

# 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](mailto: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`, 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,] 0.9500000 0.8833333
[2,] 0.3166667 0.9833333
[3,] 0.6166667 0.7166667
[4,] 0.6833333 0.3500000
[5,] 0.2500000 0.6500000
[6,] 0.2166667 0.1166667

> 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)`.
- 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`:

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

```
[1] 8.793761 6.099365 6.332542 4.559134 4.944451 2.019873
```

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

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.437	3.413	4.927	4.852	6.229	8.794

## 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] 4.155074 5.013402
```

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
0.9489178	2.7917267	3.9534481	2.2184068

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
0.9489178  2.7917267  3.9534481  2.2184068

$prior
      [,1]
[1,] 4.876107

$beta.var
      const      A      B interaction
const  0.1935295 -0.1633329 -0.2031802  0.1810364
A      -0.1633329  0.4397058  0.1837770 -0.5266948
B      -0.2031802  0.1837770  0.4205316 -0.3753206
interaction 0.1810364 -0.5266948 -0.3753206  1.0116906

$beta.marginal.sd
      const      A      B interaction
0.4399199  0.6631032  0.6484841  1.0058283

$sigmahat.square
[1] 0.3108555

$mstar.star
      [,1]
[1,] 4.745272

$cstar
[1] -0.09397795

$cstar.star
[1] -0.09312776

$Z
[1] 0.1701449

```

And that's it, really.

## References

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

## 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.
```

### Affiliation:

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