Creating a simple `emulator` case study from scratch: a cookbook

Robin K. S. Hankin

October 4, 2007

1 Introduction

Package `emulator` of bundle `BACCO` performs Bayesian emulation of computer models. This document constructs a minimal working example of a simple problem, step by step. Datasets and functions have a `.vig` suffix, representing "vignette".

This document is not a substitute for KOH or KOHa or Hankin 2005 or the online help files in `BACCO`. It is not intended to stand alone: for example, the notation used here is that of KOH, and the user is expected to consult the online help in the BACCO package when appropriate.

This document is primarily didactic, although it is informal.

Nevertheless, many of the points raised here are duplicated in the `BACCO` helpfiles.

The author would be delighted to know of any improvements or suggestions. Email me at `r.hankin@noc.soton.ac.uk`.

2 List of objects that the user needs to supply

The user needs to supply three objects:

- A design matrix, here `val.vig` (rows of this show where the code has been evaluated)
- Basis functions. Here `basis.vig`. This shows the basis functions used for fitting the prior
- Data, here `z.vig`. This shows the data obtained from evaluating the various levels of code at the points given by the design matrix and the subsets object.

Each of these is discussed in a separate subsection below.

But the first thing we need to do is install the library:

```
> library(emulator)
```

2.1 Design matrix: USER TO SUPPLY

In these sections I show the objects that the user needs to supply, under a heading like the one above. In the case of the `emulator` we need a design matrix and a vector of outputs.

The first thing needed is the design matrix `val.vig`, ie the points in parameter space at which the lowest-level code is executed. The example here has just two parameters, `a` and `b`:

```
> head(val.vig)
```

```
           [,1]       [,2]
[1,] 0.21666667 0.15000000
[2,] 0.91666667 0.01666667
[3,] 0.08333333 0.41666667
[4,] 0.18333333 0.58333333
```

```
[5,] 0.01666667 0.11666667
[6,] 0.81666667 0.28333333
```

```
> nrow(val.vig)
```

```
[1] 30
```

Notes

- Each row is a point in parameter space, here two dimensional.
- The parameters are labelled `a` and `b`

2.2 Basis functions: USER TO SUPPLY

Now we need to choose a basis function. Do this by copying `basis.toy()` but fiddling with it:

```
> basis.vig <- function(x) {
+     out <- c(1, x, x[1] * x[2])
+     names(out) <- c("const", LETTERS[1:2], "interaction")
+     return(out)
+ }
```

Notes

- This is shamelessly ripped off from `basis.toy()`, except that I've changed the basis to be `c(1,a,b,ab)`.

2.3 Data: USER TO SUPPLY

The data we have for the `.vig` example is a vector whose elements are the output of the code at the points specified in `val.vig`:

```
> head(z.vig)
```

```
[1] 2.252205 2.530358 2.560680 3.468957 1.778932 3.810707
```

```
> summary(z.vig)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.779   2.999   3.923   4.254   5.344   7.972
```

3 Data analysis

The previous section showed what data and functions the user needs to supply. These all have a `.vig` suffix. This section shows the data being analyzed.

First we will estimate the scales to use:

```
> os <- optimal.scales(val = val.vig, scales.start = c(10, 10),
+     d = z.vig, func = basis.vig)
> os
```

```
[1] 3.124618 5.576694
```

So we can estimate the coefficients. But first we have to calculate the variance matrix and invert it:

```
> A.os <- corr.matrix(xold = val.vig, scales = REAL.SCALES)
> Ainv.os <- solve(A)
```

Given this, use `betahat.fun()` to get the coeffs:

```
> betahat.fun(xold = val.vig, d = z.vig, Ainv = solve(A), func = basis.vig)

      const           A           B interaction
   1.114356    1.716853    2.877479    3.775184
```

The central function is interpolant:

```
> interpolant(x = c(0.5, 0.5), d = z.vig, Ainv = Ainv.os, scales = os,
+     xold = val.vig, func = basis.vig, give.full.list = TRUE)

$betahat
      const           A           B interaction
   1.114356    1.716853    2.877479    3.775184

$prior
         [,1]
[1,] 4.355318

$beta.var
                 const           A           B interaction
const        0.1928111 -0.1546351 -0.1912374   0.1608531
A           -0.1546351  0.3214103  0.1623138  -0.3465070
B           -0.1912374  0.1623138  0.3757169  -0.3299509
interaction  0.1608531 -0.3465070 -0.3299509   0.7051406

$beta.marginal.sd
      const           A           B interaction
  0.4391027   0.5669306   0.6129575   0.8397265

$sigmahat.square
[1] 0.2965090

$mstar.star
         [,1]
[1,] 4.128887

$cstar
[1] -0.003352397

$cstar.star
[1] -0.003299431

$Z
[1] 0.03127797
```

And that's it, really.

A Data generation

The data used in this study were created by directly sampling from the appropriate multivariate Gaussian:

```
> REAL.BETA <- 1:4
> REAL.SCALES <- c(3, 6)
> REAL.SIGMASQUARED <- 0.3
> A <- corr.matrix(xold = val.vig, scales = REAL.SCALES)
> z.vig <- as.vector(rmvnorm(n = 1, mean = crossprod(REAL.BETA,
+     apply(val.vig, 1, basis.vig)), sigma = A * REAL.SIGMASQUARED))
```

Robin K. S. Hankin
National Oceanography Centre, Southampton
European Way
Southampton SO14 3ZH
United Kingdom
E-mail: r.hankin@noc.soton.ac.uk URL: http://www.noc.soton.ac.uk