Department of Statistics and Data Science | Functional Data Analysis

Functional Principal Components Analysis and Functional Linear Models

Gong Wenwu, 12031299

April 19, 2021
Emails: 12031299@mail.sustech.edu.cn

Part I 20/20
II 19/20
III 17/20
IV 19/20
Presentation 19/20
Total 94/100

Abstract

In this report, there are two main points: *Functional Principal Components Analysis* (PCA) and *Functional Regression* (fR) for *scalar-functional data*. Firstly, I have given a principal components analysis by carrying out a smoothed and a regularized PCA to explore the data and to see the features characterizing typical functions. I have tried to find out the informative structures by showing principal component scores the plotting components as perturbations of the mean curve. By comparing the proportion of explained variance, the regularized PCA perform better and can explain more information using few components. Secondly, I have performed the functional linear regression for the scalar response *(lifespan)* w.r.t. functional dependent variable *(egg count)*. To make the results more comprehensive, I have tried to compare four regression methods: restricted basis functions(SfR), regularization with roughness penalty(PfR), principal component (PCfR) and penalized principal component regression (PPCfR). The R^2 and osR^2 results show that SfR performs better than others. Further, all of these models have the value $osR^2 > 0.9$ which shows a good performance to predict the total lifespan. Also, I have plotted the weight function and regression coefficient along with confidence intervals to see the variation of estimated parameters.

Contents

1 Introduction

Functional regression is an active area of research, and the approach depends on whether the responses or dependence are functional or vector data. In this report, the main idea is to build a useful one dimensional model for scalar-functional data (**fly.data**: you can see the data set from https://www.orkin.com/flies/fruit-fly/fruit-fly-reproduction-rates-data).To do this, I have given a principal components analysis by carrying out a smoothed and a regularized PCA to explore the data to see the features characterizing typical functions and performed three different functional linear regression. Based on the results of **Proj. A1**, I have smoothed the functional object egg count data. Truly, it is the first step to do some regressions. However, the realizations of an underlying stochastic process (functional data and is fully observed trajectories) has infinity dimensional, causing the coefficient matrix is non-singular. And even if the coefficient matrix is of full rank, there are still infinitely many sets of solutions, all giving a perfect prediction of the observed data. Then dimension reduction is key for data modeling and analysis. So, the functional principal component approach is explored in the following section. Besides, four regression methods: restricted basis functions(SfR), regularization with roughness penalty(PfR), principal component (PCfR) and penalized principal component regression (PPCfR) have also introduced. Based on these model results, I have compared with each others and found out the best prediction model.

2 Functional Principal Component Analysis

Principal component analysis (PCA) is introduced by Jolliffe [Jo2002], which is a key dimension reduction tool for multivariate data and has been extended to functional data and termed functional principal component analysis(FPCA). A more comprehensive framework for statistical inference for FPCA was first developed by Kleffe [Kleffe1973]. Since then, this approach has taken off to become the most prevalent tool in FDA. This is partly because FPCA facilitates the conversion of inherently infinite-dimensional functional data to a finite-dimensional vector of random scores [Wang2016]. Here I will give a brief description about the mechanics of functional PCA and dynamic FPCA. Functional PCA, the dimension reduction tool, is achieved through an expansion of the underlying, but often not fully observed, random trajectories $X_i(t)$ in a functional basis that consists of the eigenfunctions of the covariance operator of the process X. Given the collection of egg count curves $\{X_i(t), i = 1, \cdots, 50.\}$, we can define the covariance operator

$$\Sigma(g) = \int_I \Sigma(s,t)g(s)d_s \tag{1}$$

for any function $g \in L^2$, where the covariance function is given by $\Sigma(s,t) = cov(X(s), X(t))$. Under mild assumptions, *Mercer's theorem* implies that the spectral decomposition of Σ leads to

$$\Sigma(s,t) = \sum_{k=1}^{\infty} \lambda_k \phi_k(s) \phi_k(t) \tag{2}$$

where the convergence holds uniformly for $s, t \in I$, λ_k are the eigenvalues in descending order, and ϕ_k are the corresponding orthogonal eigenfunctions. So, the FPCA expansion, conceived by Karhunen[Karhunen1946], has the approximately formula to facilitate dimension reduction as the first K terms for large enough K provide a good approximation to the infinite sum and therefore for X_i:

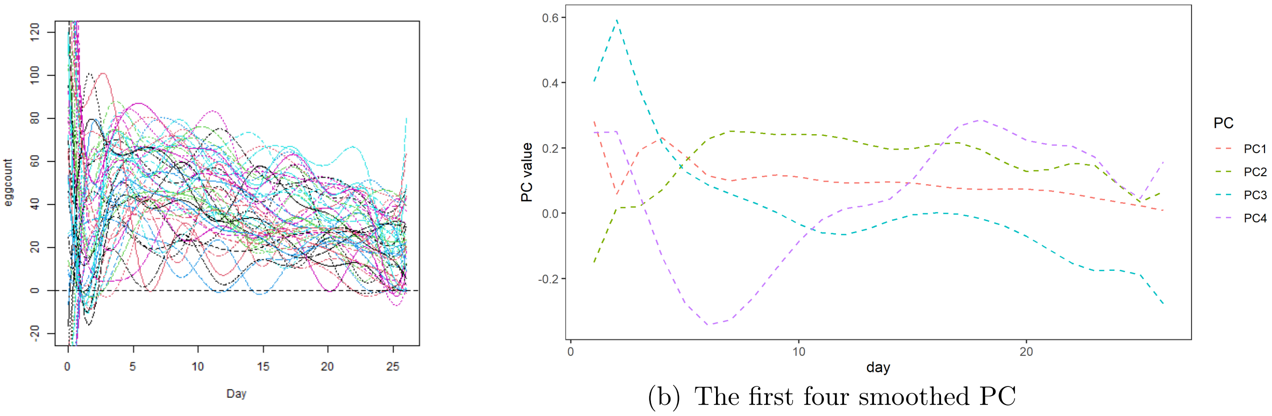
$$X_{iK}(t) = E(X(t)) + \sum_{k=1}^{K} = A_{ik} \phi_k(t). \tag{3}$$

The estimation of the eigencomponents (eigenfunctions and eigenvalues) in the FPCA framework is straightforward once the mean and covariance of the functional data have been estimated. To obtain the spectral decomposition of the covariance operator, which yields the eigencomponents, one simply approximates the estimated covariance surface $\Sigma(s,t) = cov(X(s), X(t))$ on a grid of time points, thus reducing the problem to the corresponding matrix spectral decomposition. In the computation process, we will use the *fda* package to solve the eigencomponents. The FPCA approach motivates the concept of modes of variation for functional data, a most useful tool to visualize and describe the variation in the functional data that is contributed by each eigenfunction. The kth mode of variation is the set of functions:

$$E(X(t)) \pm \sqrt{\lambda_k} \phi_k(t).$$

In this section, I first smooth the data under 12 Bspline basis (see Fig.(a) 1) function which is given by **Proj. A.1** to get the functional data (We can also choose the best number basis functions through CV procedure and this process is guided by *optim.basis* from **fda.usc** package). Then I display the eigencomponents for the egg count data across all 50 fruit flies (see Fig.(b) 1). We can see that the observed records are not very smooth, and consequently the principal component curves show substantial variability. Then a reasonable idea is to roughness or regularize the estimated principal component curves.

To do this, I apply principal components analysis using the method for regularized PCA. The procedure is to roughness the egg count data firstly and then carry



(a) The smoothed observations

(b) The first four smoothed PC

Fig. 1: The smoothed PCA of egg count.

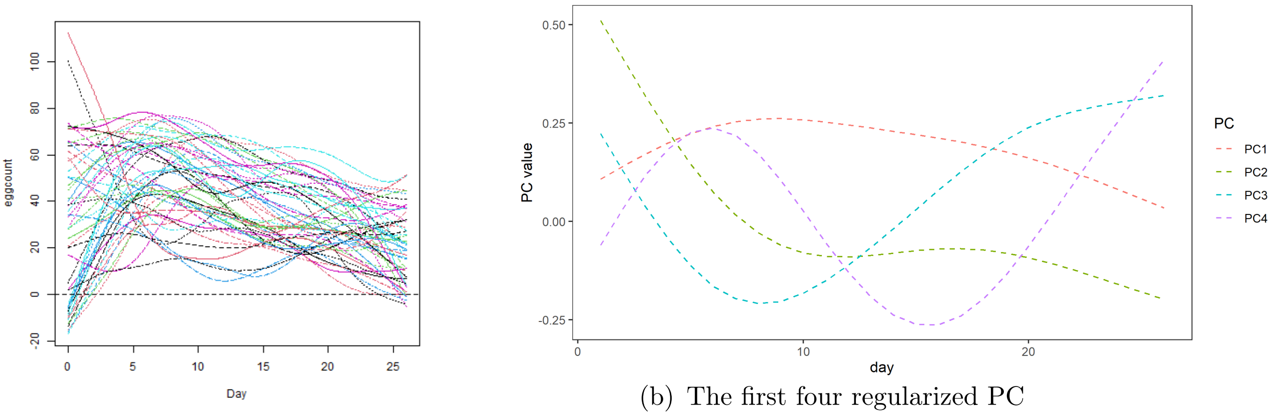
out an unsmoothed PCA. This method incorporates a smoothing parameter λ (controls the trade-off between fidelity to the functional data and smoothing) to control the amount of smoothing applied, and this has been chosen by a cross-validation method. We would obtain the roughness penalty estimate of the smooth curve by minimizing

$$PENESS = \| X - g \|^2 + \lambda \| D_g^2 \|^2 \tag{4}$$

over $g \in S$. Obviously, this is a generalization of the Bspline roughness method and we can easily get eigencomponents by setting λ. Fig. 2 shows first four regularized principal components for the egg count data, smoothed by a roughness penalty method with the same smoothing parameter $\lambda = 10$ as used for the **Proj. A1**. From Fig.(b) 2, we can see the 1st eigenfunction is nearly constant in time, implying that the largest variation between subjects is in the subject-specific average magnitude of the egg counts, so the random intercept captures the largest variation of the data. The 2nd eigenfunction shows a variation around a linear time trend with a break point near day 10, reflecting that the egg count laying is decreasing along with time, 3rd and 4th eigenfunction shows there is a cycle in the egg count laying. To examine the overall mean function and the functions obtained by adding and subtracting a suitable multiple of the principal component function is helpful to interpret the eigencomponents. Fig. 3 considerably clarifies the effects of the first two components. We can now see that the 2nd principal component reveals a linear trend and 3rd principal component corresponds to a time cycle effect combined with an overall increase in egg count.

Good explanation !

To compare these two method, Fig(b). 4 has shown that the regularized PCA can explain more information using few components and the first eigenfunction explains 50.9% of the total variation of the data and the second one an additional 36.2%



(a) The regularized observations

(b) The first four regularized PC

Fig. 2: The regularized PCA of egg count.

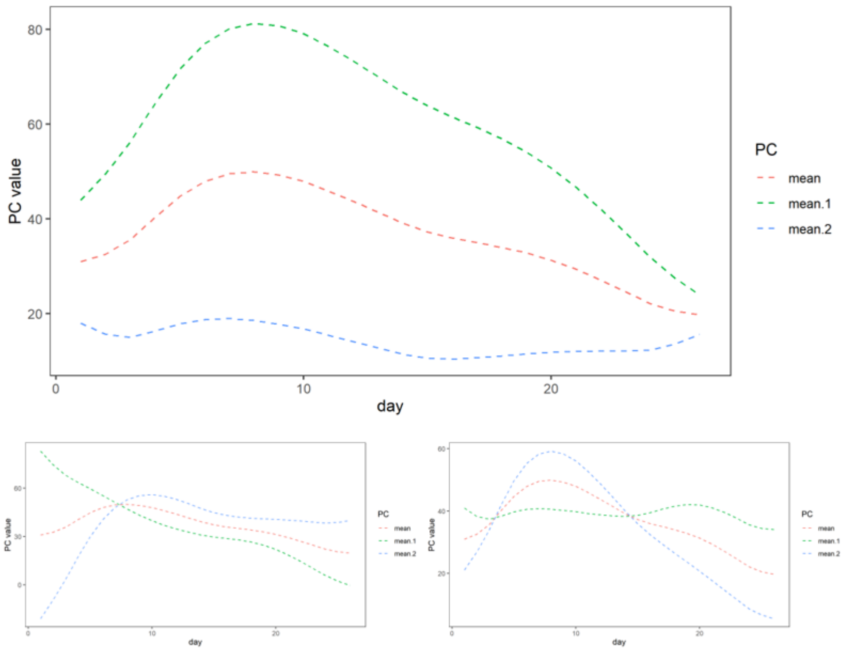
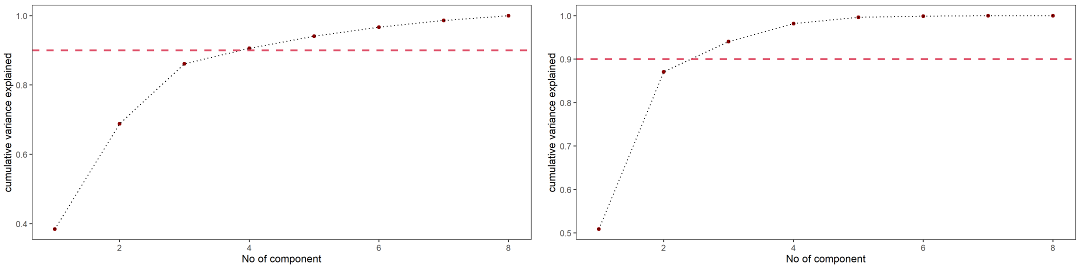


Fig. 3: The mean egg count curves and the effects of adding (green) and subtracting (blue) a suitable multiple of each PC curve.

Tab. 1: Comparison of variance proportion between fPCA and dynamic fPCA.

Method	1st	2nd	3rd	4th
fPCA	0.509	0.362	0.071	0.041
Dynamic fPCA	0.344	0.211	0.064	0.056

of the data. This result shows that we can apply *3 eigencomponents (PC1,PC2,PC3)* to recover 90% of the variation.



(a) Smoothed PCA (b) Regularized PCA

Fig. 4: The proportion of explained variance.

Applicants of functional PCA on a data with time-dependent structure is reliable, since fPCA is applied for displaying the modes of functional variation. Nevertheless, we could consider another extension of PCA, dynamic PCA which can reveal the dynamics of the underlying data structure [Oleg2016]. For the egg count data set, we can try to analysis the dynamic fPCA based on the window size is 1 day. Tab. 1 has shown the comparison results between fPCA and dynamic fPCA. We can find that the proportion of variances that fPCA can explain are bigger than those of dynamic fPCA. So, the conclusion is reached that for the data of egg count, there is no improvement for dynamic fPCA compared with fPCA and the reason may be that the fecundity of each fruit fly is independent of each other.

3 Functional Regression for Scalar Response

Functional regression is an active area of research, and the approach depends on whether the responses or dependence are functional or vector data and include combinations of functional responses with functional dependence, vector responses with functional dependence, and functional responses with vector dependence. In **Section 2**, I have been exploring the variability of a functional variable without asking how much of its variation is explainable by other variables. The main point in **Section 3** is modeling the most easier relationship between scalar response and functional dependence. In classical statistics, the analysis of variance, linear regression and the general linear model serve this purpose, so we now extend the notion of a linear model to the functional object. In the following content, I will apply four regression methods: restricted basis functions(SfR), regularization with roughness penalty(PfR), principal component (PCfR) and penalized principal component regression (PPCfR) to analysis the relationship between total lifespan of the fruit flies from their egg laying.

3.1 Principal Component Regression

Because of the finite and discrete dependence information $X(t)$, the linear transformation from the parameter space to the observation space was still specified by a design matrix Z as in the conventional multivariate general linear model. So, the functional PCA give us a naive idea to model the discrete dependence variable and then just proceed with ordinary multiple regression (see Eq. 5).

$$Y = Zb + \epsilon. \tag{5}$$

Given the first three principle components score, we have the 50×3 design matrix Z, then we can use standard multiple regression to fit this model by least squares and we named this method as principal component regression (PCfR).

We can summarize the fit in terms of the conventional $R2 = 1SSE/SST$ measure, and this is 0.9659, indicating a rather successful fitting, even taking into account the 4 parameters in the model. The corresponding $F - ratio$ is 484.5 with 4 and 46 degrees of freedom, and is significant at the 1% level.

3.2 Restricted Basis Functions Regression

On the other hands, we can apply the *FLM* (Eq. 6), introduced by Ramsay and Dalzell [Ramsay1991], to build functional regression models with scalar response.

$$\underline{Y(s) = \beta_0(s) + \int_{I_x} \beta(s,t)X(t)d_t + \epsilon(s).} \tag{6}$$

This is function-on-function model! not scalar-on-function model!

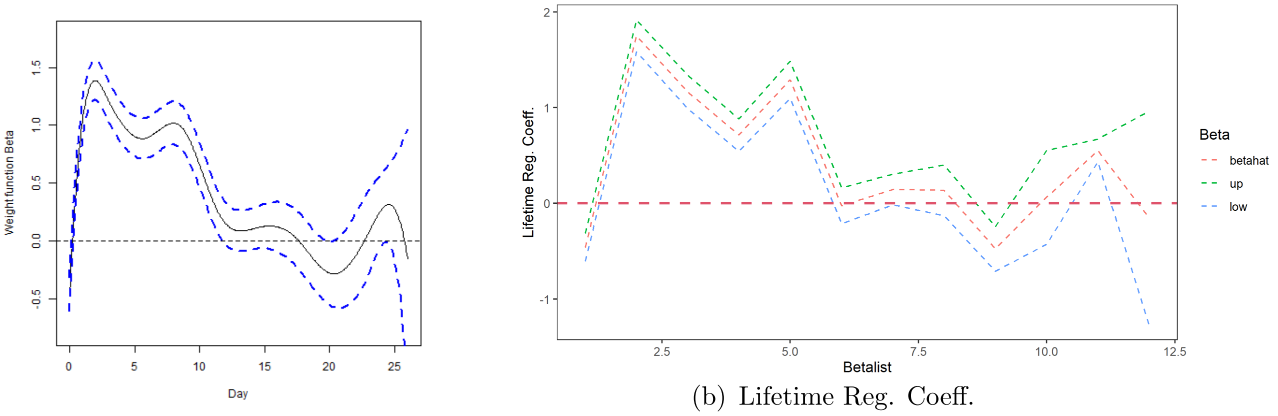
Same with the PCfR, the core idea is to expand the regression dependence $X(t)$ in terms of a set of basis functions ϕ_k, and then expand the coefficient function β in the same functional basis, such as the Bspline basis for fruit flies data (see **Proj. A1**). Specifically, consider an orthonormal basis ϕ_k of the function space, Expanding both X and β in this basis leads to $X(t) = \sum_{k=1}^{\infty} A_k\phi(t)$, $\beta(t) = \sum_{k=1}^{\infty} \beta_k\phi(t)$ and the functional regression model (Eq. 6) is seen to be equivalent to the scalar linear model of the forms

$$Y = \beta_0 + \sum_{k=1}^{\infty} \beta_k A_k + \epsilon. \tag{7}$$

Why K=12 ?

where in implementations the sum on the right side is replaced by a finite sum that is truncated at the first $K = 12$ terms (we restricts the basis function num equaling to 12) and we name this method as restricted basis functions(SfR). By splitting the fruit flies data into train and test data (7:3), we have the $R^2 = 0.9923$ and out-of-sample $osR^2 = 0.9806$ ($osR^2 = 1 - SSE/SST|_{test}$), this results show a perfect performance to predict the total lifespan. Fig. 5 has shown the estimated weight function and regression coefficient along with point-wise 95% confidence limits for predicting the total lifespan from the daily egg count.

please do not define a new terminology yourself.



(a) Weight function Beta

(b) Lifetime Reg. Coeff.

Fig. 5: Restricted basis functions regression.

The confidence intervals in the later day contain zero, suggesting that the influence of egg laying on lifespan in that period is not important. However, we see a strong peak in the day 4 and day 8 followed by a decreasing. This pattern is, in effect, favoring the egg count laying is decreasing along with time.

3.3 Regularization with Roughness Penalty Regression

To obtain more reasonable results, β must be rigidly constrained to lie in a low-dimensional parametric family, and we may worry that we are missing important features in β as a consequence. So, the roughness penalty approach is introduced to trade-off this balance and aims to avoid excessive local fluctuation in the estimated function. The penalized residual sum of squares is given by

$$PENSSE_{\lambda} = \sum_{i=1}^{N} [y_i - \beta_0 - \int x_i(s)\beta(s)d_s]^2 + \lambda \int [L\beta(s)]^2 d_s, \tag{8}$$

where L is a linear differential operator that is suitable for the problem and we name this method as regularization with roughness penalty(PfR). Further, we can choose the smoothing parameter λ by cross-validation and the CV score defined as

$$CV(\lambda) = \sum_{i=1}^{N}[y_i - \beta_0^{(-i)} - \int x_i(s)\beta_\lambda^{(-i)}(s)d_s]^2. \tag{9}$$

and minimizing $CV(\lambda)$ over λ gives an automatic choice of λ.

For fruit flies data, I have used 12 basis functions to represent the egg count curves and 35 Bspline basis functions (train data full model) to represent β. With this number of basis functions for β, it would be possible to exactly fit the data from the 35 training data. To see how well cross-validation would help us in arriving at a reasonable fit by penalizing D^2. Fig. 6 plots the cross-validation score against the logarithms of various values of λ.

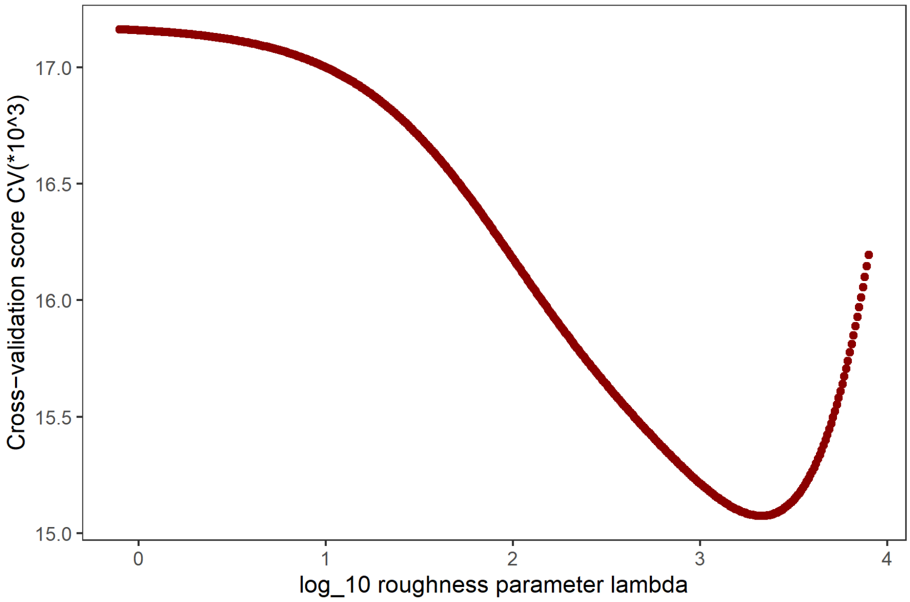
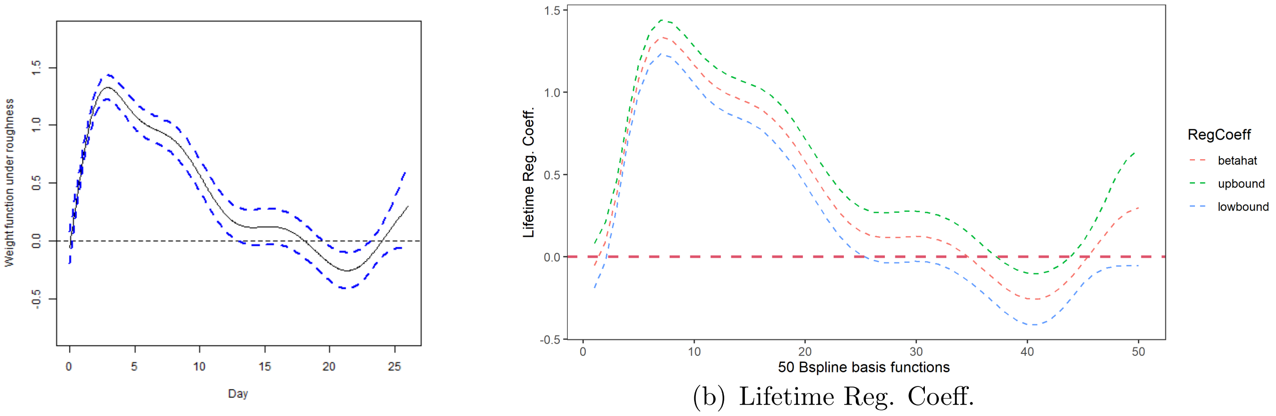


Fig. 6: The cross-validation score function $CV(\lambda)$ for fitting lifespan by daily egg count variation, with a penalty on the size of D^2. The logarithm of the smoothing parameter is taken to base 10.

From the fitting results, we have the $R^2 = 0.9871$ and out-of-sample $osR^2 = 0.9784$, this results also show a perfect performance to predict the total lifespan. Compared with SfR, the confidence intervals plot (see Fig. 7) has the same meaning to illustrate the β, but we can see a more smoothed results, i.e. we have avoided the local fluctuation in the estimated function. However, both of measure value are smaller.

3.4 Penalized Principal Component Regression

To make our results more reliable, I have also tried penalized principal component regression (PPCfR) model, these results are given by **fda.usc** package. For the no-



(a) Weight function Beta

(b) Lifetime Reg. Coeff.

Fig. 7: Regularization with roughness penalty regression.

tation, the PPCfR1 denotes penalized under fixed PC and PPCfR2 denotes that we tuning the penalty through CV and finding the min(CV) model. Tab. 2 has shown the comparison results of different regression model, which shows the restricted basis functions regression (SfR) performs better than any others, and PPCfR2 performs better than PCfR.

Good!

Tab. 2: Comparison of model fitted and predictive for different regressions

Measurement	SfR	PfR	PCfR	PPCfR1	PPCfR2
R2	0.9923	0.9871	0.9695	0.9435	0.9871
osR2	0.9806	0.9784	-	0.9195	0.9655

4 Conclusions

- Regularized PCA perform better and can explain more information using few components.
- Interpreting the eigencomponents is not always an entirely straightforward in functional PCA problems, the eigencomponents as perturbations of the mean can aid their interpretation. For egg count in this report, the 2nd eigenfunction shows a variation around a linear time trend with a break point near day 10, reflecting that the egg count laying is decreasing along with time, 3rd and 4th eigenfunction shows there is a cycle in the egg count laying.
- Dynamic PCA is useful only if $X(t)$ satisfy weakly stationary, if there is no relation between any two $X_i(t)$ and $X_{i+1}(s)$, dynamic fPCA will not perform better than fPCA.
- Functional regression depends on whether the responses or dependence are functional or vector data. For scalar-functional regression, this model can be viewed as an extension of the traditional multivariate linear model that associates vector responses with vector dependence by smoothing or regularizing the functional object.
- There are many methods to model scalar-functional data, I have outlined four parts and found out the restricted basis functions regression performs better than others.

References

[1] Jolliffe IT.. Principal Component Analysis. (2nd ed.). 2002. New York: Springer.

[2] Kleffe J.. Principal components of random variables with values in a separable Hilbert space. Math. Oper..Stat. 4:391–406.

[3] Wang, Jane-Ling and Chiou, Jeng-Min. Functional Data Analysis (June 2016). Annual Review of Statistics and Its Application, Vol. 3, Issue 1, pp. 257-295.

[4] Oleg Melnikov and Loren H. Raun. Dynamic Principal Component Analysis: Identifying the Relationship between Multiple Air Pollutants.

[5] Ramsay JO and Dalzell C. Some tools for functional data analysis. J. R. Stat. Soc. Ser. B 53:539–72.

Appendix

GWW 12031299

2021/4/16

```
rm(list = ls())
library(fda)
library(ggplot2)
library(reshape2)
library(freqdom)
setwd("E://R Files/Functional Data")
load('Data Sets/fly.Rdata')
eggcount = medfly$eggcount
day = 1:26
```

fPCA and dynamic fPCA

Smooth the data under best basis function num 12 basis

```
# eggcountfdata = fdata(t(medfly$eggcount),names=list("day","eggcount")) # convert data to fdata
# lifespan = medfly$lifetime
# nb = floor(seq(5,26,len=6))
# lam = 2^(-5:15)
# optbasis = optim.basis(eggcountfdata,lambda=lam,numbasis=nb,type.basis="bspline")
# basis_choice = data.frame(lam, t(optbasis$gcv))
# colnames(basis_choice) = c("lambda","numbasis1","numbasis2","numbasis3")
# png(file = "Basis_S.png")
# matplot(t(optbasis$gcv),type="l",main="GCV with bspline basis",xlab="num of bspline",ylab="GCV value"
# dev.off()

bbasis_ = create.bspline.basis(c(0,26), nbasis = 12, norder=4)
S_dayEggfd = with(medfly, smooth.basis(day,eggcount,
        bbasis_, fdnames=list("Day", "lifespan", "eggcount"))$fd)
# with(medfly, plotfit.fd(eggcount,day,S_dayEggfd,ask=TRUE)) # a Functional Data Object With Data
png(file = "Smooth.png")
plot(S_dayEggfd,ylim=c(-20,120)) # the aligned original recordings of the force relative to a baseline
```

```
## [1] "done"
```

```
dev.off()
```

```
## pdf
##   2
```

```
S_eggcountpcaobj = pca.fd(fdobj=S_dayEggfd,nharm=4,fdPar(bbasis_))
S_eggcountpcaobj$varprop # 0.3745757 0.2957662 0.1685607 0.0437355
```

```
## [1] 0.3745757 0.2957662 0.1685607 0.0437355
```

```
pcanum = c(1:8)
S_eigenvalues = S_eggcountpcaobj$values[1:8]
S_cum_eigenvalues = cumsum(S_eigenvalues)/sum(S_eigenvalues)

variation_S = ggplot(NULL,aes(pcanum,S_cum_eigenvalues))+geom_point(color = "darkred")+geom_line(linetyp
ggsave(variation_S,filename = "variationS.png",width = 8,height = 4)

S_harmfd = S_eggcountpcaobj$harmonics # new basis function given by PCA
S_harmvals = melt(data.frame(day,eval.fd(day,S_harmfd)),id="day") # PC values
colnames(S_harmvals) = c("day","PC","value")
pc_S = ggplot(S_harmvals,aes(day,value,group=PC,color=PC,shape=PC))+geom_line(linetype="dashed")+theme_
ggsave(pc_S,filename = "pcS.png",width = 8,height = 4)
```

Roughness smooth the data first, then carry out an unsmoothed PCA

```
bbasis = create.bspline.basis(rangeval=c(0,26), nbasis = 26, norder=4)
D2fdPar = fdPar(bbasis,Lfdobj=int2Lfd(2),lambda=10) # best lambda is given by A1
P_dayEggfd = smooth.basis(day,eggcount,D2fdPar,
                          fdnames=list("Day", "lifespan", "eggcount"))$fd
png(file = "Roughness.png")
plot(P_dayEggfd)
```

```
## [1] "done"
```

```
dev.off()
```

```
## pdf
##   2
```

```
eggcountpcaobj = pca.fd(fdobj=P_dayEggfd,nharm=4, harmfdPar=D2fdPar)
pcanum = c(1:8)
eigenvalues = eggcountpcaobj$values[1:8]
cum_eigenvalues = cumsum(eigenvalues)/sum(eigenvalues)
setwd("E://R Files/Functional Data/Projects/A2")
write.table(eggcountpcaobj$scores[,1:4],"eggcount-pc4", col.names = c("PC1","PC2","PC3","PC4"))

variation_P = ggplot(NULL,aes(pcanum,cum_eigenvalues))+geom_point(color = "darkred")+geom_line(linetype=
        theme_bw()+theme(panel.grid=element_blank())+xlab("No of component") +ylab("cumulative variance
ggsave(variation_P,filename = "variationP.png",width = 8,height = 4)

harmfd = eggcountpcaobj$harmonics # new basis function given by PCA
harmvals = data.frame(day,eval.fd(day,harmfd)) # PC values
harmvals = melt(harmvals,id="day")
colnames(harmvals) = c("day","PC","value")
pc_P = ggplot(harmvals,aes(day,value,group=PC,color=PC,shape=PC))+geom_line(linetype="dashed")+theme_bw
ggsave(pc_P,filename = "pcP.png",width = 8,height = 4)

sharmvals = eval.fd(day,harmfd)%*%diag(sqrt(eggcountpcaobj$values[1:4]))
colnames(sharmvals) = c("PC1","PC2","PC3","PC4")
sharmvals = melt(data.frame(day,sharmvals),id="day")
colnames(sharmvals) = c("day","PC","value")
pc_P_c = ggplot(sharmvals,aes(day,value,group=PC,color=PC,shape=PC))+geom_line(linetype="dashed")+theme_
ggsave(pc_P_c,filename = "pcPcentered.png",width = 8,height = 4)
```

Visualizing the results: plotting components as perturbations of the mean)

```
plot.pca.fd(eggcountpcaobj, cex.main=0.9) # the effect on the overall mean curve of adding and subtract
```

```
mean_ = eggcountpcaobj$meanfd$coefs
effects = matrix(0,26,4)
d = rep(0,4)
eigen_ = eval.fd(day,eggcountpcaobj$harmonics)
for (i in c(1:4)){
  d[i] = sqrt(eggcountpcaobj$values)[i]
  effects[,i] = 2*d[i]*eigen_[,i]
} # the variation in the functional data that is contributed by each eigenfunction
up_ = mean_+effects[,1]
lower_ = mean_-effects[,1]
perturbations = data.frame(day,mean_,up_,lower_)
perturbations = melt(perturbations,id="day")
colnames(perturbations) = c("day","PC","value")
effect_ = ggplot(perturbations,aes(day,value,group=PC,color=PC,shape=PC))+geom_line(linetype="dashed")+
ggsave(effect_,filename = "effect.png",width = 8,height = 4)
eggcountvarmx = varmx.pca.fd(eggcountpcaobj) # Varimax rotation: find a more interpretable basis
plot(eggcountvarmx)
```

Dynamic fPCA: DPCA decomposition outputs components which are uncorrelated in time

```
X_eggcount = as.matrix(eggcount)
eggcountpca = prcomp(X_eggcount, center=TRUE,rank.=4)
summary(eggcountpca) # the explained variation is smaller than no roughness PCA
```

```
## Importance of first k=4 (out of 26) components:
##                            PC1     PC2      PC3      PC4
## Standard deviation     83.6837 65.5236 36.10488 33.83685
## Proportion of Variance  0.3445  0.2112  0.06413  0.05633
## Cumulative Proportion   0.3445  0.5558  0.61989  0.67622
```

```
eggcountdpca = dpca(X_eggcount, q=5, Ndpc=4)
dvar_ = eggcountdpca$var
cat("var contribution from dpca output \n \t",dvar_[1:4])
```

```
## var contribution from dpca output
##       0.3618283 0.2146082 0.0813539 0.05870535
```

```
dvariation = ggplot(NULL,aes(pcanum,cumsum(dvar_[1:8])))+geom_point(color = "darkred")+geom_line(linetyp
ggsave(dvariation,filename = "dvariation.png",width = 8,height = 4)
```

Functional Linear Models

```
R2 = function(y,y_hat,y_train_bar){
  R2 = 1-sum((y-y_hat)^2)/sum((y-y_train_bar)^2)
  return(R2)
}

osR2 = function(y_test,y_pre,y_train_bar){
  osR2 = 1-sum((y_test-y_pre)^2)/sum((y_test-y_train_bar)^2)
  return(osR2)
}

# flm_stdlist = function(y,y_hat,flm,xfd){
#   N = length(y)
#   resid = y - y_hat
#   SigmaE = sum(resid^2)/(N-flm$df)*diag(rep(1,N))
#   y2cMap = xfd$y2cMap
#   stderrList = fRegress.stderr(flm, y2cMap, SigmaE)
#   return(stderrList) # flm results and contain the beta_std, xfd is a smooth.basis that generates the
# }

flm_stdlist = function(y,y_hat,flm,basisnum){
  N = length(y)
  resid = y - y_hat
  SigmaE = sum(resid^2)/(N-flm$df)*diag(rep(1,N))
  y2cMap = diag(N)
  stderrList = fRegress.stderr(flm, y2cMap, SigmaE)
  return(stderrList) # for scaled response variable
}

betaconfInter = function(beta_hat,beta_std,beta_num){
  betanum = c(1:beta_num) # coefficient beta num
  betaup = beta_hat+2*beta_std
  betalow = beta_hat-2*beta_std
  betaconf = data.frame(betanum,betaest,betaup,betalow)
  return(betaconf)
}
```

Scalar-on-functional regression model

Regression using restricted basis functions: No roughness lambda

```
train_ind = 1:35
X_train = eggcount[,train_ind]
X_test = eggcount[,-train_ind]
bbasis_ = create.bspline.basis(c(0,26), nbasis=12, norder=4) # not full model, this result is given by
xfd = smooth.basis(day,X_train,
        bbasis_, fdnames=list("Day", "lifespan", "eggcount")) # estimated value of x(t)
xfdobj_S = xfd$fd # this fob is relevent to the xfdlist name of flm_S

lifespan = as.numeric(medfly$lifetime[train_ind])
lifespan_bar = mean(lifespan)
y_test = as.numeric(medfly$lifetime[-train_ind])

flm_S = fRegress(lifespan~xfdobj_S) # model building in another way

lifespan_fit_S = flm_S$yhatfdobj
# Fitted_S = ggplot(NULL,aes(lifespan_fit_S,lifespan))+geom_point(color = "darkred")+geom_abline(slope=
# ggsave(Fitted,filename = "Fitted_S.png",width = 8,height = 4)

R2_S = R2(lifespan,lifespan_fit_S,lifespan_bar)
```

```
print(paste("R2 =",R2_S)) # R2 = 0.992427274348569
```

```
## [1] "R2 = 0.992292521174783"
```

```
RMSE_S = sqrt(mean((lifespan-lifespan_fit_S)^2))
print(paste("RMSE =",RMSE_S))
```

```
## [1] "RMSE = 13.1860094092325"
```

```
betaeststd = flm_stdlist(lifespan,lifespan_fit_S,flm_S,xfd) # to calculate the projection matrix: solve

X_testfd = smooth.basis(day,X_test,
        bbasis_, fdnames=list("Day", "lifespan", "eggcount"))$fd
X_testfdobj = fRegress(y_test~X_testfd, method="model")$xfdlist$X_testfd

lifespan_pre_S = inprod(flm_S$betaestlist$xfdobj_S$fd, X_testfdobj) # useage of predict
osR2_S = osR2(y_test,lifespan_pre_S,lifespan_bar)
print(paste("osR2 =",osR2_S)) # 0.980643976796598
```

```
## [1] "osR2 = 0.980643976796598"
```

```
betaestfd = betaeststd$betaestlist$xfdobj_S$fd

png(file = "Betafunc_S.png")
plot(betaestfd, xlab="Day", ylab="Weight function Beta", lwd=2, col=2,col="blue",ylim=c(-0.8,1.8))
```

```
## [1] "done"
```

```
betaestfd_std = betaeststd$betastderrlist[[2]]
lines(betaestfd+2*betaestfd_std, lty=2, lwd=2, col="blue")
lines(betaestfd-2*betaestfd_std, lty=2, lwd=2, col="blue")
dev.off()
```

```
## pdf
##   2
```

```
betaest = betaeststd$betaestlist$xfdobj_S$fd$coefs # coefficient function
# betaestfd = flm_S$betaestlist$xfdobj_S$fd
# all.equal(betaestfd1,betaestfd)
betaest_std = betaeststd$betastderrlist[[2]]$coefs # betahat_std
betaconf = betaconfInter(betaest,betaest_std,12) # confidence interval
colnames(betaconf) = c("num","betahat","up","low")
betaconf = melt(betaconf,id="num")
colnames(betaconf) = c("num","Beta","value")
Beta_S = ggplot(betaconf,aes(num,value,group=Beta,color=Beta,shape=Beta))+geom_line(linetype="dashed")+
ggsave(Beta_S,filename = "Beta_S.png",width = 8,height = 4)
```

Choice of lambda

Regularization with roughness penalties: Best lambda

```
detach(package:fda)
library(fda)
X_train = eggcount[,train_ind]
X_test = eggcount[,-train_ind]
lifespan = as.numeric(medfly$lifetime[train_ind])
lifespan_bar = mean(lifespan)
```

```
y_test = as.numeric(medfly$lifetime[-train_ind])

bbasis_ = create.bspline.basis(c(0,26), nbasis=12, norder=4)
xfd = smooth.basis(day,X_train,
        bbasis_, fdnames=list("Day", "lifespan", "eggcount"))
xfdobj = xfd$fd
model = fRegress(lifespan~xfdobj, method="model")

lambda =  22510
betabasis_bbasis = create.bspline.basis(rangeval=c(0,26), nbasis = 35, norder=4)
betafd = fd(rep(1, 35), betabasis_bbasis) # betalistlist
betafdPar = fdPar(betafd, Lfdobj=int2Lfd(2), lambda=lambda) # roughness betalist
model$betalist$xfdobj = betafdPar # replace the betalist, i.e. the regress coefficient function beta(t)
flm_P = do.call('fRegress',model)
lifespan_fit = flm_P$yhatfdobj
# Fitted = ggplot(NULL,aes(lifespan_fit,lifespan))+geom_point(color = "darkred")+geom_abline(slope=1)+t
# ggsave(Fitted,filename = "Fitted.png",width = 8,height = 4)

R2_P = R2(lifespan,lifespan_fit,lifespan_bar)
print(paste("R2 =",R2_P)) # R2 = 0.987987921981976, smaller than no penalty
```

```
## [1] "R2 = 0.985502812107544"
```

```
RMSE_P = sqrt(mean((lifespan-lifespan_fit)^2))
print(paste("RMSE =",RMSE_P))
```

```
## [1] "RMSE = 18.0841914053249"
```

```
X_testfd = smooth.basis(day,X_test,
        bbasis_, fdnames=list("Day", "lifespan", "eggcount"))$fd
X_testfdobj = fRegress(y_test~X_testfd, method="model")$xfdlist$X_testfd
lifespan_pre_P = inprod(flm_P$betaestlist$xfdobj$fd, X_testfdobj) # useage of predict
osR2_P = osR2(y_test,lifespan_pre_P,lifespan_bar)
print(paste("osR2 =",osR2_P)) # 0.976094642992361
```

```
## [1] "osR2 = 0.976094642992361"
```

```
betaeststd = flm_stdlist(lifespan,lifespan_fit,flm_P,35) # scaled response
# betaeststd = flm_stdlist(lifespan,lifespan_fit,flm_P,xfd) # functional response
betaestfd = betaeststd$betaestlist$xfdobj$fd
png(file = "Betafunc.png")
plot(betaestfd, xlab="Day", ylab="Weight function under roughness", lwd=2, col=2,col="blue",ylim=c(-0.8
```

```
## [1] "done"
```

```
betaestfd_std = betaeststd$betastderrlist[[2]]
lines(betaestfd+2*betaestfd_std, lty=2, lwd=2, col="blue")
lines(betaestfd-2*betaestfd_std, lty=2, lwd=2, col="blue")
dev.off()
```

```
## pdf
##   2
```

```
betaest = betaeststd$betaestlist$xfdobj$fd$coefs
betaest_std = betaeststd$betastderrlist[[2]]$coefs
betaconf = betaconfInter(betaest,betaest_std,35)
colnames(betaconf) = c("num","betahat","upbound","lowbound")
```

```
betaconf = melt(betaconf,id="num")
colnames(betaconf) = c("num","RegCoeff","value")
Beta = ggplot(betaconf,aes(num,value,group=RegCoeff,color=RegCoeff,shape=RegCoeff))+geom_line(linetype=
ggsave(Beta,filename = "Beta.png",width = 8,height = 4)
```

Functional principal component regression

```
eggcountpc=read.table("E://R Files/Functional Data/Projects/A2/eggcount-pc4",header = TRUE,row.names =
pcscore = as.matrix(eggcountpc)[,1:3]
pclifespan = as.matrix(medfly$lifetime)
fpclm=lm(pclifespan~pcscore)
summary(fpclm)
```

```
##
## Call:
## lm(formula = pclifespan ~ pcscore)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -51.891 -20.475  -1.538  20.493  66.004
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 466.26000    4.17936 111.562  < 2e-16 ***
## pcscorePC1    2.20706    0.06934  31.832  < 2e-16 ***
## pcscorePC2    1.69000    0.08225  20.548  < 2e-16 ***
## pcscorePC3   -0.79327    0.18747  -4.231  0.00011 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 29.55 on 46 degrees of freedom
## Multiple R-squared:  0.9693, Adjusted R-squared:  0.9673
## F-statistic: 484.5 on 3 and 46 DF,  p-value: < 2.2e-16
```

```
R2_pca = R2(pclifespan,fpclm$fitted.values,lifespan_bar)
print(paste("R2 =",R2_pca)) # R2=0.969501679534734
```

```
## [1] "R2 = 0.969501679534734"
```

Functional penalized principal component regression

```
library(fda.usc)
load('E://R Files/Functional Data/Data Sets/fly.Rdata')
train_ind = 1:35
eggcountdata = t(medfly$eggcount)
eggcountfdata = fdata(eggcountdata,names=list("day","eggcount")) # convert data to fdata
X_train = eggcountfdata[train_ind,]
X_test = eggcountfdata[-train_ind,]
lifespan = medfly$lifetime[train_ind] # Scalar response
lifespan_bar = mean(lifespan)
y_test = medfly$lifetime[-train_ind]

fpcreg = fregre.pc(fdataobj=X_train,y=lifespan,kmax=4,P=c(0,0,1))# Functional penalized PC regression u
fpcreg # R2=0.9435122
```

```
##
## -Call: fregre.pc(fdataobj = X_train, y = lifespan, P = c(0, 0, 1), kmax = 4)
##
## -Coefficients:
## (Intercept)          PC1          PC2          PC3
##    478.7143       1.3478       2.6268       0.3671
##
## -R squared:  0.9435122
## -Residual variance:  1438.717
```

```
lifespan_pre_PPCA = predict(fpcreg,X_test)
osR2_PPCA = osR2(y_test,lifespan_pre_PPCA,lifespan_bar)
print(paste("osR2 =",osR2_PPCA)) # 0.919510231267232
```

```
## [1] "osR2 = 0.919510231267232"
```

```
fppcreg = fregre.pls.cv(fdataobj=X_train,y=lifespan,kmax=8,lambda=0:5,P=c(0,0,1)) # Selects the PPLS co
fppcreg$fregre.pls # R2=0.9871092
```

```
##
## -Call: fregre.pls(fdataobj = fdataobj, y = y, l = 1:pc.opt, lambda = lambda[rn.opt],     P = P)
##
## -Coefficients:
## (Intercept)         PLS1         PLS2         PLS3
##      478.71        49.09        37.08        12.56
##
## -R squared:  0.9871092
## -Residual variance:  383.8666
```

```
lifespan_pre_PPCA_cv = predict(fppcreg$fregre.pls,X_test)
osR2_PPCA_cv = osR2(y_test,lifespan_pre_PPCA_cv,lifespan_bar)
print(paste("osR2 =",osR2_PPCA_cv)) # 0.965586093772263
```

```
## [1] "osR2 = 0.965586093772263"
```