Introducing BACCO, an R package for Bayesian analysis of computer code output

Robin K. S. Hankin

Abstract

An earlier version of this document was published as Hankin (2005).

This paper introduces the **BACCO** package of R routines for carrying out Bayesian analysis of computer code output. The package requires three packages of related functionality: **emulator** and **calibrator**, and **approximator**, computerized implementations of the ideas of Oakley and O'Hagan (2002), Kennedy and O'Hagan (2001a), and Kennedy and O'Hagan (2000) respectively. The package is self-contained and fully documented R code, and includes a toy dataset that furnishes a working example of the functions.

Package **emulator** carries out Bayesian emulation of computer code output; package **calibrator** allows the incorporation of observational data into model calibration using Bayesian techniques.

The package is then applied to a dataset taken from climate science.

Keywords: Bayesian analysis, climate modelling, uncertainty in climate prediction.

1. Introduction

In this paper I introduce **BACCO**, a new R package comprising packages **emulator** and **calibrator**, that implements Bayesian analysis of computer code output using the methods of Oakley and O'Hagan (2002) and Kennedy and O'Hagan (2001a) respectively. The **BACCO** package can be obtained from the Comprehensive R Archive Network, CRAN, at `http://cran.r-project.org/`.

Many computer models, including climate prediction models such as C-GOLDSTEIN (Marsh, Edwards, and Shepherd 2002), take many hours, or even weeks, to execute. This type of model can have tens to hundreds of free (adjustable) parameters, each of which is only approximately known. Consider a scenario in which a particular model has been run in the past with different sets of input parameters. The code output (here attention is confined to a single scalar value, such as global average temperature) is desired at a new set of parameters, at which the code has not been run. Under the Bayesian view (Currin, Mitchell, Morris, and Ylvisaker 1991), the true value of the code output is a random variable, drawn from a distribution that is conditioned by our prior knowledge, and in this case by the previous code runs; the computer code is thus viewed as a random function. Package **emulator** gives statistical inferences about this random function, and may be used to furnish computationally cheap—yet statistically rigorous—estimates of the computer code output.

Although deterministic—in the sense that running the model twice with identical inputs gives identical outputs—the Bayesian paradigm is to treat the code output as a random variable.

Formally, the code output y is represented as a function $\eta(x;\boldsymbol{\theta})$ of the input parameter vector x and parameter vector $\boldsymbol{\theta}$; if no confusion can arise, $y = \eta(x)$ is written. Although $\eta(\cdot,\cdot)$ is known in principle, in practice this is not the case. C-GOLDSTEIN, for example, comprises over 10^4 lines of computer code, and the fact that the output is knowable in principle appears to be unhelpful in practice.

It is perhaps easier to understand output from a deterministic code as a random variable if one imagines oneself to be at a computer terminal, waiting for a run to finish. The fact that both the code itself, and its input parameters, are perfectly prescribed (thus the output is *in principle* predetermined), does not reduce one's subjective uncertainty in the output; the Bayesian paradigm treats this as indistinguishable from uncertainty engendered by a conventional random variable. Of course, if the code has been run before at slightly different point in parameter space, one may be able to assert with some confidence that this particular run output shouldn't be too different from the last run's (and of course if the code is run at exactly the same point in parameter space, we can be certain that the code output will be identical[1]). Such considerations are formalized in the Gaussian process model, discussed in detail in section 1.2.

A case often encountered in practice is that the values of one or more components of x are uncertain, typically because of practical difficulties of measurement (cloud albedo is a commonly-cited example). It is therefore appropriate to consider X, the true input configuration, to be a random variable with distribution $G(x)$. Thus the output $Y = \eta(X)$ is also a random variable and it is the distribution of Y—the 'uncertainty distribution'—that is of interest.

In the case of C-GOLDSTEIN, direct Monte-Carlo simulation of Y is so computationally intensive as to become impractical: a single run typically takes 24 hours, and the parameter space is 27 dimensional.

Package **emulator** allows one to *emulate* the code: the unknown function $\eta(\cdot)$ is assumed to be a Gaussian process, and previous code runs constitute observations. I will show in this paper that emulation can be orders of magnitude faster than running the code itself; and, at least in the examples considered here, the emulator provides an estimate that is reasonably close to the value that code itself would produce.

In this paper, the object of inference is the random function that is evaluated by the computer code. Although one may question whether this object is actually of interest given that the model is unlikely to predict reality correctly, Kennedy and O'Hagan (2001a) point out that even a good model may be rendered ineffective by uncertainty in the input; and that quantification of the uncertainty distribution is the first step towards reducing uncertainty in the preditions.

1.1. Bayesian calibration of computer models

Notwithstanding that computer models are interesting and important entities in their own

[1]Many computer models are *chaotic* in the sense that running the model twice with closely separated but non-identical parameter values will result in very different outputs. Such systems are generally not amenable to the approach outlined here because the standard parameterization for the correlation function $c(\mathbf{x},\mathbf{x}')$ discussed in the next section breaks down. Such systems may be modelled using a device known as a *nugget* which breaks correlation between infinitesimally different points in parameter space. However, the climate models used here as test cases are known to be non-chaotic.

right, the ultimate object of inference is reality, not the model. Package **calibrator** implements a statistically rigorous method of incorporating observational data into uncertainty analyses, that of Kennedy and O'Hagan (2001a) and Kennedy and O'Hagan (2001b) (hereafter KOH2001 and KOH2001S respectively).

When preparing a computer model for making predictions, workers often *calibrate* the model by using observational data. In rough terms, the idea is to alter the model's parameters to make it fit the data.

A statistically unsound (Currin *et al.* 1991) methodology would be to minimize, over the allowable parameter space, some badness-of-fit measure over the physical domain of applicability of the code.

To fix ideas, consider C-GOLDSTEIN predicting sea surface temperature (SST) as a function of position x (here styled "latitude" and "longitude"). The naïve approach discussed above would be to minimize, over all parameter values $\boldsymbol{\theta}$ in a set of allowable parameters $\mathcal{P}$, the squared discrepancies between observations $z(x)$ and model predictions $\eta(x,\boldsymbol{\theta})$ summed over observational space $\mathcal{X}$:

$$\min_{\boldsymbol{\theta}\in\mathcal{P}}\left[\sum_{x\in\mathcal{X}}\left(z(x)-\eta(x,\boldsymbol{\theta})\right)^2\right]$$

From a computational perspective, minimizing the object function above necessitates minimization over a large dimensional parameter space; optimization techniques over such spaces are notorious in requiring large numbers of function evaluations, which is not be feasible in the current context. Also note that the method requires code evaluations at the same points ("x") as the observations $z(\cdot)$, which may not be available.

Other problems with this approach include the implicit assumption that observation errors (and model evaluations) are independent of one another. This assumption is likely to be misleading for closely separated observation points and for pairs of code evaluations that use parameter sets that differ by only a small amount.

Bayesian methods are particularly suitable for cases in which the above problems are present, such as the climate model predictions considered here. KOH2001 present a methodology by which physical observations of the system are used to learn about the unknown parameters of a computer model using the Bayesian paradigm.

1.2. The Gaussian process

The notion of Gaussian process underlies both **emulator** and **calibrator** packages. Consider a function $f:\mathcal{X}\longrightarrow\mathbb{R}$ with $\mathcal{X}\subseteq\mathbb{R}^p$. If $f(\cdot)$ is regarded as an unknown function, in the Bayesian framework it becomes a random function. Formally, $f(\cdot)$ is a Gaussian process if, for every $n\in\mathbb{N}$, the joint distribution of $f(\mathbf{x}_1),\ldots,f(\mathbf{x}_n)$ is multivariate normal provided $\mathbf{x}_i\in\mathcal{X}$. In particular, $f(\mathbf{x})$ is normally distributed for all $\mathbf{x}\in\mathcal{X}$.

The distribution is characterized by its mean function $m(\cdot)$, where $m(\mathbf{x})=E\left\{f(\mathbf{x})\right\}$, and its covariance function $c(\cdot,\cdot)$, where $c(\mathbf{x},\mathbf{x}')=\mathrm{cov}\left\{f(\mathbf{x}),f(\mathbf{x}')\right\}$.

It is usual to require that $f(\mathbf{x})$ and $f(\mathbf{x}')$ be closely correlated if $\mathbf{x}$ and $\mathbf{x}'$ are themselves close; here, it is assumed that $c(\cdot,\cdot)=\sigma^2 r(\mathbf{x}-\mathbf{x}')$ with $r(\mathbf{d})=\exp(-\mathbf{d}^T\boldsymbol{\Omega}\mathbf{d})$, $\boldsymbol{\Omega}$ being a symmetric positive definite matrix (which is usually unknown and has to be estimated).

2. Emulation: computer output as a Gaussian process

For any random function $\eta(\cdot)$ and set of points $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ in its domain, the random vector $\{\eta(\mathbf{x}_1), \ldots, \eta(\mathbf{x}_n)\}$ is assumed to be a multivariate normal distribution with mean

$$E\left\{\eta(\mathbf{x})|\beta\right\} = h(\mathbf{x})^T\beta,$$

conditional on the (unknown) vector of coefficients β and $h(\cdot)$, the q known regressor functions of $\mathbf{x} = (x_1, \ldots, x_p)^T$; a common choice is $h(\mathbf{x}) = (1, x_1, \ldots, x_p)^T$, but one is free to choose any function of $\mathbf{x}$. The covariance is given by

$$\operatorname{cov}\left\{\eta(\mathbf{x}), \eta(\mathbf{x}')|\sigma^2\right\} = \sigma^2 c(\mathbf{x}, \mathbf{x}')$$

where $c(\mathbf{x}, \mathbf{x}')$ is a correlation function that measures the correlation between $\mathbf{x}$ and $\mathbf{x}'$; here the form

$$c(\mathbf{x}, \mathbf{x}') = \exp\left\{-(\mathbf{x} - \mathbf{x}')^T B(\mathbf{x} - \mathbf{x}')\right\} \tag{1}$$

will be used, where B is a positive semidefinite matrix. This form has the advantage that $\eta(\cdot)$ has derivatives of all orders; other forms for the covariance function do not have this desirable property. Oakley and O'Hagan use the weak conjugate prior for β and σ^2:

$$p(\beta, \sigma^2) \propto \sigma^{(r+q+2)/2} \exp\left[\left\{(\beta - z)^T V^{-1}(\beta - z) + a\right\} / \left(2\sigma^2\right)\right]$$

and discuss methods for expert elicitation of a, r, z and V. The random function $\eta(\cdot)$ may be conditioned on observations at specific points in parameter space (ie code runs). This set of points is known as the *experimental design*, or the *design matrix*. Although it is possible to tailor design matrices to a particular problem, here a Latin hypercube system is used. With the specified prior, and an experimental design $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ on which $\eta(\cdot)$ has been evaluated giving $\mathbf{d} = \eta(\mathcal{D})$, it can be shown (Oakley 2; Oakley and O'Hagan 2002) that

$$\left.\frac{\eta(\mathbf{x}) - m^*(\mathbf{x})}{\hat{\sigma}\sqrt{c^*(\mathbf{x}, \mathbf{x})}}\right| \mathbf{d}, B \sim t_{r+n} \tag{2}$$

where

$$m^*(\mathbf{x}) = h(\mathbf{x})^T\hat{\beta} + t(\mathbf{x})^T A^{-1}(\mathbf{d} - H\hat{\beta}) \tag{3}$$

$$c^*(\mathbf{x}, \mathbf{x}') = c(\mathbf{x}, \mathbf{x}') - t(\mathbf{x})^T A^{-1} t(\mathbf{x}') + \left\{h(\mathbf{x})^T - t(\mathbf{x})^T A^{-1} H\right\} V^* \left\{h(\mathbf{x}')^T A^{-1} H\right\}^T$$

$$t(\mathbf{x}) = (c(\mathbf{x}, \mathbf{x}_1), \ldots, c(\mathbf{x}, \mathbf{x}_n))^T \qquad H = \left(h^T(\mathbf{x}_1), \ldots, h^T(\mathbf{x}_n)\right)^T$$

$$A = \begin{pmatrix} 1 & c(\mathbf{x}_1, \mathbf{x}_2) & \cdots & c(\mathbf{x}_1, \mathbf{x}_n) \\ c(\mathbf{x}_2, \mathbf{x}_1) & 1 & & \vdots \\ \vdots & & \ddots & \\ c(\mathbf{x}_n, \mathbf{x}_1) & \cdots & & 1 \end{pmatrix}$$

$$\hat{\beta} = V^*\left(V^{-1}z + H^T A^{-1}\mathbf{d}\right) \qquad \hat{\sigma}^2 = \frac{a + z^T V^{-1} z + \mathbf{d}^T A^{-1}\mathbf{d} - \hat{\beta}^T \left(V^*\right)^{-1} \hat{\beta}}{n + r - 2}$$

$$V^* = \left(V^{-1} + H^T A^{-1} H\right)^{-1} \qquad d = (\eta(\mathbf{x}_1), \ldots, \eta(\mathbf{x}_n))^T .$$

Thus $m^*(\mathbf{x})$ is a computationally cheap approximation to $\eta(\mathbf{x})$; uncertainties are given by $\hat{\sigma}\sqrt{c^*(\mathbf{x},\mathbf{x})}$. These equations are consistent in that the estimated value for points actually in the design matrix is in fact the observations (with zero error). Writing $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^T$, we have

$$m^*(\mathbf{X}) = H\hat{\beta} + t(\mathbf{X})^T A^{-1}(\mathbf{d} - H\hat{\beta}) = \mathbf{d} \tag{4}$$

where we use the fact that $h(\mathbf{X}) = H$ and $t(\mathbf{X}) = A$, and expand $\hat{\beta}$ directly. It may similarly be shown that $c^*(\mathbf{x}, \mathbf{x}) = 0$ for $\mathbf{x} \in \mathcal{D}$, as expected: the emulator should return zero error when evaluated at a point where the code output is known.

2.1. Package emulator in use: toy problem

The central function of package **emulator** is `interpolant()`, which calculates the terms of equation 3. Here, I use the package to investigate a simple toy world in which observations are a Gaussian process on six parameters named `a` to `f`: the mean is `0*a + 1*b + 2*c + 3*d + 4*e + 5*e + 6*f`, corresponding to $\beta =$ `0:6`; the residuals are correlated multivariate Gaussian, with unit variance and all scales unity[2]. The overall approach is to use this scheme to generate data, then use the data to estimate the parameters, and finally, compare the estimates with the true values.

I first consider a synthetic dataset consisting of observations of the predefined Gaussian process on a design matrix generated by **emulator**'s function `latin.hypercube(n,d)`. This function takes two arguments: the first, `n`, is the number of observations to take; the second, `d` is the number of dimensions of the design space. The output of the function `latin.hypercube()` is a matrix with each row corresponding to one point in parameter space at which the Gaussian process is observed. Taking 20 observations in a 6 dimensional design space appears to give a small yet workable toy problem:

```
> toy <- latin.hypercube(20, 6)
> head(toy)

      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
[1,] 0.875 0.775 0.425 0.425 0.975 0.625
[2,] 0.275 0.875 0.775 0.875 0.425 0.975
[3,] 0.325 0.175 0.575 0.275 0.625 0.325
[4,] 0.475 0.575 0.675 0.375 0.325 0.225
[5,] 0.725 0.125 0.975 0.925 0.275 0.775
[6,] 0.175 0.225 0.375 0.475 0.575 0.475
```

[2]Here, "scales" means the e-folding lengthscales for correlations between datapoints. The concept is a special case of equation 1 in which B is diagonal, with elements B_i, say; the relevant lengthscales S_i will be $B_i = 1/S_i^2$. Then the correlation between two points $\mathbf{x}$ and $\mathbf{x}'$ is given by

$$c(\mathbf{x}, \mathbf{x}') = \exp\left(-\sum_{i=1}^{r}\left(\frac{\left|x_i - x_j'\right|}{S_i}\right)^2\right).$$

Sometimes it is more convenient to consider the elements of matrix B, and sometimes it is better to think in terms of the lengthscales S_i (they have the same dimensions as the parameters).

It is now possible to specify the expectation of the Gaussian process as a linear product, with coefficients `0:6` acting on regression function `H1.toy()`, which prepends a 1 to its argument (viz `regressor.basis(x)=c(1,x)`). In R idiom:

```
> f <- function(x) sum((0:6) * x)
> expectation <- apply(regressor.multi(toy), 1, f)
```

where `regressor.multi()` is a vectorized version of `regressor.basis()`. Recall that we suppose `f()` to be an unknown function; one object of package **emulator** is to infer its coefficients. Next, matrix A is calculated, with scales all equal to 1:

```
> toy.scales <- rep(1, 6)
> toy.sigma <- 0.4
> A <- toy.sigma * corr.matrix(xold = toy, scales = toy.scales)
```

Thus, for example, `A[1,2]` gives the covariance between observation 1 and observation 2. We can simulate observational data by sampling from the appropriate Gaussian process directly, using the **mvtnorm** package (Hothorn, Bretz, and Genz 2001):

```
> d <- as.vector(rmvnorm(n = 1, mean = expectation, sigma = A))
```

Vector `d` is the vector of observations, with variance matrix `A`. Now function `interpolant()` of package **emulator** can used to estimate the unknown function `f()`, at a point not in the training set:

```
> x.unknown <- rep(0.5, 6)
> jj <- interpolant(x.unknown, d, toy, scales = toy.scales, g = TRUE)
> print(drop(jj$mstar.star))

[1] 10.19733
```

The estimated value (element `mstar.star`) is reasonably close to the correct value (which we know to be `0.5*sum(0:6)=10.5`). Also, the estimated value for the regression line, given by element `betahat`, is close to the correct value of `0:6`:

```
> print(jj$betahat)

    const      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
-0.497094  1.319761  2.146907  2.486594  4.774684  5.068548  5.662764
```

The package will also give an estimate of the error on $\hat{\beta}$:

```
> print(jj$beta.marginal.sd)

    const      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
0.5404718 0.4547840 0.4117208 0.3911489 0.3834957 0.4311633 0.4023278
```

showing in this case that the true value of each component of β lies within its marginal 95% confidence limits.

Estimation of scales

In the above analysis, the scales were set to unity. In real applications, it is often necessary to estimate them *ab initio* and a short discussion of this estimation problem is presented here.

The scales are estimated by maximizing, over allowable scales, the posterior likelihood. This is implemented in the package by `optimal.scales()`, which maximizes the a-postiori probability for a given set of scale parameters. The likelihood is

$$(\hat{\sigma})^{-(n-q)/2}\,|A|^{-1/2}\cdot\left|H^TA^{-1}H\right|^{-1/2} \tag{5}$$

which is evaluated in the package by `scales.likelihood()`. It is more convenient to work with the logarithms of the scales, as they must be strictly positive. Function `optimal.scales()` is essentially a wrapper for multidimensional optimizing routine `optim()`, and is used as follows:

```
> scales.optim <- optimal.scales(val = toy, scales.start = rep(1,
+     6), d = d, give = FALSE)
> print(scales.optim)

[1] 1.04623721 0.14381200 0.20780336 0.02103420 0.02745443 0.38725412
```

The estimated scales are not particularly close to the (known) real values, all of which are unity; the scales are known to be "difficult to estimate" (Kennedy and O'Hagan 2000). One interpretation might be that there is not a large amount of information residing in the residuals once β has been estimated.

However, using the estimated scales does not seem to make much difference in practice:

```
> interpolant(x.unknown, d, toy, scales = toy.scales, g = FALSE)

[1] 10.19733

> interpolant(x.unknown, d, toy, scales = scales.optim, g = FALSE)

[1] 10.07065
```

2.2. Emulation applied to a dataset from climate science

The methods above are now used for a real dataset obtained from C-GOLDSTEIN (Edwards and Marsh 2005), an Earth systems model of intermediate complexity[3].

[3]That is, intermediate between box models such as MAGICC (Wigley, Raper, Hulme, and Smith 2000) and general circulation models such as HadCM3 (Gordon *et al.* 2000), which solve the primitive equations for fluid flow on a 3D grid.

The `results.table` dataset is a dataframe consisting of 100 rows, corresponding to 100 runs of C-GOLDSTEIN. Columns 1-19 are input values and columns 20-27 are code outputs. Here, the column corresponding to global average surface air temperature (SAT) is used.

Figure 1 shows observed versus predicted SATs. The emulator was "trained" on the first 27 runs, so 27 of the points are perfect predictions, by equation 4. The other 77 points show a small amount of error, of about 0.1°C. Such an error is acceptable in many contexts.

One interpretation of the accurate prediction of SAT would be that this output variable is well-described by the Gaussian process model.

However, variables which exhibit violently nonlinear dynamics might be less well modelled: examples would include diagnostics of the thermohaline circulation (THC), which is often considered to be a bistable system (Vellinga and Wood 2002), exhibiting catastrophic breakdown near some critical hypersurface in parameter space. Such behaviour is not easily incorporated into the Gaussian process paradigm and one would expect emulator techniques to be less effective in such cases.

Computational efficiency

C-GOLDSTEIN takes 12 to 24 hours to complete a single run, depending on the parameters. The emulator software presented here takes about 0.1 seconds to calculate the expected value and estimated error—a saving of over five orders of magnitude.

One computational bottleneck is the inversion of matrix A, but this only has to be done once per emulator.

3. Bayesian calibration: package calibrator

Computer climate models generally require two distinct groups of inputs. One group is the (unknown) parameter set about which we wish to make inferences; these are the calibration inputs. It is assumed that there is a true but unknown parameter set $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_{q_2})$ with the property that if the model were run with these values, it would reproduce the observations plus a model inadequacy term plus noise. For any single model run, it is convenient to denote the calibration parameters by $\mathbf{t}$ where $\mathbf{t} = (t_1, \ldots, t_{q_2})$.

The other group comprises all the other model inputs whose value might change when the calibration set is fixed, conventionally denoted by $\mathbf{x} = (x_1, \ldots, x_{q_1})$. In the current context, this group is the latitude and longitude at which mean surface temperature is desired.

A model run is essentially an evaluation—or statistical observation—of the unknown random function $\eta(\cdot, \cdot)$ at a specified point in parameter and location space $\eta(\mathbf{x}, \mathbf{t})$; a field observation is a statistical observation of another unknown random function $\xi(\cdot)$.

This system has the notational advantage of distinguishing the true but unknown value $\boldsymbol{\theta}$ of the calibration inputs from the value $\mathbf{t}$ that were set as inputs when running the model.

The calibration data comprise the n field observations $\mathbf{z} = (z_1, \ldots, z_n)^T$ (ie observations of $\xi(\mathbf{x}_i)$, subject to error), and the outputs $\mathbf{y} = (y_1, \ldots, y_N)^T$ from N runs of the computer code evaluated at points on a design matrix D_1 where $D_1 = \{(\mathbf{x}_1^*, \mathbf{t}_1), \ldots, (\mathbf{x}_N^*, \mathbf{t}_N)\}$. Thus $\mathbf{y} = \eta(D_1)$, or $y_i = \eta(\mathbf{x}_i^*, \mathbf{t}_i)$; we write $\mathbf{d}^T = (\mathbf{y}^T, \mathbf{z}^T)$ for the full set of calibration data.

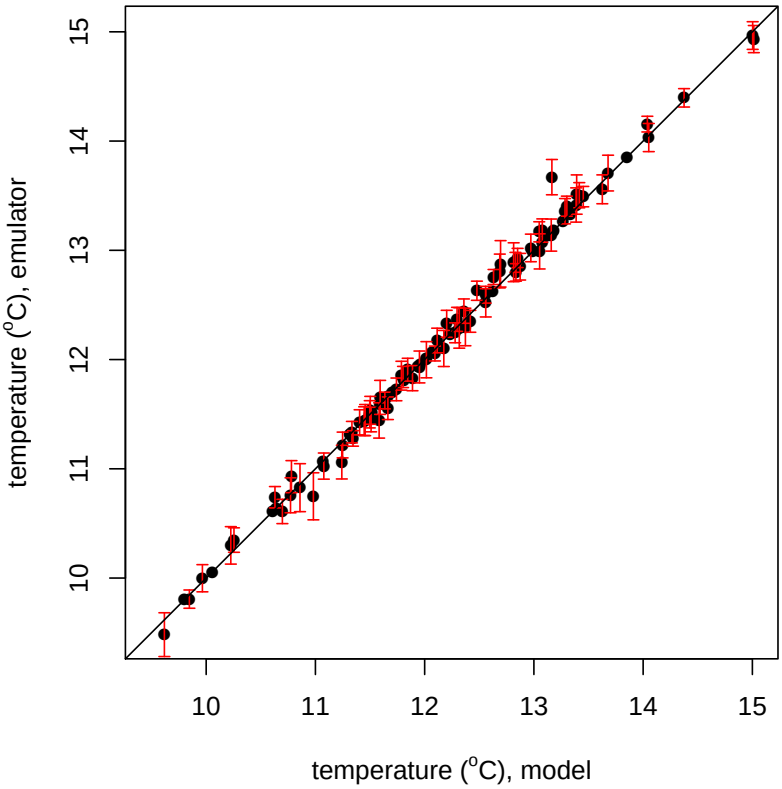

Figure 1: Observed vs predicted surface air temperature; " observed" means actual code run; "predicted" means predicted using the emulator. Error bars are 95% confidence intervals

Model

KOH2001 suggest that

$$\xi(\mathbf{x}) = \rho \cdot \eta\left(\mathbf{x}, \theta\right) + \delta\left(\mathbf{x}\right) \tag{6}$$

is an adequate representation of the relationship between observation and computer model output. Here $\delta(\cdot)$ is a model inadequacy term that is independent of model output $\eta(\cdot, \cdot)$. It is then assumed that the observations $z_i = \xi(\cdot) + N(0, \lambda)$. The true value of neither λ nor ρ is known and has to be inferred from observations and the calibration dataset.

This statistical model suggests a non-linear regression in which the computer code itself defines the regression function through the term $\rho \cdot \eta(\mathbf{x}_i, \boldsymbol{\theta})$ with parameters ρ and $\boldsymbol{\theta}$. KOH2001 consider the meaning of model fitting in this context and conclude that "the true value for $\boldsymbol{\theta}$ has the same meaning and validity as the true values of regression parameters [in a standard regression model]. The 'true' $\boldsymbol{\theta}$ is a best-fitting $\boldsymbol{\theta}$ in the sense of representing the data faithfully according to the error structure specified for the residuals".

Prior knowledge

The Bayesian paradigm allows the incorporation of *a priori* knowledge (beliefs) about the parameters of any model being used. Consider a model in which climate sensitivity λ is a free parameter[4]. In principle, we are free to 'tune' λ to give the best fit to observational data. However, most workers would consider $0.4 - 1.2\,\mathrm{K} \cdot \mathrm{W}^{-1} \cdot \mathrm{m}^2$ to be a reasonable range for this physical constant, in the sense that finding an optimal value for λ outside this range would be unacceptable.

It is possible to incorporate such 'prior knowledge' into analysis by ascribing a prior distribution to λ. One might interpret the upper and lower ends of the range as 5th and 95th percentiles of a normal, or lognormal, distribution; if the bounds are "hard", then a uniform PDF might be used.

The prior PDF for the vector of parameters $\boldsymbol{\theta}$ is denoted $p(\cdot)$. KOH2001S assume that $\boldsymbol{\theta}$ is multivariate Gaussian with $\theta \sim \mathcal{N}\left(m_\theta, V_\theta\right)$.

Hyperparameters

KOH2001S define hyperparameters ρ, λ, ψ_1, ψ_2 that specify the correlations between the observations in the dataset, and the relationship between the computer model and observations.

The variance matrix of $\mathbf{d}$ has $N+n$ rows and columns; its (j, j')-th element is $c_1((\mathbf{x}^*_j, \mathbf{t}_j), (\mathbf{x}^*_{j'}, \mathbf{t}_{j'}))$. KOH2001 derive results for the case where

$$c_1((\mathbf{x}, \mathbf{t}), (\mathbf{x}', \mathbf{t}')) = \sigma_1^2 \exp\left\{-(\mathbf{x} - \mathbf{x}')^T \boldsymbol{\Omega}_x (\mathbf{x} - \mathbf{x}') - (\mathbf{t} - \mathbf{t}')^T \boldsymbol{\Omega}_t (\mathbf{t} - \mathbf{t}')\right\}$$

where $\boldsymbol{\Omega}_x$ and $\boldsymbol{\Omega}_t$ are arbitrary functions of hyperparameter ψ_1. A similar definition gives the variance matrix of $\mathbf{x}$ in terms of a second hyperparameter ψ_2.

For the purposes of implementing KOH2001, it is convenient to use a single R object that contains not only the hyperparameters above, but also other information such as σ_1^2 and ρ. The values of these variables have to be inferred in any case and it is logical to consider them alongside the formal hyperparameters.

[4]Climate sensitivity is usually defined as the equilibrium change in global mean surface temperature following a doubling of the atmospheric CO_2 concentration (Stainforth 2005).

In the R implementation presented here, the hyperparameter object `phi` also contains extra information such as the Cholesky decomposition of the square matrices, and the prior mean and variance of $\boldsymbol{\theta}$. This device greatly simplifies many functions in the package; an example is given in the discussion of `ht.fun()` below.

4. Design of package calibrator

The overarching design philosophy of package **calibrator** is to be a direct and transparent implementation of KOH2001 and KOH2001S. Each equation appearing in the papers has a corresponding R function, and the notation is changed as little as possible. One reason for this approach is debugging: with such a structured programming style, it is possible to identify differences between alternative implementations. Speed penalties of this approach appear to be slight.

4.1. Notation

In **calibrator**, most notation is an `ascii` version of that used by KOH2001S. Function names are descriptive where possible, or (as in `p.eqn8.supp()`), named for the relevant equation number in the supplement KOH2001S.

4.2. Toy dataset

Partly to facilitate code testing, and partly as a didactic aid in the man pages, package **calibrator** includes a simple "toy" dataset. Toy objects have a `.toy` suffix, as in `theta.toy`. Each function's documentation includes a working example in which toy data is processed appropriately.

For any real application (such as the climate analysis discussed below), each member of the toy dataset must be replaced with the corresponding real dataset.

Toy datasets are loaded by `data(toys)` and `?toys` gives a complete list. The toy dataset is designed to be simple and transparent, thus offering a clear test of the package's methods. The dataset comprises the following objects:

- Design matrix. Two elements:
  - `D1.toy`, matrix of 8 rows of code run points, with five columns. The first two columns ('x' and 'y') represent the input space of ζ (nonparameters styled "latitude and longitude"); the next three are parameter values.
  - `D2.toy`, matrix of 5 rows of field observations on two variables 'x' and 'y'.
- Observational data. Three elements:
  - `y.toy`, a vector of length 8. Each element corresponds to the code output at points corresponding to the rows of design matrix `D1.toy`.
  - `z.toy`, a vector of length five. Each element corresponds to a measurement at each of the rows of `D2.toy`.
  - `d.toy`, a data vector consisting of length 13: elements 1-8 are `y.toy` and elements 9-13 are `z.toy`.

- Functions. Ten elements:
  - `create.new.toy.datasets(N,n)`, a function to create new toy datasets with `N` observations and `n` code runs; error distributions generated by directly simulating the appropriate Gaussian Process on a design matrix that is a Latin hypercube generated by `latin.hypercube()`.
  - `E.theta.toy()`, a function that returns expectation of `H(D)` with respect to theta
  - `Edash.theta.toy()` as above, but returns expectation with respect to the "dashed" normal distribution detailed on page 4 of KOH2001S.
  - `extractor.toy()`, a function that extracts $\mathbf{x}^*$ and $\mathbf{t}$ from `D2`; a toy example is needed because the extraction method depends on the nature of `D1`.
  - `H1.toy()`, a function that returns the regression basis of `D1.toy`
  - `H2.toy()` As above but for `D2.toy`.
  - `hpp.fun.toy()`, a function to create hyperparameter object `phi` in a form suitable for passing to the other functions in the library.
  - `hpp.change.toy()`, as above but modifies hyperparameter object `phi`
  - `computer.model()`, function that represents the unknown computer program. In the toy case, it is a simple Gaussian process, with mean `h.toy(x,t) %*% REAL.COEFFS` and variance matrix `REAL.SIGMA1SQUARED * diag(REAL.SCALES)`. The capitalized variable names are the true but unknown parameters of the computer model. They appear in the R source code but are not modifiable in the context of package usage, because they represent the "real world"). Estimates of these quantities should be close to the actual values.
  - `reality()`, a function to simulate observational data. This toy function is a simple Gaussian process, with internal (unknown) parameters. The mean is given by `computer.model()`, plus a Gaussian process with appropriate capitalized parameters. As above, these parameters are stand for unobservable properties of the system under examination, whose values must be inferred from observation.

The objects in the toy dataset are chosen to be small enough for the functions of the package to operate quickly. However, for the purposes of testing the optimization strategy, longer datasets are needed. Such are given by function `create.new.toy.datasets()`, which automagically creates a dataset of the same form as the toy dataset discussed above but with the number of observations supplied by the user. The observations are drawn directly from the appropriate multivariate Gaussian distribution.

4.3. Example function

In order to show the design philosophy of **calibrator**, and to illustrate some of the programming issues that occur when implementing a conceptually complex methodology, I discuss in some detail a typical function of the package. The function chosen is `ht.fun()`, which is one of many needed when calculating the conditional covariance matrix of $\mathbf{z}$. Function `ht.fun()` calculates the matrix

$$\int \mathbf{h}_1\left(\mathbf{x}_j, \boldsymbol{\theta}\right) \mathbf{t}\left(\mathbf{x}_i, \boldsymbol{\theta}\right)^T p(\boldsymbol{\theta})\, d\boldsymbol{\theta} \tag{7}$$

where $\mathbf{h}_1(\cdot,\cdot)$ is the basis function for calculating expectation of $\mathbf{y}$, and $\mathbf{t}(\cdot,\cdot)$ is the set of distances to a point with k-th element

$$c_1\left((\mathbf{x}_i,\boldsymbol{\theta}),(\mathbf{x}_k^*,\mathbf{t}_k)\right), \qquad j=1,\ldots,N. \tag{8}$$

With these definitions, KOH2001 show that the k-th column of equation 7, and thus `ht.fun(...)[,k]`, is

$$\begin{aligned}\sigma_1^2\left|\mathbf{I}+2\mathbf{V}_\theta\boldsymbol{\Omega}_x\right|^{-1/2}\exp\left\{-(\mathbf{x}_i-\mathbf{x}_k^*)^T\boldsymbol{\Omega}_x(\mathbf{x}_i-\mathbf{x}_j^*)\right\}\times\\ \exp\left\{-(\mathbf{m}_\theta-\mathbf{t}_k)^T\left(2\mathbf{V}_\theta+\boldsymbol{\Omega}_t^{-1}\right)^{-1}(\mathbf{m}_\theta-\mathbf{t}_k)\right\}E'_\Theta\left(\mathbf{h}_1(\mathbf{x}_j,\boldsymbol{\theta})\right)\end{aligned} \tag{9}$$

where E'_Θ is discussed below. Equation 9 is one of several similar equations appearing in KOH2001. In the implementation, function `ht.fun()` takes ten arguments:

```
> args(ht.fun)

function (x.i, x.j, D1, extractor, Edash.theta, H1, fast.but.opaque = TRUE,
    x.star = NULL, t.vec = NULL, phi)
NULL
```

Apart from the arguments already discussed, this function requires several further arguments:

- `extractor()` is a function that splits D_1 into $\mathbf{x}^*$ and $\mathbf{t}$. It is necessary to pass this function explicitly if the split between code input space and parameter space is to be arbitrary. In the toy example, with the code input space being $\mathbb{R}^2$ and the parameter space being $\mathbb{R}^3$, the working toy example `extractor.toy()` (supplied as part of the `toys` dataset in the package) returns a list with elements `x[1:2]` and `x[3:5]`.
- `Edash.theta()` is a function that returns expectation with respect to the normal distribution $\mathcal{N}\left(\left(\boldsymbol{V}_\theta^{-1}+2\boldsymbol{\Omega}_t\right)^{-1}\left(\boldsymbol{V}_\theta^{-1}\mathbf{m}_\theta+2\boldsymbol{\Omega}_t\mathbf{t}_k\right),\left(\boldsymbol{V}_\theta^{-1}+2\boldsymbol{\Omega}_t\right)^{-1}\right)$.
- `fast.but.opaque` is a Boolean variable, with default `TRUE` meaning to take advantage of Cholesky decomposition for evaluating quadratic forms (the hyperparameter object `phi` includes the upper and lower triangular decompositions by default). However, this requires the calling routine to supply $\mathbf{x}^*$ and $\mathbf{t}$ explicitly.

  Setting `fast.but.opaque` to `FALSE` should give numerically identical results in a slower, but more conceptually clear, manner. This option is used mainly for debugging purposes.
- `phi`. The hyperparameter object. In KOH2001S, "hyperparameters" refers to ψ_1 and ψ_2, but in this computer implementation is it convenient to augment ϕ with other objects that do not change from run to run (such as the a priori PDF for $\boldsymbol{\theta}$ and Cholesky decompositions for the various covariance matrices). This way, it is possible to pass a *single* argument to each function in the package and update it in a consistent—and object-oriented—manner.

  The package includes methods for setting and changing the components of `phi` (the toy examples provided are `hpp.fun.toy()` to create a hyperparameter object and `hpp.change.toy()` to change elements of the hyperparameter object).

This example illustrates several features of the KOH2001 approach. Firstly, the system is algebraically complex, requiring a multi-level hierarchy of concepts. The design philosophy of **emulator** is to code, explicitly, each function appearing in KOH2001; and to include working examples of these functions in the online help pages. This structured approach aids debugging and code verification.

4.4. Optimization of parameters and hyperparameters

Kennedy and O'Hagan estimate the hyperparameters first, then the parameters. In this paper, I use 'hyperparameters' to mean ψ_1 and ψ_2 together with ρ, λ, σ_1^2, σ_2^2; all have to be estimated.

The package provides functions `stage1()` and `stage2()` which use numerical optimization techniques to estimate hyperparameters ψ_1 and ψ_2 in two stages, as per KOH2001a. These functions maximize the posterior likelihood of observations $\mathbf{y}$.

Function `stage3()` optimizes the model parameters by maximizing the posterior PDF of $\boldsymbol{\theta}$; but note that KOH2001 explicitly state that point estimation of $\boldsymbol{\theta}$ is not generally desirable, and suggest that calibrated prediction is usually a better approach.

5. Package calibrator in use

Two case studies of **calibrator** are now presented. First, a simple "toy" dataset will be analysed. The object of this is to show how the software behaves when the dataset has only a simple structure that is known in advance.

The package is then applied to a real climate problem in which model parameters are assessed using model output and observational data.

5.1. The toy problem

In this section, **calibrator** is used to estimate the hyperparameters of the toy dataset, in the two-stage process outlined above and in more detail in KOH2001a; all executable R code is taken directly from the `stage1()` help page. The parameters of the model are then estimated, using the estimated hyperparameters.

One advantage of considering a simple toy example is that the covariance structure may be specified exactly. Thus one can generate observations and code runs directly from the appropriate multivariate Gaussian distribution using `rmvnorm(n=1, mean, sigma)`, where `rmvnorm()`, from package **mvtnorm** returns samples from a multivariate Gaussian distribution with specified mean and variance.

If this is done, one knows that the conditions and assumptions specified by KOH2001 are met exactly, with the additional advantage that the basis functions, scales, regression coefficients, and model parameters, are all known and adjustable.

KOH2001 state explicitly that exact determination of the hyperparameters tends to be difficult in practice, and indeed even in the toy example the numerically determined values for `phi` differ somewhat from the exact values, although they are correct to within half an order of magnitude. Note that KOH2001 consider only the case of $h_1(\mathbf{x}, \mathbf{t}) = h_2(\mathbf{x}) = 1$, corresponding to a constant prior with $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$ being scalars; but in the toy example I consider the more

complex case of $h_1(\mathbf{x},\mathbf{t}) = (1,\mathbf{x},\mathbf{t})^T$ and $h_2(\mathbf{x}) = (H(0.5-\texttt{x[1]}), H(\texttt{x[2]}-0.5))^T$ where H is the Heaviside function[5]. In the climatic case considered below, Legendre functions are needed to specify a prior for global temperatures as a function of latitude and longitude.

5.2. Results from toy problem

In this section, various parameters and hyperparameters of the toy problem are estimated using standard numerical optimization techniquies. The correct values are viewable by examining the source code, or by using `computer.model()` with the `export` argument set to to `TRUE`. Because these operations are not possible in real applications (parameters are unobservable), accessing them causes the package to give a warning, reminding the user that he is carrying out an operation that requires omniscience[6], an attribute not thought to be possessed by the typical user of **calibrator**.

In the first stage, ψ_1 is estimated by maximizing $p(\psi_1|y)$, given by equation 4 of KOH2001. This is carried out by `stage1()` of the package. In stage 2, the remaining hyperparameters ψ_2 are estimated by maximizing the posterior probability $p(\rho,\lambda,\psi_2|\mathbf{d},\psi_1)$ given by the (unnumbered) equation on page 4, evaluated by R function `p.page4()`.

Taking the example given in the help page for `stage1()`, but with 74 code runs and 78 observation points, chosen as a compromise between informative dataset and reasonable execution time, yields

```
          x           y           A           B           C
 6.82060592  0.09290103  6.25596840  0.77746434  0.01929785
sigma1squared
 0.83948328
```

comparing reasonably well with the true values of `10, 0.1, 10, 1, 0.1, 0.4`, as hard-coded in to `computer.model()`. Better agreement might be obtained by taking more observations or using more code runs, at the expense of increased runtimes.

Stage 2, in which ψ_2, λ, and σ_1^2 are estimated, yields

```
      rho       lambda            x            y sigma2squared
 0.8631548    0.1051552  3.81230769   3.40685222    0.64521921
```

comparing reasonably well with the real values of 2 and 3 for the scales, 0.2 for lambda, and 0.3 for σ_1^2, hard-coded in to `reality()`. Again, the estimated values are close to the exact values but further improvements could be obtained by taking more observations or using more code runs; but too many observations can invite numerical problems as several large, almost singular, matrices have to be inverted.

Stage 3 finds a maximum likelihood estimate for the model parameters, by maximizing the apriori PDF for θ, given by `p.eqn8.supp()` in this implementation.

[5]Thus, the model inadequacy term is the sum of two parts, the first being zero if `x[1]>0.5` and an unknown constant otherwise; and the second being zero if `x[2]<0.5` and a second unknown constant otherwise. The R idiom would be $h_2(\mathbf{x}) =$ `c( x[1]<0.5, x[2]>0.5)`

[6]Changing the parameters is not permitted without editing the source code. This would be equivalent to omnipotence.

ϕ	A	B	C
$\hat{\theta}\|\phi = \phi_{\text{true}}$	0.88	0.09	0.24
$\hat{\theta}\|\phi = \phi_{\text{best}}$	0.78	0.13	0.40
$\hat{\theta}\|\phi = \phi_{\text{prior}}$	0.51	0.22	0.71
θ_{true}	0.90	0.10	0.30

Table 1: Estimates for the parameter set using three different values for the hyperparameters ϕ

```
      A          B           C
0.78345 0.131281 0.40222366
```

comparing resonably well with the real values of `c(0.9, 0.1, 0.3)` hard coded in to `reality()`. Note that the three stages above operate sequentially in the sense that each one estimates parameters, which are subsequently held fixed.

Effect of inaccurate hyperparameters

As shown above, even with a system specifically tailored for use with KOH2001, the estimated values for the hyperparameters may differ from the true values by a substantial amount.

To gain some insight into the effect of hyperparameter misprediction, table 1 presents some predictions conditional on the true hyperparameters, the estimated hyperparameters, and several incorrect values. Observe how the true value of the hyperparameters yields the most accurate values for θ, although in this case the difference is slight. It is important to realize that the best that can possibly be expected is for the predicted value being drawn from the same distribution as generated the observation. In this case, both observations and computer predictions are Gaussian processes.

5.3. Conclusions from toy problem analysis

The toy problem analyzed above shows that the implementation is satisfactory in the sense that the true values of the parameters and hyperparameters may be estimated with reasonable accuracy if their true value is known and the assumptions of KOH2001 are explicitly satisfied.

However, certain difficulties are apparent, even in this highly simple system, which was specifically created in order to show the methods of KOH2001S in a favourable light.

The primary difficulty is execution time for the optimization process to find the best set of hyperparameters. Stage 1 is reasonably fast, but the objective function minimized in stage 2 takes about 5 minutes to evaluate with the setup described above: execution time goes as the fourth power of number of observations. In the toy case considered here, any number of observations could be taken (the generating function is very cheap), but too many renders the methods unworkably slow; and too few makes the estimates unreliable.

Note that the situation is ameliorated somewhat by stage 2 requiring only four-dimensional optimization.

6. Bayesian calibration applied to a real problem in climate science

The **calibrator** package is now applied to a problem of greater complexity, namely climate

science output from the C-GOLDSTEIN computer model. The techniques presented here are complementary to the Kalman filtering techniques of Annan, Hargreaves, Edwards, and Marsh (2005).

The problem considered here is slightly different from that presented in part 1. Here, KOH2001 is used to calibrate temperature data generated by C-GOLDSTEIN using observational data taken from the SGEN dataset.

The procedure is as follows:

1. Pick a few representative points on the Earth's surface whose temperature is of interest. This set might include, for example, the North and South Poles, and a point in Western Europe. Seven points appears to be a good compromise between coverage and acceptable runtime.

2. Choose a reasonable prior distribution for the 16 adjustable C-GOLDSTEIN parameters, using expert judgement

3. Generate an ensemble of calibration points in parameter space by sampling from the prior PDF. For each member of this ensemble, run C-GOLDSTEIN at that point in parameter space and record the predicted temperature at each of the seven representative points on Earth's surface. For this application, a calibration ensemble of about 100 code runs appears to represent a reasonable compromise between coverage and acceptable runtime.

4. Determine the hyperparameters for the dataset exactly as for the toy problem above

5. From the prior PDF, sample an ensemble of say 10000 points and, using the hyperparameters estimated in step 4 above, use the emulator package to estimate the European temperature conditional on the field observations $\mathbf{z}$ and code runs $\mathbf{y}$.

6. From equation 8, estimate the probability of each of the 10000 points in parameter space

7. Construct a weighted CDF for the temperature predictions.

Such a procedure will specify a CDF that incorporates observed temperature data from the NCEP dataset. In essence, parameter sets that give predicted temperature distributions closely matching observed data are assigned a high probability; if the match is poor, a low probability is assigned.

Although it might be possible to maximize the posterior PDF for parameter space (and thus determine a maximum likelihood estimate for the "true" parameters), such an approach would preclude techniques that require averaging over parameter space.

6.1. Results

The results section splits into two: first, numerical estimates of the scales and other hyperparameters, and second, results conditional on those hyperparameters.

Estimation of the hyperparameters

Optimization is always difficult in high dimensional spaces. Here, location of the true global minimum is an acknowledged difficulty; the emphasis in this paper is on the considerably

easier problem of locating points in hyperparameter space that are significantly better than the starting point of the optimization routine. It is also the case that the true global minimum, even if it could be found, is a statistic of the observations in the sense that it is a partition of the sample space—and thus would be different from trial to trial (if repeated sampling were admitted).

Use of the simulated annealing process helps to reduce the undesirable possibility of being "trapped" in a local minimum, but does not eliminate it. For the present, it is suggested that the difficulties of multidimensional optimization are conceptually distinct from the main thrust of this paper (and are far better dealt with in the specialist optimization literature).

In any case, as shown in the toy example section above, using incorrect scales is unlikely to lead to serious mispredictions.

The values for the hyperparameters used in the remainder of this paper were the result of a weekend's optimization time on a dual 2GHz P5.

Modification of the prior

Results are now presented which show distributions for temperatures in Northern Europe, conditional on the observed values of the observations and optimized values of the hyperparameters. For the purposes of this paper, "temperature in Northern Europe" means annual average temperatures, as predicted by C-GOLDSTEIN, at latitude 60°N, longitude 0°E. Figure 2 shows how the distribution function changes on incorporation of observed temperatures.

Note how the median temperature, for example, falls by about one degree centigrade when the observational dataset is included.

6.2. Conclusions from climate problem analysis

The primary conclusion from the above analysis is that it is *possible* to apply the methods of KOH2001 to a real problem in climate dynamics, in a computationally efficient manner.

This is, as far as I am aware, the first time that a posterior PDF has been rigorously derived for the parameter space of a climate model.

The fact that the estimated PDF changes significantly on incorporation of observational data suggests that the prior is uninformative. Such conclusions can be useful in climate science, by suggesting where new observational data can be most effective.

7. Conclusions

Viewing a deterministic computer code as a Gaussian process with unknown coefficients and applying Bayesian analysis to estimate them appears to be a useful and powerful technique. It furnishes a fast, statistically rigorous approximation to the output of any computer program.

This paper has presented **emulator**, an R package that interprets computer code output as a Gaussian process, and uses the ideas of Oakley and O'Hagan (2002) to construct an emulator: that is, a statistical approximation to the actual code.

The package is applied successfully to a toy dataset, and a problem taken from climate science in which globally averaged surface air temperature (SAT) is predicted as a function of 16 unknown input parameters.

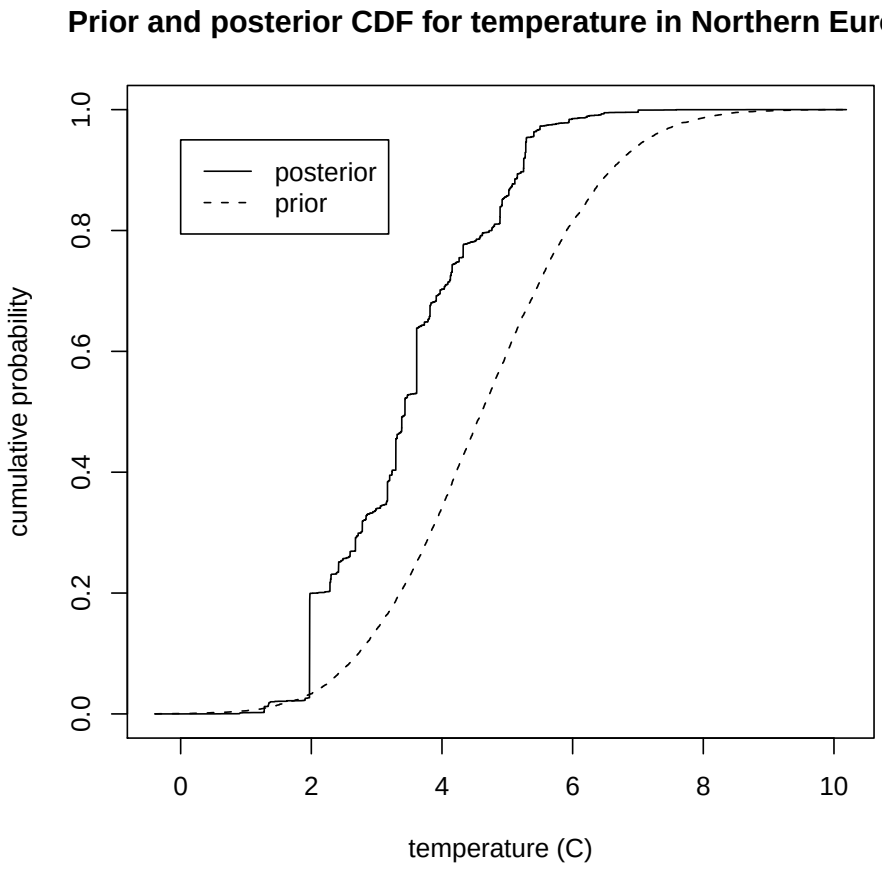

Figure 2: CDF for temperatures in Northern Europe: prior (dotted) and posterior (solid)

Average SAT is predicted well by the emulator, as this output variable is well-described by the Gaussian process model.

This paper has shown how Bayesian methods may be used interpret climate model output in a statistically rigorous way. The **BACCO** package, incorporating packages **emulator**, **approximator**, and **calibrator**, implements the formulæ of Oakley and O'Hagan (2002) and Kennedy and O'Hagan (2001a) respectively in a transparent and maintainable manner.

Both packages were demonstrated using the built in "toy" dataset, and a dataset taken from climate science.

The **emulator** package produced an approximation to C-GOLDSTEIN output that ran five orders of magnitude faster, and was accurate to within about 0.1°C.

The **calibrator** package applied formal Bayesian methods to show that predictions for temperature over Northern Europe were about 1°C cooler when observational data is taken into account.

References

Annan JD, Hargreaves JC, Edwards NR, Marsh R (2005). "Parameter estimation in an intermediate complexity Earth system model using an ensemble Kalman filter." *Ocean Modelling*, **8**(1-2), 135–154.

Currin C, Mitchell T, Morris M, Ylvisaker D (1991). "Bayesian prediction of deterministic functions, with application to the design and analysis of computer experiments." *Journal of the American Statistical Association*, **86**(416), 953–963.

Edwards NR, Marsh R (2005). "Uncertainties due to transport-parameter sensitivity in an efficient 3D ocean-climate model." *Climate Dynamics*, **24**, 415–433.

Gordon C, *et al.* (2000). "The simulation of SST, sea ice extents, and ocean heat transports in a version of the Hadley Centre coupled model without flux adjustments." *Climate Dynamics*, **16**, 147–168.

Hankin RKS (2005). "Introducing **BACCO**, an R bundle for Bayesian analysis of computer code output." *Journal of Statistical Software*, **14**(16).

Hothorn T, Bretz F, Genz A (2001). "On multivariate t and Gauss probabilities in R." *R News*, **1**(2), 27–29. URL http://CRAN.R-project.org/doc/Rnews/.

Kennedy MC, O'Hagan A (2000). "Predicting the output from a complex computer code when fast approximations are available." *Biometrika*, **87**(1), 1–13.

Kennedy MC, O'Hagan A (2001a). "Bayesian calibration of computer models." *Journal of the Royal Statistical Society, Series B*, **63**(3), 425–464.

Kennedy MC, O'Hagan A (2001b). "Supplementary details on Bayesian calibration of computer models." Internal Report. URL http://www.shef.ac.uk/~st1ao/ps/calsup.ps.

Marsh R, Edwards NR, Shepherd JG (2002). *Development of a fast climate model (C-GOLDSTEIN) for Earth system science.* National Oceanography Centre, Southampton, European Way, Southampton SO14 3ZH.

Oakley J (2). *Bayesian uncertainty analysis for complex computer codes*. Ph.D. thesis, University of Sheffield.

Oakley J, O'Hagan A (2002). "Bayesian inference for the uncertainty distribution of computer model outputs." *Biometrika*, **89**(4), 769–784.

Stainforth DA (2005). "Uncertainty in predictions of the climate response to rising levels of greenhouse gases." *Nature*, **433**(27), 403–406.

Vellinga M, Wood RA (2002). "Global climatic impacts of a collapse of the Atlantic thermohaline circulation." *Climatic Change*, **54**(3), 251–267.

Wigley TML, Raper SCB, Hulme M, Smith SJ (2000). *The MAGICC/SCENGEN climate scenario generator: Version 2.4 technical manual*.

Affiliation:

Robin K. S. Hankin
University of Cambridge
19 Silver Street
Cambridge CB3 9EP
United Kingdom
E-mail: hankin.robin@gmail.com