

# An introduction to stochastic emulation

Andrew Parnell  
andrew.parnell@mu.ie



[https://github.com/andrewcparnell/intro\\_emulators](https://github.com/andrewcparnell/intro_emulators)

# Outline

- ▶ What are emulators?
- ▶ Some jargon and notation
- ▶ A simple example
- ▶ A reminder on Gaussian Processes
- ▶ A more complex example: emulating an SEIR model
- ▶ Other complications

## What are emulators?

- ▶ Suppose we are in a situation where we have a complicated system from which we can simulate
- ▶ The system will have input and output variables
- ▶ Suppose that the simulations are very slow
- ▶ An emulator is a statistical short-cut to the system which produces estimated values of the outputs for a given set of inputs
- ▶ Examples include: climate models, power plants, bombs, biological systems, etc

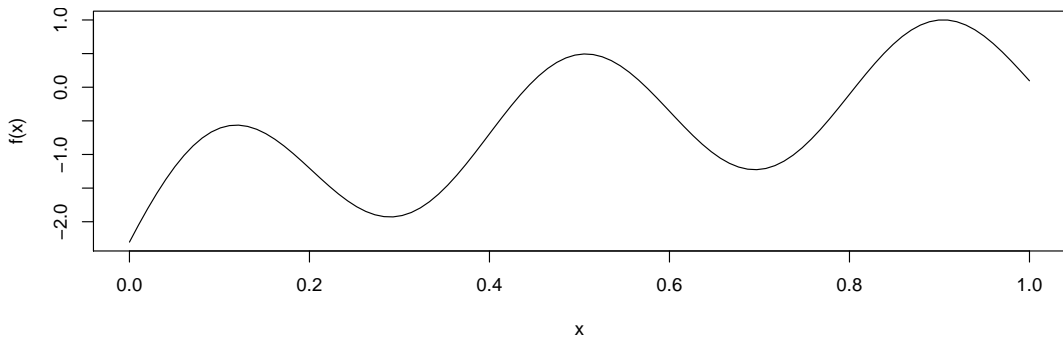
## Some jargon and notation

- ▶ The complicated system is called the *simulator*
- ▶ The statistical short-cut is called the *emulator*
- ▶ Let the inputs to the simulator be  $x$ , the outputs  $y$ , and the simulator  $f$
- ▶ Usually the simulator is deterministic so  $y = f(x)$  with no noise
- ▶ The emulator is  $\hat{f}$
- ▶ Your mission is to find  $\hat{f}$  given that it's very expensive to call  $f$

## A simple example

Suppose our simulator looked like this:

```
x = seq(0, 1, length = 100)
f = function(x) log(x+0.1)+sin(5*pi*x)
plot(x, f(x), type = 'l')
```



## Building a simple emulator

The usual approach is:

1. Decide on how many runs you can afford
2. Specify a few design points  $x^*$  at which to run  $f$ . Obtain  $y^* = f(x^*)$
3. Fit a *Gaussian Process* to the outputs
4. Use the Gaussian Process to predict where the biggest uncertainties are to specify your next set of design points
5. Repeat from 3 until you have used up all your runs

The final Gaussian Process can be used as the emulator and the simulator can be thrown away!

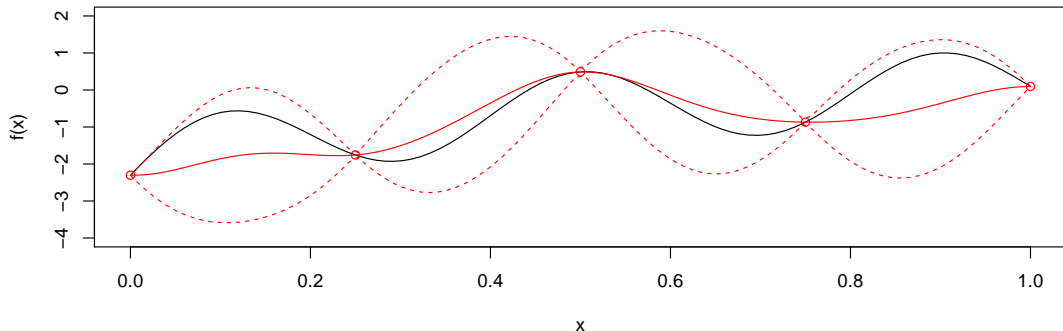
## Back to the example

- Suppose we decided that we have 10 points at which to run the emulator, and we would start with 5 runs

```
x_star = seq(0, 1, length = 5)
y_star = f(x_star)
library(GPfit)
f_hat = GP_fit(x_star, y_star)
x_new = seq(0, 1, length = 100)
pred = predict(f_hat, x_new)
```

# Plot

```
plot(x, f(x), type= 'l', ylim = c(-4, 2))  
points(x_star, y_star, col = 'red')  
lines(x_new, pred$Y_hat, col = 'red')  
lines(x_new, pred$Y_hat - 2*sqrt(pred$MSE), lty = 2, col = 'red')  
lines(x_new, pred$Y_hat + 2*sqrt(pred$MSE), lty = 2, col = 'red')
```





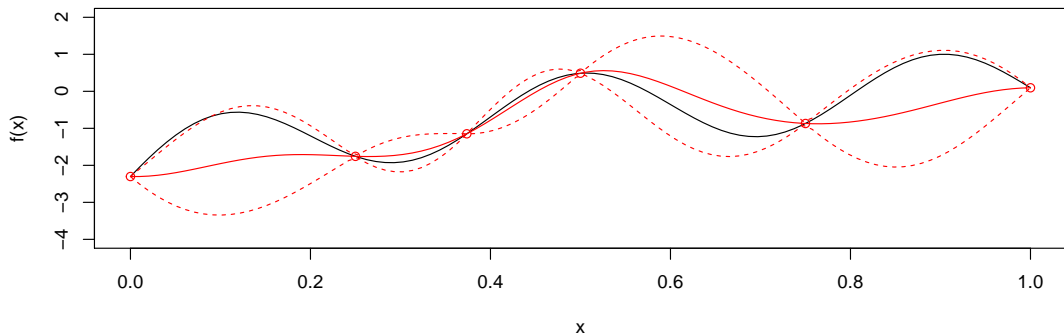
## Step 2

- Now find the place (or places) where the simulator has done badly and re-run

```
x_star = c(x_star, x_new[which.max(pred$MSE)])  
y_star = c(y_star, f(x_star[6]))  
f_hat = GP_fit(x_star, y_star)  
x_new = seq(0, 1, length = 100)  
pred = predict(f_hat, x_new)
```

## Plot 2

```
plot(x, f(x), type= 'l', ylim = c(-4, 2)); points(x_star, y_star, col = 'red')  
lines(x_new, pred$Y_hat, col = 'red')  
lines(x_new, pred$Y_hat - 2*sqrt(pred$MSE), lty = 2, col = 'red')  
lines(x_new, pred$Y_hat + 2*sqrt(pred$MSE), lty = 2, col = 'red')
```



... and repeat

## What is going on in the background?

- ▶ The GPfit package is fitting a Gaussian Process
- ▶ This is the model:

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

- ▶ So there are 3 parameters:  $\mu, \phi, \tau$
- ▶ The parameter  $\tau^2$  controls the variance of the Gaussian process (bigger values have more uncertainty)
- ▶ The parameter  $\phi$  controls how smooth the curve is
- ▶ The function  $\rho$  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 prediction:

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

- ▶ Here  $\Sigma^*$  and  $\Sigma^{**}$  are the covariance matrix of the prediction points and their variance respectively
- ▶ This means that you can get predicted means and uncertainties from every point

## 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
- ▶ The 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 predicting

## A more complicated example

- ▶ Suppose we now want to emulate a more complicated model
- ▶ Example: an SEIR infectious disease model that predicts the time to extinction of a disease
- ▶ The inputs are the basic reproduction number  $R_0$ , the initial number of exposed individuals  $E$ , the number of infected individuals  $I$ , and the number of recovered individuals so far  $R$
- ▶ The output is a set of quantiles for the length of time before the virus is extinct
- ▶ The model takes a long time to run

## Set up

- ▶ I'm willing to run the simulator over 24 hours (maybe 500 runs if well coded)
- ▶ I need good values to start the simulator off
- ▶ In 4 input dimensions I need the starting values to cover the space well
- ▶ Use a Latin hypercube design
- ▶ (NB: the `GP_fit` function requires all input values to be scaled between 0 and 1)
- ▶ There are other input parameters too but these are fixed values

## Building the emulator

1. File `run_simulator.R`
2. File `build_emulator_*.R`
3. File `out_lt1.txt`
4. File `fit_emulator.R`
5. File `run_emulator.R`
6. File `app.R`



## Other issues

- ▶ Discontinuities in the simulator
- ▶ Design of experiments for input space (and adaptive design)
- ▶ Priors on input space
- ▶ Multivariate responses
- ▶ Emulating time series data
- ▶ Emulating stochastic systems