<!--

# Constructing the emulators

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 points $x$ 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

$$c(x,x^{\prime}) = \exp\left(-\frac{(x\_A-x\_A^{\prime})^2}{\theta^2}\right).$$

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 input space. The $\theta$ hyperparamater is the correlation length for the emulator (more of which later).

To construct the emulators, a few steps are required:

- We first create a set of 'initial' emulators, providing the estimates for the regression surface parameters and the basic correlation specifications;

- The emulator variance is compared to the ensemble variability, and their ratio is used as an estimate for $\delta$;

- The emulators are then trained on the data using the Bayes Linear update formulae.

The function `emulator\_from\_data` does much of this work: given some observations, the regression parameters and active inputs are found for each output. Model selection is performed using stepwise addition or deletion (as appropriate), using the AIC criterion to find the minimal best fit. The standard error from this model is used as an estimate for $\sigma^2$.

``` {r}

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

train0 <- wave0[samples,1:9]

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

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

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

```

The `emulator\_from\_data` function will take any of the specifications as parameters, if the default behaviour does not provide an acceptable set of emulators.

Given this, we can find an estimate for the deltas and create a new set of emulators:

``` {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 at the end provides an overview of the emulator specifications, including active variables, their contribution to the regression surface, and the second-order specifications for $\beta$ and $u(x)$. We could look at the emulators' behaviour over the input space, but it will not be iluminating:

``` {r}

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

names(ems0) <- output\_names

emulator\_plot(ems0)

emulator\_plot(ems0, 'sd')

```

The emulator expectation plots just show the structure of the regression surface, which is at most quadratic in its parameters. The emulator variance (or equivalently, standard deviation) is simply constant across the parameter space for each output.

We can now train these emulators fully using the `adjust` method:

```{r}

wave1 <- map(seq\_along(ems0), ~ems0[[.]]$adjust(train0, output\_names[[.]]))

```

The corresponding print statement will not indicate in this case that anything has changed: since by default the `emulator\_from\_data` function assumes that $\text{Var}[\beta]=0$ (i.e. that the regression surface is known), the coefficients are fixed. The correlation structure has changed, but it is difficult to show how it has changed. Due to the update formulae, the correlation structure is now not stationary: the value of $\sigma^2$ is now $x$-dependent (which is not easy to demonstrate in a print statement!). 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}

names(wave1) <- output\_names

emulator\_plot(wave1)

emulator\_plot(wave1, var = 'sd')

```

We can see that the trained 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 <- wave1$I10$get\_exp(train0[,names(ranges)])

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

all(wave1$I10$get\_cov(train0[,names(ranges)]) < 10^(-12))

```

Note the comparative speeds of evaluation, here. The initial $80$ points we generated from the simulator 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.

# Emulator diagnostics

For a given set of emulators, we can consider how accurately they reflect the simulator's outputs over the space. To do so, we can consider a set of diagnostic tests. First, however, we need to set up a framework for evaluating implausibility.

For a given output, and an observed value, the implausibility is defined as the difference between the predicted output and the actual observation, taking into account all sources of uncertainty; for input values $x$ the schematic form for the implausibility $I(x)$ is

$$I(x) = \frac{|f(x)-z|}{\sqrt{\sum \sigma^2}},$$

where $f(x)$ is the predicted output, $z$ the observation, and the $sigma$ terms represent uncertainties. To use the implausibility, therefore, we need to define a set of observations at the output times. As this is a synthetic dataset, we will take as our observations the model runs from a chosen input point, with a large number of repetitions; the means will provide the observations and the standard deviations at each point will give some 'observation error'.

```{r}

targets = list(

I10 = list(val = 240, sigma = 25.27),

I15 = list(val = 396, sigma = 40.99),

I20 = list(val = 453, sigma = 46.48),

I25 = list(val = 428, sigma = 43.98),

I30 = list(val = 392, sigma = 40.30)

)

```

Then, for instance, we can find the emulator implausibility for the first output:

```{r, eval=FALSE}

emulator\_plot(wave1[[1]], 'imp', targets = targets[[1]])

```

The uncertainties that go into the denominator of the emulator implausibility are the observation uncertainties defined, and the emulator variance at the given point.

The first three diagnostics are relatively straightforward, and can be presented together. For a given validation set, we can consider the following:

- Within uncertainties, does the emulator output accurately represent the equivalent simulator output?

- What are the standard errors of the emulator outputs in light of the simulator outputs?

- Does the emulator adequately classify points as implausible or non-implausible?

These are encapsulated in the `validation\_diagnostics` function.

``` {r, eval=FALSE}

which\_invalid <- validation\_diagnostics(wave1, valid0, output\_names, targets = targets)

which\_invalid

```

The first column of plots gives an indication of the emulator outputs against the simulator outputs: the emulator outputs are plotted against the simulator outputs with a 3sd confidence interval overlaid. An 'ideal' emulator would exactly reproduce the simulated results: this behaviour is represented by the green line $f(x)=E[f(x)]$. Any points whose emulated prediction lies more than $3\sigma$ away from the simulated output is highlighted in red. The second column gives the standard errors: generally we would like the standard errors to be within $-2$ and $2$. Finally, the third column compares the emulator implausibility at the points to the equivalent simulator implausibility. There are three cases to consider:

- The emulator and simulator both classify a point as implausible/non-implausible: this is fine. Both are giving the same classification for the set of points.

- The emulator classifies a point as non-implausible, while the simulator rules it out: this is also fine. The emulator should not be expected to shrink the parameter space as much as the simulator does, at least not on a single wave. Points classified in this way will survive this wave, but may be removed on subsquent waves as the emulators grow more accurate on a reduced parameter space.

- The emulator rules out a point, but the simulator does not: these are the problem points, suggesting that the emulator is ruling out parts of the parameter space that it should not be ruling out.

The function itself, along with producing the plots, also returns a data.frame consisting of those points which failed one or more diagnostic tests. It is often worth considering these points, particularly if they lie close to the boundary of the space; having a few points which fail diagnostics is not the end of the world, but we should at least consider whether the emulator is failing in parts of the space we would want it to be performing well on.

A helper for visualising these problem points is provided in the function `validation\_pairs`: this give pairs plots of the validation points for each parameter pair, colouring the points by their diagnostic success (bottom left) and predicted implausibility (top right).

``` {r, eval=FALSE}

vp <- validation\_pairs(wave1, valid0, targets)

```

We can see that the points that are struggling with diagnostics are indeed on the boundaries of the space, particularly on the boundary of the $(\delta,\mu)$ space. Examination of the upper half of this plot also shows that a large proportion of the points are due to be ruled out as non-implausible, so they lie in parts of the parameter space that will have no impact on the overall history matching process.

One final thing that we may consider is the choice of cut-off for the implausibility measure. The implausibility is a metric for evaluating how far out from being a good fit any input point is: there is no hard-and-fast rule for deciding at what point we decide a point is too implausible. Indeed, there are two things to consider when we have multiple univariate emulators:

- What cutoff should we impose? A rough rule of thumb loosely follows Pukelsheim's $3\sigma$ rule, which states that any unimodal distribution can be treated normally, in the sense that a $5\%$ confidence interval corresponds to $3\sigma$ around the mean. This is only the case for a single such distribution; for multiple univariate emulators it is slightly more involved. However a rough starting cut-off $m$, for confidence interval $1-\alpha$ and $N$ emulators, would be

$$m = \Phi^{-1}\left(\frac{1+(1-\alpha^{1/N})}{2}\right)$$

where $\Phi^{-1}$ is the inverse of the normal distribution CDF.

- Given multiple emulators, how do we measure overall implausibility? We want a single measure for the implausibility at a given point, but for each emulator we obtain an individual value for $I$. The simplest way to combine them is to consider maximum implausibility at each point:

$$I\_M(x) = \max\_{i=1,\dots,N}I\_{i}(x),$$

where $I\_i(x)$ is the implausibility at $x$ coming from the $i$th emulator. For large collections of emulators, it may be useful to instead consider the second-, or third-maximum implausibility; where some outputs are deemed more important than others (for instance, putting greater weight on emulation of the peak of an epidemic), we may instead take a weighted average across the implausibity measures.

The default behaviour of the diagnostics and plots here is to take a cutoff of $3$, and take maximum implausibility across the outputs: where multiple outputs are concerned, emulator functions make a call to `nth\_implausible` with default $n=1$. This can be modified in any function call that uses it. For instance, the above diagnostic plot could consider minimum implausibility in its upper plots via

``` {r, eval=FALSE}

vp2 <- validation\_pairs(wave1, valid0, targets, n=length(wave1))

```

One way we can get a feel for what cut-off value is reasonable is via the `space\_removed` function, which for a given set of emulators will determine how much of the space will be removed by a particular implausibility cut-off. We can also consider what would happen if we were to modify the structural discrepancy, the emulator variances, or the correlation lengths. This evaluates over a fairly large set of points and, in the cases of varying variance or correlation lengths, has to retrain multiple sets of emulators, so does take a while to run. For the purposes of speed, here, we set `n\_points` to $5$; this creates a set of $5^d$ points to evaluate over, where $d$ is the dimensional of the space. By default, it considers varying the structural discrepancy, and does so in steps of $10\%$ around $100\%$ - to change this look at the `u\_mod` argument in the function call.

``` {r, eval=FALSE}

sp1 <- space\_removed(wave1, valid0, targets)

sp2 <- space\_removed(wave1, valid0, targets, u\_mod = c(0.75, 1, 1.25), modified = 'var', n\_points = 5)

sp3 <- space\_removed(wave1, valid0, targets, modified = 'corr', n\_points = 5)

```

A cut-off of $3$ here, using maximum implausibility, would be sufficient to remove around $75\%$ of the current parameter space. This is a reasonable level of removal for a first wave: however, if the expected amount of removal was much lower we could consider whether it is sensible to reduce the cut-off (a companion plot that shows how many diagnostic failures would result from a particular cutoff value is in the pipeline). Finally, we can see that changing the correlation lengths has a minimal impact on the resultant space reduction: this should not be surprising in this case as the linear part of the emulators (i.e. the $g(x)\beta$ term) is sufficient to capture much of the dynamics of the points:

```{r, eval=FALSE}

map\_dbl(ems0, ~summary(.$model)$adj.r.squared)

```

The adjusted $R^2$ of the linear models are all high, so the correlation structure is having to do very little 'work' for the emulators to match the data. If we instead had a model where the linear part cannot accurately represent the output surface, the choice of correlation lengths would be a much more important choice, and the final plot above would be a much stronger indicator of suitability.

In any event, the diagnostics here give an indication of the suitability of the emulators in emulating the outputs at this wave. If there are particular outputs for which the emulators do not give a good fit, then we can modify the specifications for that emulator directly (for example, modifying the correlation length, the variance, or the regression surface) and re-train; if the emulator simply cannot provide a good fit to the output, we can choose not to emulate this output for the wave in question.

# Point Generation

Having generated emulators based on the first wave of points, evaluated their suitability, and considered a means by which to rule out points, we can now produce a new set of points to pass to the simulator. The function `generate\_new\_runs` is designed for this purpose; its default behaviour is as follows.

- If no prior points are provided, a set of points is generated using a Latin Hypercube Design, rejecting implausible points;

- Using these as seeding points, more points are generated using importance sampling to attempt to fully cover the non-implausible region;

- Finally, pairs of points are selected at random and more points are sampling from lines connecting these points, with particular importance given to those that are close to the non-implausible boundary.

All of these steps can be overridden or modified, but the default behaviour allows for a good rudimentary search of the non-implausible space.

```{r}

new\_points <- generate\_new\_runs(wave1, ranges, n\_points = 120, z = targets)

plot(new\_points, pch = 16, cex = 0.5)

```

We can start to see the structure of the non-implausible region, here. The `wave\_points` function provides a better indication of the difference between the two sets of wave points

``` {r}

wave\_points(list(wave0, new\_points), in\_names = names(ranges))

```

Finally, we can put these new points into the simulator and obtain the outputs:

```{r}

next\_wave <- getOutputs(new\_points, seq(10,30,by=5))

```

Finally, we can see how much better the 2nd wave points perform compared to the original points using `simulator\_plot`.

``` {r}

w1points <- data.frame(cbind(new\_points,next\_wave)) %>% setNames(c(names(ranges), paste0("I",seq(10,30,by=5)), paste0("EV",seq(10,30,by=5))))

all\_points <- list(wave0[1:9], w1points[1:9])

simulator\_plot(all\_points, targets)

```

We can see that, compared to the space-filling random points used to train the first emulators, the new points are in much closer agreement with the data we wish to match to. Subsequent waves, trained on these new points, will be more confident in the new non-implausible region and will therefore refine the region in light of the greater certainty.

# Further waves

We would follows the same procedure for subsequent waves, with a couple of caveats. We follow the same procedure for training a new set of emulators:

``` {r}

sampling <- sample(nrow(w1points), 40)

train1 <- w1points[sampling,1:9]

valid1 <- w1points[!seq\_along(w1points[,1])%in%sampling,1:9]

new\_ranges <- map(names(ranges), ~c(min(w1points[,.]), max(w1points[,.]))) %>% setNames(names(ranges))

ems1 <- emulator\_from\_data(train1, output\_names, new\_ranges, quadratic = T)

deltas <- apply(w1points[,10:14], 2, mean)/map\_dbl(ems1, ~.$u\_sigma)

ems1 <- emulator\_from\_data(train1, output\_names, new\_ranges, deltas = deltas, quadratic = TRUE)

for (i in 1:length(ems1)) ems1[[i]]$output\_name <- output\_names[i]

wave2 <- map(seq\_along(ems1), ~ems1[[.]]$adjust(train1, output\_names[[.]]))

names(wave2) <- output\_names

```

We would apply diagnostics to this as before, using `valid1` as the validation set. Assuming that the diagnostics are acceptable, we could then proceed to consider implausibility - however, we need the implausibility over the whole input space, and the new emulators have only been trained on a subset thereof. We must therefore consider implausibility across all waves, rather than just the wave under consideration at the time.

``` {r}

all\_waves <- c(wave1, wave2)

all\_targets <- c(targets, targets)

emulator\_plot(all\_waves, var = 'maximp', targets = all\_targets)

```

This may seem an unwieldy way to approach this (and it is, at present); however, it is important to remember that the number of emulators at each wave may not be the same; for example, if we have had to remove an output at wave 1, then the targets would be accordingly changed. In this illustration case, we did not have to worry about doing so since we have assumed that all targets can be emulated.

The remainder of the analysis proceeds much as in the first wave. In generating new points, we would of course provide `all\_waves` to the point generation function.

Finally, a basic wave generation function is provided to avoid stepping through the sets presented here: the function `full\_wave` will take training data, validation data, and a desired number of points and generate the next set of points (for details, as always, try `?full\_wave`).

``` {r}

test\_full\_wave <- full\_wave(train0, valid0, ranges, output\_names, targets, 120, sample\_method = 'importance')

wave\_points(list(new\_points, test\_full\_wave$next\_sample), names(ranges))

```

This returns four objects: the trained emulators, the base emulators used for calibration, the new parameter ranges, and the new sample points to pass to the simulator. We can see that the performance is comparable to the step-through of the functions; however, we have less control over expert judgement of suitability of emulators, cutoff values, and other considerations.

-->