Emulation workshop tutorial notes

I. Andrianakis and I. Vernon

September 14, 2016

1 Theoretical background

This section provides some theoretical background behind the development of emulators. Understanding the material below will help you make the most out of the practical session that follows. We will first present a simple case of a zero mean-unit variance emulator and then proceed with the more general case that we will use in the practical session.

1.1 A zero mean-unit variance emulator

According to the definition in (Rasmussen and Williams, 2006) a Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.

Suppose that $f(\mathbf{x})$ is a zero mean and unit variance Gaussian process; we can write this as

$$f(\mathbf{x}) \sim \mathcal{GP}(0, c(\mathbf{x}, \mathbf{x}')). \tag{1}$$

The $c(\mathbf{x}, \mathbf{x}')$ term is known as the covariance function and plays a fundamental role in Gaussian processes as it determines the association between $f(\mathbf{x})$ and $f(\mathbf{x})'$ based on the distance between $\mathbf{x}$ and $\mathbf{x}'$. If we denote this distance as W, the covariance function can take a multitude of forms, some of the most common are shown in table 1. Despite the flexibility in their form, covariance functions still have to satisfy a fundamental constraint, that of positive definiteness. For more on this point the reader may consult chapter 4 of (Rasmussen and Williams, 2006).

Let us now examine the weighted distance W. Suppose that we have two p-dimensional points $\mathbf{x} = [x_1, x_2, \ldots, x_p]$ and $\mathbf{x}' = [x'_1, x'_2, \ldots, x'_p]$. Their weighted distance is defined as

$$W = \left[\sum_{i=1}^{p} \left(\frac{x_i - x'_i}{\delta_i} \right)^2 \right]^{1/2}. \tag{2}$$

The parameters $\delta = [\delta_1, \delta_2, \ldots, \delta_p]$ are known as *correlation lengths* and essentially weigh the distance between $\mathbf{x}$ and $\mathbf{x}'$. A large δ_i implies that $f(\mathbf{x})$ and $f(\mathbf{x}')$ will be strongly correlated along the i^{th} dimension of $\mathbf{x}$ and vice versa.

Correlation function	Formula
Gaussian	$c(\mathbf{x}, \mathbf{x}') = \exp\left(-W^2\right)$
α-exponential ($\alpha < 2$)	$c(\mathbf{x}, \mathbf{x}') = \exp\left(-W^\alpha\right)$
Matérn 3/2,	$c(\mathbf{x}, \mathbf{x}') = \left(1 + \sqrt{3}W\right) \exp\left(-\sqrt{3}W\right)$
Matérn 5/2,	$c(\mathbf{x}, \mathbf{x}') = \left(1 + \sqrt{5}W + 5W^2/3\right) \exp\left(-\sqrt{5}W\right)$

Table 1: Some common correlation functions $c(\mathbf{x}, \mathbf{x}')$, with W denoting a weighted distance between $\mathbf{x}$ and $\mathbf{x}'$.

Suppose now that we observe the Gaussian process at n discrete points $D = \{\mathbf{x}_i : i = [1, \ldots, n]\}$ and we get the vector of observations $f(D) = [f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n)]$. D and $f(D)$ here represent the emulator's training points. According to the Gaussian process in eq. 1, the observations $f(D)$ will follow a joint Gaussian distribution,

$$p(f(D)|\delta) = \frac{|\tilde{A}|^{-1/2}}{(2\pi)^{n/2}} \exp\left[-\frac{1}{2} f(D)^{\mathrm{T}} \tilde{A}^{-1} f(D)\right], \tag{3}$$

where $\tilde{A}$ is the covariance matrix of the design points $f(D)$, which is defined as

$$\tilde{A} = \begin{pmatrix} 1 & c(\mathbf{x}_1, \mathbf{x}_2) & \ldots & c(\mathbf{x}_1, \mathbf{x}_n) \\ c(\mathbf{x}_2, \mathbf{x}_1) & 1 & \ldots & c(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ c(\mathbf{x}_n, \mathbf{x}_1) & \ldots & \ldots & 1 \end{pmatrix}.$$

The distribution of $f(D)$ is conditioned on the covariance function hyper-parameters δ. One way of training the emulator could be to search for a value of δ that maximises equation 3, i.e.

$$\hat{\delta} = \arg\max_{\delta} \left[p(f(D)|\delta)\right].$$

A more involved approach, would be to define a prior distribution for δ and draw samples from the posterior distribution $p(\delta|f(D))$. For simplicity, we assume here the first approach.

We are now interested in estimating the value of the Gaussian process at an unknown point $\mathbf{x}$, conditional on the observed data $f(D)$ and $\hat{\delta}$. This distribution is

$$p(f(\mathbf{x})|f(D), \hat{\delta}) = \mathcal{N}(\mathrm{E}^*[f(\mathbf{x})], \mathrm{Var}^*[f(\mathbf{x})]) \tag{4}$$

where $\mathcal{N}(\mu, \Sigma)$, is a normal distribution with mean μ and variance Σ, and

$$\begin{aligned} \mathrm{E}^*[f(\mathbf{x})] =& c(\mathbf{x})^{\mathrm{T}} \tilde{A}^{-1} y \\ \mathrm{Var}^*[f(\mathbf{x})] =& c(\mathbf{x}, \mathbf{x}') - c(\mathbf{x})^{\mathrm{T}} \tilde{A}^{-1} c(\mathbf{x}') \end{aligned}$$

with the vector $c(\mathbf{x})$ defined as $c(\mathbf{x}) = [c(\mathbf{x}, \mathbf{x}_1), c(\mathbf{x}, \mathbf{x}_2), \cdots, c(\mathbf{x}, \mathbf{x}_n)]^{\mathrm{T}}$. The above distribution is the posterior distribution of a zero mean - unit variance emulator.

1.2 General case

We will now present a more general case of a Gaussian process emulator, with a non-zero mean and arbitrary variance. The model assumed is

$$f(\mathbf{x}) \sim \mathcal{GP}(h^{\mathrm{T}}(\mathbf{x})\beta, \sigma^2 c(\mathbf{x}, \mathbf{x}')). \tag{5}$$

which is similar to equation 1, with the addition of σ^2 that controls the scaling (variance) of the process and the mean term $h^{\mathrm{T}}(\mathbf{x})\beta$. The term $h(\mathbf{x})$ is a $q \times 1$ vector of functions of $\mathbf{x}$ (regressors). For example

$$\begin{aligned} h^{\mathrm{T}}(\mathbf{x}) &= [1] \\ h^{\mathrm{T}}(\mathbf{x}) &= [1, x_1, \ldots, x_p] \\ h^{\mathrm{T}}(\mathbf{x}) &= [1, x_1, x_2^2, \ln(x_3)], \text{ etc.} \end{aligned}$$

β is a $q \times 1$ vector of regression parameters.

Assuming again a set of n observations $f(D)$, the likelihood of this model is

$$\begin{aligned} p(f(D)|\beta, \sigma^2, \theta_c) &= \mathcal{N}(H\beta, \sigma^2 A) \\ &= \frac{|A|^{-1/2}}{(2\pi\sigma^2)^{n/2}} \exp\left[-\frac{1}{2\sigma^2}(f(D) - H\beta)^{\mathrm{T}} A^{-1} (f(D) - H\beta)\right] \end{aligned} \tag{6}$$

where

$$H = [h(\mathbf{x}_1), h(\mathbf{x}_2), \cdots, h(\mathbf{x}_n)]^{\mathrm{T}}.$$

The matrix A is defined as $A = \tilde{A} + \nu\mathcal{I}$, where $\mathcal{I}$ a diagonal matrix and ν is a parameter, commonly known as the nugget (Andrianakis and Challenor, 2012). The role of the nugget is to account for the existence of uncertainty in the training data (e.g. when emulating the mean output of a stochastic model, as calculated from averaging individual runs), while it can also safeguard against numerical instabilities that can occur in the calculations involved in training and fitting the emulator. The parameters δ and ν are jointly referred to as $\theta_c = [\delta, \nu]$.

One way of training this emulator could be to obtain estimates for $\{\beta, \sigma^2, \theta_c\}$ by maximising the likelihood function in eq. 6. However, the parameters β and σ^2 can be marginalised in the Bayesian sense analytically. We opt for the marginalisation of these two parameters because: a) the resulting emulator has less hyper-parameters to deal with b) it tends to make the estimates of the non-marginalised parameters more robust and c) marginalisation takes into account the uncertainty about the 'true' value of the marginalised hyper-parameters. An analytical marginalisation of the θ_c parameters is not possible, and the computational cost of a numerical marginalisation typically outweighs its benefits in this case. We therefore use maximum likelihood estimates for θ_c.

If we assume a non informative prior for σ^2 and β, $p(\sigma^2, \beta) \propto \sigma^{-2}$ (Kennedy and O'Hagan, 2001), these two parameters can be marginalised, yielding

$$p(f(D)|\theta_c) \propto \frac{|A|^{-1/2}|H^{\mathrm{T}}A^{-1}H|^{-1/2}}{(\hat{\sigma}^2)^{(n-q)/2}} \tag{7}$$

where $\hat{\sigma}^2 = f(D)^{\mathrm{T}}[A^{-1} - A^{-1}H(H^{\mathrm{T}}A^{-1}H)^{-1}H^{\mathrm{T}}A^{-1}]f(D)$. This is the expression we optimise w.r.t. θ_c when we train an emulator.

In the same fashion, one can obtain the posterior distribution for the emulator $p(f(\mathbf{x})|f(D), \hat{\theta}_c)$, which will be a multivariate t-distribution, with $n - q$ degrees of freedom and mean

$$\mathrm{E}^*[f(\mathbf{x})] = h^{\mathrm{T}}(\mathbf{x})\hat{\beta} + c^{\mathrm{T}}(\mathbf{x})A^{-1}(f(D) - H\hat{\beta}) \tag{8}$$

where $\hat{\beta} = (H^{\mathrm{T}}A^{-1}H)^{-1}H^{\mathrm{T}}A^{-1}f(D)$ and variance

$$\mathrm{Var}^*[f(\mathbf{x})] = \frac{\hat{\sigma}^2}{n-q-2}c_1(\mathbf{x}, \mathbf{x}'), \tag{9}$$

with

$$\begin{aligned} c_1(x, x') = {} & c(x, x) - c^{\mathrm{T}}(x)A^{-1}c(x) \\ & + (h^{\mathrm{T}}(x) - c^{\mathrm{T}}(x)A^{-1}H)(H^{\mathrm{T}}A^{-1}H)^{-1}(h^{\mathrm{T}}(x) - c^{\mathrm{T}}(x)A^{-1}H)^{\mathrm{T}}. \end{aligned}$$

If the degrees of freedom $(n - q)$ of the t-distribution are sufficiently large (e.g. $n - q > 20$) we can consider $p(f(\mathbf{x})|f(D), \hat{\theta}_c)$ as being a multivariate normal distribution with the same mean and variance as above.

2 Session 1

The emulator described in section 1.2 is implemented in the `GPCore` class. In this part of the tutorial we will emulate simple 1-D models and investigate the effect of the correlation and mean functions. We will also look into some validation techniques.

2.1 Preparation

- Start R and move in the directory where the files `GPCore.R` and `Models.R` are, using the `setwd()` function.
- Source the files `GPCore.R` and `Models.R` so that the necessary functions are loaded in memory, e.g. `source('Models.R')`.

2.2 Building the emulator

We will first emulate the simple 1-input 1-output model

$$y = \sin(2\pi x) \tag{10}$$

i.e. the sine function. This toy model is implemented in the function `Model1`.

Generate the design points `X` (the points where we will run the model)

```
X = matrix(c(0,0.25,0.5,0.9,1), ncol = 1)
```

The `GPCore` class accepts the input training data `X` in the form of a *matrix* with n rows (number of design points) and p columns (number of inputs, in this case 1).

Then 'run' the model at these points to obtain the output data `Y`

```
Y = Model1(X)
```

You can also 'run' the model for a range of values between -0.2 and 1.2, which we will use to evaluate the emulator

```
xplot = matrix(seq(-0.2, 1.2, length=100), ncol=1)
yplot = Model1(xplot)
```

Let us now build the emulator. The first step is to load the data

```
E = GPCore()     # define the emulator
E$InputData(X)   # load the input data
E$OutputData(Y) # load the output data
```

Typing `E` on R's command prompt, will display an overview of the emulator we have build so far.

2.3 Varying the correlation length δ

We will now examine the effect of the correlation length (δ) on the emulators. Set the correlation length to 1 and then plot the emulator's posterior mean and confidence intervals, together with the actual simulator for comparison:

```
E$SetHyper(1)  # Set the correlation length to 1
E$Plot(xplot)  # Plot the emulator's posterior
lines(xplot,yplot) # Plot the simulator
```

In the plot that showed up, the black line is the simulator (sine function) the orange x's are the training points `X`, `Y` $(D, f(D))$, the orange continuous line is the emulator's posterior mean ($\mathrm{E}^*[f(\mathbf{x})]$) and the blue shade represents $\pm$ 2 standard deviations of the posterior ($[\mathrm{Var}^*[f(\mathbf{x})]]^{0.5}$).

- What do you think of this emulator? Does it capture the simulator adequately? Ideally we'd like the simulator (black line) to fall within the blue shaded area, without the latter being excessively large.

Try varying the correlation length value in the range $(0, 1.3]$. δ should be *positive* and for $\delta > 1.3$ this emulator will run into numerical problems (we will see later how we can overcome this).

- What can you say about the effect of the correlation length value on the emulator?
- Which value do you think is optimal for this problem?

2.4 Maximum likelihood

One way of finding an 'optimal' value for δ is by maximising the likelihood $p(f(D)|\theta_c)$, with respect to δ. Note that so far we've kept the nugget ν fixed, so we can consider that $\theta_c \equiv \delta$. The `GPCore` class returns the log-likelihood of an emulator via the method `LogLik`. The following code snippet will calculate the log likelihood for several values in the $(0, 1.3)$ range, plot the results and find the maximum.

```
N = 1000
# preallocate the values of delta
delta = seq(from=0.0001, to=1.3, length.out=1000)
# preallocate storage for the log likelihood
LogLik = rep(0, N)
for (k in 1:N) {
  # calculate the likelihood for delta[k]
  LogLik[k] = E$LogLik(delta[k])$L
}

plot(delta, LogLik)          # plot
deltahat =   delta[which.max(LogLik)]   # find the maximum
```

- What is the value of δ that maximises the likelihood?
- How does this compare to the optimal value you found before?

2.5 The nugget ν

The nugget is a (typically small) number that we add on the main diagonal of the correlation matrix A. Although it is often neglected, it can have some useful roles in an emulator: a) to alleviate numerical problems, b) to model the existence of noise in the observations `Y` and c) to allow for inactive variables. Let's consider first the numerical problems.

The `GPCore` class calculates the emulator's posterior distribution at points `x` via the method `Predict`

```
r = E$Predict(x)
```

Note that `x` must be an $n' \times p$ *matrix*, containing the n' p-dimensional points where we wish to evaluate the emulator's posterior distribution. The function `Predict` returns a list containing a field `m`, which is a $n' \times 1$ matrix with the posterior mean values and a field `V`, which is the $n' \times n'$ posterior covariance matrix. The main diagonal of `V` contains the posterior variance of the n' points, which we use to plot the shaded areas of the emulator.

Set $\delta = 1.5$ and $\nu = 0$ by typing

```
E$SetHyper(1.5,0)
```

Calculate the emulator's posterior at the points `xplot`

```
r = E$Predict(xplot)
```

You must have received a warning message about numerical errors. Try plotting the diagonal of the `V` matrix, e.g.

```
plot(diag(r$V))
```

What is the minimum value on the main diagonal of `V`? Theoretically, the posterior variance cannot be less than zero, so if you find it is, then it is because of a numerical error.

Repeat the above by setting $\nu = 10^{-8}$. Compare the diagonals of the posterior covariance matrices for the two cases. How do they look like? They should be similar apart from the second diagonal being non-negative (as it should!). One function of the nugget therefore, is to alleviate numerical problems.

For the emulators we build in this workshop, we require that the emulator's posterior mean at the design points `X` equals the model output `Y` (in other words, the orange line should pass through the crosses in the emulator plots). However, this need not always be the case, as for example, if we believe that our data `Y` are inaccurate (noisy), we can still be happy if the emulator passes nearby the design points but does not match them exactly.

To see this effect, plot the emulator for $\delta = 0.7$ and start increasing the value of ν from 10^{-14} to 10^{-1}. What do you see? Does the emulator always interpolate the design points `X`, `Y`? To achieve this you can use the following code:

```
E$SetHyper(0.7,1e-14)
E$Plot(xplot)
lines(xplot,yplot)
```

For the rest of this tutorial we'll keep the nugget fixed.

2.6 Correlation functions

In this part we will investigate the effect of using different correlation functions. So far we have been using the Gaussian correlation function (see table 1). The `GPCore` class also supports the Matérn 3/2 and Matérn 5/2 functions. To change the emulator's correlation function use the command

```
E$SetCorFun('funname')
```

where `funname` is either `GausCor`, `Mat32`, or `Mat52`.

For this exercise, we will use a different model, 200 runs of which are stored in `Model2.RData`. Load this data (`load('Model2.RData')`). You should see 2 variables, `x_M2` and `y_M2`; these are the inputs (`X`) and outputs (`Y`) of the 200 model runs.

- Plot the `Model2` data, (`plot(x_M2,y_M2,'l')`). How do they look? Note that this model's output is less smooth compared to the model we used so far (the sine function).

We will select 7 points out of the 200 to build the emulator. Type the following

```
ind = c(1,30,60,90,120,150,180)
X = matrix(x_M2[ind,1],ncol=1)
Y = matrix(y_M2[ind],ncol=1)
```

Now the variables `X` and `Y` contain the 1st, 30th, 60th,... input output pairs from the 200 model runs. These variables will be used for building the emulator. Below is a sample code for doing this:

```
E = GPCore()              # Define the emulator
E$InputData(X)            # Load the input data
E$OutputData(Y)           # Load the output data
E$SetCorFun('GausCor')    # Set the correlation function
```

Having setup the emulator, we will find the maximum likelihood estimate for the correlation length δ. Note that the nugget ν remains fixed at 10^{-12}. The following code plots the log likelihood and saves the maximum in the variable `deltahat`:

```
N = 1000
delta_max = 2
delta = seq(from=0.0001, to=delta_max, length.out=N)
LogLik = rep(0,N)

for (k in 1:N) {
  LogLik[k] = E$LogLik(delta[k])$L
}
plot(delta,LogLik)
deltahat = delta[which.max(LogLik)]
deltahat
```

You can experiment with different values of `delta_max`, just make sure before moving on, to use a `delta_max` that is relatively close to the mode, so that its exact location is approximated accurately enough.

You can finally plot the resulting emulator with the commands:

```
E$SetHyper(deltahat)
E$Plot(x_M2)
lines(x_M2,y_M2)
```

Repeat the above procedure for the 3 correlation functions supported by the `GPCore` class. Plot the final emulators in 3 different windows, so that you can compare them (the command `dev.new()` might be useful for this).

- How different are the maximum likelihood correlation lengths for the 3 cases?
- How different are the resulting emulators?
- What can you say about the different correlation functions?

The Gaussian correlation function is *infinitely differentiable*, meaning that the resulting emulator has mean square derivatives of all orders and is very smooth. In fact the Gaussian correlation function is equivalent to the Matérn with $\kappa \to \infty$ (instead of $\kappa = 3/2$ or $\kappa = 5/2$). The Gaussian correlation function is the simplest of the 3 and has been used extensively in the field. However, for many models its smoothness assumptions can be too restrictive, in which case, the Matérn family can be a good alternative.

2.7 Mean functions

In this section we will investigate the effect of the mean function on the emulators. To this end, we will use the `Model3`, which is in the file `Models.R` and implements the function

$$y = \sin(10\pi x) + 0.1\exp(5 * x). \tag{11}$$

For values of x close to 0, the function is dominated by the behaviour of the sine term, while for $x \to 1$, the exponential term takes over.

We will emulate this 'model' with the three mean functions offered by the `GPCore` class. These are

- `RegFun0`: $h^{\mathrm{T}}(\mathbf{x}) = 1$
- `RegFun1`: $h^{\mathrm{T}}(\mathbf{x}) = [1, \mathbf{x}]$
- `RegFun3`: $h^{\mathrm{T}}(\mathbf{x}) = [1, \mathbf{x}, \mathbf{x}^2, \mathbf{x}^3]$

You can change the mean function with the `SetRegFun` method, e.g.

```
E$SetRegFun('RegFun0')
```

Create the following design points,

```
X = matrix(c(0.0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.8, 0.9, 1),ncol=1)
```

and run the model for these values

```
Y = Model3(X)
```

Also run the model for a range of values that we will then use for plotting and evaluating the emulator

```
xplot = matrix(seq(-0.2,1.2,length.out=100),ncol=1)
yplot = Model3(xplot)
```

Follow the steps of the previous section to a) build an emulator with the default correlation function and nugget value, and each of the three mean functions, b) find the maximum likelihood value of the correlation length δ and c) plot the emulator against the model. Plot all three emulators in separate windows to allow comparing them. What conclusions can you draw about the resulting emulators?

The `RegFun3` is a 3rd order polynomial, which in this case captures (most of) the exponential term of the model, and the part of the Gaussian process that is determined by the correlation function ($c(\mathbf{x}, \mathbf{x})$) captures the sine part of the model. In the case of `RegFun0`, the mean function is a simple constant, that captures the mean level of the model output (e.g. the average value of `Y`), and the correlation part of the Gaussian process has to capture both the steep rise that is due to the exponential term as well as the fluctuations that are due to the sine. As a result, the `RegFun0` emulator is more uncertain between the values 0.2 and 0.8. Also note that the `RegFun3` emulator, does a better job in extrapolating the model (i.e. for $x > 1$). We should note however, that this behaviour is particular to this example, and an increase in the complexity of the mean function does not guarantee better extrapolation properties (sometimes it is the opposite!).

Note that the design here was deliberately chosen not to be space filling, to mimic the case that often occurs in higher dimensions, where the distance between design points can be quite large.

2.8 Validation

After building an emulator, we need to test how accurate its predictions are. To achieve this we typically run the simulator a few more times and compare the model output with the emulator's predictions. A paper that discusses various validation techniques is Bastos and O'Hagan (2009). Here, we will consider two measures: the standardised prediction errors and the Mahalanobis distance.

Suppose that we run the model at the n_p validation points $\mathbf{x}'$ and we obtain the model output $f(\mathbf{x}')$. The standardised prediction errors are given by:

$$e_i = \frac{f(\mathbf{x}'_i) - \mathrm{E}^*[f(\mathbf{x}'_i)]}{\mathrm{Var}^*[f(\mathbf{x}'_i)]^{1/2}}, \tag{12}$$

for $i \in [1, n_p]$. That is, the standardised prediction errors represent the difference between the observed simulator outputs and the emulator's posterior mean at the same inputs $\mathbf{x}'_i$, divided by the posterior standard deviation (uncertainty) of the emulator at the same inputs $\mathbf{x}'_i$. If the number of training data is large enough, we can assume that the standardised prediction errors follow a standard normal distribution. Therefore, any errors with an absolute value larger than 2 indicate that there may be a problem with the emulator.

An extension that allows summarising the diagnostic into a single number and also takes into account correlations between errors is the Mahalanobis distance. This is given by

$$M = (f(\mathbf{x}') - \mathrm{E}^*[f(\mathbf{x}')])^{\mathrm{T}} \mathrm{Var}^*[f(\mathbf{x}')]^{-1} (f(\mathbf{x}') - \mathrm{E}^*[f(\mathbf{x}')]) \tag{13}$$

If we define a 'scaled' Mahalanobis distance as

$$M_s = \frac{(n-q)M}{n_p(n-q-2)} \tag{14}$$

it can be shown that M_s follows an F distribution with parameters n_p and $n-q$, i.e.

$$M_s \sim F_{n_p, n-q} \tag{15}$$

The mean of the above distribution is equal to $(n-q)/(n-q-2)$, which for practical purposes can be approximated by 1 (as we expect that the number of training data n will be much larger than the number of regression functions q). Therefore, if the diagnostic M_s is very different to 1, then this indicates some problem with the emulator. This difference can be assessed by comparing the M_s statistic to the cumulative distribution function $F_{n_p, n-q}$ which should provide the probability to observe such a value for M_s, had the emulator been valid.

We will now use the above diagnostics to validate an emulator of the model we built in the previous section.

- Run `Model3` at the design points $[0, 0.18, 0.39, 0.63, 0.8, 1]$.
- Build an emulator with the above data and the default mean and correlation functions.

- Set the nugget to 10^{-10}, e.g.

```
E$SetHyper(1,1e-10)
```

  otherwise you might run into numerical problems!

- Find the maximum likelihood.
- Plot the emulator's output against the model. How does the emulator look?

We'll now run `Model3` at points $[0.05, 0.25, 0.42, 0.7]$. These will be our validation points.

```
Xv = matrix(c(0.05, 0.25, 0.42, 0.7),ncol=1)
Yv = Model3(Xv)
```

You can add these points on the plot by typing `points(Xv,Yv)`.

Let us now calculate the standardised prediction errors. First evaluate the emulator at `Xv`

```
R = E$Predict(Xv)
```

The variable `R` is a list with a field `m`, which is an $n_p \times 1$ matrix containing the posterior mean and a field named `V`, which is a $n_p \times n_p$ matrix that contains the posterior covariance.

The standardised prediction errors can be computed with the following command:

```
spe = (Yv - R$m) / diag(R$V)^0.5
```

What are their values? Do they fall outside the [-2,2] interval?

The Mahalanobis distance can be calculated with the command

```
M = t(Yv - R$m) %*% solve(R$V,Yv - R$m)
```

and its scaled version with

```
Ms = (E$n - E$q)/(dim(Yv)[1]*(E$n - E$q - 2))*M
```

- How large is `MS`? If the emulator were valid it should have been close to 1. Its value can also be compared against the $F_{n_p,n-q}$ distribution to see what are the chances of observing this value of M_s for a valid emulator.

The `GPCore` class calculates and plots the above 2 diagnostics automatically, with the command

```
E$Validate(Xv,Yv)
```

You can compare the results of the `Validate` method with the ones you calculated before.

You can now train the emulator using the points $[0, 0.05, 0.18, 0.25, 0.39, 0.42, 0.63, 0.7, 0.8, 1]$ and validate it using points $[0.1, 0.33, 0.55, 0.90]$.

- Repeat the above procedure. How do the diagnostics look now?

Although this one dimensional example can be validated by a simple inspection of the emulator's output, the above validation diagnostics are extremely important in high dimensional (multi-input) models, where such a visual inspection is not possible.

3 History matching

As you may remember from the previous lecture, history matching is a way to determine the input values of a model that allow it to match some empirical observations. History matching is centred around the concept of *implausibility*. The implausibility measures the distance between the measurements z and the emulator's posterior mean for an input $\mathbf{x}$, weighted by all the uncertainties that are present in the system, namely the Code Uncertainty (V_c), the Observation Uncertainty (V_o), the Model Discrepancy (V_m) and in case of stochastic models, the Ensemble Variability (V_s). Since the models we deal with here are deterministic, the ensemble variability is zero. For the sake of simplicity, we will ignore the model discrepancy as well, although in practice it is very important and it should be taken into account. The general form of implausibility is (Vernon et al., 2010; Andrianakis et al., 2014)

$$I(\mathbf{x}) = \frac{|z - \mathrm{E}^*[f(\mathbf{x})]|}{[V_o + V_c(\mathbf{x}) + V_s + V_m]^{1/2}}, \tag{16}$$

where z is the observation. In this example we will consider the simpler version:

$$I(\mathbf{x}) = \frac{|z - \mathrm{E}^*[f(\mathbf{x})]|}{[V_o + V_c(\mathbf{x})]^{1/2}}. \tag{17}$$

As explained in the lecture, a natural cut-off for the implausibility is $I_c = 3$. We will use this cut-off here, meaning that any $\mathbf{x}$ with $I(\mathbf{x}) > 3$ will be implausible, and vice versa.

We will now history match `Model2` from section 2.6. Suppose that we have an observation $z = -35$ with observation uncertainty $V_o = 0.025$.

We first build an emulator using the following points

```
ind = c(1, 40, 80, 120, 160, 200)
X = matrix(x_M2[ind,1], ncol=1)
Y = matrix(y_M2[ind,1], ncol=1)
```

and then as usual

```
E = GPCore()
E$InputData(X)
E$OutputData(Y)
```

leaving the mean function and the nugget at its default values. We also choose the Matérn 3/2 correlation function

```
E$SetCorFun('Mat32')
```

- Find the maximum likelihood correlation length and build the emulator as in the previous exercises. Plot it against all the model runs. Does it validate? (You can assess this visually.)

The Observation can be loaded onto the emulator with the command

```
E$Obs = -35
```

and the observation uncertainty with the command

```
E$OE  = 0.025
```

The implausibility can be readily calculated with the function

```
I =  E$Implausibility(x_M2)
```

If you have time, you can calculate the implausibility manually and check your results against those of the `GPCore` class function. For this you will need the `Predict` method of the `GPCore` class.

- Plot the implausibility against the model input values, e.g. `plot(x_M2,I)`. You can also use `plot(x_M2,I, ylim = c(0,10))` to zoom in the region of interest.
- For which input values is the implausibility less than 3? You can find the indices of those values with the command `ni_ind = which(I<3)`.

The following code will plot the model, the emulator, the observation data with ± 2 error standard deviations and the *actual* model runs that the emulator thinks are acceptable ($I < 3$) coloured in green.

```
xlim = c(0,1)
ylim = c(-55,10)
E$Plot(x_M2, xlim=xlim, ylim=ylim)
lines(x_M2, y_M2)
points(x_M2[ni_ind,], y_M2[ni_ind], col='green')
abline(h = c(E$Obs - 2*E$OE^0.5, E$Obs, E$Obs + 2*E$OE^0.5),
       lty = c(1, 3, 1))
points(x_M2, rep(ylim[1], length = dim(x_M2)[1]), col='red')
points(x_M2[ni_ind,], rep(ylim[1], length = length(ni_ind)),
       col='green')
```

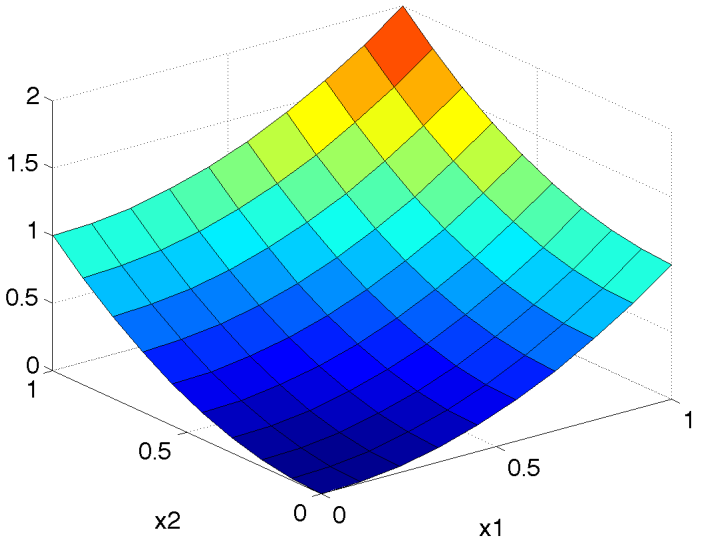

Figure 1: Model 4 ($y = x_1^2 + x_2^2 + x_3$) with $x_1, x_2 \in [0, 1]$ and $x_3 = 0$.

Let us now add some wave 2 non-implausible points in the training data of our emulator. We'll add the wave 2 points 50, 60, and 70 and build the emulator with the following data:

```
ind = c(1, 40, 50, 60, 70, 80, 120, 160, 200)
X = matrix(x_M2[ind,1], ncol=1)
Y = matrix(y_M2[ind,1], ncol=1)
```

Repeat the procedure above. Which indices result in an implausibility less than 3? What is the value of the model output for these indices? What is the value of the input? You will see that we manage to pinpoint the value of the model input for which the model matches the observation $z = -35$ just with 9 model evaluations.

3.1 A 3 dimensional model

In this section we will emulate and history match a 3 dimensional model, trying to highlight some of the issues that arise when we increase the number of inputs. The model we will use is the simple function

$$y = x_1^2 + x_2^2 + x_3, \text{ with } x_1, x_2, x_3 \in [0, 1] \tag{18}$$

If we fix x_3, this model is quarter of a quadratic 'bowl' with its output lying in the interval $[x_3, x_3+2]$ (figure 1). We will try to find the input values for which the model output takes the value 2.

We start by emulating the model. We first draw 30 3-dimensional design points using a maximin Latin hypercube, with the command

```
X = maximinLHS(30,3)
```

You can see the distribution of these points by typing `pairs(X)`. We then run the model at these points with the command

```
Y = matrix(Model4(X), ncol=1)
```

and then build the emulator as per usual, using the default mean function and nugget and the Matérn 3/2 correlation function:

```
E = GPCore()
E$InputData(X)
E$OutputData(Y)
E$SetCorFun('Mat32')
```

In this exercise we will try to calibrate the model to the 'observation' $z = 2 \pm 0.02$, so we can upload this information on the emulator with the commands

```
E$Obs = 2
E$OE = 1e-4
```

Here we assume that the interval 0.02 represents 2 standard deviations, and recall that the observation uncertainty (V_o) (the `OE` field above), represents the variance.

The next step is fitting the emulator to the data. Unlike the 1-d case, finding the maximum likelihood estimate of the 3 correlation lengths (1 per input) by visual inspection is not easy. Instead, we can use an optimisation algorithm to find it. This facility is provided by the `GPCore` class with the member function `Optimise`. This function works by accepting a $p \times 1$ vector of initial guesses for the δ's and then uses this starting point to find the maximum value of the likelihood. If successful, it stores this value in the emulator, which is then ready to use. The syntax for this function is

```
E$Optimise(c(d1, d2, ..., dp))
```

The likelihood surface of multidimensional emulators can have multiple local maxima. It is therefore advisable to run the above algorithm several times, until a sufficiently high local maximum is found. Once such a maximum is found, the emulator has then to be validated, a step which we can skip in this exercise for brevity.

You can run the following code a number of times, until a large enough local maximum is found. (For this example, anything around 110 should do!)

```
delta.init = runif(n=3, min=0, max=1)
optim.result = E$Optimise(delta.init)
print(c("Log Likelihood = ", E$L))
```

One way of visualising the non-implausible space in high dimensional models is via 2-d minimum implausibility and optical depth plots. To create a minimum implausibility plot for 2 inputs, we can draw a large number of points that span the p-dimensional model input space, and place them in the bins of a 2-d rectangular $n_b \times n_b$ grid (e.g. 30×30), depending on the value the 2 inputs of interest

take for each of these points. We then evaluate the implausibility of all the points and calculate the minimum implausibility in each bin. This yields a $n_b \times n_b$ matrix of minimum implausibilities, and if we plot this as an image it gives us the minimum implausibility plot. If on the other hand, for each bin we calculate the ratio of the non-implausible ($I < 3$) over the total number of points in the bin we get the optical depth plot. The first plot tells us what is the minimum implausibility we can expect to find if we were to fix 2 inputs to particular values. The second plot tells us what is the proportion of non-implausible points we can expect to find, again if we were to fix the two inputs to a certain value. These plot are very useful in the analysis of high dimensional models.

Let us now try to create some of these plots. The first step is to draw a large number of points in the unit hypercube, for which we will evaluate the implausibility. This can be achieved with the command

```
Xp = matrix(runif(n=900000, min=0, max=1), ncol=3)
I = E$Implausibility(Xp)
```

(note that in the above lines we evaluated an emulator 900000 times. How long did it take? Imagine doing this with an even moderately slow computer model!)

We will now make a minimum implausibility plot for the 2 first inputs (the quadratics). To achieve this, we will divide their input space ($[0,1]^2$) into a 30×30 grid and create a matrix `M` that will contain the minimum implausibility of the points that fall within each grid bin. At the same time, we will create a matrix `D` that will contain the ratio of the non-implausible points per each grid bin. We can do this with the function `BinData` which is loaded from the `Models.R` file, and can be called as

```
input1 = 1
input2 = 2
nb = 30
BD = BinData(I,Xp,input1,input2,nb)
```

where `I` is the implausibility, `Xp` are the input points, the next two arguments represent the inputs (columns of the `Xp` matrix) we want to make the plots for, and the last argument is the number of bins. The variable `BD` will be a list containing the matrices `M` and `D` described above. Input 1 will vary across the *rows* of these matrices and input 2 across the *columns*.

We can now make the minimum implausibility plot. We first create a vector that will label the axes of our plot with the command

```
xplot = (1:nb)/nb - 0.5/nb;
```

And finally make the plot with the command

```
filled.contour(xplot, xplot, BD$M, xlab="Input 1",
               ylab="Input 2", levels=seq(0, max(BD$M), len=30),
               color.palette=rev.rain.colors)
```

You should now see the minimum implausibility plot with the implausibility ranging between 0 and

90. We would like now to visualise the non-implausible space (i.e. $I(\mathbf{x}) < 3$)). One way of doing this is the following:

Define a temporary `M` matrix, that you can clip at a `cutoff` value, so that you can zoom-in the colorscale of the previous plot, e.g.

```
cutoff = 3
Mtemp = BD$M
Mtemp[Mtemp>cutoff] = cutoff
filled.contour(xplot, xplot, Mtemp, xlab="Input 1", ylab="Input 2",
               levels=seq(0, max(Mtemp), len=30),
               color.palette=rev.rain.colors)
```

You should now see an implausibility plot where all the non-implausible space lies outside the first quadrant quarter of the circle $x^2 + y^2 = 1$. This implies, that if in our model $x_1^2 + x_2^2 < 1$, it is impossible to match the observation $z = 2$. You can also experiment with drawing similar plots for the other input combinations.

You can also make the optical depth plot in the same way, where you should see that all the area within the $x_1^2 + x_2^2 < 1$ circle has a 0 probability of matching the calibration target. The following code will produce the optical depth plot

```
filled.contour(xplot, xplot, BD$D, xlab="Input 1",
               ylab="Input 2", nlevels=25,
               color.palette=cold1)
```

As a final step, you can visualise the 3 dimensional non-implausible space, using the `rgl` package and typing the following commands

```
require('rgl') # Only need to call this once!

X.ni = Xp[I<3,]
plot3d(X.ni[,1], X.ni[,2], X.ni[,3], col='green')
```

First, resize the window that shows up to make it a bit bigger, and you can then rotate it with your mouse/trackpad. You can see that all the implausible points lie on the part of the surface $x^2 + y^2 + z = 2$ that falls within the unit cube.

4 Conclusion

We have covered quite some ground on the emulation and history matching of computer models. We have not discussed however, models with several outputs or stochastic models. For more on these topics the interested reader can consult (Vernon et al., 2010; Andrianakis et al., 2014).

References

I. Andrianakis and P. Challenor. The effect of the nugget on Gaussian process emulators of computer models. *Computational Statistics & Data Analysis*, 56:4215–4228, 2012.

I. Andrianakis, I. Vernon, N. McCreesh, TJ McKinley, J. Oakley, R. Nsubuga, M. Goldstein, and R.G. White. Bayesian history matching and calibration of complex infectious disease models using emulation: a tutorial and a case study on HIV in Uganda. *Submitted to PLoS Computational Biology*, 2014.

L.S. Bastos and A. O'Hagan. Diagnostics for Gaussian process emulators. *Technometrics*, 51: 425–438, 2009.

M.C. Kennedy and A. O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society. Series B*, 63(3):425–464, 2001.

C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*. The MIT press, 2006.

I. Vernon, M. Goldstein, and R.G. Bower. Galaxy formation: a Bayesian uncertainty analysis. *Bayesian Analysis*, 5(4):619–670, 2010.