Karhunen-Loeve Implementation in R

James Briant

April 2021

1 Introduction

These notes should be read in conjunction with the appendix from the Bayesian Inverse Problems notes in Uncertainty Quantification Notes.

2 Adjusting the Solution for 2D samples

The notes outline how to sample from a 1D GP. But what if you wanted a 2D sample?

The natural way to parameterise a 2D domain into cells, ie. using (i, j) notation, leads to problems when defining the matrix $\mathcal{C}$ as described in the UQ notes. Consider the 3×3 example below

cell (1,3)	cell (2,3)	cell (3,3)
cell (1,2)	cell (2,2)	cell (3,2)
cell (1,1)	cell (2,1)	cell (3,1)

Instead, introduce the following parameterisation where each cell is numbered sequentially.

cell 7	cell 8	cell 9
cell 4	cell 5	cell 6
cell 1	cell 2	cell 3

Provided we know how many cells exist in the x-direction, which is known because we will define this, then we can uniquely determine a location in the grid using only 1 value, say $m \in \{1, 2, ..., M = 9\}$. Mathematically, the full domain $D = [0, L_x] \times [0, L_y]$ can now be represented by M cells,

$$D = \bigcup_{m=1}^{M} D_m.$$

The definition of $\mathcal{C}$ in equation (A.7) now makes sense using this parameterisation. The challenge now is how find the centre of the m^{th} cell using only m.

3 Converting Between the Number Line and Cartesian Coordinates

First, suppose we wants to draw a 2D GP sample that has I points in the x-direction and J points in the y-direction. Then we require a grid with I cells horizontally and J cells vertically. This gives $IJ = M$ cells in total.

Each cell has centre η_m but we require this in Cartesian form. Fortunately, since we know I we can easily convert between the two systems.

Since R starts counting the index of vectors from 1, we will develop a system that numbers the cells from 1 through to I (or 1 through to J in the y-direction). This can easily be changed for implementations in other programming languages that start counting from 0, but it is not presented here.

$$\begin{aligned} i_m &= \big(m - 1(\text{mod } I)\big) + 1 \\ j_m &= \text{ceiling}\Big(\frac{m}{I}\Big) \end{aligned} \tag{1}$$

Using these indices, we can get the x and y Cartesian coordinates.

$$\begin{aligned} \dot{x}_{i_m} &= h_x\Big(i_m - 0.5\Big), \qquad i_m = 1, ..., I \\ \dot{y}_{j_m} &= h_y\Big(j_m - 0.5\Big), \qquad j_m = 1, ..., J \end{aligned} \tag{2}$$

where $h_x = \frac{L_x}{I+1}$ and $h_y = \frac{L_y}{J+1}$. Now we have, $\eta_m = (\dot{x}_{i_m}, \dot{y}_{j_m})$.

We can also convert back from Cartesian index notation to the number line notation through the following formula,

$$m = (j - 1)I + i$$

4 R Implementation

Finding the eigenvalues and eigenvectors (technically eigenfunctions but each one is evaluated at η_m) should be straight forward. But, the new parameterisation means $\mathcal{C}$ has dimensions $(IJ \times IJ)$. Even for modest I and J this becomes huge. eg, $I = J = 100 \implies \mathcal{C}$ has dimensions $10,000 \times 10,000$.

Fortunately, we only require the positive eigenvalues, and most of the eigenvalues of the system are negative. (This was found only through observation and **may not be true in general!**)

The package `RSpectra` in `R` provides a useful function for finding only the first few eigenvalues (and eigenvectors) of a given matrix. See `eigs_sym()` for details.

The rest of the algorithm is straight forward as denoted in equation A.11 (note J means something else in the notes) and then convert the vector in to a $I \times J$ matrix (in the correct order!).