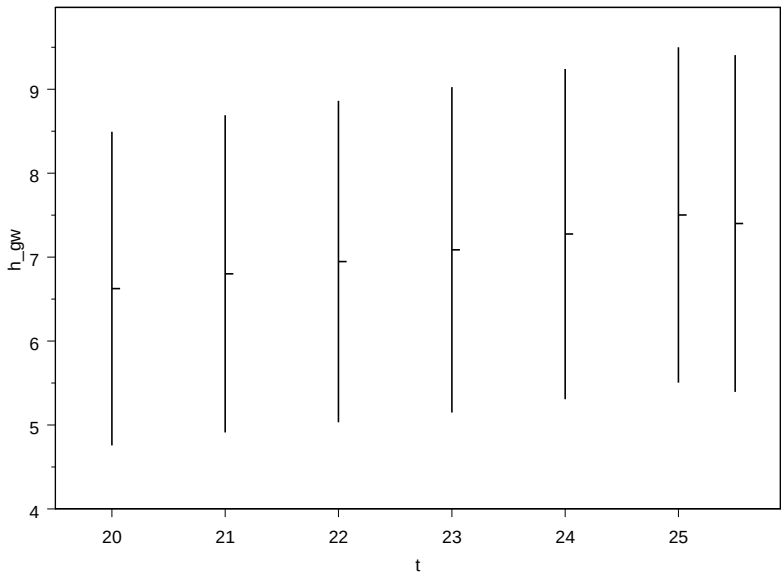

Figure 3: UA results for h_{gw}.

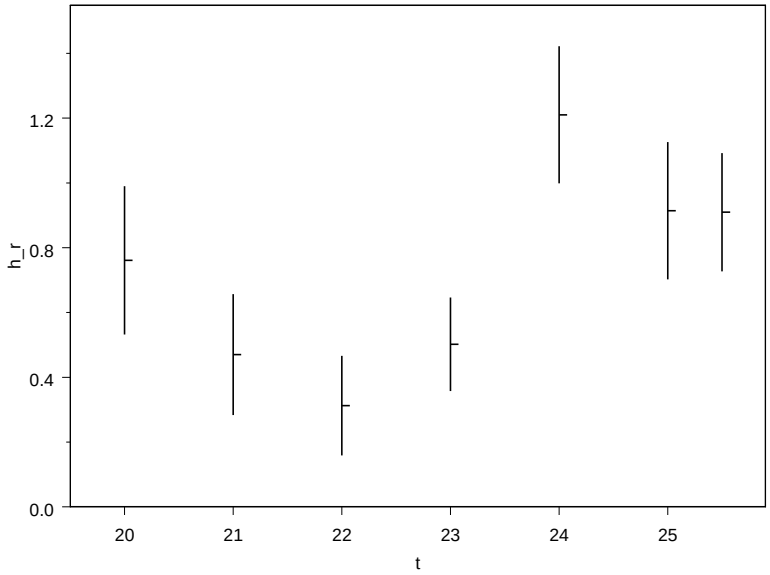

Figure 4: UA results for h_r.

standard emulation techniques:

$$E_{\mathbf{f}(.)}\left[E_{\mathbf{X}}\left(\mathbf{Y}_t|\mathbf{X}\right)\right], \tag{5}$$

$$Var_{\mathbf{f}(.)}\left[E_{\mathbf{X}}\left(\mathbf{Y}_t|\mathbf{X}\right)\right], \tag{6}$$

$$E_{\mathbf{f}(.)}\left[Var_{\mathbf{X}}\left(\mathbf{Y}_t|\mathbf{X}\right)\right]. \tag{7}$$

You will notice $\mathbf{f}(.)$ and $\mathbf{X}$ have swapped roles in equations (5) to (7). However, it is easy to see that

$$\begin{aligned} &Var_{\mathbf{X}}\left[E_{\mathbf{f}(.)}\left(\mathbf{Y}_t|\mathbf{X}\right)\right] + E_{\mathbf{X}}\left[Var_{\mathbf{f}(.)}\left(\mathbf{Y}_t|\mathbf{X}\right)\right] \\ &\qquad = Var_{\mathbf{f}(.)}\left[E_{\mathbf{X}}\left(\mathbf{Y}_t|\mathbf{X}\right)\right] + E_{\mathbf{f}(.)}\left[Var_{\mathbf{X}}\left(\mathbf{Y}_t|\mathbf{X}\right)\right], \end{aligned} \tag{8}$$

using the law of total variance. The nice thing about (6) is that it can be interpreted as the portion of uncertainty due to the lack of knowledge about $\mathbf{f}(.)$;

that is, uncertainty in the emulation process. Similarly, (7) can be viewed as the portion uncertainty due to the lack of knowledge about the inputs.

One extremely crude way to estimate (6) and (7) could be to use the following:

$$Var_{\mathbf{f}(.)}\left[E_{\mathbf{X}}\left(\mathbf{Y}_t|\mathbf{X}\right)\right] \approx Var_{\mathbf{f}(.)}\left[\mathbf{Y}_t|\mathbf{X}^*\right], \tag{9}$$

where $\mathbf{X}^*$ is the mode of our distribution for $\mathbf{X}$. We could plug the approximation into (8) to get a value for $E_{\mathbf{f}(.)}\left[Var_{\mathbf{X}}\left(\mathbf{Y}_t|\mathbf{X}\right)\right]$. I cannot stress how wrong it would be to do this (as a matter of fact, Gosling (2006) spends several pages showing why you should not use this approach); however, I included this thought to stress how much we want to get at expressions like (6) and (7).

We would like to have analytical expressions for equations (5) to (7), which could be calculated without the computational effort of Monte Carlo, for all time-steps under consideration. This is infeasible for all but the simplest input distributions when using standard emulation techniques. To circumnavigate this, the Monte Carlo scheme can be reversed: first, we draw a complete single-step function then draw thousands of possible input configurations and use the drawn function to obtain Monte Carlo estimates of the required quantities. However, in order to do this, the single-step function must be emulated to a high degree of accuracy over all possible input configurations (not just the initial value ranges). This may make this approach too expensive for all but the most well-behaved functions.

A sizeable challenge in the uncertainty analysis of complex computer models is accounting for uncertainty in the forcing inputs. In the rainfall-runoff model, there are two forcing inputs being specified at each time, rainfall and evapotranspiration potential (PET). If we look at the evolution of the state parameters over a year, we need to consider our beliefs about 730 forcing inputs. However, it is clear in Figure 5 that there is probably strong correlation between the two forcing inputs in the example and possible autocorrelation. If we could fit some

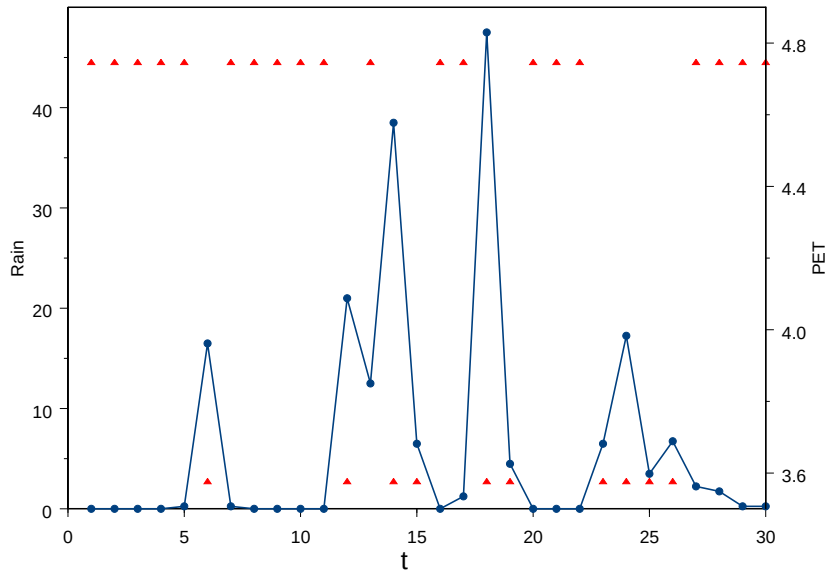

Figure 5: Forcing inputs over 30 time-steps; the red triangles are PET and the blue circles are rainfall.

sort of time series model to our beliefs about the forcing inputs, we could make conditional draws from that distribution at each step of our Monte Carlo scheme.

Another model that is available to MUCM is a dynamic vegetation model. This model has a forcing input called degree days that are very important for the output of the model. Over a certain period of the year, the model stores up the number of days that cross a certain temperature threshold. Once it reaches a certain point, a process inside the model switches on and the value of that forcing input is then ignored (until the next year). I've seen many inputs like this in models; it is not clear how we can model them. It may be the case that this set of forcing inputs are directly related to a few model-parameters and can be ignored; only the model builders can tell us.

3 SA for dynamic computer codes

What are we really interested in: the behaviour of the single-step function or the behaviour of the simulator over multiple time-steps? A sensitivity analysis of the single-step function will tell you a lot about how the state variables will evolve from one time point to the next. This could be done using standard SA results given in Oakley and O'Hagan (2004). However, drawing from my own experience with this type of model, I believe that the value of the state variables will be of paramount importance in determining their value at the next time-step. In addition to this, I believe that the single-step function maps $\mathbf{y}$ to $\approx \mathbf{y}$. A SA of the single-step function could reveal a lot about the impact of the forcing and model parameters.

If we consider the sensitivity of an output to the function inputs for a particular t, then we can just use the results of Oakley and O'Hagan (2004). However, we would like to avoid separate emulation for every t — this is the whole point of Conti et al. (2007). An analytical formulation of sensitivity measures would be ideal, and, if UA can be performed analytically, there will be scope for doing this. In calculating sensitivity measures for each t, we could answer important questions about the simulator. For instance, we could measure how many steps it takes for the simulator to forget the initial values of the state variables; it is often assumed in practice that the initial values are forgotten if you run the simulator over enough time steps. Figure 6 shows h_r, from the aforementioned rainfall-runoff model, evolve over 25 time-steps from five different starting values.

Most of the sensitivity measures in Oakley and O'Hagan (2004) are made possible by the input parameters being independent. It might be possible to construct an independent parameterisation for the model-parameters and the initial state variables; however, it is not clear how this could be achieved for forcing inputs with their complex correlation structure. We may need to drop the independence assumption and develop sensitivity measures that can cope

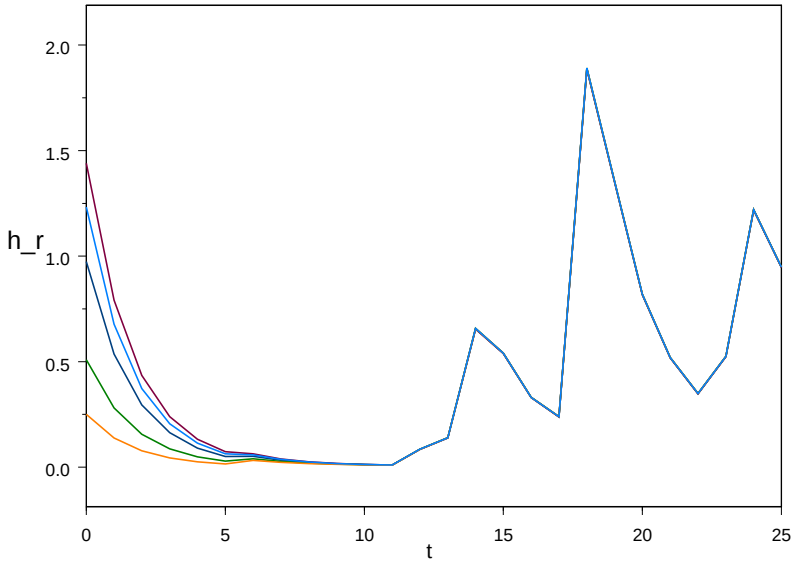

Figure 6: h_r evolving over 25 time-steps from 5 different starting points.

with highly structured input distributions.

4 To be developed

Here is all the previous pages' content put in bullet form to give a checklist of interesting questions.

- Can we avoid Monte Carlo when performing UA?
- Do we need to use the approximation to the exact simulation approach when performing UA?
- Forcing inputs require more than independent distributions; how can we model them in a sensible manner?
- SA for single-step functions is just applying known probabilistic SA techniques. Can we derive similar measures for dynamic simulators?

- What does variance decomposition mean when the input parameters are related in a potentially complex way?
- Can we combine our UA approach with calibration and data assimilation to create a sophisticated model for the model-reality discrepancy?

The final point here is potentially the biggest advance that emulation of dynamic simulators will provide. We could break down the problem of model discrepancy into single-step discrepancies where judgements might be more easily made and data could be used to inform this process on a step-by-step basis.

References

Conti, S., Gosling, J., Oakley, J., and O'Hagan, A. (2007). "Gaussian process emulation of dynamic computer codes." Technical report, Department of Probability and Statistics, The University of Sheffield.

Gosling, J. (2006). "Differences between estimates of expected computer code output with GEM-SA." Technical report, Department of Probability and Statistics, The University of Sheffield.

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

— (2004). "Probabilistic sensitivity analysis of complex models: a Bayesian approach." *J. R. Statist. Soc. Ser. B*, 66: 751–769.

Saltelli, A., Chan, K., and Scott, E. (eds.) (2000). *Sensitivity Analysis*. New York: Wiley.