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)

emulator\_plot(ems0, 'sd')

```

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. The emulator variance (or equivalently, standard deviation) is simply constant across the parameter space for each output. We now use the `adjust` method on our emulators to obtained 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.

# Emulator diagnostics

For a given set of emulators, we want to assess how accurately they reflect the model's outputs over the space. To do so, we can consider a number 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 our dataset is synthetic, we will take as our observations the model runs from a chosen parameter set, with a large number of repetitions; the means will provide the observations and the standard deviations at each parameter set will give the 'ensemble variability' representing the stochastic nature of the model.

```{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)

)

```

The uncertainties that go into the denominator of the emulator implausibility are the ensemble variability defined, and the emulator variance at the given parameter set. Note that if our data were not synthetic, we would also include the observation uncertainty, to represent the finite accuracy with which real data is observed and the model discrepancy, accounting for the fact that no model perfectly represents reality.

An important thing to 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 parameter set is: there is no hard-and-fast rule for deciding at what point a parameter set is too implausible. Indeed, there are two things to consider when we have multiple univariate emulators.

First of all: what cut-off 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.

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

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

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 we will see here is to take a cut-off of $3$ (following Pukelsheim's $3\sigma$ rule), and take maximum implausibility across the outputs. For instance, to find the emulator implausibility for the first output we use the `emulator\_plot` function specifying 'imp' for implausibility and passing it the target for the first output:

```{r}

emulator\_plot(ems0\_adjusted[[1]], 'imp', targets = targets[[1]])

```

This is a 2D slice through the input space: chosen a pair $(\bar\beta,\bar\gamma)$, the plot shows the implausibility of the input parameter set $(\bar\beta, \bar\gamma, \delta\_M, \mu\_M)$, where $\delta\_M$ denotes the mid-range value of the delta parameter and similarly for $\mu\_M$. Parameter sets with a high implausibility (orange region) are highly unlikely to give a good fit and will be discarded when forming the parameters sets for the next wave.

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 model output?

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

- Does the emulator adequately classify parameter sets as implausible or non-implausible?

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

``` {r}

which\_invalid <- validation\_diagnostics(ems0\_adjusted, valid0, output\_names, targets = targets)

```

The first column of plots gives an indication of the emulator outputs against the model outputs: the emulator outputs are plotted against the model outputs with a $3\sigma$ confidence interval overlaid. An 'ideal' emulator would exactly reproduce the model results: this behaviour is represented by the green line $f(x)=E[f(x)]$. Any parameter set 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 parameter sets to the equivalent model implausibility. There are three cases to consider:

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

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

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

The function `validation\_diagnostic`, along with producing the plots, also returns a data.frame consisting of those parameters sets which failed one or more diagnostic tests.

``` {r}

which\_invalid

```

It is often worth considering these points, particularly if they lie close to the boundary of the space: having a few parameter sets 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 parameter sets is provided in the function `validation\_pairs`: this gives pairs plots of the validation points for each parameter pair, colouring the points by their diagnostic success (bottom left) and predicted implausibility (top right). The diagnostics part gives the maximum standard error at each point: the standard error is $$\frac{|\text{emulator value}-\text{model value}|}{\sqrt{\text{emulator variance}}}$$ for each output and we maximise over the outputs.

``` {r}

vp <- validation\_pairs(ems0\_adjusted, 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 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.

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 will consider minimum implausibility in its upper plots if we set $n$ to be the number of emulators:

``` {r}

vp2 <- validation\_pairs(ems0\_adjusted, valid0, targets, n=length(ems0))

```

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. By default, `space\_removed` considers varying the structural discrepancy, and does so in steps of $10\%$ around $100\%$. Being our data synthetic, the `space\_removed` function will vary the ensemble variability, which we can consider as a form of internal discrepancy.

``` {r}

sp1 <- space\_removed(ems0\_adjusted, valid0, targets)

```

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).

We can also consider what would happen if we were to modify the emulator variances or the correlation lengths. To do this we set `modified ='var'` or `modified='corr'`. This evaluates over a fairly large set of parameter sets and 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 dimension of the input space. As before, the default behaviour considers steps of $10\%$ around $100\%$. Here we set the `u\_mod` argument in the function call to choose different steps for the variance.

``` {r}

sp2 <- space\_removed(ems0\_adjusted, valid0, targets, u\_mod = c(0.75, 1, 1.25), modified = 'var', n\_points = 5)

sp3 <- space\_removed(ems0\_adjusted, valid0, targets, modified = 'corr', n\_points = 5)

```

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}

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 data, evaluated their suitability, and considered a means by which to rule out parameter sets, we can now produce a new set of parameter sets to pass to the model. The function `generate\_new\_runs` is designed for this purpose; its default behaviour is as follows.

- If no prior parameter sets are provided, a set is generated using a [Latin Hypercube Design](https://en.wikipedia.org/wiki/Latin\_hypercube\_sampling), rejecting implausible points;

- Using these as seeding points, more parameter sets are generated using [importance sampling](https://en.wikipedia.org/wiki/Importance\_sampling) to attempt to fully cover the non-implausible region;

- Finally, pairs of parameter sets are selected at random and more sets are sampled from lines connecting them, 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(ems0\_adjusted, 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 data.

``` {r}

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

```

Here `wave0` points are in yellow and `new\_points` are in purple. The plots in the main diagonal show the distribution of points in `wave0` and that of `new\_points`.

Now we can put `new\_points` into the model and obtain the outputs:

```{r}

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

```

Binding together `new\_points` and `next\_wave` we obtain `wave1`, the full data for the next wave. We can see how much better the `wave1` parameter sets perform compared to the original `wave0` parameter sets using `simulator\_plot`.

``` {r}

wave1 <- 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], wave1[1:9])

simulator\_plot(all\_points, targets)

```

We can see that, compared to the space-filling random parameter sets used to train the first emulators, the new parameter sets are in much closer agreement with the data we wish to match to. Subsequent waves, trained on these new parameter sets, 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 follow the same procedure for subsequent waves, with a couple of caveats.

First of all we train a new set of emulators, in the same way we did for `ems0`:

``` {r}

sampling <- sample(nrow(wave1), 40)

train1 <- wave1[sampling,1:9]

valid1 <- wave1[!seq\_along(wave1[,1])%in%sampling,1:9]

new\_ranges <- map(names(ranges), ~c(min(wave1[,.]), max(wave1[,.]))) %>% setNames(names(ranges))

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

deltas <- apply(wave1[,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]

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

names(ems1\_adjusted) <- output\_names

```

We can apply diagnostics to this as before, using `valid1` as the validation set. Assuming the diagnostics are acceptable, we 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(ems0\_adjusted, ems1\_adjusted)

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 parameter sets, 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 parameter sets and generate the next set of parameter sets (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 parameter sets to pass to the model. 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, cut-off values, and other considerations.