

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

Robin K. S. Hankin

May 10, 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,] 0.4166667 0.3833333
[2,] 0.2500000 0.5166667
[3,] 0.5166667 0.8166667
[4,] 0.2833333 0.7500000
```

```
[5,] 0.3166667 0.8833333
[6,] 0.1166667 0.4500000
```

```
> 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] 3.588634 3.892817 5.514857 4.642174 5.380999 3.463987
```

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

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.623	3.079	4.141	4.257	5.224	8.804

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] 4.693994 6.081641
```

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	
1.175035	1.489734	2.765415	4.413821

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
1.175035  1.489734  2.765415  4.413821
```

```
$prior
      [,1]
[1,] 4.406064
```

```
$beta.var
      const      A      B interaction
const  0.3439542 -0.3133214 -0.3686314  0.3697488
A      -0.3133214  0.6465902  0.3742094 -0.7771240
B      -0.3686314  0.3742094  0.7778579 -0.8097566
interaction 0.3697488 -0.7771240 -0.8097566  1.6364453
```

```
$beta.marginal.sd
      const      A      B interaction
0.5864761  0.8041083  0.8819625  1.2792362
```

```
$sigmahat.square
[1] 0.4655164
```

```
$mstar.star
      [,1]
[1,] 3.555337
```

```
$cstar
[1] -0.1841977
```

```
$cstar.star
[1] -0.1781809
```

```
$Z
[1] 0.2880037
```

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>