GpOutput2D : An R Package for metamodelling models with Two-Dimensional functional Output by using FPCA and Gaussian Process Regression Models

PERRIN Tran Vi-vi Élodie

Contents

1 Introduction

The aim of GpOutput2D package is to build metamodels for models with two-dimensional functional output, by using Gaussian Process (GP) regression methods [Williams and Rasmussen, 2006] (also called kriging models). The following simulator is considered:

$$\begin{array}{rccc} f & : & \Omega \subseteq \mathbb{R}^d & \rightarrow & \mathbb{L}^2(\mathcal{Z}) \\ & & \mathbf{x} & \mapsto & y_{\mathbf{x}}(\mathbf{z}) \end{array} \tag{1}$$

We assume that we know $n \in \mathbb{N}^\star$ simulations of f: $\{(\mathbf{x}_i, y_i(\mathbf{z})), i = 1, \ldots, n\}$, with $y_i(.) = y_{\mathbf{x}_i}(.)$. We aim at predicting the map $f(\mathbf{x}^\star)$ for a new point $\mathbf{x}^\star$.

The functions of GpOutput2D come from the PhD work of Perrin et al. [2020]. The main lines of the method are:

1. To reduce the infinite dimension of $\mathbb{L}^2(\mathcal{Z})$ to a finite dimensional space of size $K \in \mathbb{N}^\star$, by representing the functional output by an orthonormal basis functions, which is denoted $\Phi(\mathbf{z}) = (\phi_1(\mathbf{z}), \ldots, \phi_K(\mathbf{z}))^\top$:

$$y_{\mathbf{x}}(\mathbf{z}) = \sum_{k=1}^{K} \alpha_k(\mathbf{x})\phi_k(\mathbf{z}) = \boldsymbol{\alpha}(\mathbf{x})^\top \Phi(\mathbf{z}) \tag{2}$$

with $\boldsymbol{\alpha}(\mathbf{x})$ the coefficients vector of $y_{\mathbf{x}}(.)$ on $\Phi(\mathbf{z})$.

2. To truncate the number of basis coefficients, by keeping those which influence the most the energy.
3. To apply a standard multivariate PCA to the selected coefficients.
4. To build GP models for each principal component score (or coordinate).
5. To predict the scores of $y_{\mathbf{x}^\star}(.)$ on principal components, by using their associated kriging models.
6. To predict $y_{\mathbf{x}^\star}(.)$ by embedding the estimated scores into the initial functional space: $\mathbb{L}^2(\mathcal{Z})$.

An other **R** package exists, called FPCA2D, with functions for performing FPCA on two-dimensional functional data (even for three-dimensional functional data, with FPCA3D). However, the only implemented basis functions is the Fourier basis. Furthermore, we do not have control over the Fourier decomposition. In GpOutput2D, other basis functions have been implemented: two-dimensional wavelet and B-splines basis. B-splines basis is not orthonormal. Then, it can be orthonormalized by performing Gram-Schmidt method [Björck, 1994], or specific procedures for splines [Liu et al., 2019, Qin, 2000, Redd, 2012].

Wavelet basis is commonly used in image processing [Mallat, 1999]. Indeed, a key advantage over Fourier transform is the capture of both frequency and location information. Spline functions are piecewise polynomials. They are commonly chosen for approximation of non-periodic functional data [Ramsay, 2006, Ramsay and Silverman, 2007]. Basis system have been developed for spline functions. The B-splines basis has been used in the package. GpOutput2D depends on waveslim package, for two-dimensional wavelet transform, and on orthogonalsplinebasis package, for two-dimensional B-splines basis (and for its orthonormalization).

To approximate at best the functional data, the basis dimension K may be high dimensional. Therefore, a preliminary selection step is added, by choosing the $\tilde{K}$ basis terms which are most influential based on the energy decomposition (see [Perrin et al., 2020]. Then, PCA is applied to them. The non-selected basis terms are estimated by empirical mean. With this idea, the method can be performed on large dimensional vectors, which contain basis coefficients.

The construction of GP models, for each principal component score, is based on km of DiceKriging package, or gp of kergp package. Both functions fit kriging model on data. The difference is that kergp lets the user to define his own covariance kernel. In particular, building a GP metamodel is possible for models which have categorical input variables [Roustant et al., 2020]. However, kergp is a laboratory package, and may evolve in its future. Therefore, GpOutput2D also depends on DiceKriging, for user which are interested in stable software.

The main functionalities of GpOutput2D are implemented as S3 methods. They are shown in Table 1.

Method Name	Description
Fpca2d	Applying FPCA to two-dimensional functional data.
km_Fpca2d	Building a kriging models on each score obtained by using **Fpca2d**, with the **km** function from **DiceKriging** package.
gp_Fpca2d	Building a kriging models on each score obtained using **Fpca2d**, with the **gp** function from **kergp** package.
predict	Prediction of the two-dimensional functional output of the objective function at a new point using GP model on scores of FPCA.

Table 1: Main functions of GpOutput2D package.

2 An analytical test case

Here, we illustrate how GpSpatialOutput works on a 2D toy example. The toy function, called Campbell2D, has eight scalar inputs ($d = 8$) and a spatial map as output (e.g. a function which depends on two inputs ($\mathbf{z} = (z_1, z_2)$) corresponding to spatial coordinates). The Campbell2D function was first introduced by Marrel et al. [2010]. (3) shows the model, and (4) is the formula of the spatial output.

$$\begin{array}{rccc} f & : & [-1,5]^8 & \to & \mathbb{L}^2([-90,90]^2) \\ & & \mathbf{x} = (x_1, \ldots, x_8) & \mapsto & y_{\mathbf{x}}(\mathbf{z}) \end{array} \tag{3}$$

where $\mathbf{z} = (z_1, z_2) \in [-90, 90]^2$, $x_j \in [-1, 5]$ for $j = 1, \ldots, 8$, and

$$\begin{array}{rl} y_{\mathbf{x}}(z_1, z_2) = & x_1 \exp\left[-\frac{(0.8z_1+0.2z_2-10x_2)^2}{60x_1^2}\right] + (x_2 + x_4) \exp\left[\frac{(0.5z_1+0.5z_2)x_1}{500}\right] + \\ & x_5(x_3 - 2) \exp\left[-\frac{(0.4z_1+0.6z_2-20x_6)^2}{40x_5^2}\right] + \\ & (x_6 + x_8) \exp\left[\frac{(0.3z_1+0.7z_2)x_7}{250}\right] \end{array} \tag{4}$$

This function has been implemented in GpSpatialOutput. For the application, the spatial domain $[-90, 90]^2$ is discretized on an uniform grid of dimension $n_{\mathbf{z}} \times n_{\mathbf{z}}$, with $n_{\mathbf{z}} = 64$.

```
> # inputs of the Campbell2D function
> x1<-rep(-1,8); x2<-rep(5,8); x3<-c(5,3,1,-1,5,3,1,-1)
> X <- rbind(x1,x2,x3) # inputs
> #
> # spatial domain
> nz<-64 # nz^2 is the size of the spatial domain
> z<-seq(-90,90,length=nz) # spatial coordinates
> #
> # Campbell2D function
> Y = Campbell2D(X,z,z) # outputs
```

We use the raster package to plot spatial data. A rotation matrix function, called rot90, will be performed on matrix, in order to get maps in the right rotation in a raster object.

```
> library(raster) # Geographic Data Analysis
> #
```

```
> # a raster model
> rst <- raster(xmn = -90, xmx = 90, ymn = -90, ymx= 90,
+               nrows=nz,ncols=nz)
> #
> # matrix rotation function
> rot90 <- function(x){(apply(t(x), 2, rev))}
```

Figure 1 shows examples of Campbell2D outputs. The output map presents strong spatial heterogeneities, sometimes with sharp boundaries. Furthermore, the spatial distribution is different according to the $\mathbf{x}$ values.

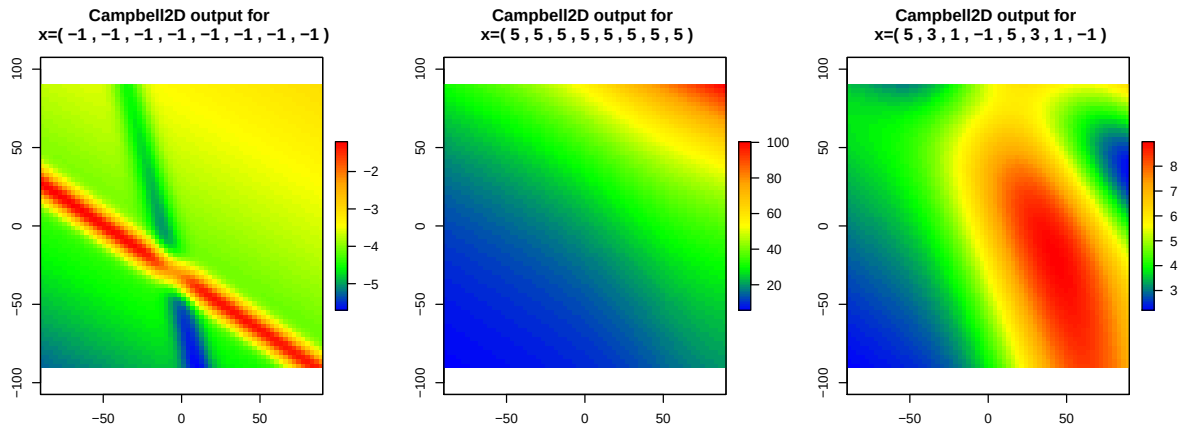

Figure 1: Example of Campbell2D outputs. From left to right, inputs are $\mathbf{x} = (-1,-1,-1,-1,-1,-1,-1,-1)$, $\mathbf{x} = (5,5,5,5,5,5,5,5)$, and $\mathbf{x} = (5,3,1,-1,5,3,1,-1)$.

3 Functional Principal Component Analysis (FPCA)

Fpca2d is a wrapper function to perform FPCA on two-dimensional functional data (images, maps, etc.), given a projection method. The two implemented functional basis are : orthonormal B-splines and wavelet.

To illustrate how to use FPCA, a learning sample of size $n = 200$ is considered, with a space-filling design of experiment (doe), which is constructed by using Latin Hypercube Sampling (LHS) design, and optimized by the SA algorithm [Dupuy et al., 2015], (implemented on the DiceDesign R package).

```
> library(lhs) # Latin Hypercube Sample
> library(DiceDesign) # Design of Computer Experiments
> #
> # design of experiment
> n<-200 # size of the learning sample
> doe <- maximinLHS(n=n,k=8) # LHS design
> doe_learn <-maximinSA_LHS(doe)$design # optimized design
> #
> # range of Campbell2D inputs
> rg <- c(-1,5)
```

```
> doe_learn<-doe_learn*(rg[2]-rg[1]) + rg[1]
> #
> # Output Spatial domain
> nz<-64
> z <- seq(-90,90,length=nz)
> #
> # Campbell2D output
> Y<-Campbell2D(doe_learn,z,z)
```

3.1 Functional basis decomposition

GpOutput2D depends on waveslim package for wavelet decomposition, by using dwt.2d function [Mallat, 1999, Vetterli and Kovacevic, 1995].

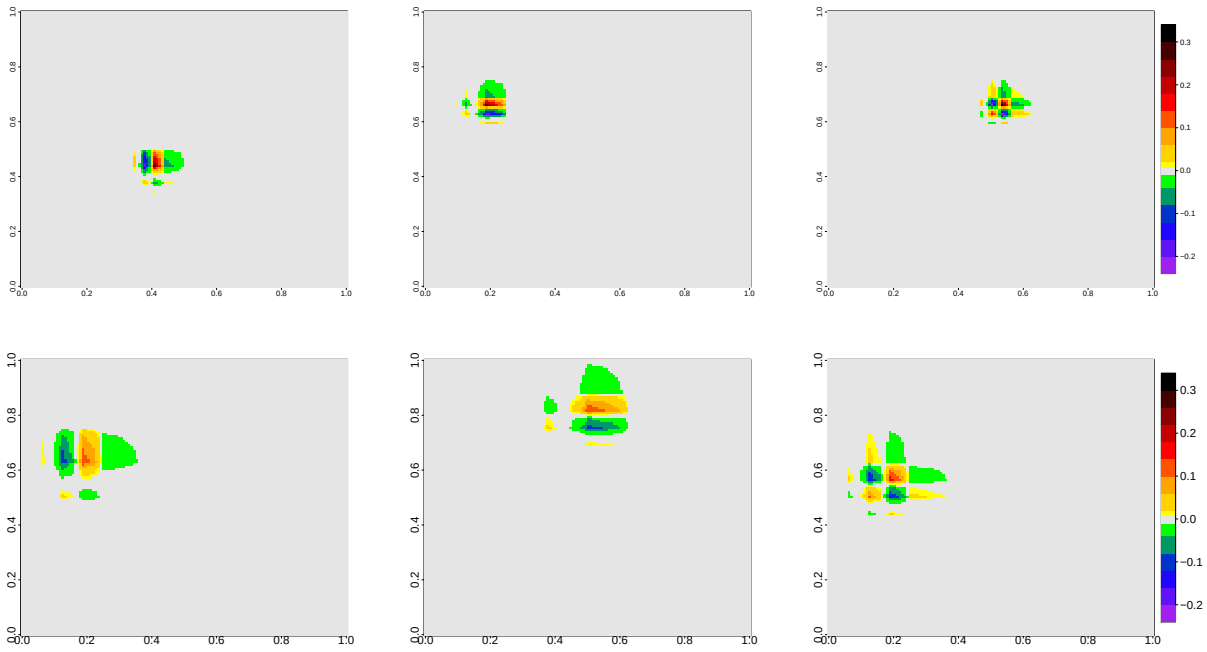

Figure 2: Examples of D4 Daubechies wavelets on $[0,1]^2$, which is discretized into a grid of size 128×128. From left to right, the figures represent examples of horizontal, vertical and diagonal wavelets. The top figures are wavelets at scale 3. The bottom figures are wavelets at scale 4.

In GpOutput2D, we also propose to use B-splines basis to approximate data. However, the truncation of the number of coefficients for PCA needs to work on orthonormal basis [Perrin et al., 2020]. Therefore, OrthoNormalBsplines2D builds a B-splines basis, which is orthonormalized according to a given orthogonalization method. The available methods are "GS", for the Gram-Schmidt method [Björck, 1994], and "Redd", for a specific method for B-splines, which is based on matrix representation of the functional basis [Redd, 2012]. We detail here how OrthoNormalBsplines2D builds the B-splines basis.

The orthonormalized B-splines is build as follows :

1. Two 1D B-splines basis are built for each dimension of the spatial domain. They are denoted $\boldsymbol{\psi}(\mathbf{z}) = (\psi_1(\mathbf{z}), \ldots, \psi_K(\mathbf{z}))$ and $\boldsymbol{\psi}'(\mathbf{z}) = (\psi'_1(\mathbf{z}), \ldots, \psi'_{K'}(\mathbf{z}))$, with $K, K' \in \mathbb{N}$.

2. $\boldsymbol{\psi}(\mathbf{z})$ and $\boldsymbol{\psi}'(\mathbf{z})$ are orthogonalized. If ortho = "GS", the standard Gram-Schmidt method is performed by using the gramSchmidt function from the pracma package. If ortho = "Redd", the ortogonalization is performed by using the orthogonalsplinebasis package. The method consists to represent the B-splines basis in matrix form, and to perform a matrix product with the root of the Gram matrix $\mathbf{G} = (\int \psi_k(\mathbf{z})\psi'_k(\mathbf{z})d\mu(\mathbf{z}))_{1 \leq k,k' \leq K}$ (see [Redd, 2012]). The new basis are respectively denoted $\boldsymbol{\psi}_\perp(\mathbf{z})$ and $\boldsymbol{\psi}'_\perp(\mathbf{z})$. The default orthogonalization method is "GS".

3. $\boldsymbol{\psi}_\perp(\mathbf{z})$ and $\boldsymbol{\psi}'_\perp(\mathbf{z})$ are then normalized :

$$\tilde{\boldsymbol{\psi}}(\mathbf{z}) = \frac{\boldsymbol{\psi}_\perp(\mathbf{z})}{||\boldsymbol{\psi}_\perp(\mathbf{z})||_2} \quad \text{and} \quad \tilde{\boldsymbol{\psi}}'(\mathbf{z}) = \frac{\boldsymbol{\psi}'_\perp(\mathbf{z})}{||\boldsymbol{\psi}'_\perp(\mathbf{z})||_2}$$

4. We consider the 2D basis which is obtained by tensorization,

$$\phi_{k,l}(\mathbf{z})) = \tilde{\boldsymbol{\psi}}_k(\mathbf{z})\tilde{\boldsymbol{\psi}}'_l(\mathbf{z}),\ k = 1, \ldots K,\ l = 1, \ldots, K'$$

The following code show how to build an orthonormal B-splines basis.

```
> #=======================================
> # To build an orthonormal B-splines basis
> #=======================================
>
> z.knots <-seq(-90,90,length=35)# knots for the B-splines basis
> OPhi_GS <- OrthoNormalBsplines2D(z,z,z.knots,z.knots,ortho="GS")
> OPhi_Redd <- OrthoNormalBsplines2D(z,z,z.knots,z.knots,ortho="Redd")

> #===============
> # Raster model
> #===============
>
> library(raster)
> # raster model
> rst <- raster(xmn=-90,xmx=90,ymn=-90,ymx=90,
+               nrows=64,ncols=64)
> # rotation matrix function
> rot90 <- function(x){apply(t(x),2,rev)}

> #======================
> # plot functions of OPhi
> #======================
>
> # indices of the OPhi basis
> k<-c(10,20,30)
> l<-c(5,15,25)
> par(mfrow=c(2,3),mar=(c(3,4,4,2)+0.1))
```

```
> # with ortho="GS"
> for(i in 1:3){
+   rst[]<- rot90(OPhi_GS[,k[i],,l[i]])
+   plot(rst, col = rainbow(128), horizontal=TRUE,
+        zlim=c(-0.19,0.681),legend.width=1.5,
+        main="orthogonalization \n with Gram-Schmidt")
+ }# end for i
> # with ortho="Redd"
> for(i in 1:3){
+   rst[]<- rot90(OPhi_Redd[,k[i],,l[i]])
+   plot(rst, col = rainbow(128), horizontal=TRUE,
+        zlim=c(-0.19,0.681),legend.width=1.5,
+        main="orthogonalization \n with orthogonalsplinebasis")
+ }# end for i
```

Figures 3 and 4 show examples of orthonormal basis functions, respectively by using ortho ='GS', and ortho ='Redd'. Basis functions seems similar. However, after performing a substraction of both basis (see Figure 5), we see a difference in range.

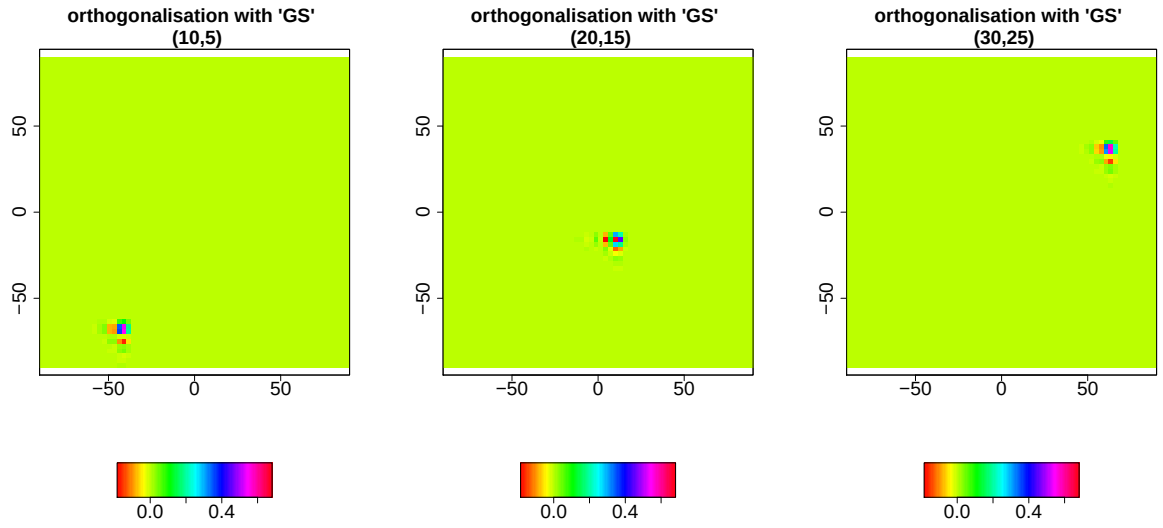

Figure 3: Example of functions of the orthonormal B-splines basis, which are obtained with the Gram-Schmidt method. From left to right, coordinates in the B-splines basis are : $(10, 5)$, $(20, 15)$, $(30, 25)$.

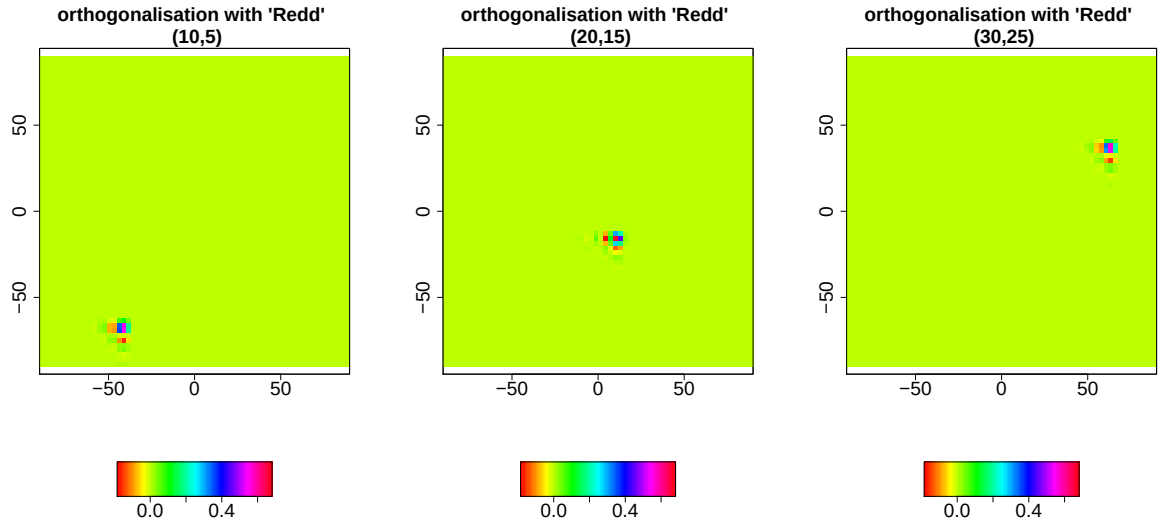

Figure 4: Example of functions of the orthonormal B-splines basis, which are obtained with the orthogonalsplinebasis package. From left to right, coordinates in the functional basis are : $(10, 5)$, $(20, 15)$, $(30, 25)$.

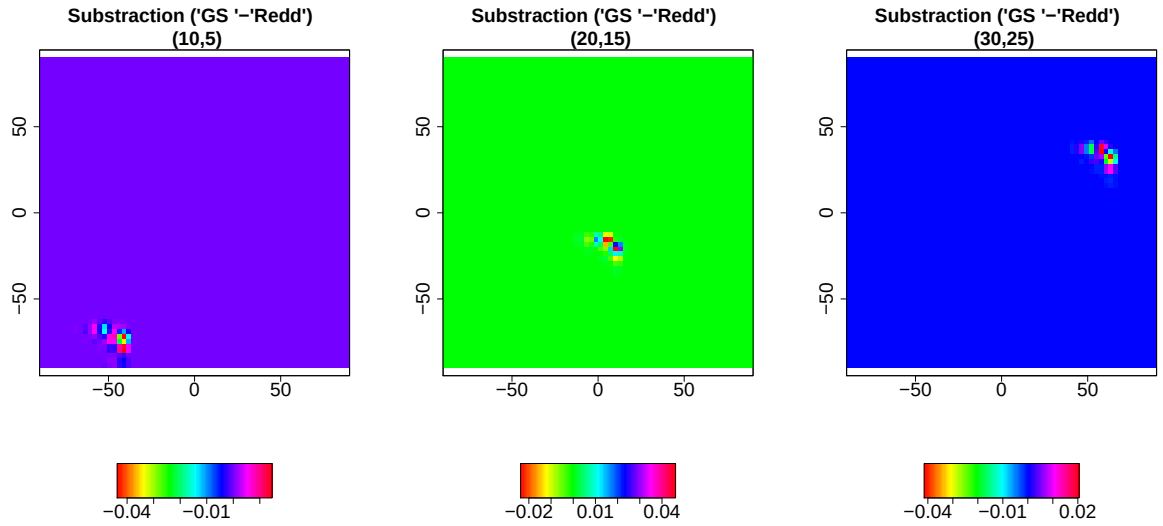

Figure 5: Substraction of orthonormal B-splines basis functions obtained by 'GS' and by 'Redd'. From left to right, coordinates in the functional basis are : $(10, 5)$, $(20, 15)$, $(30, 25)$.

3.2 Application of FPCA

According to Ramsay [2006], after performing wavelet or orthonormal B-splines decompositions, FPCA is equivalent to apply a standard multivariate PCA on the coefficients. In order to speed up FPCA, standard PCA is applied on the most "important" coefficients, which are selected according to their associated mean part of energy, as in Perrin et al. [2020]. Therefore, in Fpca2d, the total mean proportion of energy, called p, can be fixed. The number of coefficients is then calibrated according to its value. Otherwise, instead of giving p, the number of coefficients can be directly given. The basis parameter allows to choose the projection method : "Wavelet", for wavelet basis, and "Bsplines", for orthonormal B-splines.

```
> # parameters for B-splines
> K<-35
> z.breaks <- seq(-90,90,length=K) #knots fot B-splines
> norder<-2 # B-splines order
> # parameter for wavelet decomposition
> J<-1# depth of the decomposition
> wf<-"d4" # type of wavelets, here Daubechies 4
> # number of principal components
> nPC <- 5
> #### FPCA ####
>
> # using wavelets
> FPCA_dwt <- Fpca2d(method = "Wavelets", x=Y,
+                    J=J,wf=wf, # wavelet parameters
+                    rank.=nPC,ncoeff=4000)
> # using Bsplines
> FPCA_bs <- Fpca2d(method = "Bsplines", x=Y,
+                    z1=z, z2=z, z1.knots=z.breaks, z2.knots=z.breaks,
+                    norder=2, # B-splines parameters
+                    rank.=nPC,p=1)
```

A S3 method has been implemented, in order to plot characteristics of FPCA. Different plots can be done according to the specified type : c("inertia", "energy", "MeanPoe", "eigenfunctions"). For "inertia" and "eigenfunctions", the argument PC allows to specify the plotted principal components by giving a vector with theirs numbers. The parameter "inertia" gives a barplot (see Figure 7), which illustrates the proportion of explained variance of each principal component. "eigenfunctions" gives the eigenfunction images (see Figure 6). "MeanPoe" corresponds to an image of coefficients (from wavelet or B-splines basis) mean proportion of energy (see Figure 8). The parameter "energy" gives a barplot (see Figure 9), which shows the number of coefficients used on PCA according to the total mean proportion of energy. The given percentages correspond to the percentage of selected coefficients.

```
> par(mfrow=c(1,3))
> plot(z1=z,z2=z,FPCA_dwt,type="eigenfunctions",PC=1:3)

> plot(FPCA_dwt,type="inertia",PC=1:nPC)

> plot(FPCA_dwt,type="MeanPoe")

> plot(FPCA_dwt,type="energy",p=seq(0.5,1,by=0.05))
```

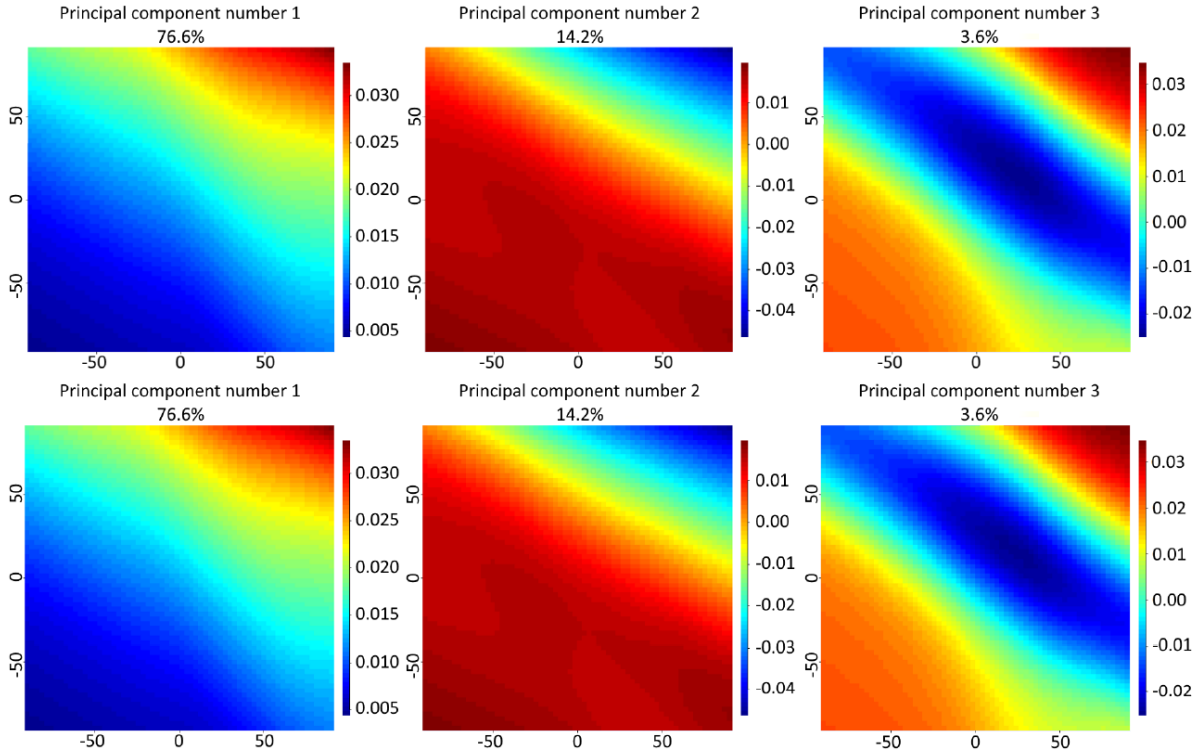

Figure 6: From left to right, the three first eigenfunctions, which correspond respectively to 76, 6%, 14.2%, and 3.6% of the explained variance (same for both lines). First line correspond to the eigenfunctions of FPCA by using wavelet basis. The second line correspond to those obtained by using an orthonormal B-splines basis (with ortho ='GS').

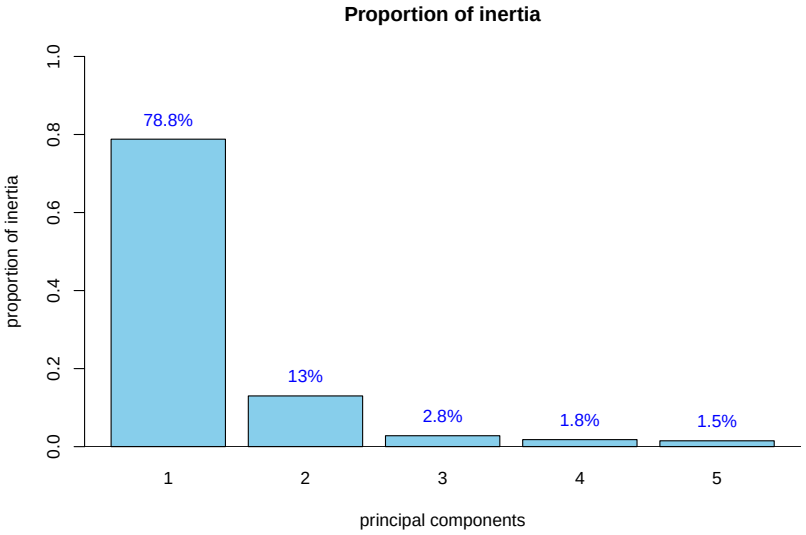

Figure 7: Proportion of explained variance of the five first principal components.

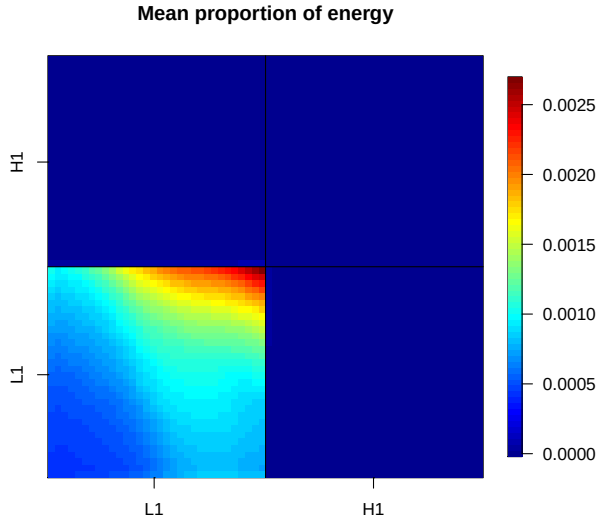

Figure 8: Mean proportion of energy of wavelet coefficients, with the depth of decomposition at 1. "L1" corresponds to scale function on x-axis (horizontal) or y-axis (vertical). "H1" corresponds to wavelet function on x-axis (horizontal) or y-axis (vertical). Square at the bottom left of the figure corresponds to coefficients associated with the 2D scale function. Squares at the top left, the top right and the figures' bottom right correspond to coefficients, respectively associated with vertical, diagonal, and horizontal wavelet (translation direction).

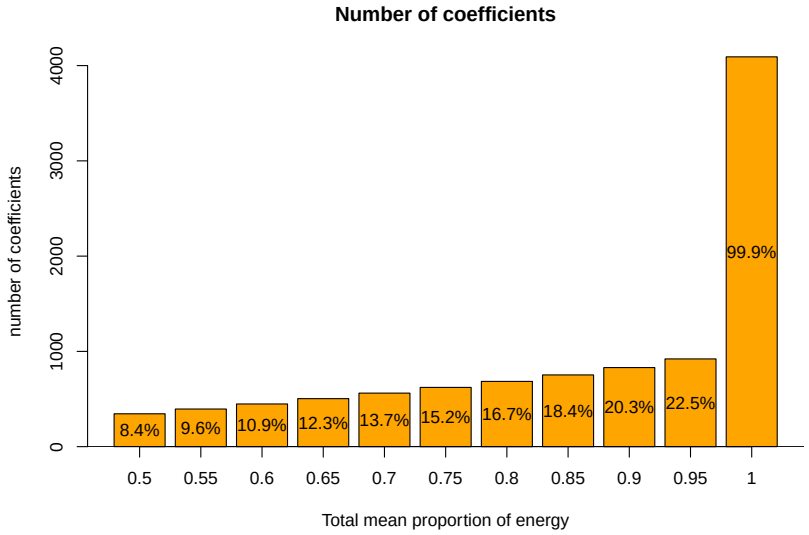

Figure 9: Number of coefficients, which are used on PCA, according to the total mean proportion of energy. The given percentages correspond to the percentage of selected coefficients.

4 Prediction

4.1 Fitting Gaussian Process Regression models

In GpOutput2D, two functions, called km_Fpca2d and gp_Fpca2d, fit Gaussian process (GP) regression model (also called kriging model) on each principal component, which are obtained by Fpca2d. km_Fpca2d uses the km function from DiceKriging package. gp_Fpca2d uses the gp function from kergp package. Both packages allow to create GP models.

kergp is a Laboratory package which performs GP interpolation with an emphasis on user-defined covariance kernels. Furthermore, categorical variables can also be treated as model inputs. However, kergp may evolve in the future. Users interested in stable software for the Analysis of Computer Experiments can used instead DiceKriging package.

km_Fpca2d

```
> # design of experiment in data.frame
> colnames(doe_learn)<-paste("x",1:8,sep="")
> doe_learn<-data.frame(doe_learn)
> # using wavelet basis
> mw <- km_Fpca2d(design=doe_learn,response=FPCA_dwt,control=list(trace=FALSE))
> # using othonormalized B-splines basis
> mB <- km_Fpca2d(design=doe_learn,response=FPCA_bs,control=list(trace=FALSE))
```

gp_Fpca2d

For the example, we use the same scale and variance parameters as estimated in km_Fpca2d. In order to assign scale and variance parameters on each principal component kriging model, a list of kernel function is given in gp_Fpca2d.

```
> library(doFuture)
> #============================
> #   By using wavelet basis
> #============================
>
> ## kernel for each principal component
> myCov <- c()
> for(l in 1:nPC){
+   # model
+   cov_ml <- mw[[l]]@covariance
+
+   #kernel
+   kl<-covTS(inputs = colnames(doe_learn),
+           kernel = "k1Matern5_2",
+           dep = c(range = "input"))
+
+   # allocation of scale and variance parameters
+   coef(kl)<-c(range = cov_ml@range.val, sigma2 = rep(cov_ml@sd2,8))
+
+   myCov <- c(myCov,list(kl) )
+ } # end for l
> ## model by using wavelet basis
>
> gp_w<- gp_Fpca2d(design=doe_learn, response=FPCA_dwt, cov=myCov,estim=FALSE)
> #============================
> #   By using B-splines basis
> #============================
>
> ## kernel for each principal component
> myCov <- c()
> for(l in 1:nPC){
+   # model
+   cov_ml <- mB[[l]]@covariance
+
+   #kernel
+   kl<-covTS(inputs = colnames(doe_learn),
+           kernel = "k1Matern5_2",
+           dep = c(range = "input"))
+
+   # allocation of scale and variance parameters
+   coef(kl)<-c(range = cov_ml@range.val, sigma2 = rep(cov_ml@sd2,8))
+
+   myCov <- c(myCov,list(kl) )
+ } # end for l
```

```
> ## model by using B-splines basis
> gp_B<- gp_Fpca2d(design=doe_learn, response=FPCA_bs, cov=myCov,estim=FALSE)
```

4.2 Prediction

To analyse prediction accuracy, a test sample of size $n_{test} = 1000$ has been built with inputs randomly chosen according to a uniform distribution $\mathcal{U}([-1, 5]^8)$.

```
> ntest<-1000
> # inputs
> NewX <- matrix(runif(ntest*8,min=-1,max=5),ncol=8)
> # outputs
> Ytest <- Campbell2D(NewX,z,z)
> # inputs in data.frame
> colnames(NewX)<-colnames(doe_learn)
> NewX <-data.frame(NewX)
```

A S3 method predict has been developed for km_Fpca2d and gp_Fpca2D. By specifying compute = FALSE, only the kriging mean is computed. If TRUE, the kriging variance (here, the standard deviation is returned) and confidence intervals are computed too. Here, compute is TRUE only for the model built by km_Fpca2d with wavelet basis.

```
> #============================
> #   By using wavelet basis
> #============================
>
> # By using km_Fpca2d
> pw_km <- predict(mw,newdata=NewX,type="UK")
> # By using gp_Fpca2d
> pw_gp <- predict(gp_w,newdata=NewX,type="UK", compute = FALSE)
>
> #============================
> #   By using B-splines basis
> #============================
>
> # By using km_Fpca2d
> pB_km <- predict(mB,newdata=NewX,type="UK", compute = FALSE)
> # By using gp_Fpca2d
> pB_gp <- predict(gp_B,newdata=NewX,type="UK", compute = FALSE)
```

4.3 An example of prediction

```
> i = 20 # example simulation
> par(mfrow=c(1,3))
> # z-axis limit
> zlims <- c(7.9,27.5)
> # number of colors
> nlevel=20
> # real matrix
> rst[]<-rot90(Ytest[,,i])
```

```
> plot(rst,col=rainbow(nlevel),zlim=zlims,legend.width=2,
+      main="Real map")
> # prediction
> rst[]<-rot90(pw_km$mean[,,i])
> plot(rst,col=rainbow(nlevel),zlim=zlims,legend.width=2,
+      main="Estimated map")
> # kriging variance
> library(RColorBrewer)
> color <-colorRampPalette(c("white","yellow","orange","red","darkred","black"))
> rst[]<-rot90(pw_km$sd[,,i]**2)
> plot(rst,col=color(nlevel),legend.width=2,
+      main="kriging variance")
> # 95% confidence interval
> par(mfrow=c(1,2))
> ## lower bound
> rst[]<-rot90(pw_km$lower95[,,i])
> plot(rst,col=rainbow(nlevel),zlim=zlims,legend.width=2,
+      main="95% confidence interval (lower bound)")
> ## upper bound
> rst[]<-rot90(pw_km$upper95[,,i])
> plot(rst,col=rainbow(nlevel),zlim=zlims, legend.width=2,
+      main="95% confidence interval (upper bound)")
>
```

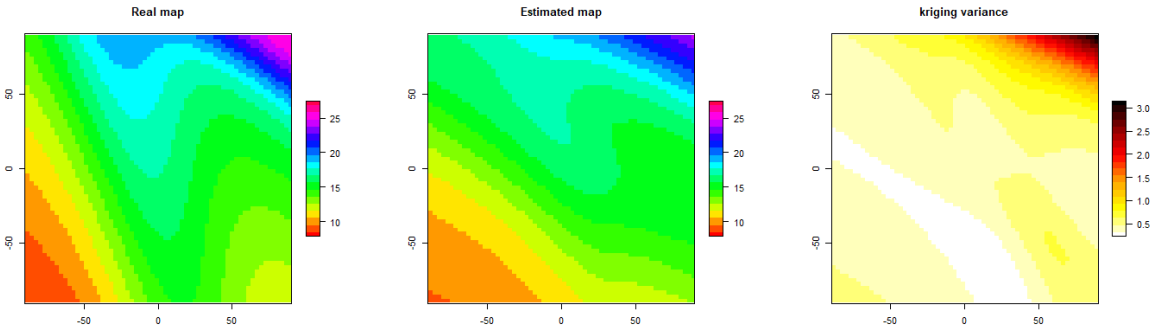

Figure 10: Example of estimated matrix. From left to right : the real matrix, the estimated matrices (which is obtained by km_Fpca2d with wavelet basis), and the prediction variance (it can be assimilated to the kriging variance).

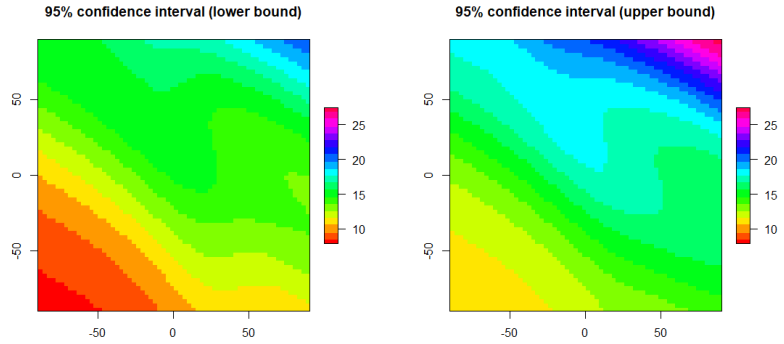

Figure 11: From left to right, lower and upper bound of the 95% confidence interval of the estimated matrix (see Figure 10)

4.4 Prediction accuracy

The function error.predict measures the prediction accuracy by computing the spatial RMSE and Q^2 as follows :

$$\text{RMSE}(\mathbf{z}) = \sqrt{\frac{1}{n_{test}} \sum_{i=1}^{n_{test}} \left(y_i(\mathbf{z}) - \hat{y}_i(\mathbf{z})\right)^2}$$

$$Q^2(\mathbf{z}) = 1 - \frac{\frac{1}{n_{test}} \sum_{i=1}^{n_{test}} (y_i(\mathbf{z}) - \hat{y}_i(\mathbf{z}))^2}{\frac{1}{n_{test}} \sum_{i=1}^{n_{test}} (y_i(\mathbf{z}) - \overline{y}(\mathbf{z}))^2}$$

with $\hat{y}_i(.)$, the estimation of $y_i(.)$, and $\overline{y}(\mathbf{z}) = \frac{1}{n} \sum_{i=1}^{n} y_i(\mathbf{z})$. By specifying rtx.scores=FALSE, the prediction accuracy of each principal component kriging model is also measured by RMSE and Q^2. The rtx.scores default is FALSE. To get RMSE and Q^2, we must give the real matrices, the predictions, and the Fpca2d object used to build the kriging models. For the example, error.predict is run on the prediction obtained with km_Fpca2d, for wavelet and B-splines basis. Figure 12 shows the RMSE and Q^2 matrices of both metamodels.

```
> #=========
> # wavelet
> #=========
> err_pw <- error.predict(Ytest,pw_km,FPCA_dwt,rtx.scores=TRUE)
> # Prediction accuracy of score estimations
> print(err_pw$scores$rmse) # RMSE

      PC1       PC2       PC3       PC4       PC5
19.970950 15.373994 11.868038  8.427110  9.718261

> print(err_pw$scores$Q2) # Q2

      PC1       PC2       PC3       PC4       PC5
0.9944059 0.9779924 0.9478785 0.9552067 0.9454740

> #============
> # B-splines
> #============
```

```
> err_pB <- error.predict(Ytest,pB_km,FPCA_bs,rtx.scores=TRUE)
> # Prediction accuracy of score estimations
> print(err_pB$scores$rmse) # RMSE

     PC1      PC2      PC3      PC4      PC5
19.97097 15.37399 11.86802  8.42704  9.71815

> print(err_pB$scores$Q2) # Q2

      PC1       PC2       PC3       PC4       PC5
0.9944059 0.9779924 0.9478786 0.9552073 0.9454752

> #==================
> # RMSE and Q2 matrices
> #==================
>
> # RMSE
> par(mfrow=c(1,2))
> zlims=c(0.31,2.38)
> ## Fpca-wavelet
> rst[]<-rot90(err_pw$y$rmse)
> plot(rst, zlim=zlims, col=rainbow(100),
+      main="RMSE with wavelet basis",legend.width=2)
> ## Fpca-Bsplines
> rst[]<-rot90(err_pB$y$rmse)
> plot(rst, zlim=zlims,col=rainbow(100),
+       main="RMSE with B-splines basis",legend.width=2)
> # Q2
> par(mfrow=c(1,2))
> zlims=c(0.83,0.99)
> color<-colorRampPalette(c("orange","yellow","green","darkgreen","black"))
> ## Fpca-wavelet
> rst[]<-rot90(err_pw$y$Q2)
> plot(rst, zlim=zlims, col=color(100),
+      main="Q2 with wavelet basis",legend.width=1.5)
> ## Fpca-Bsplines
> rst[]<-rot90(err_pB$y$Q2)
> plot(rst, zlim=zlims,col=color(100),
+       main="Q2 with B-splines basis",legend.width=1.5)
>
```

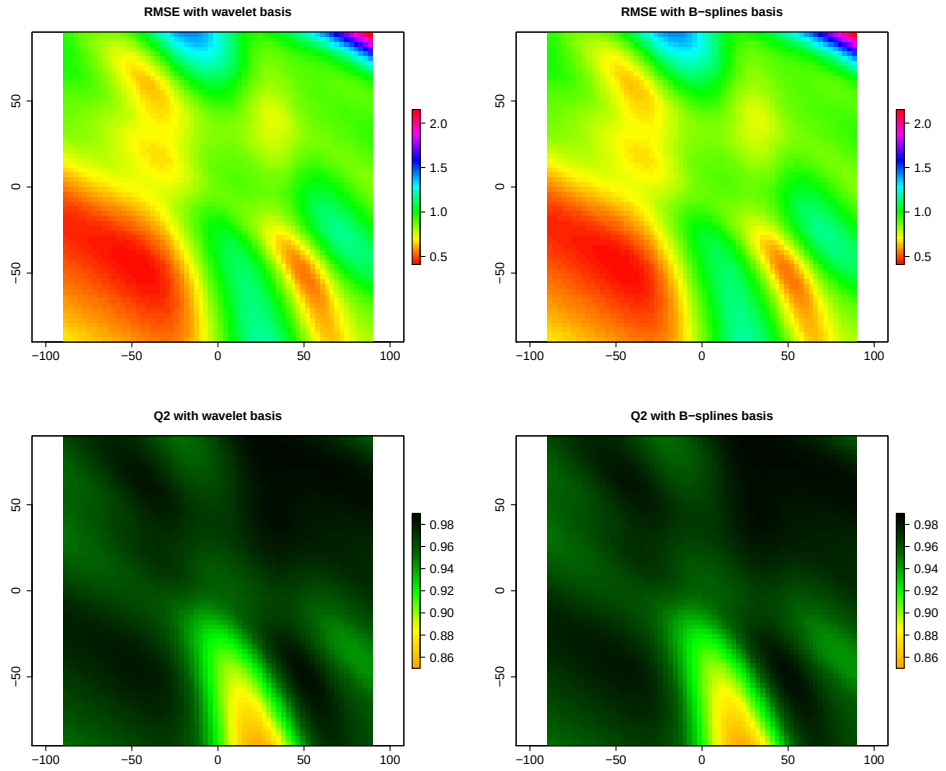

Figure 12: From left to right, the results for FPCA based on wavelets and for FPCA based on B-splines. The top figures are the RMSE matrices. The bottom figures are the Q^2 matrices.

References

Björck, Å. (1994). Numerics of gram-schmidt orthogonalization. *Linear Algebra and Its Applications*, 197:297–316.

Dupuy, D., Helbert, C., Franco, J., et al. (2015). Dicedesign and diceeval: Two r packages for design and analysis of computer experiments. *Journal of Statistical Software*, 65(11):1–38.

Liu, X., Nassar, H., and Podgórski, K. (2019). The ob-splines–efficient orthonormalization of the b-splines.

Mallat, S. (1999). *A wavelet tour of signal processing*. Elsevier.

Marrel, A., Iooss, B., Jullien, M., Laurent, B., and Volkova, E. (2010). Global sensitivity analysis for models with spatialy dependent outputs. *Environmetrics*, 22(3):383–397.

Perrin, T., Roustant, O., Rohmer, J., Alata, O., Naulin, J., Idier, D., Pedreros, R., Moncoulon, D., and Tinard, P. (2020). Functional principal component analysis for global sensitivity analysis of model with spatial output. *arXiv preprint arXiv:2005.10285*.

Qin, K. (2000). General matrix representations for b-splines. *The Visual Computer*, 16(3-4):177–186.

Ramsay, J. O. (2006). *Functional data analysis*. Wiley Online Library.

Ramsay, J. O. and Silverman, B. W. (2007). *Applied functional data analysis: methods and case studies*. Springer.

Redd, A. (2012). A comment on the orthogonalization of b-spline basis functions and their derivatives. *Statistics and Computing*, 22(1):251–257.

Roustant, O., Padonou, E., Deville, Y., Clément, A., Perrin, G., Giorla, J., and Wynn, H. (2020). Group kernels for gaussian process metamodels with categorical inputs. *SIAM/ASA Journal on Uncertainty Quantification*, 8(2):775–806.

Vetterli, M. and Kovacevic, J. (1995). *Wavelets and subband coding*. Number BOOK. Prentice-hall.

Williams, C. K. I. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*. MIT Press Cambridge, MA.