Using Emulators to Estimate Uncertainty in Complex Models

Peter Challenor

National Oceanography Centre
Southampton, United Kingdom
p.challenor@noc.ac.uk

Abstract. The Managing Uncertainty in Complex Models project has been developing methods for estimating uncertainty in complex models using emulators. Emulators are statistical descriptions of our beliefs about the models (or simulators). They can also be thought of as interpolators of simulator outputs between previous runs. Because they are quick to run, emulators can be used to carry out calculations that would otherwise require large numbers of simulator runs, for example Monte Carlo uncertainty calculations. Both Gaussian and Bayes Linear emulators will be explained and examples given. One of the outputs of the MUCM project is the MUCM toolkit, an on-line recipe book for emulator based methods. Using the toolkit as our basis we will illustrate the breadth of applications that can be addressed by emulator methodology and detail some of the methodology. We will cover sensitivity and uncertainty analysis and describe in less detail other aspects such as how emulators can also be used to calibrate complex computer simulators and how they can be modified for use with stochastic simulators.

Keywords: emulator, Gaussian process, Bayes linear, sensitivity, uncertainty, calibration

1 Introduction

The increase in computing power over the last few decades has led to an explosion in the use of complex computer codes in both science and engineering. Almost every area of science and engineering now uses complex numerical simulators to solve problems that could not have been tackled only a few years ago. Examples include engineering [2], climate science, [4], and oceanography [12]. At the present time most of these simulators are deterministic but there is an increasing use of stochastic simulators as well [24]. These computer codes comprise many thousands (or even millions) of lines of code and take long times to compute even on the fastest supercomputers available today. Questions we would like to ask of the simulators include: for a given uncertainty in the inputs to the simulator what is the uncertainty in the outputs; which inputs have the most effect on the outputs; are all the inputs important; how can we relate the simulator to reality? In this paper we look at how emulators can be used to

address these problems. For a more detailed examination of the background and theory of what is discussed here see [5].

2 Uncertainty

Assuming for the moment that our simulators are deterministic rather than stochastic where does the uncertainty come from? We know that the predictions we make are not exact. (If you are tempted to think of your simulator as perfect, how much you would be prepared to bet on its result being the same as a physical experiment?) Some of the 'error' may be numerical, running on different computer architectures or in different precisions will give different answers. But for well written code these differences will be small. In general the uncertainty in our simulator outputs comes from two sources: from uncertainty in its inputs and from uncertainty in its structure. By inputs we mean all the external inputs to the simulator; these include the initial conditions, boundary conditions and parameters. Although these may have very different properties, for example initial conditions are often spatial fields while the parameters are normally collections of single numbers, we will generally treat them the same. Similarly we take a Bayesian approach and make no distinction between aleatoric uncertainty, arising from genuine randomness, and epistemic uncertainty which is a measure of our ignorance.

The input uncertainty can be thought of as the internal uncertainty within the simulator. The structural uncertainty is how our particular simulator relates to other simulators and more importantly to reality. This is discussed further below and in [5].

3 Quantifying Uncertainty

Before we consider structural uncertainty and the relationship between simulators and reality lets first consider the input uncertainty. We take a Bayesian approach to the problem, but it can be reformulated in terms of frequentist statistics [20]. The formulae generally stay the same but the justification is different.

We have some variables (y) that we are interested in and which we will call outputs. These are related to another set of variables (x), which we will call inputs. We can find the values of the outputs corresponding to the values of the inputs by running a complex computer program which we will denote by f. Mathematically we can write

$$y = f(x)$$

We will assume for now that f is deterministic, so if we run f with the same set of inputs we will obtain the same set of outputs. We assume that any numerical error is small enough to be ignored. Some or all of our inputs might be uncertain. This might arise from genuine randomness, which we call aleatoric uncertainty; or the uncertainty could be a result of lack of knowledge, called

epistemic uncertainty. The distinction between these two forms on uncertainty is not as clear as might appear. For example consider a coin toss. While the coin is in the air our uncertainty is aleatoric, but once it has landed, but is hidden from us, the uncertainty becomes epistemic. Because of this interchangeability we treat all sources of uncertainty the same and describe them with probability density functions ($\pi(x)$). If we want to know the uncertainty on y given $\pi(x)$ we need to calculate the transformation of $\pi(x)$ induced by f. If f is linear this is an easy problem and can be solved analytically, but the complex codes we are interested in are non-linear and an analytical solution does not exist. The naive solution is to use Monte Carlo methods: draw a sample from $\pi(x)$, x_i, propagate this through the program to produce $y_i = f(x_i)$. The resulting y_i are then a sample from $\pi(y)$ from which we can estimate $\pi(y)$. Such methods are effective but large samples are needed, particularly for high dimensional x and y. This means that Monte Carlo methods are not viable for expensive computer codes.

Our solution to this problem is to build a fast approximation to the full numerical simulator. This is known as an *emulator*. We not only want fast approximators, we want fast approximators that estimate their own error. It cannot be stressed too much how useful it is having knowledge of the uncertainty in the emulator estimate. When we come to validate our emulators it is invaluable as it gives us a measure against which we can gauge how far the estimator may be from the truth. Similarly it is very helpful in sequential design, where we can put the next point at the most uncertain current point and when we come to estimate the uncertainty of the outputs we need to include the uncertainty arising from the process of emulation.

4 Gaussian Processes and Emulators

Our requirement that we have an emulator that is fast and contains an estimate of its own uncertainty is satisfied by the Gaussian process, [18]. Gaussian processes are very adaptable stochastic processes which can be used to fit non-linear data. A Gaussian process (GP) is the infinite dimensional analogue of a Gaussian distribution. It is defined by a mean function and a covariance function. The mean function gives the expected value of the GP at any point. The covariance function then gives the covariance between any two points. The form of the covariance function dictates how 'smooth' a realisation of the GP is. For example if we use the 'exponential' form of the covariance function

$$cov(x_1, x_2) = \sigma^2 e^{-|x_1 - x_2|} \tag{1}$$

we get Brownian motion and any realisation does not possess any derivatives. On the other hand if we use the squared exponential covariance

$$cov(x_1, x_2) = \sigma^2 e^{-(x_1 - x_2)^2} \tag{2}$$

then all derivatives exist and we get a very smooth set of realisations. There are other forms of the covariance function, see [18] for further details. For uncertainty

quantification a limited selection of covariance functions tend to be used. We rarely believe our simulators do not possess any derivatives so we tend not use the exponential form. The squared exponential is probably the most used but this can give rise to numerical problems (see below). Other options include the Matern and generalised exponential form $\sigma^2 exp(-||x_1 - x_2||^\alpha)$.

The smoothness of the GP is dictated by its covariance function but its large scale properties come from the mean function. In general a linear model is used as the mean function

$$\mu(x) = h(x)^T \beta \tag{3}$$

where the $h(x)$ are basis functions and the β's are coefficients. The $h(x)$ can be any basis functions but are usually taken to be monomials $\{1, x, x^2, \ldots\}$. However more complex functions such as Fourier bases could be used if for example our output was on a circle. There is some discussion within the emulation community on how much effort should be put into building a good mean, or regression, model and how much the GP can be allowed to fit the large and small scale variation in the data. For example [11] make the case for not including a linear model term at all. However most practitioners do include regression terms even if they are only low order polynomials.

To analyse our simulator we use a GP as a prior and combine it with the simulator runs to produce a posterior emulator. Our prior has a mean function as given above

$$\mu(x) = h(x)^T \beta \tag{4}$$

where the β are parameters that will need to have their own prior specified. We then specify a covariance function

$$cov(x_1, x_2) = \sigma^2 c(x_1, x_2; C) \tag{5}$$

where σ^2 is a variance term and C is a matrix of 'smoothing' terms for the correlation function, c. For example if we have a squared exponential correlation function (sometimes called the Gaussian correlation function for obvious reasons) C might be a diagonal matrix such that

$$c(x_1, x_2; C) = exp(-(x_1 - x_2)^T C(x_1 - x_2)) \tag{6}$$

C does not have to be diagonal but it reduces the number of parameters to estimate and making this assumption does not appear to impact on our emulators.

5 Bayes Linear Methods

When we decided to use a Gaussian process as our emulator we made more assumptions than we needed to. The assumption that all the points on our stochastic process have a multivariate Normal distribution is not necessary. All we need assume is that second moments exist and we can then specify the mean and covariance functions without making any assumptions about the statistical distribution of the process.

Such methods that use Bayes theorem but do so in terms of first and second moments are known as Bayes Linear methods and are described in [6]. The clear advantage over a full Bayes solution is that the priors are also specified only in terms of first and second moments so we do not need to elicit full probability distributions from experts. The second advantage as we shall see is that the calculations are much simpler and faster. However nothing is free and without adding some distributional assumptions we cannot make realisations of the process or produce any form of probabilistic credibility or uncertainty limits. Because Normal distributions are defined by their first two moments there is often confusion between Bayes linear methods and making an assumption of Normality. A true Bayes Linear analysis refuses to make any distributional assumptions; although the results may look similar they are conceptually very different.

As with the GP emulator our basic form is

$$f(x) = \sum_j \beta_j h_j(x) + w(x) \tag{7}$$

The regression terms are identical but the $w(x)$ is not a Gaussian process but rather a general second order process, defined by its covariance function.

The equivalent to Bayes Theorem for Bayes Linear are the Bayes Linear update equations. If θ is a vector of our parameters (in our case this will be the β's, σ^2 and the length scales) and x are the results of our runs

$$E(\theta|x) = E(\theta) + Cov(\theta, x)V(x)^{-1}(x - \sum_j \beta_j h_j(x)) \tag{8}$$

for the adjusted expectation and by

$$V(\theta|x) = V(\theta) - Cov(\theta, x)V(x)^{-1}Cov(x, \theta) \tag{9}$$

The full equations for the posterior moments of the Bayes Linear emulator are rather more complicated and are given in full in the Core BL Emulator Thread of the MUCM toolkit [14]

6 MUCM and the Toolkit

The Managing Uncertainty in Complex Models (MUCM) consortium consists of five UK research institutions (University of Sheffield, University of Durham, University of Aston, London School of Economics and the National Oceanography Centre). As part of its research activities MUCM has set up an on-line toolkit. This consists of more than 300 pages describing most aspects of emulation. The toolkit is not a software package. The best analogy is a recipe book, one of those good recipe books that encourage you to experiment. There are worked examples so you can check that your code works and there are links to existing software packages. Part of an example page is shown in Fig. 1. The URL for the toolkit is `http://www.mucm.ac.uk/toolkit`.

Fig. 1. The top page for the MUCM toolkit

The toolkit is arranged in thirteen threads plus three large case studies. The threads include two 'core' threads on emulating with Gaussian processes and Bayes linear emulation. Further threads explore how to extend these two core procedures when there are multiple outputs, dynamic emulators, two level emulators (one more complex than the other), combining multiple, independent emulators and using and obtaining derivative information. The issues of model discrepancy, history matching, calibration and sensitivity analysis are also covered as well as the important problems of experimental design and screening. In the near future we expect at least one additional thread on emulators for stochastic simulators. Within what is a relatively short paper it is impossible to cover the breadth of material in the toolkit so we will concentrate on a few highlights.

7 Building an Emulator

In the MUCM Toolkit we have two threads that cover what is described as the core problem. The core problem is in many ways the simplest application for an emulator. We have a single output of interest, we do have multivariate inputs but no information on the derivative of the output. There are no real world data to compare the simulator to and we are not concerned with making inferences about reality; we are only interested in the simulator. This is not as uncommon as might be thought. Simulators, particularly relatively simple ones, are often used to study an idealised version of the system rather than to make inferences about the real world.

The steps in building an Gaussian process emulator are:

1. Set up the initial GP model. Decide on the form of the regression terms and the correlation function.
2. Decide what priors we are going to use for the GP parameters.
3. Design and run the initial experiment. This is sometimes called the training experiment. Design is an important issue. Computer time is usually limited and each simulator run is expensive so every run must be made to count. Design has its own thread on the MUCM toolkit and is discussed further below.
4. Build the emulator.
5. Validate the emulator. This is a crucial step. Our first attempts at building good emulators are often failures and we need to establish, and possibly convince others, that we have built an emulator which we can have confidence in. There are two approaches. The first is to use a leave one out cross validation. Each point in the training set is left out in turn and an emulator is built using the remaining points. This reduced emulator is used to predict the left out point. The mean square error of the differences between these predictions and left out runs gives a global measure of the quality of the emulator, while a plot of the individual differences may reveal areas of input space where the emulator is performing badly. The advantage of leave one out is that we do not need to perform any more runs of the simulator. The disadvantage is that we are not testing the full emulator but emulators built using reduced datasets. Our carefully crafted space filling design is continually compromised by having one point at a time omitted.
   The alternative is to run a second experiment with an independent set of simulator runs. These are then used to test the emulator. This can be done by simply comparing the difference between the predicted values and the truth (scaled by the predicted uncertainty) similar to the leave out method or there are more rigorous methods based on regression diagnostics that take into account the correlation between points [1]. Once we are satisfied that we have an emulator that validates we can combine the training and validation runs and create a single emulator.

The procedure for a Bayes linear emulator is similar but the specification of the priors is in terms of moments rather than full prior distributions.

8 Design

One of the first steps in any experiment is the design. This is as true of computer experiments as it is of field trials. Our methods have been developed with expensive simulators in mind, so our designs need to minimise the number of runs required. However we also we wish to cover the complete input space. This seems like an impossible task. If we were to use a traditional factorial design with only 2 levels for each input the number of runs required would be 2^p where

p is the number of inputs. Two levels of each input gives us very poor coverage and 2^p becomes unaffordable for even a small number of inputs.

We therefore have to look at other designs. There are two main families of design: those based on the Latin Hypercube and those based on Quasi-Monte Carlo sequences.

The traditional design is the Latin Hypercube [13]. In a Latin Hypercube design we first decide how many evaluations of the simulator we can afford in total, let this be n. We then divide the range for each input variable into n equal sections. The simulator is evaluated once, and only once, in each of these settings. This means that we have good marginal coverage of each of the variables. The Latin Hypercube design is now produced by permuting the numbers $1, ..., n$ for each variable separately. Note the randomisation is to produce a design rather than to randomise for external factors as it is used in field experiments. All Latin Hypercubes have good marginal properties but are not necessarily the space filling. There is no algorithm for an optimal space-filling Latin Hypercube. In general an additional space-filling criteria, maximising the minimum distance between points [9] or minimising the sum of inverse distances for example, is imposed on the design. Alternatively we can use Latin Hypercubes based on orthogonal arrays [17].

An alternative to the Latin Hypercube is to use a low discrepancy sequence to define the design points. Such sequences were originally devised to efficiently compute multidimensional integrals [15]. These sequences are space filling but for small numbers of simulator runs there can be problems with certain projections having 'holes' in them. Examples include the Halton sequences [7] and Niederreiter nets. For the design of computer experiments the most commonly used low discrepancy sequence is the Sobol sequence [22,**?**].

Current work in the design of computer experiments explored in the MUCM toolkit [14] includes sequential design where we use an initial space filling design to learn about the system and then use this information to guide us on where future runs should be carried out.

9 An Example

As a simple example consider a energy balance model of the Earth's climate. The Earth is reduced to a line of grid boxes all ocean with a single box below to represent the deep ocean. If it becomes cold enough in a box sea ice forms this has a different albedo to water so a different amount of radiation is reflected from sea ice. As the surface water becomes colder it becomes denser and can sink at a variable location in the North. Deep water upwells back to the surface at a fixed location in the South. Heat is transferred between the surface grid boxes by both advection and diffusion. The system is driven by incoming short wave radiation from the Sun. As in the real world this is greater at the equator than the poles. The total amount of incoming radiation is set by the Solar constant. As this is a simple example we only vary one input, the Solar constant, and we

only use a single output, the mean surface temperature. Note as is often the case the output we emulate is a function of the simulator not a state variable.

A space-filling Latin Hypercube in 1-d is simply a set of evenly spaced points. Using 6 points we get the emulator shown in panel (a) in Fig. 2. The dashed line shows the expected value of the emulated and the shaded region gives 95% uncertainty bands. Because we do not have a nugget term in the emulator the uncertainty collapses to zero where we have simulator runs. This makes perfect sense as here we *know* the value of the simulator output. Note that because this simulator is very cheap to run we can run it across all of input space and plot the true simulator value across the range and this is shown by the solid line in the figure.

We now do a further three runs of the simulator. Fig. 2 shows the standardised residuals for each of these new points. This is the distance between the expected value from the emulator and the true value divided by emulator standard deviation. If the emulator were correct we expect these numbers to have Normal distributions (for simplicity we are ignoring correlation here; for a full solution see [1]). One of the points is outside the limits shown in the figure, indicating that we do not have a good emulator. We therefore combine our existing points, both the six original training points and the three new validation points, into a single dataset and build a new emulator. This is shown in Fig. 2 panel (c). We now need to validate this new emulator. A further three runs are carried out and this time all three points are within the bounds for the standard used residuals. The emulator in panel (c) therefore validates. However we now have three additional runs which we can use to build an even better emulator. This, our final emulator, is shown in panel (d) of Fig. 2.

10 Sensitivity Analysis

One important application of emulators is in sensitivity analysis. Although computer simulators may have large numbers of inputs, often the outputs are dependent on only a few. Formally we can look at sensitivity analysis as answering the question: if the inputs x are changed by a small amount δx what is the effect on the simulator output $f(x)$. One way of looking at this problem is to vary each input in turn, run the simulator and see what effect the change has on the output. This is known as one at a time sensitivity analysis. If we could guarantee that all the inputs were completely independent of each other it might not be a bad idea, but we are dealing with large, complex, non-linear simulators and it would be foolhardy to make such an assumption. Any approach to measuring the sensitivity of a simulator must acknowledge that there will interactions between the inputs and should at least estimate not only the effect of single inputs (main effects) but also at least the first order interactions.

Traditionally this has been tackled by looking at the derivatives of $f(.)$ w.r.to x. This gives the *local* sensitivity since $\partial f(x)/\partial x$ depends on x and may change radically as we move around input space. An alternative is to use variance based

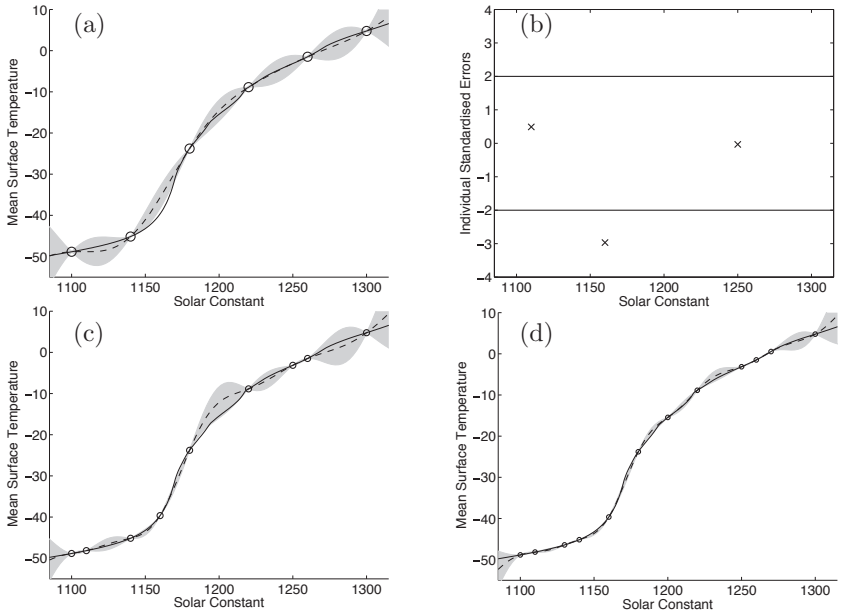

Fig. 2. Panel (a) shows the expected value of the emulator (dashed) and the true simulator based on six runs (o). The shaded region is a 95% uncertainty band around the expectation. Panel (b) shows the values for 3 additional validation runs with the emulator uncertainty. Note that one of the points is outside the uncertainty limits. Panel (c) shows the emulator built using all nine points (the original training set plus the three validation points). This emulator validates with an additional three points (not shown). The final emulator based on 12 points (6+3+3) is shown in panel (d)

sensitivity analyses, see for example [19]. [16] extend these methods so that they can be used with emulators.

11 History Matching and Calibration

After sensitivity analysis, probably the most important application of emulators is in comparing simulators with data and calibrating the simulators. We make a distinction between simulator comparison, often referred to as history matching, where we simply rule out areas of input space that are incompatible with the data and calibration where we estimate the 'best' input values from the data.

Both rely on the concept of 'model discrepancy'. The simulator is attempting, in most cases, to simulate some property of the real world. But if we measure that property do we expect the simulator to give an exact fit to the data? The answer is almost certainly not. In building simulators we make assumptions, we parameterise processes and there are real world processes we do not include, or even no about. All these omissions and approximations mean that we do not expect our simulator to explain the data perfectly. We describe this difference between the real world and the simulator as the model discrepancy. Model discrepancy will change as the input values change and often will have to be elicited from experts rather than being estimated from the data. Statistically it is possible to think of model discrepancy in two ways: as a bias between the simulator/emulator and the data, or as an increase in variance around the simulator/emulator.

First consider the problem of history matching. Some data are collected and we wish to know which values if any of the inputs could have produced them. Translating this into the negative language of statisticians, who only ever reject hypotheses and never accept them, we are looking for input values that are *implausible* given the data. This is done via an implausibility measure

$$I_{mp} = \sqrt{\sum_{all\ x} \frac{(x - x_{emul})^2}{\sigma^2_{data} + \sigma^2_{emul} + \sigma^2_{discrep}}}$$

where x is a data point; x_{emul} is the emulator value corresponding to that data point; σ^2_{data} is the variance of the data; σ^2_{emul} is the variance of the emulator; and $\sigma^2_{discrep}$ is the model discrepancy expressed as a variance. Values of implausibility greater than 3 (or 5 depending on how conservative we want to be) are ruled to be implausible. Additional simulator runs are then done within the input space not yet deemed implausible (sometimes to as NROY, Not Ruled Out Yet, space). By refining the emulator and adding additional runs in waves within NROY space the volume that hasn't yet been deemed implausible falls rapidly. Table 1, taken from [23], shows the volume of NROY space at each wave of the experiment as well as the number of runs in each wave.

If we want to go beyond history matching to model calibration we need to reformulate the problem. We follow [10]. First we split the inputs into two: control inputs and calibration inputs. The calibration inputs are those inputs we are trying to calibrate. Control inputs on the other hand are inputs that

Wave	Runs	% Space
1	993	14.9%
2	1414	5.9 %
3	1620	1.6 %
4	2011	0.26 %

Table 1. Table showing the reduction in Not Ruled Out Yet space by application of an implausibility measure to the Galform simulator of galaxy formation. For details see [23]

control the simulation but which can't or won't be calibrated. For example in an environmental simulator we may have inputs that give the spatial position of the outputs, these would be control inputs. We now say that the simulator $f(x_{con}, x_{cal})$ is the sum of reality $y(x_{con})$ and discrepancy $d(x_{con})$. Note that in this formulation both reality and the discrepancy are functions only of the control inputs not the calibration inputs. We have some measurements of reality, $z(x_{con})$ which are given by

$$z(x_{con}) = y(x_{con}) + \epsilon \tag{10}$$

We can now build an equation that links the data (z) and the simulator.

$$z(x_{con}) = f(x_{con}, x_{cal}) + d(x_{con}) + \epsilon \tag{11}$$

We now use Gaussian processes not only to build an emulator for $f(x_{con}, x_{cal})$ but also for $d(x_{con})$. The posterior of x_{cal} is the calibrated distribution inputs. For full details see the toolkit [14] or [10].

12 Beyond the Core Problem

The core problem as discussed above gives us a basis on which to expand to more complex problems. Examples dealt with in the toolkit include:

1. Multiple outputs where we are interested in more than a single output from the simulator.
2. Dynamic emulators, where we are interested in an output that is itself changing over time.
3. Multiple level emulators, where we have a number of simulators ranging from a coarse, fast simulator through a hierarchy to a slow, but more accurate, simulator.
4. Derivatives. Often we have derivative information available from the simulator in the form of an adjoint model. We can use this information to improve our emulator. However there is a trade off between the extra expense in calculating the derivatives and producing more runs of the simulator without derivatives. In other cases we may not have derivative information from our simulator but we are interested in the form of the derivatives. For instance we

may be interested in local sensitivity. Even with the use of automatic differentiating compilers [8] the production of an adjoint for any reasonably sized simulator is a major undertaking. An alternative is to produce an emulator for the derivative. This is relatively easy as the derivative of a Gaussian process is another Gaussian process. Validating the derivative emulator is more difficult if we do not have an adjoint to compare with.

13 Conclusions

I hope that I have managed to show in this short introduction that both Gaussian process and Bayes Linear emulators are powerful tools in the quantification of uncertainty. The MUCM Toolkit [14] is a good reference for these methods in sufficient detail to allow code to be developed. The toolkit is not static and is updated approximately quarterly. In the near future we expect it to be extended to cover stochastic simulators

Acknowledgements This paper wouldn't have been possible without the help and support of the MUCM team, in particular Ioannis Andrianakis (NOC), Ian Vernon (U. Durham) and Hugo Maruri-Aguilar (QMUL). John Shepherd (U. Southampton) kindly provided the simple climate model used in section 9.

References

1. Bastos, L., O'Hagan, A.: Diagnostics for Gaussian Process Emulators. Technometrics 51(4), 425–438 (2009)
2. Bates, R., Kenett, R., Steinberg, D.: Achieving robust design from computer simulations. Quality Technology and Quantitative Mangement 3(2), 161–177 (2006)
3. Bratley, P., Fox, B.: ALGORITHM 659: implementing Sobol's quasirandom sequence generator. ACM Transactions on Mathematical Software (TOMS) 14(1), 88–100 (1988)
4. Collins, W., Bitz, C., Blackmon, M., Bonan, G., Bretherton, C., Carton, J., Chang, P., Doney, S., Hack, J., Henderson, T., Kiehl, J., Large, W., McKenna, D., Santer, B., Smith, R.: The Community Climate System Model: CCSM3. J Climate 19, 2122–2143 (2006)
5. Goldstein, M.: Bayesian analysis for complex physical systems modelled by computer simulations: current status and future challenges. In: IFIP (2011)
6. Goldstein, M., Wooff, D.: Bayes Linear Statistics, Theory and Methods. Wiley (2007)
7. Halton, J.: On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. Numerische Mathematik 2, 84–90 (1960)
8. Hascoët, L., Greborio, R.M., Pascual, V.: Computing adjoints by automatic differentiation with Tapenade. In: Sportisse, B., LeDimet, F.X. (eds.) Ecole INRIA-CEA-EDF Problemes non-lineaires appliques. Springer (2005)
9. Johnson, M., Moore, L., Ylvisaker, D.: Minimax and maximin distance designs. Journal of Statistical Planning and Inference 26(2), 131–148 (1990)

10. Kennedy, M., O'Hagan, A.: Bayesian calibration of computer models. Journal of the Royal Statistical Society: B 63, 425–464 (2001)
11. Lim, Y., Sacks, J., Studden, W., Welch, W.: Design and analysis of computer experiments when the output is highly correlated over the input space. Canadian Journal of Statistics-Revue Canadienne De Statistique 30(1), 109–126 (2002)
12. Maltrud, M.E., McClean, J.: An eddy resolving global $1/10^\circ$ ocean simulation. Ocean Modelling 8, 31–54 (2005)
13. McKay, M., Beckman, R., Conover, W.: A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 21(2), 239–245 (1979)
14. MUCM: MUCM Toolkit, `http://www.mucm.ac.uk/toolkit`
15. Niederreiter, H.: Random number generation and quasi-Monte Carlo methods. In: CBMS-NSF Regional Conference Series in Applied Mathematics. vol. 63. SIAM, Philadelphia (1992)
16. Oakley, J., O'Hagan, A.: Probabilistic sensitivity analysis of complex models: a Bayesian approach. Journal Of The Royal Statistical Society Series B-Statistical Methodology 66(3), 751–769 (2004)
17. Owen, A.: Orthogonal arrays for computer experiments, integration and visualization. Stat Sinica 2(2), 439–452 (1992)
18. Rasmussen, C., Williams, C.: Gaussian processes for machine learning. MIT Press (2006)
19. Saltelli, A., Chan, K., Scott, E.: Sensitivity Analysis: Gauging the Worth of Scientific Models. Wiley (2000)
20. Santner, T., Williams, B., Notz, W.: The Design and Analysis of Computer Experiments. Springer-Verlag New York (2003)
21. Shakun, J., Carlson, A.: A global perspective on Last Glacial Maximum to Holocene climate change. Quaternary Science Reviews 29, 1801–1836 (2010)
22. Sobol, L.: On the distribution of points in a cube and the approximate evaluation evaluation of integrals. USSR Comput. Math. and Math. Phys. 7, 86–112 (1967)
23. Vernon, I., Goldstein, M., Bower, R.G.: Galaxy Formation: a Bayesian Uncertainty Analysis. Bayesian Analysis 5(4), 619–669 (2010)
24. Wilkinson, D.: Stochastic Modelling for Systems Biology. Chapman and Hall (2006)

DISCUSSION

Speaker: Peter Challenor

Maurice Cox : This seems a very useful toolkit. I understand you are on your second grant (MUCM2). I would be interested in hearing plans for your maintenance of the toolkit beyond MUCM2.

Peter Challenor : That is a very pertinent question that is at the top of our agenda. The MUCM2 project runs until October 2012. Maintaining the toolkit frozen from that point would not be expensive and even further developments could be done for little cost and we are looking for funding to put the toolkit on a long term sustainable basis. We would like to encourage non-MUCM participants to contribute to the toolkit and make it a true community resource.

John Reid : You have said nothing about parallel programming. It strikes me that you scope for "embarrassingly parallel" (ideal) execution.

Peter Challenor : Because we rely on ensembles of simulator runs it is quite correct that have an embarrassingly parallel problem. However some of the simulators we are working with are so large that we do not have the computer resources to make use of this. Parallel computing brings up some interesting questions in the design of experiments. I mentioned sequential designs above. Traditionally sequential designs would involve additional single simulator runs, but if we have access to parallel computing it is much more efficient to use 'batch sequential' designs where we run n additional simulations at a time.

John Rice : Have you used your simulator to explore the possible sources of the unusual climate changes observed in historical data? For example, the "little ice age" that occurred a few centuries ago or the major climate change that occurred about 12000 years ago and which appears to have occurred several times earlier with a periodicity of (as I recall) about 125000 years? Historically speaking, we appear to be near the end of a long warm period if this cycle is persistent.

Peter Challenor : The simple Earth radiation balance simulator we use in the example is not suitable for reproducing the history of the Earth's climate. It is much too simple and is lacking too many process. For a good description of the Earth's climate over the last 20000 years see [21]

Will Welch : The MUCM Toolkit provides recipes for developing code. To test that code it would be useful to have test cases complete with data and results. What plans do you have to provide such test cases?

Peter Challenor : This is a very good point and it is our intention to supply such test cases with most, if not all, the pages. We currently have worked examples for some pages; the example in section 9 is taken from the coreGP pages. We currently have nine test cases plus the three large case studies. Over the next year I hope we will see many more being produced.