Gaussian Process Function Data Analysis

R Package 'GPFDA'

Jian Qing Shi & Yafeng Cheng
School of Mathematics and Statistics
Newcastle University
Newcastle NE1 7RU, UK
E-mail: j.q.shi@ncl.ac.uk & yafeng.cheng@ncl.ac.uk

October 14, 2014

Gaussian Process Function Data Analysis

R Package 'GPFDA', Version 1.1

This version includes

- Gaussian process regression analysis for a single curve, and
- Gaussian process functional regression analysis for repeated curves

More will be added shortly in the next version, including

- Gaussian process classification and clustering
- Mixture Gaussian process functional regression models

This manual details how to install the package and how to use the package to conduct Gaussian process regression analysis for functional data of a single batch (single curve) or multiple batches (repeated curves). The detailed description can be found in Shi et al. (2007) andShi and Choi (2011).

Contents

Chapter 1

Introduction

1.1 Installation

The package **GPFDA** can be installed using the usual way in **R**. The simplest way is to install it from the repository of CRAN together with all dependency packages by

```
install.packages('GPFDA', dependencies=T)
```

You can also install the package from the source file locally end with '.tar.gz' in both Linux-like system and Windows system by

```
install.packages('GPFDA', repos=NULL)
```

Package path is omitted here. You need to add the path by yourself. Alternatively, you can use the following way

```
install.packages(file.choose(), repos=NULL)
```

It will open a window. You can then choose the source file from the related folder.

The package used several R dependency packages:

- fda.usc (Functional Data Analysis and Utilities for Statistical Computing , Febrero-Bande and de la Fuente (2012))
- fda (functional data analysis, Ramsay and Silverman (2005). Involved in package fda.usc)
- data.table (fast data frame indexing, ordering etc.)
- spam (sparse matrix calculation)

- MASS (basic functions from 'Modern Applied Statistics with S', Venables and Ripley (2002), used in some examples and demo's)

All those packages need to be installed or updated as the dependencies of **GPFDA** . This will be done automatically if it is installed from repository (CRAN). But you need to install them manually if you install **GPFDA** locally from source file.

1.2 Load the package

To load the package in **R** , use

```
library(GPFDA)
```

or

```
require(GPFDA)
```

To update the package, click the 'package' icon in menu, and select 'update packages'; or reinstall the package (need to detach the package by using R command `detach(package:GPFDA)`, if the package is loaded into the workspace).

Chapter 2

Gaussian Process Regression Analysis

2.1 Gaussian process regression model and inference

Let y be a response variable and $\boldsymbol{x}$ be a Q-dimensional covariates. A nonparametric regression model is expressed as

$$y = f(\boldsymbol{x}) + \epsilon, \quad \epsilon \sim N(0, \sigma^2),$$

where $f(\cdot)$ is unknown. However, most of the nonparametric methods suffer from the *curse of dimensionality* when they are applied to the problem with multi-dimensional covariates (i.e. Q is large). A variety of alternative approaches has been developed to overcome this problem. Examples include the additive model, the projection pursuit regression, the sliced inverse regression, the neural network model, the varying-coefficient model and Gaussian process regression (GPR) model.

Gaussian process regression model is a nonparametric model and has some nice features; see the details in Shi and Choi (2011). Suppose we have a data set

$$\mathcal{D} = \left\{ \begin{pmatrix} y_1 \\ \mathbf{x}_1 \end{pmatrix} \begin{pmatrix} y_2 \\ \mathbf{x}_2 \end{pmatrix} \cdots \begin{pmatrix} y_n \\ \mathbf{x}_n \end{pmatrix} \right\}.$$

The discrete form of a GPR model is defined as follows.

$$\begin{aligned} y_i &= f(\boldsymbol{x}_i) + \epsilon_i, \ i = 1, \ldots, n, \\ \epsilon_i &\sim \textit{i.i.d.}\ N(0, \sigma^2), \\ f(\cdot) &\sim GP(\mu(\cdot), k(\cdot, \cdot)) \text{ and } \mathrm{Cov}(f(\boldsymbol{x}_i), f(\boldsymbol{x}_j)) = k(\boldsymbol{x}_i, \boldsymbol{x}_j), \end{aligned} \tag{2.1}$$

where $GP(\mu(\cdot), k(\cdot,\cdot))$ is a Gaussian process prior with mean $\mu(\cdot)$ and the covariance function $k(\cdot,\cdot)$. The following is one of the most commonly used covariance functions in practice

$$k(x_i, x_j; \boldsymbol{\theta}) = v_1 exp\left(-\frac{1}{2}\sum_{q=1}^{Q} w_q(x_{i,q} - x_{j,q})^2\right) + \sum_{q=1}^{Q} a_q x_{i,q} x_{j,q} \tag{2.2}$$

which is the combination of a squared exponential and a linear covariance kernel. The above covariance kernel depends on a set of hyper-parameters $\{v_1, w_q, a_q,\ q = 1, \ldots, Q\}$.

2.1.1 Fitted values and predictions

We now temporarily assume that the noise variance σ^2 is known, and the covariance function $k(\cdot,\cdot)$ is predetermined with fixed hyper-parameters known in advance. We use $\boldsymbol{\theta}$ to denote σ^2 and the hyper-parameters. They can be estimated by using for example the empirical Bayesian approach which will be discussed in the next section. It is also common to assume a zero mean function, i.e. $\mu(\cdot) = 0$.

Let $\boldsymbol{f} = (f(x_1), \ldots, f(x_n))^T$. When the value of the hyper-parameters $\boldsymbol{\theta}$ is given, the posterior distribution, $p(\boldsymbol{f}|\mathcal{D}, \sigma^2)$, is a multivariate normal distribution with

$$\begin{aligned} \mathrm{E}(\boldsymbol{f}|\mathcal{D}, \sigma^2) &= \boldsymbol{K}(\boldsymbol{K} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{y} \\ \mathrm{V}ar(\boldsymbol{f}|\mathcal{D}, \sigma^2) &= \sigma^2\boldsymbol{K}(\boldsymbol{K} + \sigma^2\boldsymbol{I})^{-1}, \end{aligned}$$

where the covariance matrix $\boldsymbol{K}$ is calculated by using the kernel covariance function. Its (i,j)-th element is calcualted by

$$\boldsymbol{K}(i,j) = \mathrm{Cov}(f_i, f_j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j; \boldsymbol{\theta}). \tag{2.3}$$

Note that the mean vector of the Gaussian process prior is assumed to be zero.

It is straightforward to predict an output for a new data points i.e. the points other than $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$. We also call them as *test data* and call $\mathcal{D}$ as *training data*. Let $\boldsymbol{x}^*$ be a new input and let $f(\boldsymbol{x}^*)$ be the related nonlinear function. Since $(f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n), f(\boldsymbol{x}^*))$ constitutes a $(n+1)$-variate normal vector. Consequently, the posterior distribution of $f(\boldsymbol{x}^*)$ given the training data $\mathcal{D}$ is also a Gaussian distribution, with mean and variance given by

$$\mathrm{E}(f(\boldsymbol{x}^*)|\mathcal{D}) = \boldsymbol{\psi}^T(\boldsymbol{x}^*)\boldsymbol{\Psi}^{-1}\boldsymbol{y}, \tag{2.4}$$

$$\mathrm{Var}(f(\boldsymbol{x}^*)|\mathcal{D}) = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{\psi}^T(\boldsymbol{x}^*)\boldsymbol{\Psi}^{-1}\boldsymbol{\psi}(\boldsymbol{x}^*), \tag{2.5}$$

where $\boldsymbol{\psi}(\boldsymbol{x}^*) = (k(\boldsymbol{x}^*, \boldsymbol{x}_1), \cdots, k(\boldsymbol{x}^*, \boldsymbol{x}_n))^T$ is the covariance between $f(\boldsymbol{x}^*)$ and $\boldsymbol{f} = (f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n))$, and $\boldsymbol{\Psi}$ is the covariance matrix of $(y_1, \cdots, y_n)$ given by

$$\boldsymbol{\Psi} = \boldsymbol{K} + \sigma^2\boldsymbol{I}. \tag{2.6}$$

If y^* is the related output or response to $\boldsymbol{x}^*$, then its predictive distribution is also Gaussian, with the mean given by (2.4) and the variance

$$\hat{\sigma}^{*2} = \mathrm{Var}(f(\boldsymbol{x}^*)|\mathcal{D}) + \sigma^2. \tag{2.7}$$

If we use the posterior mean $\mathrm{E}(f(\boldsymbol{x}^*)|\mathcal{D})$ in (2.4) as the prediction of $f(x^*)$, it satisfies posterior consistency, i.e. it is a consistent estimation of the true function $f_0(\cdot)$ (see the detailed discussion in Shi and Choi, 2011).

2.1.2 Empirical Bayes estimates

In Bayesian inference, we usually select the values of hyper-parameters based on our prior knowledge. We however should be cautious on doing so for the GPR model since the dimension of $\boldsymbol{\theta}$ is usually quite large and we don't usually know the meaning or physical interpretation of $\boldsymbol{\theta}$. An alternative way is to estimate them using the observed data. This is so called empirical Bayes estimates (Carlin and Louis, 1996; Shi and Choi, 2011).

Using empirical Bayesian approach, we estimate $\boldsymbol{\theta}$ from the marginal distribution of $\boldsymbol{y} = (y_1, \ldots, y_n)^T$:

$$p(\boldsymbol{y}|\boldsymbol{\theta}) = \int p(\boldsymbol{y}|\boldsymbol{f})p(\boldsymbol{f}|\boldsymbol{\theta})d\boldsymbol{f}, \tag{2.8}$$

where $p(\boldsymbol{y}|\boldsymbol{f}) = \prod_{i=1}^n g(f_i)$ and $\boldsymbol{f} \sim N(\boldsymbol{0}, \boldsymbol{K})$. The (i,j)-th element of the covariance matrix $\boldsymbol{K}$ is calculated by (2.3). Consequently, for the continuous response with normal distribution as given in (2.1), the marginal distribution (2.8) has an analytical form as a multivariate normal. The marginal distribution of $\boldsymbol{y}$ is a normal distribution

$$\boldsymbol{y} \sim N(\boldsymbol{0}, \boldsymbol{\Psi}),$$

with covariance matrix $\boldsymbol{\Psi}$ given in (2.6). Hence, the marginal log-likelihood of $\boldsymbol{\theta}$ is given by

$$l(\boldsymbol{\theta}|\mathcal{D}) = -\frac{1}{2}\log|\boldsymbol{\Psi}(\boldsymbol{\theta})| - \frac{1}{2}\boldsymbol{y}^T\boldsymbol{\Psi}(\boldsymbol{\theta})^{-1}\boldsymbol{y} - \frac{n}{2}\log 2\pi. \tag{2.9}$$

Thus $\boldsymbol{\theta}$ is estimated by maximizing the above log-likelihood.

Remarks.

- The conjugate gradient optimization method is used in the package.
- In practice we often estimate σ^2 and the hyper-parameters at the same time. This package therefore treats σ^2 as one of the elements in $\boldsymbol{\theta}$.

We may sometime introduce a hyper-prior for the hyper-parameters. For example, we may use the following prior distribution for hyper-parameters involved in the squared exponential covariance kernel

(2.2)

$$\begin{aligned} w_q^{-1} &\sim \Gamma(\alpha_w, \alpha_w/\mu_w), \quad d \in [1, D] \\ &\quad i.e.\ E(w^{-1}) = \mu_w, var(w^{-1}) = \mu_w^2/\alpha_w \\ a_q &\sim N(\mu_a, \sigma_a^2), \quad d \in [1, D] \\ log(v_1) &\sim N(\mu_{v_1}, \sigma_{v_1}). \end{aligned}$$

Let $p(\boldsymbol{\theta})$ be the density function of the above hyper-prior distribution, we can estimate $\boldsymbol{\theta}$ by MAP (maximum a posteriori probability), i.e. maximizing $l(\boldsymbol{\theta}|\mathcal{D}) + \log p(\boldsymbol{\theta})$.

2.2 Examples

This section contains two examples using simulated data and one example using real data. We need to load the package before we run the code.

```
library(GPFDA)
```

It may depend on a few other packages. They should be installed before running the examples; check the details in Section 1.1

Demo files for the examples are included in the package. We can use the following command to demonstrate the examples.

```
demo(DemoName,package="GPFDA")
```

For example, demonstrate Example 1 in the next section using

```
demo(gpr_ex1,package="GPFDA")
```

2.2.1 Example 1

This example involves a one-dimensional covariate. The data is generated by

$$y = \sin(6x) + \epsilon; \quad x \in [0, 1] \text{ and } \epsilon \sim GP(0, k(\cdot, \cdot)),$$

where $k(\cdot, \cdot)$ is specified by (2.2) which depends on x. The samples used to learn the model is non-equally spaced drawn from $x \in [0, 1]$. Sample size of the training data is 21. Test data are equally spaced from 0 to 1 with sample size of 34.

The following code is used to generate data. Seed is set to be a fixed number for reproducing the result.

```
library(GPFDA)
library(MASS)
rm(list=ls())
set.seed(3)
hp <- list("pow.ex.w"=log(10),"linear.a"=log(10),"pow.ex.v"=log(5),"vv"=log(1))
c <- seq(0,1,len=40)
n <- 21
ntest=34
idx <- sort(sample(1:40,n))
x <- c[idx]
y <- (mvrnorm(n=n,mu=x-x,Sigma=.01*(cov.linear(hp,x)+cov.pow.ex(hp,x)))[,1])+sin(x*6)
y <- as.matrix(y)
x <- as.matrix(x)
test <- as.matrix(seq(0,1,len=ntest))
```

Model learning is obtained using

```
a <- gpr(x,y,c("linear","pow.ex"),hp)
```

The above function is to estimate the parameters using Empirical Bayesian approach. This will return a '.gpr' object, which is a list of:

- CovFun: Covariance function type
- fitted: Fitted value for training data
- fitted.sd: Standard deviation of the fitted value for training data
- gamma: Parameter used in powered exponential covariance function
- hyper: Estimates of the hyper-parameters
- I: Variance of the estimated hyper-parameters
- train.x: Training data of the covariates
- train.y: Training data of the response

For example the estimates of the hyper-parameters are:

```
> a$hyper
  $linear.a
```

```
[1] 0.5134467

$pow.ex.v
[1] 0.6114809

$pow.ex.w
[1] 1.592719

$vv
[1] -2.724992
```

For convenience we took a log transformation in the package. So the actual values of the estimates are

```
> sapply(a$hyper,exp)
  linear.a   pow.ex.v   pow.ex.w         vv
1.67104084 1.84315896 4.91709944 0.06554671
```

Thus the estimates of the hyper-parameters in (2.2) are

$$\hat{a} = 1.67104084,\ \hat{v} = 1.84315896\ \hat{w} = 4.91709944 \text{ and } \hat{\sigma}^2 = 0.06554671.$$

To calculate predictions for a set of test data, we use the function `gppredict()`

```
b <- gppredict(a,test)
```

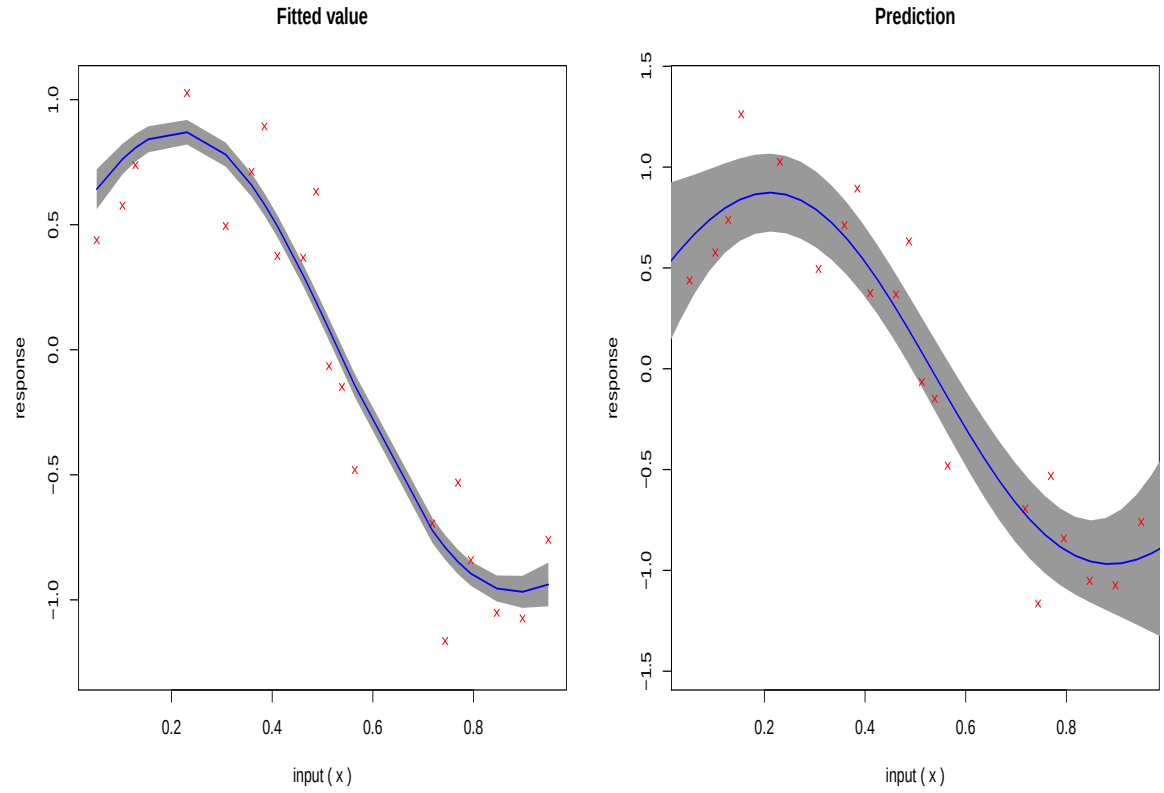
where `test` is the set of test data. This will return a list of objects, in addition to all the objects obtained from the model learning process, including:

- mu: Predictive mean
- sigma: Predictive variance

Thus `b$mu` can be used to calculate prediction. The prediction interval with e.g. 95% level can be calculated by

```
upper=b$mu+1.96*sqrt(b$sigma)
lower=b$mu-1.96*sqrt(b$sigma)
```

Fig 2.1 shows the fitted values with their 95% confidence interval for the training data and prediction with their 95% prediction interval for the test data.



(a) Fitted line

(b) Prediction

Figure 2.1: Fitted values and predictions: the red crosses are the training data; Solid lines are the fitted values or prediction; Shadow areas are the confidence interval or the the prediction interval.

The default setting of `gppredict()` is to use the estimation of the hyper-parameters obtained in the training precess `gpr`. It can also accept custom input if the values of the parameters are pre-determined. The first argument `train` controls the way of input.

If we have already run the command:

```
a <- gpr(x,y,c("linear","pow.ex"),hp)
```

the default setting is to use the output of `gpr`:

```
b <- gppredict(train = a, Data.new = test)
```

or equivalently

```
b <- gppredict(a, test)
```

Alternatively, we can use the following way:

```
b <- gppredict(train = F, Data.new = test, hyper = a$hyper,
            Cov = a$CovFun, Y = a$train.y, Data = a$train.x)
```

Here, the training data, covariance matrix, the values of parameters could use custom input or the pre-stored output of `gpr`.

We can use the following command to show the demo of this example.

```
demo(gpr_ex1,package="GPFDA")
```

Note: when you see the information `Waiting to confirm page change ...` you need to click the graph window to continue the demo.

2.2.2 Example 2

The second example involves 4-dimensional covariates. The data is generated as:

$$y = 0.2x_1 \times |x_1|^{1/3} - 4\sin(x_2) + \exp(x_3) + \log(x_4) + \epsilon, \quad \epsilon \sim GP(0, k(\cdot,\cdot)),$$

where $k(\cdot,\cdot)$ is specified by (2.2) which depends on $\boldsymbol{x} = (x_1, x_2, x_3, x_4)^T$. Again the training data are non-equally spaced and the sample size is 100. Seed is set up for reproducing the results.

```
set.seed(2)
hp <- list("pow.ex.w"=rep(log(10),4),"linear.a"=rep(log(10),4),"pow.ex.v"=log(5),
            "vv"=log(1))
kernal <- c('linear','pow.ex')
nn=100; mm=1000;
```

```
p=dnorm((rnorm(800)))
idx=sort(sample(1:800,nn,prob=p/sum(p)))
X=matrix(0,ncol=4,nrow=800)
X[,1]=seq(-5,10,len=800)
X[,2]=seq(0,1,len=800)
X[,3]=seq(-15,-10,len=800)
X[,4]=seq(1,2,len=800)
Y=(mvrnorm(n=800,mu=as.matrix(X[,1]-X[,1]),Sigma=(cov.linear(hp,X)
    +cov.pow.ex(hp,X)))[,1])*0.002+(0.2*sign(X[,1])*abs(X[,1])^(1/3)
    -4*sin(X[,2])+exp(X[,3])+log(X[,4]))*3
X=X[idx,];Y=as.matrix(Y[idx])
x=matrix(0,ncol=4,nrow=mm)
x[,1]=seq(-5,10,len=mm)
x[,2]=seq(0,1,len=mm)
x[,3]=seq(-15,-10,len=mm)
x[,4]=seq(1,2,len=mm)
```

The model training and prediction calculating for the test data are carried out by:

```
a <- gpr(X,Y,kernal,trace=2)
b <- gppredict(a,x)
```

The argument `trace` is used to control whether you want to print out every `trace` step of the optimization precess. Convergence information is included in

```
a$conv
```

It has two possible values: 0 and 1 where 0 means converged successfully and 1 means failed.

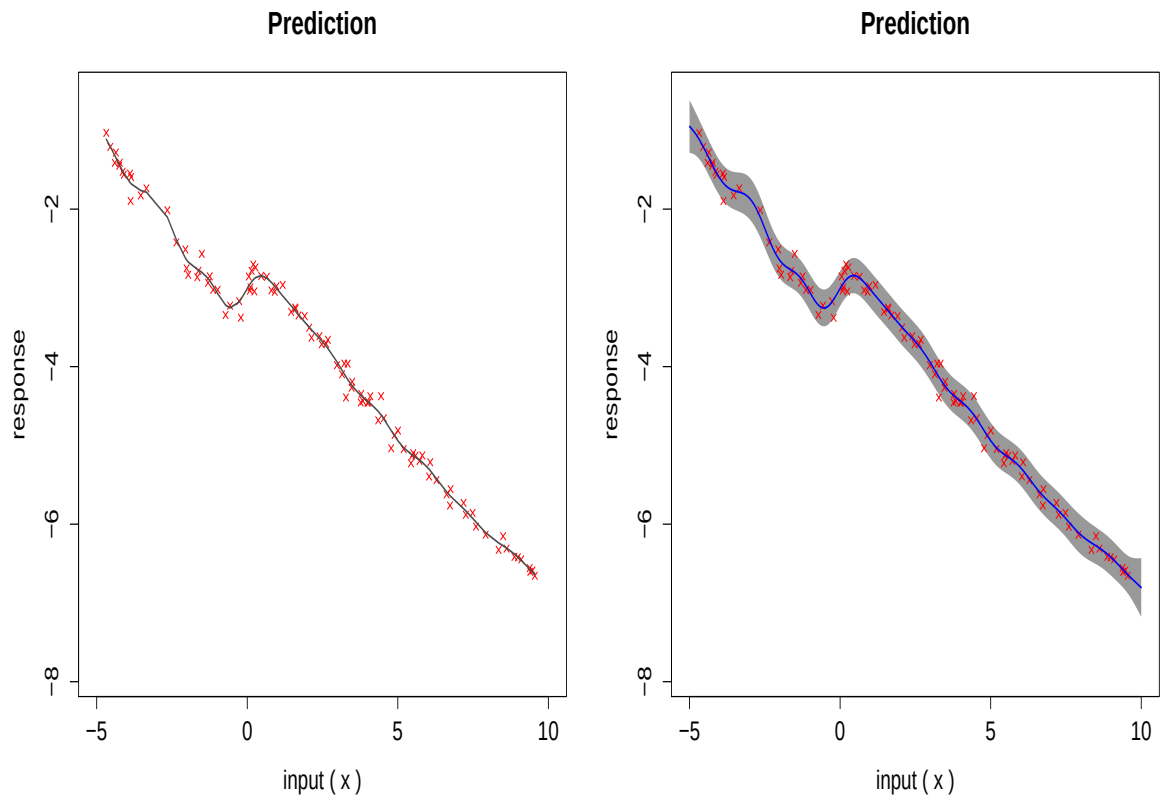
The estimates of the hyper-parameters are

```
> unlist(sapply(a$hyper,exp))
linear.a1    linear.a2    linear.a3    linear.a4     pow.ex.v
0.015916366  0.004562655  9.644476266  4.280208192  4.822551884
pow.ex.w1    pow.ex.w2    pow.ex.w3    pow.ex.w4           vv
9.147491678  9.993546889  8.654467722 10.024756532  0.013280801
```

Figure 2.2 shows the fitted values and the predicted value against one covariate.

You may use the following command to run the demo

```
demo(gpr_ex2,package="GPFDA")
```



(a) Fitted values

(b) Predictions and the 95% predictive interval

Figure 2.2: Plots of the fitted values and predictions against x_1 (solid lines): the crosses stand for training data and the the shadow area in the right panel is the predictive interval.

2.2.3 Example 3: atmospheric CO_2 concentrations

This example used a real data set. Three types of covariance kernels are used, including a custom defined kernel.

The data is included in the package (it can also be download from `http://cdiac.esd.ornl.gov/ftp/trends/co2/maunaloa.co2`). It is the recording of monthly average atmospheric Carbon Dioxide concentrations (per million by volume (ppmv)) collected at the Mauna Loa Observatory, Hawaii, between 1958 and 2003. Five missing values are removed in this example.

We can load the data by

```
data(co2)
```

This gives a 51×13 matrix. Each row includes the records for 12 months as well as the annual average for the year. There are total 51 years from 1958 to 2003. We use the following R code to remove the

missing data and rearrange the data as the format used in the package.

```
y=data.matrix(co2[,!names(co2)%in%"Annual_Average"])
y=matrix(t(y),ncol=1)
x=1:612/12; x[y<0]=NA
mat=cbind(y,x)
mat=na.omit(mat)

X=as.matrix(mat[,2])
Y=as.matrix(mat[,1])
x=seq(1,612,len=1000)/12
```

The time scale is set to be from 1/12 to 51, which can be changed easily. The data is presented in Figure 2.3, showing a clear periodic pattern.

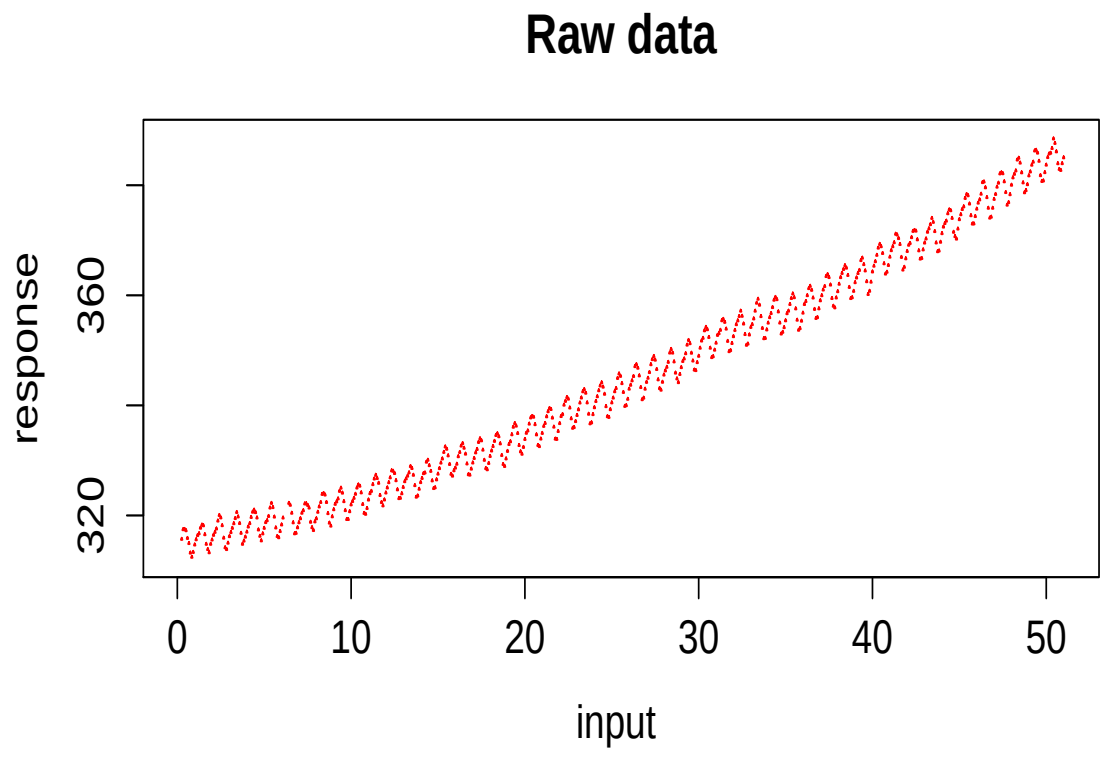


Figure 2.3: CO2 data: monthly average atmospheric Carbon Dioxide concentrations

We first fit the data using the squared exponential or the rational quadratic covariance function. The prediction is calculated at the 1000 equally spaced time points.

```
system.time(a1 <- gpr(as.matrix(X),as.matrix(Y),c("pow.ex"),mean="t"))
system.time(b1 <- gppredict(a1,Data.new=as.matrix(x)))

hp2 <- list('rat.qu.a'=2,'rat.qu.s'=0.1,'rat.qu.w'=-1,'vv'=1.5)
a2 <- gpr(as.matrix(X),as.matrix(Y),c('rat.qu'),mean='t',trace=2,hyper = hp2)
b2 <- gppredict(a2,Data.new=as.matrix(x))
```

Note that `system.time` is to show the time of running the code. The last argument `mean` is to define the mean structure. Here `mean="t"` defines a GPR model using a linear trend line as a mean function. It can also take the value of 0 or 1, standing for zero mean or a constant mean. The default setting is 0. You may use `plot(b1)` or `plot(b2)` to plot the predictions. The predictions and their 95% predictive interval are given in Figure 2.4. Neither of them can model the preiodic pattern.

To capture the periodic pattern, we use the following covariance function (Rasmussen and Williams, 2006):

$$k(x, x') = v * exp(-w * (x - x')^2 - u * sin^2(\pi * (x - x'))). \tag{2.10}$$

This is not a standard covariance kernel included in the package. So we need to define it manully.

- The first step is to define the covariance kernel itself. The name of the kernel must be constituted by the prefix 'cov.' and 6 letters (e.g., 'cov.custom').

```
cov.custom=function(hyper,Data,Data.new=NULL){
  hyper=lapply(hyper,exp);
  datadim=dim(Data)
  if(is.null(Data.new)) Data.new=Data
  A1=xixj_sta(Data,Data.new,hyper$custom.w) #exp(w)*||x-x'||^2

  mdim=dim(Data);mdim.new=dim(Data.new)
  cov.=sapply(1:mdim[1],function(i) matrix(rep(Data[i,], mdim.new[1]),
              nrow=mdim.new[1],byrow=T)-Data.new)
  cov..=matrix(0,ncol=mdim[1],nrow=mdim.new[1])
  if(mdim[2]>1){
    for(i in 1:(mdim[2]-1)){
      cov..=cov..+cov.[1:mdim.new[1],];cov.=cov.[-(1:mdim.new[1]),]}
    cov.=cov..+cov.   # x-x'
  }
  A2=hyper$custom.u*(sin(pi*cov.))^2
```

```
  return(hyper$custom.v*exp(-A1-A2))
}
```

Here the function `xixj_sta` is used to calculate $exp(w) * ||x - x'||^2$. It is defined in the package, and the details can be found in the package description.

- The second step is to define the first derivative of the likelihood in terms of all the parameters involved in the covariance kernel. The formulas are omit here. The name of the functions should look like `DCov.custom.w`, where the middle affix is the custom defined name, and the last letter i.e. `w` names the vector of the parameters which can use one letter only.

```
DCov.custom.w=function(hyper,data,AlphaQ){
  Dcov=cov.custom(hyper,data)
  A1=-xixj_sta(data,data,hyper$custom.w)
  out=Dcov %*% A1
  out=sum(out*AlphaQ)
  return(out)
}


DCov.custom.u=function(hyper,data,AlphaQ){
  Dcov=cov.custom(hyper,data)
  hyper=lapply(hyper,exp)
  mdim=dim(data);mdim.new=mdim
  cov.=sapply(1:mdim[1],function(i) matrix(rep(data[i,], mdim.new[1]),
        nrow=mdim.new[1],byrow=T)-data)
  cov..=matrix(0,ncol=mdim[1],nrow=mdim.new[1])
  if(mdim[2]>1){
  for(i in 1:(mdim[2]-1)){
    cov..=cov..+cov.[1:mdim.new[1],];cov.=cov.[-(1:mdim.new[1]),]}
    cov.=cov..+cov.
  }
  A2=-hyper$custom.u*(sin(pi*cov.))^2
  out=Dcov%*%A2
  out=sum(out*AlphaQ)
  return(out)
}
```

```
DCov.custom.v=function(hyper,data,AlphaQ){
  out=cov.custom(hyper,data)
  out=sum(out*AlphaQ)
  return(out)
}
```

In this case, only the first derivative of the covariance matrix against the parameters need to be calculated. If define it as 'out', the result should be the trace of 'out' multiply 'AlphaQ', which is same as 'sum(out*AlphaQ)'. The argument 'AlphaQ' is calculated in the main function, which will attach to these functions. $AlphaQ = (Q^{-1}y)^2 - Q^{-1}$, where Q is the covariance matrix.

- The third step is to define the second derivative of the likelihood.

```
D2custom.w=function(hyper,data,inv.Q,Alpha.Q){
  Dcov=cov.custom(hyper,data)
  A1=-xixj_sta(data,data,hyper$custom.w)
  wD2=Dcov%*%(A1^2+A1)
  wD1=Dcov%*%A1
  D2c.w=D2(wD1,wD2,inv.Q,Alpha.Q)
  return(D2c.w)
}
```

```
D2custom.w=function(hyper,data,inv.Q,Alpha.Q){
  Dcov=cov.custom(hyper,data)
  A1=-xixj_sta(data,data,hyper$custom.w)
  wD2=Dcov%*%(A1^2+A1)
  wD1=Dcov%*%A1
  D2c.w=D2(wD1,wD2,inv.Q,Alpha.Q)
  return(D2c.w)
}
```

```
D2custom.u=function(hyper,data,inv.Q,Alpha.Q){
  Dcov=cov.custom(hyper,data)
  hyper=lapply(hyper,exp)
```

```
  mdim=dim(data);mdim.new=mdim
  cov.=sapply(1:mdim[1],function(i) matrix(rep(data[i,],
               mdim.new[1]),nrow=mdim.new[1],byrow=T)-data)
  cov..=matrix(0,ncol=mdim[1],nrow=mdim.new[1])
  if(mdim[2]>1){
    for(i in 1:(mdim[2]-1)){
      cov..=cov..+cov.[1:mdim.new[1],];cov.=cov.[-(1:mdim.new[1]),]}
    cov.=cov..+cov.
  }
  A2=-hyper$custom.u*(sin(pi*cov.))^2
  uD2=Dcov%*%(A2^2+A2)
  uD1=Dcov%*%A2
  D2c.u=D2(uD1,uD2,inv.Q,Alpha.Q)
  return(D2c.u)
}

D2custom.v=function(hyper,data,inv.Q,Alpha.Q){
  vD1=cov.custom(hyper,data)
  vD2=vD1
  D2c.v=D2(vD1,vD2,inv.Q,Alpha.Q)
  return(D2c.v)
}
```

- Finally we define a function including only the diagonals of the covariance matrix. This makes computation more efficient.

```
diag.custom=function(hyper,data){
  Qstar=rep(exp(hyper$custom.v),dim(data)[1])
  return(Qstar)
}
```

We can now define a new GPR model, in which the covariance function combines three kernels: the custom defined (2.10), the squared exponential and the rational quadratic covariances. The model is defined by

```
hp3 <- list(custom.u= 1,custom.v=.1,custom.w=-.1,pow.ex.v=.1,pow.ex.w=.1,
            rat.qu.a=.2,rat.qu.s=.1,rat.qu.w=-.1, vv=-2.5)
```

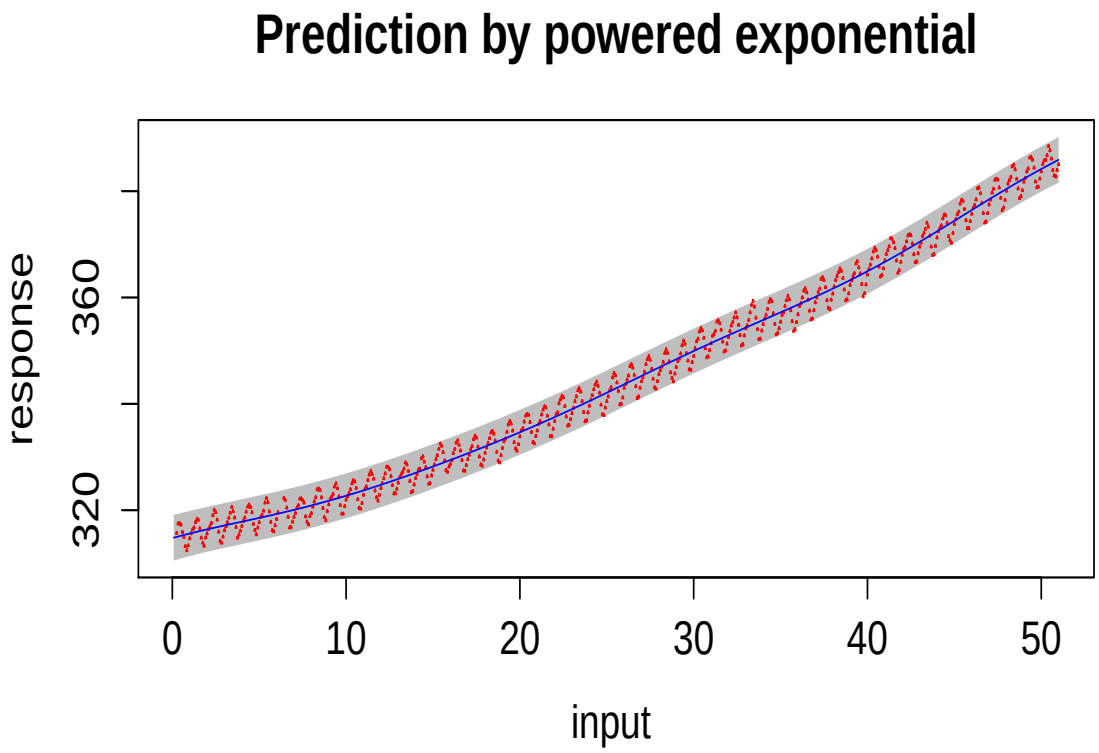
```
a3 <- gpr(as.matrix(X),as.matrix(Y),Cov=c('pow.ex','custom','rat.qu'),
      NewHyper=c('custom.w','custom.u','custom.v'),mean='t',trace=2,hyper = hp3)
b3 <- gppredict(a3,Data.new=as.matrix(x))
```

To define the new covariance kernel, the name of the custom defined one should be included, and the parameters used in the custom defined kernel should also be included in the argument 'NewHyper'.

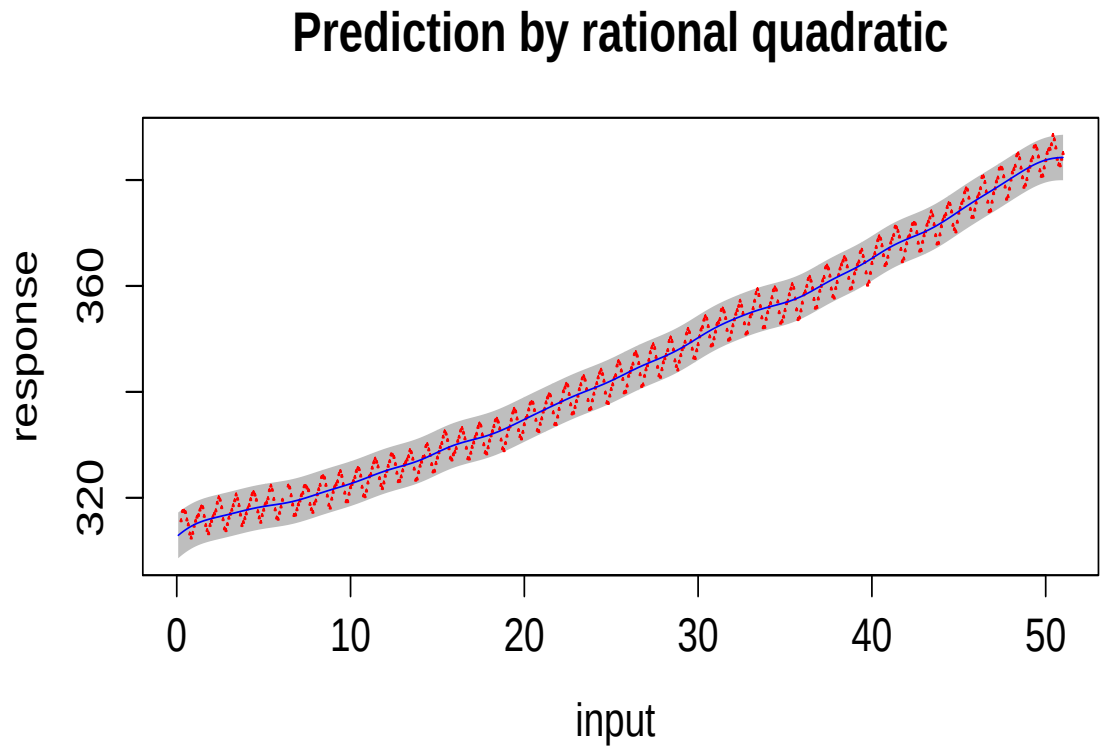
The prediction using the new model is shown in Figure 2.5. The periodic pattern has been modeled this time.

The demo of the example can be carried out by:

```
demo(co2,package="GPFDA")
```



(a) Using the squared exponential cov function



(b) Using the rational quadratic cov function

Figure 2.4: Co2 data: the raw data (dotted point, in red), the fitted line (the blue line) with its 95% predictive interval (in shade).

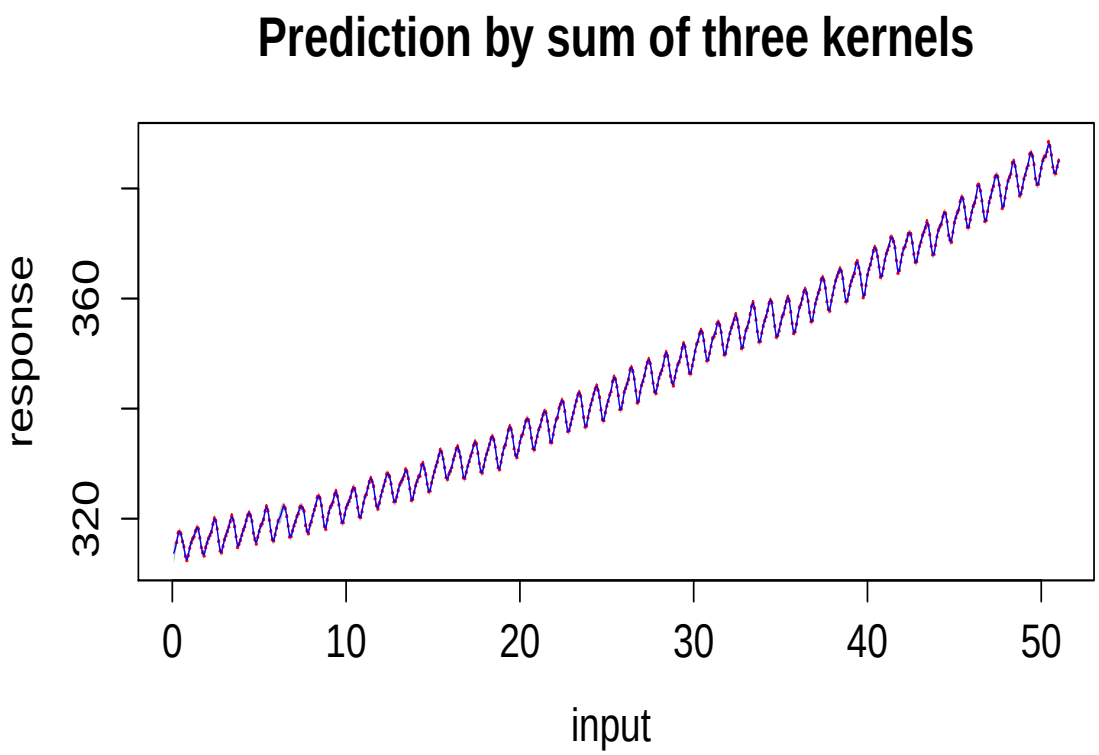


Figure 2.5: Co2 data: the raw data (dotted point, in red), the fitted line (the blue line) with its 95% predictive interval (in shade) using the combination of three covariance functions.

Chapter 3

Gaussian Process Function Regression Analysis

3.1 Gaussian process functional regression model

The GPR models discussed in the previous chapter is for a single curve (which can also be regarded as a batch with only one realization). We now turn our attention to consider batch data or data with repeated curves.

Assume that we have a functional response variable $y_m(t)$ for $m = 1, 2, \ldots, M$, and we also have a set of functional covariates $\boldsymbol{x}_m(t)$ and a set of scalar covariates $\boldsymbol{u}_m$, where

$$\boldsymbol{x}_m(t) = (x_{m1}(t), x_{m2}(t), \ldots, x_{mQ}(t))^T \quad \text{and} \quad \boldsymbol{u}_m = (u_{m1}, u_{m2}, \ldots, u_{mp})^T.$$

A general nonlinear regression model for the m-th batch is defined as

$$y_m(t) = f(t, \boldsymbol{x}_m(t), \boldsymbol{u}_m) + \epsilon_m(t), \tag{3.1}$$

where $\epsilon_m(t)$'s are random errors which are independent at different t's. Here, we indistinctively refer to this m-th *replication* as m-th batch or m-th *curve* if we need to emphasize its different aspects.

A Gaussian process functional regression (GPFR) model is defined by

$$y_m(t) = \mu_m(t) + \tau_m(\boldsymbol{x}_m) + \epsilon_m(t), \tag{3.2}$$

where $\mu_m(t)$ models the common mean structure across different curves, while $\tau_m(\boldsymbol{x}_m)$ defines the covariance structure of $y_m(t)$ for the different data points within the same curve. We use a Gaussian process regression model to define the covariance structure:

$$\tau_m(\boldsymbol{x}_m) \sim GPR_m[0, k_m(\boldsymbol{\theta}_m)|\boldsymbol{x}_m], \quad m = 1, \ldots, M, \tag{3.3}$$

where $GPR_m[0, k_m(\boldsymbol{\theta}_m)|\boldsymbol{x}_m]$, as defined in (2.1), denotes a Gaussian process regression model with a covariance function k_m and hyper-parameters $\boldsymbol{\theta}_m$. Equations (3.2) and (3.3) jointly define a Gaussian process functional regression model; it is denoted by (Shi and Choi, 2011)

$$y_m(t) \sim GPFR[\mu_m(t), k_m(\boldsymbol{\theta}_m)|\boldsymbol{x}_m(t), \boldsymbol{u}_m].$$

3.2 GPFR model with a linear functional mean model

In this section, we consider a special case of Gaussian process functional regression model by using a linear functional mean model. We assume that the mean model $\mu_m(t)$ depends on the scalar covariates $\boldsymbol{u}_m$ and t only, i.e. $\mu_m(t) = \boldsymbol{u}_m^T\boldsymbol{\beta}(t)$. Thus, the Gaussian process functional regression model with a linear functional regression mean is given by

$$y_m(t) = \boldsymbol{u}_m^T\boldsymbol{\beta}(t) + \tau_m(\boldsymbol{x}_m) + \epsilon_m(t). \tag{3.4}$$

Assume that all the functional variables in the same batch are observed at the same data points $\{t_{mi}, i = 1, \ldots, n_m\}$ for $m = 1, \ldots, M$. The data observed in each batch are

$$\mathcal{D}_m = \{(y_{mi}, t_{mi}, x_{m1i}, \ldots, x_{mQi}) \text{ for } i = 1, \ldots, n_m; \text{ and } (u_{m1}, \ldots, u_{mp})\}, \tag{3.5}$$

where $y_{mi} = y(t_{mi})$ is the observation of $y_m(t)$ at t_{mi} and $x_{mqi} = x_q(t_{mi})$ is the measurement of the q-th input variable for $q = 1, \ldots, Q$. Likewise, the observations of the scalar variables for the m-th batch are given by $\boldsymbol{u}_m = (u_{m1}, \ldots, u_{mp})^T$. If any of the functional variables have not been recorded at the same data points, they can still be used in the analysis but they need to be treated beforehand; see Ramsay and Silverman (2005) for further details.

In (3.4) $y_m(t)$ and $\boldsymbol{\beta}(t)$ can be approximated by using a set of basis functions:

$$y_m(t) \approx \tilde{y}_m(t) = \boldsymbol{A}_m^T\boldsymbol{\Phi}(t), \quad \text{and} \quad \boldsymbol{\beta}(t) \approx \boldsymbol{B}^T\boldsymbol{\Phi}(t),$$

where $\boldsymbol{\Phi}(t) = (\phi_1(t), \ldots, \phi_H(t))^T$ are a set of H basis functions, $\boldsymbol{A}_m$ is an H-dimensional coefficient vector and $\boldsymbol{B}$ is a $H \times p$ matrix. For example, $\boldsymbol{\Phi}(t)$ could be a set of B-spline basis functions.

Based on the data $\{\mathcal{D}_m\}$ given in (3.5), we can evaluate the marginal likelihood for the model (3.4), and then calculate all unknown parameters using empirical Bayesian approach (see Shi et al., 2007; Shi and Choi, 2011).

In practice, we may use a fast approximating approach. The coefficients $\boldsymbol{A}_m$ and $\boldsymbol{B}$ can be estimated by

$$\begin{aligned} \hat{\boldsymbol{A}}_m &= (\boldsymbol{\Phi}_m^T\boldsymbol{\Phi}_m)^{-1}\boldsymbol{\Phi}_m^T\boldsymbol{y}_m, \\ \hat{\boldsymbol{B}}^T &= (\boldsymbol{U}^T\boldsymbol{U})^{-1}\boldsymbol{U}^T\boldsymbol{A}, \end{aligned}$$

where $\boldsymbol{y}_m = (y_{m1}, \ldots, y_{mn_m})^T$, $\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_M)^T$ and $\boldsymbol{\Phi}_m$ is an $n_m \times H$ matrix with elements $(\phi_h(t_{mi}))$. The details can be found in Section 5.3 in Shi and Choi (2011).

3.2.1 Predictions

We now consider how to calculate the prediction $y^* = y(t^*)$ at a new point $(t^*, \boldsymbol{x}^*, \boldsymbol{u}^*)$ with $\boldsymbol{x}^* = \boldsymbol{x}(t^*)$. From (3.4), the mean is estimated by

$$\hat{\mu}(t) = \boldsymbol{u}^T \hat{\boldsymbol{B}}^T \boldsymbol{\Phi}(t), \tag{3.6}$$

and the prediction of y^* include two parts

$$\hat{y}^* = \hat{\mu}(t^*) + \hat{\tau}(\boldsymbol{x}^*), \tag{3.7}$$

where $\tau^* = \tau(\boldsymbol{x}^*)$ is predicted by its conditional mean $\mathrm{E}(\tau^*|\mathcal{D})$ from the Gaussian process regression model defined in (3.3) (the formula is given by (2.4)).

Type I prediction. We suppose that we have already observed some data for a batch and want to predict $y(t^*)$ at a new point, denoting it as the $(M+1)$-th curve or batch. In addition to the training data observed in the first M batches, assume that n observations have also been obtained in the new curve at $\boldsymbol{t} = (t_1, t_2, \ldots, t_n)^T$, providing the data

$$\mathcal{D}_{M+1} = \{(y_{M+1,i}, t_{M+1,i}, x_{M+1,1,i}, \ldots, x_{M+1,Q,i}),\ i = 1, \ldots, n;\ \boldsymbol{u}_{M+1}\}.$$

Thus the training data is $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_M, \mathcal{D}_{M+1}\}$, which is used to estimate the parameters using the methods discussed earlier. To predict y^* at a new data point t^*, we assume that y^* and the observed data $\{y_{M+1,i}, i = 1, \ldots, n\}$ have the same model (3.4), and thus τ^* and $\{\tau_{mi}, i = 1, \ldots, n\}$ have the same GPR model structure.

The predictive mean $\hat{y}^*$ and predictive variance $\hat{\sigma}^{*2}$ are respectively given by equations (5.26) and (5.27) in Section 5.3.1 in Shi and Choi (2011).

Type II prediction. The second case is to calculate prediction for a completely new curve. Notationally, we still refer to the new curve as the $(M+1)$-th curve, with scalar covariate $\boldsymbol{u}_{M+1}$. Our objective is to predict y^* at $(t^*, \boldsymbol{x}^*)$ in the $(M+1)$-th batch. In this case, there is no data observed in the $(M+1)$-th batch, and thus the training data are $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_M\}$. One simple method is to predict it using the mean part only, so that

$$\hat{y}^* = \hat{\mu}_{M+1}(t^*) = \boldsymbol{u}_{M+1}^T \hat{\boldsymbol{B}}^T \boldsymbol{\Phi}(t^*). \tag{3.8}$$

An alternative way is to assume that curves $1, 2, \ldots, M$ provide an empirical distribution of the set of all possible curves (Shi et al., 2005), considering that

$$P(y^* \text{ belongs to the } m\text{-th curve}) = w_m, \tag{3.9}$$

for $m = 1, 2, \ldots, M$.

If we assume that y^* is generated from the m-th curve means, predictive mean and variance of y^* is calculated from the above Type I prediction procedure which are denoted by $\hat{y}_m^*$ and $\hat{\sigma}_m^{*2}$ respectively.

Based on the above assumption for the empirical distribution, a prediction for the response associated with a new input $\boldsymbol{x}^*$ at t^* in a completely new curve can be calculated by

$$\hat{y}^* = \sum_{m=1}^{M} w_m \hat{y}_m^*, \tag{3.10}$$

and the related predictive variance is

$$\hat{\sigma}^{*2} = \sum_{m=1}^{M} w_m \hat{\sigma}_m^{*2} + \left(\sum_{m=1}^{M} w_m \hat{y}_m^{*2} - \hat{y}^{*2} \right). \tag{3.11}$$

We usually take equal empirical probabilities i.e. $w_m = 1/M$. Unequal weights can be considered using an allocation model (see e.g. Shi and Wang, 2008).

Remarks.

1. In the package `GPFDA`, we used equal empirical probabilities to calculate Type II prediction.

2. The current version of `GPFDA` consides the special case (3.4) only (i.e. with a linear function mean model). A more general model will be available in tne next version.

3.3 Examples

This section contains a few simulated examples, the computation time is also included in the code. Package 'ggplot2' may be required to draw some graphs.

3.3.1 Example 1

Use the following command to run the demo:

```
demo(gpfr,package="GPFDA")
```

The details will be added shortly.

Appendix

A.1 Covariance functions

The details of derivatives of kernel functions

- linear kernel:

$$\mathbf{\Psi_{linear}}(i,j) = \sum_{q=1}^{Q} exp(a_q) x_q(t_i) x_q(t_j)$$

There are a series of a_q in linear kernel, for each a_q, the first derivative is:

$$\frac{\partial \mathbf{\Psi}}{\partial a_q}(i,j) = exp(a_q) x_q(t_i) x_q(t_j)$$

The second derivative is:

$$\frac{\partial^2 \mathbf{\Psi}}{\partial a_q^2}(i,j) = exp(a_q) x_q(t_i) x_q(t_j)$$

- powered exponential:

$$\mathbf{\Psi_{pow.ex}} = exp(v) exp\left(-\sum_{q=1}^{Q} exp(w_q)||x_q - x'_q||^{\gamma}\right)$$

There are a series of w_q and one v in powered exponential, for each w_q, the first derivative is:

$$\frac{\partial \mathbf{\Psi}}{\partial w_q} = -exp(v) exp\left(-\sum_{q=1}^{Q} exp(w_q)||x_q - x'_q||^{\gamma}\right) exp(w_q)||x_q - x'_q||^{\gamma}$$

The second derivative is:

$$\frac{\partial^2 \mathbf{\Psi}}{\partial w_q^2} = exp(v) exp\left(-\sum_{q=1}^{Q} exp(w_q)||x_q - x'_q||^{\gamma}\right) \left(exp(2w_q)||x_q - x'_q||^{2\gamma} - exp(w_q)||x_q - x'_q||^{\gamma}\right)$$

For v, the first derivative is:

$$\frac{\partial \mathbf{\Psi}}{\partial v} = exp(v) exp\left(-\sum_{q=1}^{Q} exp(w_q)||x_q - x'_q||^{\gamma}\right)$$

The second derivative is:

$$\frac{\partial^2 \mathbf{\Psi}}{\partial v^2} = exp(v) exp\left(-\sum_{q=1}^{Q} exp(w_q)||x_q - x'_q||^{\gamma}\right)$$

- rational quadratic

$$\mathbf{\Psi_{rat.qu}} = (1 + \sum_{q=1}^{Q} exp(s_{\alpha,q})||x_q - x'_q||)^{-exp(\alpha)}$$

There are a series of $s_{\alpha,q}$ in rational quadratic, for each $s_{\alpha,q}$, the first derivative is:

$$\frac{\partial \mathbf{\Psi}}{\partial s_{\alpha,q}} = -exp(\alpha)(1 + \sum_{q=1}^{Q} exp(s_{\alpha,q})||x_q - x'_q||)^{-exp(\alpha)-1} exp(s_{\alpha,q})||x_q - x'_q||$$

The second derivative is:

$$\begin{aligned}\frac{\partial^2 \mathbf{\Psi}}{\partial s_{\alpha,q}^2} &= -exp(\alpha)\Bigg[(-exp(\alpha) - 1)\left(1 + \sum_{q=1}^{Q} exp(s_{\alpha,q})||x_q - x'_q||\right)^{-exp(\alpha)-2} exp(2s_{\alpha,q})||x_q - x'_q||^2 + \\ &\qquad \left(1 + \sum_{q=1}^{Q} exp(s_{\alpha,q})||x_q - x'_q||\right)^{-exp(\alpha)-2} exp(s_{\alpha,q})||x_q - x'_q||\Bigg]\end{aligned}$$

For α, the first derivative is:

$$\frac{\partial \mathbf{\Psi}}{\partial \alpha} = log(1 + \sum_{q=1}^{Q} exp(s_{\alpha,q})||x_q - x'_q||)(1 + \sum_{q=1}^{Q} exp(s_{\alpha,q})||x_q - x'_q||)^{-exp(\alpha)}$$

The second derivative is:

$$\frac{\partial^2 \mathbf{\Psi}}{\partial \alpha^2} = log(1+\sum_{q=1}^{Q} exp(s_{\alpha,q})||x_q-x'_q||)log(1+\sum_{q=1}^{Q} exp(s_{\alpha,q})||x_q-x'_q||)(1+\sum_{q=1}^{Q} exp(s_{\alpha,q})||x_q-x'_q||)^{-exp(\alpha)}$$

- noise term. The first derivative is:

$$\frac{\partial \mathbf{\Psi}}{\partial \epsilon} = exp(\epsilon)$$

The second derivative is:

$$\frac{\partial^2 \mathbf{\Psi}}{\partial \epsilon^2} = exp(\epsilon)$$

To see the examples of using these three different kernel functions for example 1, do:

```
example('cov.linear', package='GPFDA')
example('cov.pow.ex', package='GPFDA')
example('cov.rat.qu', package='GPFDA')
```

Bibliography

Carlin, B. P. and Louis, T. A. (1996). *Bayes and Empirical Bayes Methods for Data Analysis*. Chapman & Hall/CRC, London.

Febrero-Bande, M. and de la Fuente, M. O. (2012). Statistical computing in functional data analysis: the r package fda. usc. *Journal of Statistical Software*, 51(4):1–28.

Ramsay, J. O. and Silverman, B. W. (2005). *Functional Data Analysis*. Springer, New York, second edition.

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

Shi, J. Q. and Choi, T. (2011). *Gaussian Process Regression Analysis for Functional Data*. Chapman & Hall/CRC, London, first edition.

Shi, J. Q., Murray-Smith, R., and Titterington, D. M. (2005). Hierarchical Gaussian process mixtures for regression. *Statistics and Computing*, 15:31–41.

Shi, J. Q. and Wang, B. (2008). Curve prediction and clustering with mixtures of Gaussian process functional regression models. *Statistics and Computing*, 18:267–283.

Shi, J. Q., Wang, B., Murray-Smith, R., and Titterington, D. M. (2007). Gaussian process functional regression modeling for batch data. *Biometrics*, 63:714 – 723.

Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S*. Springer, London, fourth edition.