Building Emulators

Andrew Parnell and Philip Cardiff
andrew.parnell@mu.ie

Maynooth University
National University of Ireland Maynooth

https://andrewcparnell.github.io/intro_emulators/

Introduction

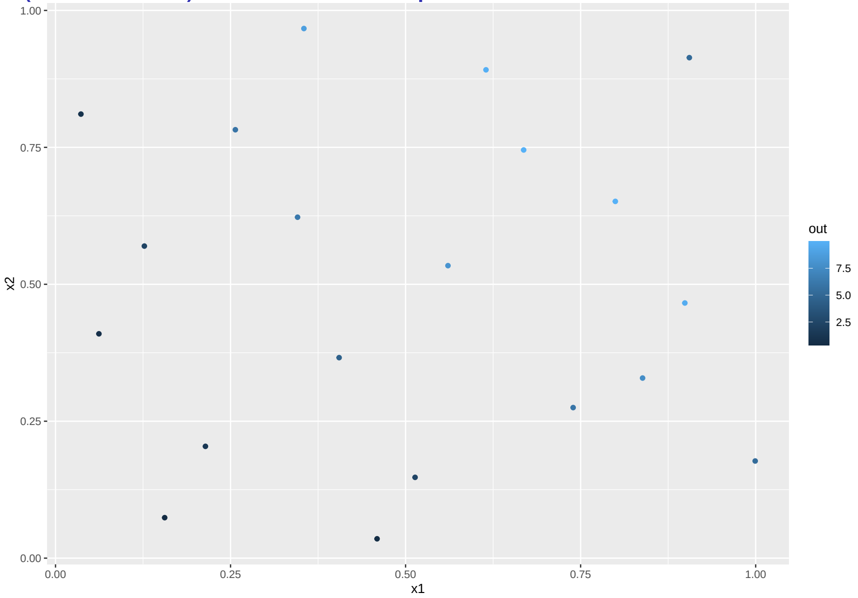
- We have now chosen our best input values to run the simulator at
- We now need to run the simulator. This may take a while. . .
- . . . but then we can build our emulator!
- We will use a Gaussian Process to build the emulator

First step: run the simulator

- Assume we have chosen our input points using a suitable design and that we have chosen our design points
- We now run our simulator for sine wave example with:

```r
n_runs <- 20
n_inputs <- 2
initial_grid <- maximinLHS(n_runs, n_inputs)
# Set up container and run
out <- rep(NA, length = n_runs)
for (i in 1:n_runs) {
  out[i] <- f(initial_grid[i, 1], initial_grid[i, 2])
}
```

Example 1 (sine wave) simulator output



Example 2 (2D Navier-Stokes) simulator run

```r
for (i in 1:n_runs) {
  system("./dockerClean2.sh")
  curr_answer <- system(paste("./dockerRun.sh",
                              df_grid$largeInletVelocityMagnitude[i],
                              df_grid$smallInletVelocityMagnitude[i],
                              df_grid$kinematicViscosity[i]),
                        intern = TRUE)
  df_grid$out[i] <- as.numeric(curr_answer[length(curr_answer) - 1])
  print(df_grid$out[i])
}
```

Fitting the emulator

- We now have a set of outputs (y) and a set of inputs (X)
- We assume as before that y is univariate and X is multivariate
- This is now a standard machine learning problem! (Go and see the Rfternoon series of introductory R lectures course if you want to solve machine learning problems)
- ...but mostly these will not work!
- This is because the simulator is deterministic - we want our emulator to reproduce $f(X) = y$ when we choose values of X that we have already run
- Most machine learning algorithms do not do this, except for Gaussian Processes...

What is a Gaussian Process?

- A Gaussian Process is just a fancy version of linear regression where we also model the correlation between the response values using a **multivariate normal distribution**
- This is the model:

$$y|x \sim MVN(\mu 1, \Sigma);\ \Sigma_{ij} = \tau^2 \rho_\phi(\|x_i - x_j\|)$$

- There are 3 parameters: μ, ϕ, τ
- The parameter τ^2 controls the variance of the Gaussian process (bigger values have more uncertainty)
- The parameter ϕ controls how smooth the curve is
- The function ρ controls the correlation function; how the correlation between the y values decreases as x gets further away

Gaussian process predictions

- The GP has a wonderful formula for predicting at new input values x^*:

$$\hat{y}^*|y \sim MVN(\mu 1 + \Sigma^* \Sigma^{-1}(y - \mu 1), \Sigma^{**} - \Sigma^* \Sigma^{-1}(\Sigma^*)^T)$$

- Here Σ^* and Σ^{**} are the covariance matrix of the prediction points and their variance respectively
- If you are predicting for a known point then $\Sigma^{**} = \Sigma^* = \Sigma$ and the prediction is just y. This means it will go through the points you have already run in your simulator!
- The other advantage is that you get predicted means and uncertainties from every point you choose

Why Gaussian processes?

- There is nothing special about the fact that it's a Gaussian Process used here. Any statistical model that can produce predictions with quantified uncertainties should be OK, provided it produces zero variance predictions at given input values
- Another nice thing about GPs is that they are smooth which often matches the differential equations underlying the simulator
- The bad thing about them is that they can be slow to fit - we need to solve the covariance matrix when optimising the parameters and creating predictions

The `GPFit` package

- We will use the `GPfit` package to create our Gaussian Process predictions because it's very simple
- Go back and see part 1 for richer alternatives
- One downside to using `GPfit` is that it requires all inputs to be scaled between 0 and 1. This just requires an extra few lines of coding

Fitting

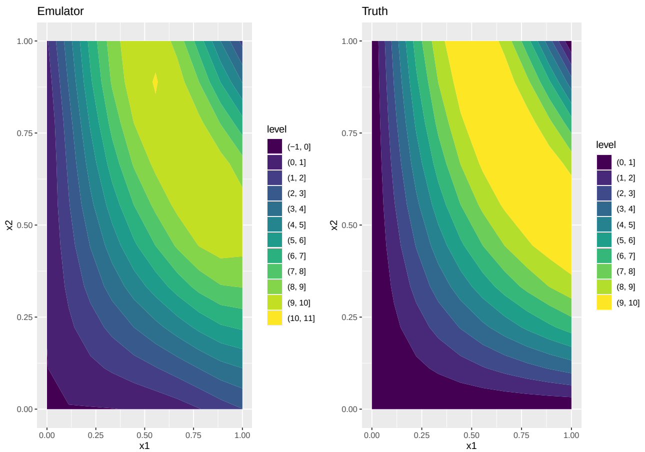
Fitting the model is just one line!

```
emulator <- GP_fit(initial_grid, out)
```

For a 2 input example we can plot the emulator, but we need to create our own plots after that

Predicting new values for example 1

```r
x_new <- seq(0, 1, length = 10) # Beware - we will create 10^6 new values!
X_new <- expand.grid(x_new,
                     x_new)
pred <- predict(emulator, X_new)
```



Emulating via other machine learning models

- You don't have to use a GP to fit your emulator, you could use your favourite machine learning algorithm
- But remember that it will likely not give the true simulator values at the same input
- Using a different machine learning approach will work well if (a) you have a stochastic simulator, or (b) you can perform a very large number of simulation runs

Summary

- We have now fitted a Gaussian Process emulator to our example data
- Gaussian processes are just a clever machine learning tool that are both smooth and exhibit zero variance at the input values
- Using the `GPfit` package it is only one line to fit an emulator
- Can then go and predict for any new values of the simulator more or less instantly
- Next: plotting, checking, and calibrating the emulator...