# Constructing the emulators

## Background: the structure of an emulator

An [emulator](https://en.wikipedia.org/wiki/Emulator) is a way of representing our beliefs about the behaviour of an unknown function. In our case, where the model is stochastic, the unknown function is taken to be the mean of each of the model outputs over multiple runs. Given a set of model runs and an estimate of the parameters, we use the emulator to get a probability distribution for the mean of a model's output at any input parameter set, without the need to run the model at the chosen input.

In this tutorial, we will construct an emulator for each of the model outputs separately. The general structure of a univariate emulator is as follows:

$$f(x) = g(x) \beta + u(x),$$

where $g(x)$ are the regression functions, $\beta$ the regression parameters, and $u(x)$ is the correlation structure. We split the correlation structure further into two pieces - for two parameter sets $x$ and $x^{\prime}$, the structure is:

$$u(x) = \sigma^2 (1-\delta) c(x,x^{\prime}) + \delta I\_{\{x=x^\prime\}}.$$

Here $\sigma^2$ is the (prior) emulator variance and $c(x,x^{\prime})$ is a correlation function; the simplest such function is squared-exponential

$$c(x,x^{\prime}) = \exp\left(-\frac{\sum\limits\_{i}(x\_{i,A}-x\_{i,A}^{\prime})^2}{\theta^2}\right).$$

The subscript $A$ indicates that the correlation function operates only on the active inputs for the emulator: that is, inputs that contribute to the regression surface. To ensure the correlation structure is well-defined, the 'nugget' term $\delta I\_{\{x=x^{\prime}\}}$ is included: this operates on all the parameters in the input space and represents the proportion of the overall variance due to the ensemble variability. The $\theta$ hyperparamater is the correlation length for the emulator: the larger $\theta$ is, the smoother the local variations of the emulators will be. Note that one can also consider multiple correlation lengths, one for each input direction.

## Constructing the emulators step by step

To construct the emulators, three steps are required:

1) We create a set of 'initial' emulators `ems0` by fitting a regression surface to `train0` and setting $\delta$ to zero. These simple emulators will provide us with estimates for the regression surface parameters and the basic correlation specifications;

2) The ensemble variability in `train0` is compared to the variance of the 'initial' emulators, and their ratio is used as an estimate for $\delta$. We then train the 'initial' emulators again with the estimated value for $\delta$;

3) The new emulators `ems0` constitute our prior which we adjust to the data through the Bayes Linear update formulae. In this way we obtain the final version of our first wave emulators: `ems0\_adjusted`.

Let us now go through each step in detail.

### Step 1

We start by splitting `wave0` into training and validation sets, each consisting of 40 observations.

``` {r}

samples <- sample(nrow(wave0), 40)

train0 <- wave0[samples,1:9]

valid0 <- wave0[!seq\_along(wave0[,1])%in%samples,1:9]

```

The function `emulator\_from\_data` creates the initial emulators for us. We pass `emulator\_from\_data` the training data, the name of the outputs we want to emulate and the list of parameter ranges. We also specify that we want to fit a quadratic surface, rather than a hyperplane (which is the default setting). Taken this information, `emulator\_from\_data` finds both the regression parameters and the active inputs for each of the indicated outputs.

``` {r}

output\_names <- paste0("I", seq(10,30, by=5))

ems0 <- emulator\_from\_data(train0, output\_names, ranges, quadratic=TRUE)

```

### Step 2

Now that we have our simple `ems0`, we can find an approximation for the value of the deltas in our model. As recalled in the background section above, the delta parameter represents the proportion of the total variance due to the ensemble variability. The ensemble variability for a given output is just the mean of the relative column $EV$ in `wave0`. The overall variance $\sigma$ is estimated through the standard error of the relative emulator in `ems0`. Once their ratio is taken, we are ready to create a new set of emulators, which we still call `ems0`, using the estimated deltas:

``` {r}

delts <- apply(wave0[10:ncol(wave0)], 2, mean)/map\_dbl(ems0, ~.$u\_sigma)

ems0 <- emulator\_from\_data(train0, output\_names, ranges, quadratic = TRUE, deltas = delts)

ems0[[1]]

```

The print statement provides an overview of the emulator specifications and correlation structure, including

- Basis Functions (here we have the four parameters beta, gamma, delta, mu and products of them, since we chose quadratic regression),

- Active variables,

- first and second order specifications for $\beta$ and $u(x)$. Note that by default `emulator\_from\_data` assumes that $\text{Var}[\beta]=0$: the regression surface is known and coefficients are fixed. This explains why Beta Variance and Mixed Covariance (which shows the covariance of $\beta$ and $u(x)$) are both zero. One can set beta.var=TRUE in order to consider the beta parameters as random variables.

We can plot the emulators to see how they represent the output space: the `emulator\_plot` function does this for emulator expectation, variance, standard deviation, and implausibility (more on which later).

``` {r}

for (i in 1:length(ems0)) ems0[[i]]$output\_name <- output\_names[i]

names(ems0) <- output\_names

emulator\_plot(ems0)

```

The emulator expectation plots show the structure of the regression surface, which is at most quadratic in its parameters, through a 2D slice of the input space. In particular the two parameters $\beta$ and $\gamma$ are selected and the plot shows the expected value of each output for all possible pairs $(\beta,\gamma)$.

To plot the emulators standard deviation we just use `emulator\_plot` passing 'sd' as second argument:

``` {r}

emulator\_plot(ems0, 'sd')

```

Here we immediately see that the emulator variance (or equivalently, standard deviation) is simply constant across the parameter space for each output. This is not what we want though, since one would expect emulators to be less uncertain around training points, where the model was evaluated. This will be taken care of in the next step.

### Step 3

```{task}

Add 2 and 2 together

```

We now use the `adjust` method on our emulators to obtain the final Bayes Linear version of our `wave0` emulators:

```{r}

ems0\_adjusted <- map(seq\_along(ems0), ~ems0[[.]]$adjust(train0, output\_names[[.]]))

```

Note that the `adjust` method works with the data in `train0` exactly as the function `emulator\_from\_data` did: it performs Bayes Linear adjustment, given the data. This function creates a new emulator object with the adjusted expectation and variance of beta as the primitive specifications, and supplies the data for the new emulator to compute the adjusted expectation and variance of $u(x)$, and the adjusted $Cov[\beta, u(x)]$. Due to the update formulae, the correlation structure isn't stationary anymore: the value of $\sigma^2$ is now $x$-dependent.

```{r}

names(ems0\_adjusted) <- output\_names

emulator\_plot(ems0\_adjusted)

emulator\_plot(ems0\_adjusted, var = 'sd')

```

We can see that the adjusted emulators more reasonably show the structure of the model. The variance has been updated: the closer the evaluation point is to a training point, the lower the variance (as it 'knows' the value at this point). In fact, evaluating these emulators at the training points demonstrates this fact:

```{r}

em\_evals <- ems0\_adjusted$I10$get\_exp(train0[,names(ranges)])

all(abs(em\_evals - train0$I10) < 10^(-12))

all(ems0\_adjusted$I10$get\_cov(train0[,names(ranges)]) < 10^(-12))

```

Note the comparative speeds of evaluation, here. The initial $80$ parameter sets we generated from the model took around 45 seconds on a relatively powerful laptop; evaluating the emulator expectation over a $40\times40$ grid takes less than 5 seconds; evaluating the emulator variance over the same grid takes 30 seconds.

This is all fine, but we need to consider whether these emulators are actually performing as we would expect them to. For this, we need to consider emulator diagnostics.