Bayesian Analysis of Computer Code Outputs: A Tutorial

A. O'Hagan
University of Sheffield, UK

August 11, 2004

Abstract

The Bayesian approach to quantifying, analysing and reducing uncertainty in the application of complex process models is attracting increasing attention amongst users of such models. The range and power of the Bayesian methods is growing and there is already a sizeable literature on these methods. However, most of it is in specialist statistical journals. The purpose of this tutorial is to introduce the more general reader to the Bayesian approach.

Keywords: Bayesian statistics; calibration; dimensionality reduction; emulator; Gaussian process; roughness; screening; sensitivity analysis; smoothness; uncertainty analysis; validation.

1 Introduction

Over the last 5 to 10 years, a range of tools have been developed using Bayesian statistics to tackle many important problems faced by developers and users of complex process models. These methods can be seen as developments of work on Design and Analysis of Computer Experiments in the 1980s, which introduced the fundamental idea of building a statistical *emulator* of a simulation model. The Bayesian Analysis of Computer Code Outputs (BACCO) extends this approach both in terms of emulation of more complex model behaviour and in terms of methodology to employ emulators to address a wide range of practical questions in the development and use of process models.

1.1 Simulators

Complex models are built in almost all fields of science and technology (and increasingly in social sciences and commerce) to simulate the behaviour of real-world systems. These models may be empirical or represent detailed scientific understanding of the real-world process. The latter may be called process models or mechanistic models, but the distinction is somewhat artificial because nearly all process models contain equations whose forms or coefficients are empirically determined. These models are usually implemented in computer programs, which can run to many thousands of lines of code and can take from a fraction of a second to several hours to run. We will refer to both the mathematical model and the computer program that implements it as a *simulator*.

We will be concerned in this tutorial with deterministic simulators, i.e. simulators that produce the same outputs every time if they are given the same inputs. (The distinction is again rather artificial, because even a stochastic simulator could be made deterministic by making the random number seed an input.) Such a simulator can be regarded as a mathematical function $f(.)$, that takes a vector $\mathbf{x}$ of inputs and produces an output vector $\mathbf{y} = f(\mathbf{x})$.

The outputs $\mathbf{y}$ of a simulator are a prediction of the real-world phenomena that are simulated by the model, but as such will inevitably be imperfect. There will be uncertainty about how close the true real-world quantities will be to the outputs $\mathbf{y}$. This uncertainty arises from many sources, particularly uncertainty in the correct values to give the inputs $\mathbf{x}$ and uncertainty about the correctness of the model $f(.)$ in representing the real-world system; see Kennedy and O'Hagan (2001) for a more complete taxonomy of the uncertainties involved in using simulators. The primary objective of the methods outlined in this tutorial is to quantify, analyse and reduce uncertainty in simulator outputs.

1.2 Bayesian methods

The Bayesian approach to statistics is experiencing a surge in interest and demand in almost every area of application of statistics. There are many reasons for this, but this tutorial will not address the often complex distinctions between the Bayesian and the more well-known frequentist approaches. Nevertheless, the word 'Bayesian' appears in the title and it is appropriate to say a few words about what is Bayesian in the methods described here, and why they are preferred to frequentist methods for these problems.

Frequentist statistics earns its name through its emphasis on interpreting the probability of an event as the long-run limiting frequency with which the event occurs if repeated an infinite number of times. To most people who are not specialists in probability or statistics, this is the only interpretation of probability that they have ever learnt, but it is a severely limiting notion. The frequentist definition of probability can only apply to events that are, at least in principle, repeatable an indefinite number of times. The uncertainty in repeatable events, which arises from their intrinsic randomness and unpredictability, is sometimes called *aleatory* uncertainty.

Most of the uncertain quantities that we wish to learn about through statistical analysis are, in contrast, not repeatable; they are one-off things. To the user of a model of groundwater flow, the permeability of some particular block of soil or rock is unknown but not an instance of something repeatable. It is a one-off, the permeability of that specific block. The model user's uncertainty is not due to intrinsic randomness, but to his or her lack of knowledge about that particular block. The uncertainty in non-repeatable events that is due simply to our lack of knowledge of them is called *epistemic* uncertainty.

Almost all the uncertainties in the analysis of simulator outputs are epistemic. In a particular application of the model, the correct values of the model inputs are one-off, specific to this application, and so subject to epistemic uncertainty. Furthermore, no model is perfect, so that even if we have the correct inputs $\mathbf{x}$ the outputs $\mathbf{y}$ will differ from the true values of the phenomena being simulated; uncertainty about this *model discrepancy* is also epistemic.

Bayesian statistics is based on a much broader definition of probability. Within the Bayesian framework we are free to quantify all kinds of uncertainty, whether aleatory or epistemic, through probabilities.[1] All of the techniques described in this tutorial are statistical, and all are grounded firmly in the Bayesian framework.

1.3 Objectives of this tutorial

Section 2 explains the idea of an emulator as a statistical representation of a simulator, while Section 3 describes how the Gaussian process (GP) emulator works and discusses the key steps in building a GP emulator. Emulation is rarely an end in itself; the purpose of building an emulator is almost always to facilitate other calculations that would not be practical to do using the simulator itself. In Section 4, we explain the use of emulation as a computationally efficient device to address various problems associated with the use of simulators, in the context of uncertainty analysis. Emulation-based techniques for sensitivity analysis and calibration are outlined in Section 5. The Bayesian methods are undergoing quite rapid development to address new challenges posed by complex simulation models, and some of these extensions and challenges are explained in Section 6. Section 7 offers a summary of the main points and discussion of related work.

The primary objective of the tutorial is to convey, with as little technical statistical detail as possible, the way that the Bayesian methods work and the key tasks and challenges in their use. The intended audience is developers and users of models in all fields of science and technology, rather than statisticians. The aim is not to teach how to actually employ these methods. To do so would have entailed an article many times as long as this tutorial. However, references are given throughout to journal articles, usually in the mainstream statistics literature, where the interested reader may find the necessary technical details.

We will deal almost exclusively with the case when the model produces a single output y, rather than a vector $\mathbf{y}$. This is for ease of exposition, and we will only refer briefly in Section 6 to extra issues that arise in the multi-output case.

[1] See O'Hagan and Oakley (2004) for further discussion of how the distinction between aleatory and epistemic uncertainty is fundamental to that between frequentist and Bayesian statistics.

2 Principles of emulation

2.1 What is an emulator?

A major concern of the SAMO meetings is to understand the sensitivity of model outputs to variation or uncertainty in their inputs. Sensitivity analysis (SA) and uncertainty analysis (UA) are powerful general tools for both the model developer and the model user. Unfortunately, the standard techniques of UA/SA that are described in Saltelli *et al* (2000) demand a very large number of model runs, and when a single model run takes several minutes these methods become impractical. Even for a model that takes just one second to run, a comprehensive variance-based sensitivity analysis may require millions of model runs, and just one million runs will take 11.5 days of continuous CPU time. A major research strand in SAMO is therefore the search for more efficient computational tools to perform SA. The main reason for interest in BACCO is that the Bayesian methods are enormously more efficient than other existing approaches. This efficiency is achieved through emulation.

An emulator is a statistical approximation of the simulator.

Remembering that the simulator is just a function $f(.)$ that maps inputs $\mathbf{x}$ into an output $y = f(\mathbf{x})$, we could imagine using an approximation $\hat{f}(.)$ instead of $f(.)$ for the UA/SA calculations. If the approximation is good enough, then the uncertainty and sensitivity measures produced by the analysis will be sufficiently close to those that would have been obtained using the original simulator $f(.)$. If the approximation is simpler than the original function, and more importantly, if $\hat{f}(\mathbf{x})$ can be computed much faster than $f(\mathbf{x})$, then this offers the solution to the infeasibility of applying SA to a complex simulator. It is clearly preferable to have an approximate sensitivity analysis than no sensitivity analysis at all.

However, an emulator is not just an approximation, but a *statistical* approximation. An approximation to $f(\mathbf{x})$ for any input configuration $\mathbf{x}$ is a single value $\hat{f}(\mathbf{x})$. An emulator provides an entire probability distribution for $f(\mathbf{x})$. We can regard the mean of that distribution as the approximation $\hat{f}(\mathbf{x})$, but the emulator also provides a distribution around that mean which describes how close it is likely to be to the true $f(\mathbf{x})$. In fact, an emulator is a probability distribution for the entire function $f(.)$.

Although such a thing may seem incredibly complex, statisticians and probabilists have a very well established theory of *stochastic processes* with which to describe uncertainty about functions.

2.2 Building an emulator

In practice an emulator is statistical in two senses. Not only is it a statistical approximation to the simulator, but it is built using statistical methods. Given a set of *training runs* of the model, in which outputs $y_1 = f(\mathbf{x}_1), y_2 = f(\mathbf{x}_2), \ldots, y_N = f(\mathbf{x}_N)$ are observed, we treat these as *data* with which to *estimate* $f(.)$.

Two natural criteria that the emulator should satisfy are as follows.

1. At a *design point* $\mathbf{x}_i$, the emulator should reflect the fact that we know the true value of the simulator output, so it should return $\hat{f}(\mathbf{x}_i) = y_i$ with no uncertainty.
2. At other points, the distribution for $f(\mathbf{x})$ should give a mean value $\hat{f}(\mathbf{x})$ that represents a plausible interpolation or extrapolation of the training data, and the probability distribution around this mean should be a realistic expression of uncertainty about how the simulator might interpolate/extrapolate.

Criterion 1 is easy to check. The principal way to check criterion 2 is to make extra runs of the model, and to confirm that the true simulator outputs do lie appropriately and consistently within the emulator's probability distributions for those outputs.

Some approaches to emulation that may be familiar to some readers include fitting regression models to the data, or using them to create a neural network. It is easy to see the sense in which a regression model might be an emulator. The statistical analysis provides estimates of the regression parameters, and plugging these estimates into the regression equation provides the approximation $\hat{f}(.)$. Furthermore, we could construct confidence intervals around the regression parameters to describe uncertainty in the fitted approximation. However, it is also very easy to see that regression models do not satisfy the above criteria. (For instance, a high order polynomial may satisfy the first criterion but would then fail the second.) It is less obvious that neural networks also fail, but we find in practice that neural network interpolations typically fail criterion 2 by not allowing for enough uncertainty about how the true simulator will behave.

The form of emulator used in the BACCO approach is the Gaussian process, which is an analytically very tractable form of stochastic process. A properly fitted Gaussian process emulator will satisfy the two fundamental criteria.

2.3 Code uncertainty

When we use an emulator to perform SA instead of the simulator, we need to acknowledge an extra source of uncertainty. The measures of sensitivity that we obtain will differ from the true measures that would have been obtained had we been able to carry out the SA on the simulator. We are uncertain about how far those true measures might be from the values we obtained using the emulator. A key point about the emulator being a statistical approximation is that we can quantify that uncertainty. Indeed, since the emulator gives a probability distribution for the true function $f(.)$, this induces a probability distribution for any SA or other measures that might be derived from $f(.)$.

This additional uncertainty is called *code uncertainty*, because we are uncertain about what the true simulation code would produce. It is not really a new concept in SA, since it is implicit in the Monte Carlo methods that underlie the UA/SA techniques in Saltelli *et al* (2000). Any Monte Carlo estimate has a standard error, that can be reduced by increasing the sample size. The code uncertainty in emulation operates in the same way. Given a large enough training sample, we will effectively know $f(.)$ with negligible uncertainty.

Whichever method we use, we wish in practice to take a sufficiently large sample of model runs to achieve acceptable accuracy in SA and other analyses.

The key to the greater efficiency of BACCO methods is that for a given training sample size N the code uncertainty in SA estimates using an emulator is much smaller than that obtained from Monte Carlo methods using the same sample size. Where methods based on Monte Carlo may require hundreds of thousands of model runs to achieve acceptable accuracy, BACCO methods can typically achieve the same accuracy with a few hundred runs.

3 Gaussian process emulation

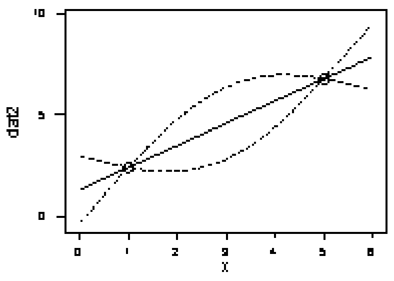
This section describes the principles of Gaussian process (GP) emulation in non-technical terms. The statistical theory can be found, with full mathematical details, in Kennedy and O'Hagan (2001) and Oakley and O'Hagan (2002, 2004).

3.1 The GP and how it works — one input

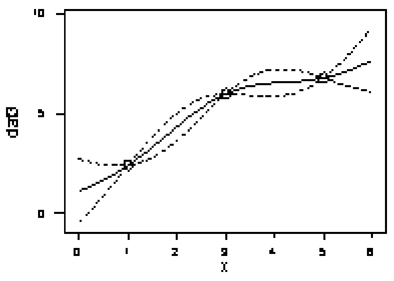
Figure 1 illustrates how the GP emulator approximates a simple function. For the purposes of this example, in addition to assuming that the model produces a single output we further suppose that it has a single input. The model is then a simple function $y = f(x)$. The function in this case is $f(x) = x + 3\sin\frac{x}{2}$. This is of course not in any sense a complex function, but we imagine it as representing the kind of situation where $f(.)$ is indeed a complex function and may take several seconds or even minutes of computer time to evaluate for a single input x.

Figure 1(a), (b) and (c) shows a sequence in which the number of training data points is increased from 2 to 3 to 5. The data points are shown as circles. In each case the solid line is the mean of the emulator distribution, which plays the role of the approximation $\hat{f}(.)$. Notice how it always passes through the data points. With only 2 points, it is simply a straight line through those points. However, as we increase the number of points it adapts to the shape of the true function.

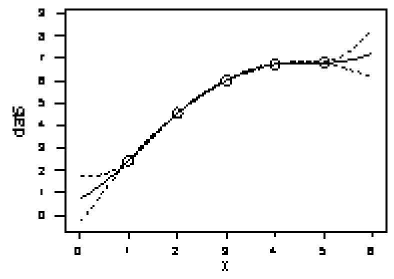
The dashed lines are bounds set at plus and minus two standard deviations, and so are approximately 95% probability bounds. Notice that the uncertainty is pinched in to zero at the data points. With only 2 points, there is still considerable uncertainty about where the true function $f(.)$ lies, apart from at the two data points. As the number of points increases, not only does the mean function adapt to the shape of the true $f(.)$, but uncertainty decreases. With 5 points there is negligible uncertainty about how the true function interpolates the training data points, but notice that uncertainty rapidly increases if we try to extrapolate outside the data.



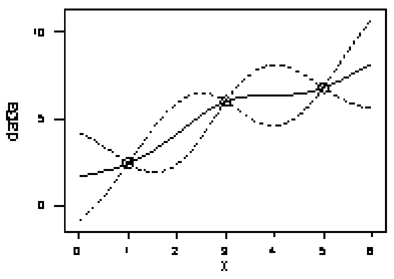
(a)



(b)



(c)



(d)

Figure 1. Examples of GP emulation.

Formally, a Gaussian process is an extension of the familiar normal, or Gaussian, distribution. The multivariate normal distribution is a distribution for several variables, each of which marginally has a normal distribution. The GP is a distribution for a function, where each point $f(\mathbf{x})$ has a normal distribution. The normal distribution is the most widely used distribution in Statistics, partly because it is mathematically very easy to work with. Those same nice mathematical properties carry over to the GP, and contribute further to the efficiency of the BACCO approach; see Section 4.

3.2 Smoothness

A fundamental assumption in the kind of GPs that have been used in BACCO is that the model $f(.)$ is a smooth, continuous function of its inputs. It is true that many complex computer codes do not meet this condition, either because the underlying real world phenomenon experiences rapid phase changes, or because switches between alternative equations have been coded into the model. The latter is common in environmental models and others built on empirical relationships. We return to the assumption of continuity in Section 6.

This basic assumption of a (homogeneously) smooth, continuous function gives the GP major computational advantages over Monte Carlo methods. The implication of smoothness can be seen in Figure 1. Given that we know the function value at $x = 1$, smoothness implies that $f(x)$ must be close to that same value for any x sufficiently close to 1. This is why the uncertainty about $f(x)$ is less when x is close to a design point than when it is far from any design point. It is also linked with the fact that the uncertainty decreases as the number of design points increases, simply because the maximum distance from any point decreases.

Each point in the training data provides substantial information about the function $f(.)$ for inputs close to that design point. It is this extra information, that is not used in Monte Carlo methods, which accounts for the greater efficiency of BACCO methods.

The GP incorporates a parameter that specifies the degree of smoothness, in terms of how far a point needs to go from a design point before the uncertainty becomes appreciable. In Figure 1 (a), (b) and (c) this parameter has the same value, whereas in Figure 1 (d) it has a different value implying much less smoothness. Comparing with Figure 1 (b), in which the same 3 design points are used, we see that the uncertainty bounds are now much wider. The interpolation is also a little less smooth, although this is scarcely noticeable in practice.

The actual degree of smoothness concerns how rapidly the function "wiggles". A rough function responds strongly to quite small changes in inputs, and we need many more data points to emulate accurately a rough function over a given range. Again this is clear in the comparison between Figure 1 (b) and (d). So the efficiency of BACCO methods increases with the degree of smoothness.

In practice, BACCO methods estimate the smoothness parameter from the training data. This is obviously a key GP parameter to estimate, and we need robust estimation methods. The chosen value can be validated by predicting new data points. That is, having fitted the GP, and in particular having estimated the smoothing parameter, we run the model at some new input points $x'_1, x'_2, \ldots$, and compare the resulting outputs $y'_1, y'_2, \ldots$ with the probability distributions predicted by the emulator for those values. If the smoothness parameter in the GP is too high, the emulator will make predictions with understated uncertainty, and the new output values will be further from the emulator approximations than the emulator expects. Conversely, a too low smoothness parameter makes predictions with overstated uncertainty.

3.3 Higher dimensions

The same general behaviour applies with more than one input, although it is now harder to show it visually as in Figure 1. With two or more inputs, we are fitting a surface through the data. At the actual data points, the fitted emulator passes exactly through the data values, and there is zero uncertainty. As we move the input vector $\mathbf{x}$ away these points, the uncertainty increases. Adding more data points causes the fitted function to adapt itself to the shape of the true function, and uncertainty decreases. It is clear how, with enough training data, the uncertainty can be reduced to negligible levels over the range of input values covered by the data.

The GP now includes smoothness parameters to describe how rapidly the output responds to changes in each input. It requires one parameter for each input dimension, at least. Again, accurate and robust estimation of the values of these parameters is crucial to constructing an effective emulator.

An important question for the user of almost any computational tool is how much the computational burden increases with dimensionality. In order to emulate a model with many inputs, how many training data points are required? In many dimensions there is much more 'space' between data points, suggesting that the number of training runs required would escalate rapidly with the number of inputs. However, an important compensating factor is that in practice, models never respond strongly to all of their inputs. Pragmatically, we get a high level of smoothness in all but a few dimensions.

By estimating the smoothness parameters, the GP automatically identifies the inputs to which the output is insensitive. Effectively, it projects points down through those smooth dimensions into the lower dimensional space of inputs that matter. It is certainly true that 200 points in 25 dimensions covers the space very sparsely, but in 5 dimensions 200 points can fill the space quite densely enough to get good emulation. If only 5 of the 25 inputs influence the output appreciably, then the GP automatically reduces the dimensionality.

Another dimensionality question is the number of model outputs. A model with many outputs effectively has many functions of the inputs, and in principle we can emulate each output separately. However, it can be important to recognise correlations between the outputs, and this is an area of ongoing research.

3.4 GP with regression

We can improve the emulator by using it in combination with regression. Effectively, we fit a regression model and then use a GP to smoothly interpolate the residuals. Using a good regression model means that much of the variation of the model output in response to its inputs is explained by the regression function. The GP is then called on to represent only the variation that is not explained by the regression, and this typically allows the fitted GP to have increased smoothness, so that fewer training runs are required.

Practical experience suggests that 'good' means 'parsimonious'; there should not be more complexity in the regression model than is needed. In statistical language, the regression model would not overfit the data. The analyses shown in Figure 1 were all done using a GP in combination with a simple linear regression. It may be that a quadratic regression would have improved the fit, and so might have led to narrower error bounds on this particular function. In practice, however, we find that curvature in responses to inputs rarely fits a simple quadratic form well enough to justify the extra complexity of including such terms in the regression.

Combining the GP with a good regression fit is even more useful for models with many inputs than in one dimension.

It is interesting to note that regression coefficients (usually from simple linear regression fits) are widely interpreted as measures of the sensitivity of the output to the corresponding parameters. It has also been suggested that in GP emulation the smoothness parameters indicate sensitivity (in the sense that the inverse of a smoothness parameter can be seen as a measure of sensitivity). The truth is that neither smoothness parameters nor regression parameters alone are adequate representations of sensitivity. The way that the output responds to each input, as represented in the emulator, involves both the regression and GP parts.

3.5 Design

We need to choose input configurations $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ at which to run the model to get training data. These do not need to be random. The objective is to learn about the function $f(.)$, and well spaced points that cover the region of interest are much better than random points.

In principle, we can even choose an optimal set of points, subject to any suitable criterion, but this is itself computationally demanding for large N and/or a high dimensional input space. In practice, simple patterned allocations are very effective, and some randomisation is frequently used. For example, one simple method is to generate, say, 100 random Latin Hypercube samples and choose the one having largest minimum distance between points. There are no doubt many better algorithms, but it is not clear whether they would yield much improvement in the emulator's accuracy. See for instance the range of designs developed in Santner *et al* (2003).

4 Use of emulators — Uncertainty Analysis

The Bayesian approach is a two-stage approach. First, we build the emulator, then we use the emulator to compute any desired analyses. Only one set of runs of the simulator is used, to build the emulator. After the emulator is built, we do not need any more simulator runs, no matter how many analyses are required of the simulator's behaviour.

This contrasts with the conventional methods of sensitivity analysis described in Saltelli *et al* (2000). The Monte Carlo-based methods described there require a fresh set of simulator runs for each analysis. For instance, to compute measures of sensitivity for the various inputs or subsets of inputs will typically require fresh runs of the simulator for each input or set of inputs.

The simplest way to use the emulator is to treat the emulator's mean function $\hat{f}(.)$ as if it were the simulator $f(.)$, and just to apply the conventional methods to $\hat{f}(.)$. Although these methods still demand a fresh set of runs for each SA measure, these are ‘runs’ of the emulator, not the simulator. As such, the computations are typically enormously faster.

This approach ignores the statistical nature of the emulator, though, and fails to quantify the potential error due to code uncertainty, whereas an important feature of the BACCO philosophy is to account for all sources of uncertainty.

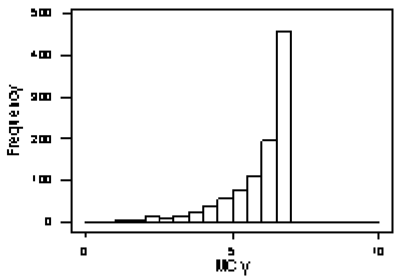
4.1 Uncertainty analysis

To show how the BACCO methods work, it is useful to consider first the simplest kind of analysis. Uncertainty analysis (UA) seeks to quantify the uncertainty in model outputs induced by uncertainty in inputs. Thus, if the input vector $\mathbf{x}$ is uncertain, we can consider it as a random vector. In statistics, we denote this by writing it as $\mathbf{X}$. The output is of course now also a random variable, and we write $Y = f(\mathbf{X})$ to show how the random output Y is related to the random input $\mathbf{X}$ through the model $f(.)$. Given a probability distribution for $\mathbf{X}$, that we denote by G, the task of UA is to characterise the probability distribution of Y.

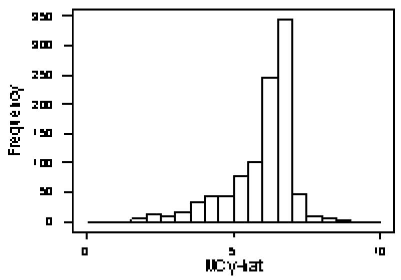
For instance, we may wish to find the expected value $M = E(Y)$ or the variance $V = \text{var}(Y)$. We will concentrate on the evaluation of M, and will illustrate the various approaches using the same very simple model $f(x) = x + 3\sin\frac{x}{2}$ that was the basis of Figure 1. We will suppose that the uncertainty in the single input X is described by a normal distribution with mean 3.5 and standard deviation 1.2. Formally, we write $X \sim N(3.5, 1.44)$, where the variance 1.44 is the square of the standard deviation.

4.2 Monte Carlo UA

The simple Monte Carlo (MC) way to perform UA is to sample values of $\mathbf{X}$ from the distribution G, and to run the simulator. Thus, if the sampled input configurations are $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, the outputs are $y_1 = f(\mathbf{x}_1), y_2 = f(\mathbf{x}_2), \ldots, y_n = f(\mathbf{x}_n)$. The sample mean $\bar{y}$ of the outputs is then an estimate of M and the sample variance is an estimate of V. The sample variance divided by n represents uncertainty (due to code uncertainty) about M.



(a)



(b)

Figure 2. Histograms of model outputs

Using the illustrative model, Figure 2(a) shows a histogram of MC outputs y_i obtained from 1000 runs of the simulator with inputs generated by 1000 draws from the $N(3.5, 1.44)$ distribution of X. The sample mean is 5.959, which is therefore the MC estimate of M. The sample variance is 1.295, and the standard error of the MC estimate of M is therefore $\sqrt{1.295/1000} = 0.036$. We therefore have a 95% confidence interval for M of $5.959 \pm 1.96 \times 0.036 = (5.888, 6.030)$. If the sample size were increased, the confidence interval and estimate would converge to the true value of M, which is 5.966.

4.3 MC applied to the emulator mean

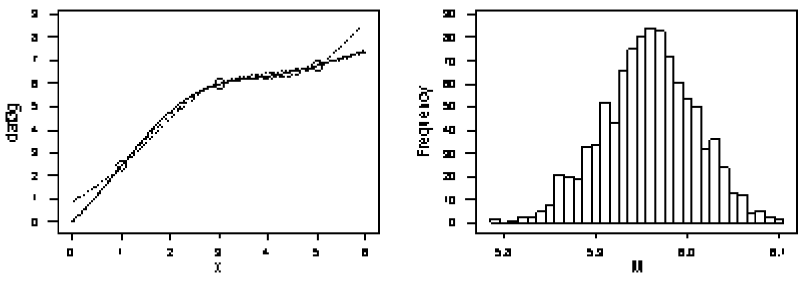
The simplest use of the emulator to do UA is to sample the inputs as before but to evaluate the emulator mean instead of the simulator, obtaining $\hat{y}_1 = \hat{f}(\mathbf{x}_1), \hat{y}_2 = \hat{f}(\mathbf{x}_2), \ldots, \hat{y}_n = \hat{f}(\mathbf{x}_n)$. Then for instance the sample mean $\widehat{\overline{y}}$ of these values estimates M. Figure 2(b) shows the histogram of $\hat{y}_i$ values obtained using the emulator mean from 3 simulator runs shown in Figure 1(b), and using the same 1000 draws from the $N(3.5, 1.44)$ distribution of X. The sample mean this time is 5.939, and a 95% interval is $(5.867, 6.011)$. The two histograms differ appreciably. In particular, because $\hat{f}(.)$ increases for $5 < x < 7$ (see Figure 1(b)), the histogram in Figure 2(b) has values of $\hat{y}_i$ greater than 7, whereas the true function decreases over this range. The estimate and confidence interval for M are very similar to those obtained by MC, but with increasing sample size they would not converge on the true value of M. Instead they would converge on what would be the true value of M if $\hat{f}(.)$ were the true function. This value is $\hat{M} = 5.958$. The difference is small, and it would require a MC sample of size 100,000 before the confidence interval failed to include the true value $M = 5.966$.

If the simulator is sufficiently complex, even 1000 runs would not be feasible and the simple MC approach would be impractical. More efficient stratified MC would still be able to estimate M to reasonable accuracy with perhaps a few tens of runs, but the accuracy we have found by just using the emulator mean is based on only 3 simulator runs. The emulator itself is so simple that 100,000 or more MC runs are entirely feasible. However, another important feature of the GP emulator is that we do not need to use MC to evaluate $\hat{M}$. The emulator mean $\hat{f}(.)$ is such a simple mathematical function that we can derive the mean $\hat{M}$ with respect to any normal input distribution analytically. So the value $\hat{M} = 5.958$ is obtainable almost instantaneously.

It is thus trivial to compute the value of $\hat{M}$ based on alternative training samples. A training sample of 13 simulator runs at $x = 0.5, 1, 1.5, \ldots, 6.5$ produces $\hat{M}$ within 0.0004 of the true M, accuracy that would require a sample of about 10 million simulator runs using simple MC.

4.4 Code uncertainty

The more correct SACCO analysis recognises uncertainty about how close the emulator is to the true simulator, i.e. code uncertainty. Figure 1(b) shows the mean function $\hat{f}(.)$, but it also shows uncertainty bounds around it. Figure 3(a) shows three possible functions that pass through the three training data points. These are random functions drawn from the emulator distribution. Of course we would expect each of these functions to give a slightly different value of the output mean M, and Figure 3(b) is a histogram of the values of M from 1000 such randomly sampled functions.



(a) (b)

Figure 3. Uncertainty analysis

In fact, we do not need to generate these random distributions, since again the nice properties of the GP mean that we can derive the distribution exactly. It is a normal distribution with mean equal to $\hat{M} = 5.958$ and standard deviation 0.051. This standard deviation makes it clear that the estimate is somewhat fortuitously close to the true M. We can now compare its accuracy rather more fairly with that of Monte Carlo. The MC method, using a sample of just 3 simulator runs, would have a standard error of $\sqrt{1.295/3} = 0.657$, whereas the BACCO estimate, also using just 3 simulator runs, has a standard deviation of 0.051. To achieve the same standard error using MC would require about 500 simulator runs.

4.5 Summary

This section has illustrated the basic features of the BACCO method using a very simple example, but the same ideas carry over to much more complex functions with many inputs. Considering only the simple output mean M, MC generates an estimate (the MC sample mean) and a standard error (the square root of the MC sample variance divided by the sample size). The BACCO approach is analogous in that it produces an estimate $\hat{M}$ (the value of M for the emulator mean function) and a standard deviation (quantifying how the value of M would vary over random functions drawn from the emulator). Both the standard error of the MC approach and the BACCO standard deviation represent code uncertainty, and can be reduced to any desired level by making enough simulator runs. In the case of MC, these are the sample runs, and it will typically take thousands (or even tens or hundreds of thousands) of simulator runs to achieve acceptable accuracy. In the case of BACCO, the simulator runs are used to build the emulator, and to achieve acceptable accuracy can require only a handful of runs in a model with just one or two inputs, or up to a few hundred with a complex function of many inputs.

The BACCO estimate and standard deviation can be computed using a 'brute force' approach of generating many random functions and applying Monte Carlo to each of these functions. Although this means potentially a huge number of 'runs' these are runs of the emulator, and even this brute force approach can be entirely feasible because the emulator 'runs' essentially instantaneously. However, the good mathematical properties of the emulator make it possible to avoid this and to evaluate the mean and standard deviation exactly without sampling. The theory for this and for similar computation for the UA variance V, are presented in Haylock and O'Hagan (1996) and O'Hagan and Haylock (1997). They present other examples, based on real simulators, that demonstrate the substantial computational advantages of the emulator-based BACCO method, compared with MC. Theory for the whole UA uncertainty distribution is given by Oakley and O'Hagan (2002) and for percentiles of that distribution by Oakley (2004).

The same properties apply to the other kinds of analysis considered in the next section. Brute force computation is practical in BACCO because we only run the simulator itself a relatively small number of times, but theoretical results typically allow the analysis to be done without sampling.

5 Use of emulators — other analyses

5.1 Sensitivity analysis

The BACCO approach has been applied very successfully to sensitivity analysis (SA), and there is now considerable experience in applying both UA and SA to complex models. The SA theory is set out in Oakley and O'Hagan (2004), giving results not only for variance-based SA measures but also for regression measures and plots of main effects and interactions. The fact that all these SA computations can be produced from a single set of training runs of the simulator is a major advantage of the BACCO approach.

Oakley and O'Hagan (2004) present a synthetic example with 15 inputs. Using only 250 simulator runs to build the emulator, they obtain a comprehensive set of SA analyses. To compute just the variance-based sensitivity indices and total sensitivity indices to the same accuracy using the FAST Monte Carlo technique for all 15 inputs would have required 15,360 simulator runs. The BACCO analysis produces not only all of these but also any desired sensitivity indices for combinations of two or more inputs with no extra simulator runs. It also gives plots such as Figure 4, which shows the main effect of varying each input, averaged over the uncertainty in all the other inputs.

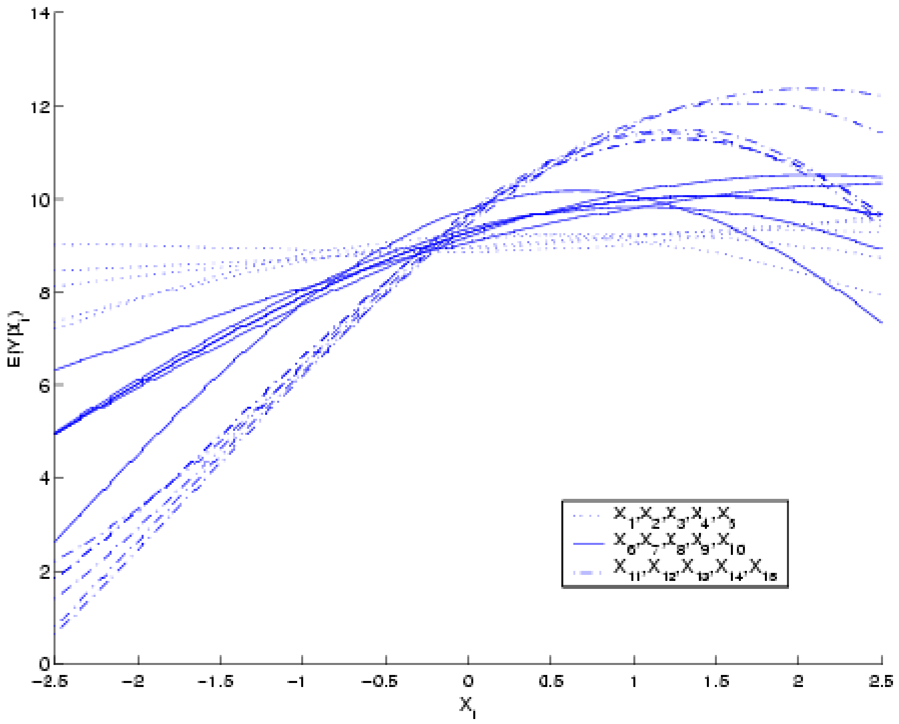


Figure 4. Main effects of 15 model inputs, from Oakley and O'Hagan (2004).

Nonlinearity of effects is clearly seen in Figure 4, in a way that is not revealed by variance-based measures. Oakley and O'Hagan (2004) show how nonlinearity can be quantified in terms of the difference between the variance-based sensitivity index and a regression-based index. Oakley (2002) shows how variance-based measures fit within a broader notion of value of information based sensitivity measures, and presents an application in which the BACCO approach achieves a 1000-fold increase in efficiency over MC methods.

It is also worth noting that it is trivial to perform 'what if' analyses in which we vary the uncertainty distribution G for the model inputs. These can be done using the emulator, and in contrast to FAST would require no new simulator runs.

Software for building the emulator and performing UA/SA is currently in beta testing and will be available for general use soon.

5.2 Bayesian calibration

A whole range of new questions arise for the model user when simulator output can be combined with observations of the real-world process that the simulator is designed to represent. In particular, *calibration* is the process of using observational data to learn about uncertain model inputs. Conventional calibration is usually done without explicit quantification of uncertainty in the inputs. Instead, the inputs to be calibrated are varied until the model output fits the observational data as closely as possible (according to some criterion), with other inputs held fixed. The best fitting values of the calibrated inputs are then treated as the true values.

Theory in Kennedy and O'Hagan (2001) presents a Bayesian approach to calibration in which uncertainty in inputs is explicit. Calibration then reduces, but does not eliminate, the uncertainty in the calibrated inputs. Emulation again plays an important part when the simulator is complex, but the introduction of observational data makes it necessary to model the relationship between the simulator and reality. This is done via a model discrepancy function (which becomes a second GP in the analysis), and another product of the Bayesian calibration process is to learn about model discrepancy. As a result, the estimate of the model discrepancy function allows the user to 'correct' the simulator and to reduce uncertainty about how well it represents reality. (More importantly in practice, and of greater long term value, this estimate of model discrepancy can be used by the model developer to inform further model improvements.) This approach has substantial potential, but experience to date is based on only a few substantial applications, such as those described in Kennedy and O'Hagan (2001) and Kennedy, O'Hagan and Higgins (2002).

Other uses of observational data include data assimilation and model validation, which are discussed in the next section.

6 Extensions and challenges

Work is ongoing or planned on a number of extensions to the basic BACCO methodology, to address a range of challenges posed by practical applications.

- *Dimension reduction* has been described as happening automatically through estimation of smoothness parameters. Thus, inputs having little influence on the output have high estimated smoothness, and the emulator concentrates on the dimensions with more influence. But this is only automatic if we have the right coordinate system. For instance, if the model is effectively a function of the sum of 15 inputs it will then appear to respond equally strongly to all of them, and no projection-based dimension reduction occurs. We need to rotate coordinates, and finding the right coordinate system is a challenge. Some progress has been made on identifying additivity and near additivity of model components, which may lead to ways of identifying this kind of structure.

- *Computation.* Building and using the emulator requires inversion of a variance matrix whose dimension is the number of simulator runs. This can lead to numerical problems. Gaussian processes with less smooth covariance structures than the default gaussian form used in BACCO methods are numerically more stable but may produce less accurate emulation.

- *Smoothness* estimation is fundamental to good emulation, and work is continuing on combining a variety of estimation methods to produce a robust and well validated choice. Allowing for uncertainty in smoothness parameters is also possible (see Bayarri *et al*, 2002) but increases the computational complexity.

- *Multiple outputs* can be dealt with by building separate emulators for each output, but this may lose information about how the outputs are correlated. This is proving to be important for some ongoing work on aggregating model outputs spatially.

- *Model inadequacy.* The model inadequacy function is important wherever we wish to link model outputs to observational data. There is usually very little information about the nature of this term, yet it is a very important component of uncertainty for model users. Ongoing work of Jonathan Rougier at the University of Durham is investigating realistic ways to model this function.

- *Dynamic models.* Most environmental models are dynamic, in the sense that they describe evolving behaviour of a system. The model itself is iterative. At each time step it takes the current state vector as part of its input vector, together with driving data and other parameters, and it outputs the updated state vector. Work is ongoing to emulate such models on a single time step, in order to handle the high dimensionality of driving data, to be able to make predictions over arbitrary time steps, and to address data assimilation.

- *Data assimilation* is a term used in dynamic models to describe learning about the current value of the state vector. It is a form of calibration, but usually done over a series of time steps. Simple methods to do this in real time are under investigation.

- *Discontinuities*. The basic GP emulator assumes that the simulator responds smoothly to its inputs. This is often not the case in practice. Computer models may have switches that cause the response to be discontinuous, or to have sharp changes of gradient. The emulator will smooth these out, leading to locally poor representation of the simulator. This may of course imply better representation of reality if reality behave more smoothly! Ideas are under investigation to emulate functions with discontinuities or sharp changes of direction, particularly where these are intentionally there to model unstable real behaviour.

- *Validation* is a term that is widely used in computer modelling, but often means little more than 'we compared the model output with real observations and look, they fit quite well'. A BACCO concept of validation is that we can predict reality, using model output with correction of model discrepancy where possible, in a statistically well validated way. That is, we can estimate real behaviour with uncertainty around the estimate (accounting for all sources of uncertainty), such that comparison with observational data suggests the expressed uncertainty is neither too large nor too small.

- *Software*. The final challenge is to create BACCO software that makes the methods readily applicable by non-specialist statisticians. The theory is complex, and demands a high level of mathematical and statistical sophistication to apply at present. As we gain more experience in the use of these methods it should become possible to transfer the technology to a wider variety of users through software. We have currently reached this point for applying UA/SA methods to reasonably well behaved models with modest numbers of inputs, and software will be released soon.

7 Discussion

The Bayesian method offers a powerful and efficient way of addressing a range of questions that are relevant to the developer or user of complex process models. By a two-step approach in which first an emulator is built as a statistical representation of the simulator and then this emulator is used to derive relevant analyses, the BACCO approach encompasses all kinds of techniques in one coherent framework.

The technology is already well established for uncertainty and sensitivity analyses, with software to be released soon. Theory is also in place for calibration and other techniques, but more practical experience is needed before these techniques are available for general use and coded in software.

For details of ongoing work and preprints of papers, see my website <http://www.shef.ac.uk/~st1ao>. Information about Marc Kennedy's GEM-SA software will appear on this website in due course.

This tutorial has concentrated on recent and ongoing work by myself and colleagues at the University of Sheffield. However, it would be seriously incomplete without placing it in the context of related past and present research by other groups. BACCO methods arise from the strong tradition of DACE (Design and Analysis of Computer Experiments), which dates back to the 1980s. An important review article on DACE research is Sacks *et al* (1989), and some more recent papers are Morris *et al* (1993) and Bates *et al* (1995). Although much DACE work is philosophically non-Bayesian, there is a clear recognition of the Bayesian interpretation and some work that is explicitly Bayesian (see in particular Currin *et al*, 1991). Much of the effort has been directed at emulation for the purpose primarily of predicting simulator output, for instance to optimise a process. Again, however, there are overlaps with BACCO work in applications to SA; see Welch *et al* (1992). Ongoing work in the Research Triangle, North Carolina, by Jerry Sacks, Jim Berger and others has benefited both from Jerry's long involvement with DACE, the strong Bayesian perspective of Jim Berger and the BACCO skills of Marc Kennedy; see Bayarri *et al* (2002). Further development in this area is continuing at Ohio State University by a group including Bill Notz and Thomas Santner. Santner *et al* (2003) is a fine text on Gaussian process emulation.

The BACCO methods are being actively explored at Los Alamos by David Higdon and others, as shown in the paper by Kathy Campbell in this volume.

Another strong group, based at the University of Durham, including Michael Goldstein, Peter Craig and Jonathan Rougier, has produced a substantial body of innovative BACCO research. The only reason that their work has not been covered more prominently in this tutorial is that their approach has emphasised Bayes linear methods rather than fully Bayesian analysis, and has sometimes taken a one-step view that involves emulation only implicitly. See Craig *et al* (1997, 2001), Goldstein and Rougier (2003, 2004).

References

Bayarri, M., Berger, J., Higdon, D., Kennedy, M., Kottas, A., Paulo, R., Sacks, J., Cafeo, J., Cavendish, J., Lin, C. and Tu, J. (2002). A framework for validation of computer models. In *Proceedings of the Workshop on Foundations for Verification and Validation in the 21st Century*, Pace, D. and Stevenson, S. (eds.). Society for Modeling and Simulation International.

Bates, R. A., Buck, R. J., Riccomagno, E. and Wynn, H. P. (1995) Experimental design and observation for large systems. *Journal of the Royal Statistical Society B* **58**, 77–94.

Craig, P. S., Goldstein, M., Rougier, J. C. and Seheult, A. H. (2001). Bayesian forecasting for complex systems using computer simulators. *Journal of the American Statistical Association* **96**, 717–729.

Craig, P. S., Goldstein, M., Seheult, A. H. and Smith, J. A. (1997). Pressure matching for hydrocarbon reservoirs: a case study in the use of Bayes linear strategies for large computer experiments. In *Case Studies in Bayesian Statistics: Volume III*, Gatsonis, C., Hodges, J. S., Kass, R. E., McCulloch, R., Rossi, P. and Singpurwalla, N. D. (eds.), 36–93. New York: Springer-Verlag.

Currin, C., Mitchell, T. J., Morris, M. and Ylvisaker, D. (1991). Bayesian prediction of deterministic functions with applications to the design and analysis of computer experiments. *Journal of the American Statistical Association* **86**, 953–963.

Goldstein, M. and Rougier, J. C. (2003). Calibrated Bayesian forecasting using large computer simulators. Available for download from http://www.maths.dur.ac.uk/stats/physpred/pape

Goldstein, M. and Rougier, J. C. (2004). Probabilistic formulations for transferring inferences from mathematical models to physical systems. Available for download from http://www.maths.dur.ac.uk/stats/physpred/papers/directSim.ps.

Haylock, R. G. and O'Hagan, A. (1996). On inference for outputs of computationally expensive algorithms with uncertainty on the inputs. In *Bayesian Statistics 5*, J. M. Bernardo *et al* (eds.). Oxford University Press, 629–637.

Kennedy, M. C. and O'Hagan, A. (2001). Bayesian calibration of computer models (with dis8cussion). *Journal of the Royal Statistical Society B* **63**, 425–464.

Kennedy, M. C., O'Hagan, A. and Higgins, N. (2002). Bayesian analysis of computer code outputs. In *Quantitative Methods for Current Environmental Issues*. C. W. Anderson, V. Barnett, P. C. Chatwin, and A. H. El-Shaarawi (eds.), 227–243. Springer-Verlag: London.

Morris, M. D., Mitchell, T. J. and Ylvisaker, D. (1993) Bayesian design and analysis of computer experiments: use of derivatives in surface prediction. *Technometrics* **35**, 243–255.

Oakley, J. (2002). Value of information for complex cost-effectiveness models. Research Report No. 533/02 Department of Probability and Statistics, University of Sheffield.

Oakley, J. (2004). Estimating percentiles of computer code outputs. *Applied Statistics* **53**, 83–93.

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

Oakley, J. and O'Hagan, A. (2004). Probabilistic sensitivity analysis of complex models: a8 Bayesian approach. *Journal of the Royal Statistical Society B* **66**, 751–769.

O'Hagan, A. and Haylock, R. G. (1997). Bayesian uncertainty analysis and radiological protection. In *Statistics for the Environment 3, Pollution Assessment and Control*, 109–128. V. Barnett and K. F. Turkman (eds.). Wiley: Chichester.

O'Hagan, A. and Oakley, J. E. (2004). Probability is perfect, but we can't elicit it perfectly. *Reliability Engineering and System Safety* (in press).

Sacks, J., Welch, W. J., Mitchell, T. J. and Wynn, H. P. (1989). Design and analysis of computer experiments. *Statistical Science* **4**, 409–435.

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

Santner, T. J., Williams, B. and Notz, W. (2003). *The Design and Analysis of Computer Experiments.* New York: Springer-Verlag.

Welch, W. J., Buck, R. J., Sacks, J., Wynn, H. P., Mitchell, T. J. and Morris, M. D. (1992). Screening, predicting, and computer experiments. *Technometrics* **34**, 15–25.