Creating a simple emulator case study from scratch:

a cookbook

Robin K. S. Hankin
Auckland University of Technology

Abstract

This document constructs a minimal working example of a simple application of the emulator package, step by step. Datasets and functions have a .vig suffix, representing “vignette”.

Keywords: emulator, BACCO, R.

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 Kennedy and O’Hagan (2001a) or Kennedy and O’Hagan (2001b) 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 Kennedy and O’Hagan (2001a,b), 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 hankin.robin@gmail.com.

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:
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, i.e., the points in parameter space at which the lowest-level code is executed. The example here has just two parameters, a and b:

```r
> head(val.vig)

     [,1]   [,2]
 [1,] 0.91666667 0.31666667
 [2,] 0.01666667 0.51666667
 [3,] 0.75000000 0.01666667
 [4,] 0.85000000 0.85000000
 [5,] 0.08333333 0.61666667
 [6,] 0.35000000 0.41666667

> 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:

```r
> 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).
- in the function, out is a vector of length four: c(1,x[1],x[2], x[1]*x[2]).
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:

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

```r
> summary(z.vig)
  Min. 1st Qu.  Median   Mean 3rd Qu.   Max.  
```

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:

```r
> os <- optimal.scales(val=val.vig, scales.start=c(10,10), d=z.vig, func=basis.vig)
> os
[1] 2.773261 4.874626
```

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

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

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

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

     const     A     B interaction
  1.492383 1.432296 1.757222     4.054100
```

The central function is interpolant:

```r
> 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)
```
An emulator cookbook

$\textbf{betahat} \begin{array}{ccc} \text{const} & A & B \text{ interaction} \\ 1.492383 & 1.432296 & 1.757222 & 4.054100 \end{array}$

$\textbf{prior}$

\[[,1] \begin{array}{c} 4.100667 \end{array} \]

$\textbf{beta.var} \begin{array}{cccc} \text{const} & A & B \text{ interaction} \\ \text{const} & 0.2516350 & -0.2443716 & -0.2709754 & 0.2827198 \\ A & -0.2443716 & 0.4237241 & 0.2821975 & -0.4870871 \\ B & -0.2709754 & 0.2821975 & 0.4733052 & -0.4859880 \\ \text{interaction} & 0.2827198 & -0.4870871 & -0.4859880 & 0.8571268 \end{array}$

$\textbf{beta.marginal.sd} \begin{array}{ccc} \text{const} & A & B \text{ interaction} \\ 0.5016324 & 0.6509410 & 0.6879718 & 0.9258114 \end{array}$

$\textbf{sigmahat.square}$

\[[1] 0.2622381 \]

$\textbf{mstar.star} \begin{array}{c} [,1] \end{array} \begin{array}{c} 5.188474 \end{array}$

$\textbf{cstar}$

\[[1] -0.02097982 \]

$\textbf{cstar.star}$

\[[1] -0.01620184 \]

$Z$

\[[1] 0.06518235 \]

And that’s it, really.

References


A. Data generation

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

```r
> 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))
```

Affiliation:

Robin K. S. Hankin
Auckland University of Technology
Wakefield Street, Auckland, NZ
E-mail: hankin.robin@gmail.com