BIOS 7720: Applied Functional Data Analysis

Lecture 6: fPCA

Andrew Leroux

March 30, 2021

Roadmap

1. Brief Recap of fPCA
2. Review Methods for Estimation

Functional Principal Component Analysis (fPCA)

- Observe p (noisy) realizations of $x(s)$ a function

$$\text{Cov}(x(s), x(u)) = \Sigma(s, u); \quad s, u \in \mathcal{S}$$

- $x(s)$ theoretically continuous, observed discretely

$$\begin{aligned} y(s) &= \mu(s) + x(s) + \epsilon(s) \\ &= \mu(s) + \sum_{k=1}^{\infty} \xi_k \phi_k(s) + \epsilon(s) \quad \text{Karhunen-Loève Theorem} \\ &\approx \mu(s) + \sum_{k=1}^{K} \xi_k \phi_k(s) + \epsilon(s) \end{aligned}$$

- $\mu(s)$ is the mean function
- $\text{Cov}(\xi_l, \xi_k) = 0$ for $l \neq k$, ϕ_k are orthogonal

Functional Principal Component Analysis (fPCA)

- Basically, PCA with smoothness on the eigenfunctions
- Principal directions are $\phi_k(s)$
- Scores are $\xi_k = \int_S x(s)\phi_k(s)ds$
- Normalized eigenfunctions $\int \phi_k(s)\phi_k(s) = 1$
- Orthogonality $\int \phi_j(s)\phi_k(s) = 0$ for $j \neq k$
- How to choose K?
  - Fix K a-priori
  - Percent variance explained (e.g. 95%)

Simulating Data

- Simulate data according to the model:

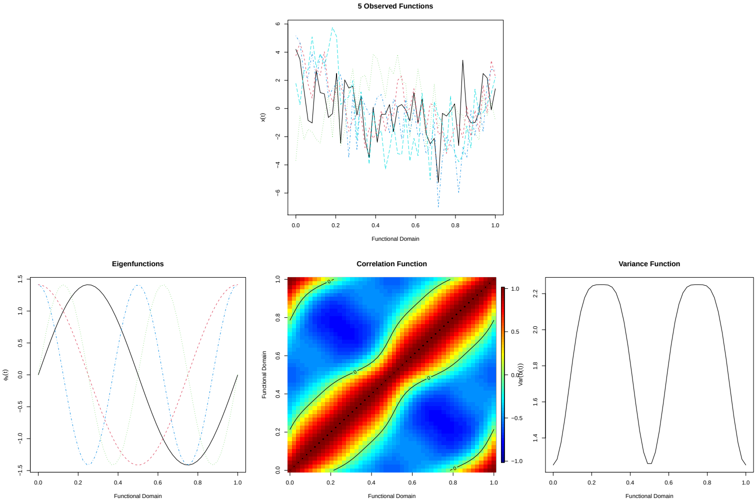
$$y_i(s) = \mu(s) + \sum_{k=1}^{4} \xi_{ik}\phi_k(s) + \epsilon_i(s)$$

- $\mu(t) = 0$
- $\mathcal{S} = [0, 1]$
- ϕ_k alternating sin/cos with decreasing period
- $\xi_k \sim N(0, 0.5^(k-1))$
- Observe $y_i(s)$ on a regular grid with $J = 50$ observations per curve

Simulating Data


```
set.seed(19840)
# simulation settings
N  <- 100  # number of functions to simulate
ns <- 50   # number of observations per function
sind <- seq(0,1,len=ns) # functional domain of observed functions
K  <- 4    # number of true eigenfunctions
lambda <- 0.5^(0:(K-1))   # true egenfunctions
sig2 <- 2  # error variance
# set up true eigenfunctions
Phi <- sqrt(2)*cbind(sin(2*pi*sind), cos(2*pi*sind),
                     sin(4*pi*sind), cos(4*pi*sind))
# simulate coefficients
# first, simulate standard normals, then multiply by the
# standard deviation to get correct variance
xi_raw <- matrix(rnorm(N*K), N, K)
xi     <- xi_raw %*% diag(sqrt(lambda))
# simulate functional responses as \sum_k \xi_ik \phi_k(t)
x <- xi %*% t(Phi)
y <- x + matrix(rnorm(N*ns, mean=0, sd=sqrt(sig2)), N, ns)
```

Simulating Data



Estimation

- Covariance smoothing
  - Kernel smoothing [Staniswalis and Lee, 1998, Yao et al., 2005]
  - Penalized splines [Di et al., 2009, Xiao et al., 2016]
- Maximum Likelihood [Peng and Paul, 2009]

fPCA: Kernel Smoothing

- Follow [Yao et al., 2005]
- Formulated for sparse/irregularly sampled data
- Some additional notation required:

$$
\begin{aligned}
y_i(S_{ij}) &= \mu(S_{ij}) + \sum_{k=1}^{\infty} \xi_{ik}\phi_k(S_{ij}) \\
&\approx \mu(S_{ij}) + \sum_{k=1}^{K} \xi_{ik}\phi_k(S_{ij}) \\
&= \mu(S_{ij}) + \boldsymbol{\Phi\xi}
\end{aligned}
$$

- S_{ij} random, iid with some marginal density
- Estimate μ, $\boldsymbol{\Sigma}$ $(\boldsymbol{\Phi})$ via kernel smoothing

fPCA: Kernel Smoothing

1. Estimate $\mu(s)$ using univariate kernel smoothing
2. Estimate the off-diagonal of $\boldsymbol{\Sigma}$ using bivariate kernel smoothing
3. Estimate error variance σ^2 as a weighted average of the difference between smoothed covariance and raw covariance
4. Kernel smoothing methods have a tuning parameter "bandwidth"
5. Select using leave-one-curve-out cross-validation

fPCA: Kernel Smoothing in R

```
library("fdapace")
# create inputs for the fdapace::FPCA function
Ly <- Lt <- vector(mode="list", length=N)
for(i in 1:N){
  # create vector of observed data
  Ly[[i]] <- y[i,]
  # create vector of observed T_ij (functional domain)
  # note that because we have no missing data and all observations
  # are on the same grid, this is the same for each function
  Lt[[i]] <- sind
}
time_start_ks <- Sys.time()
fit_ks        <- FPCA(Ly=Ly, Lt=Lt,
                      optns=list(dataType="Sparse", error=TRUE,
                                 FVEthreshold=0.95))
time_end_ks   <- Sys.time()
difftime(time_end_ks, time_start_ks, units="mins")

## Time difference of 3.201055 mins
```

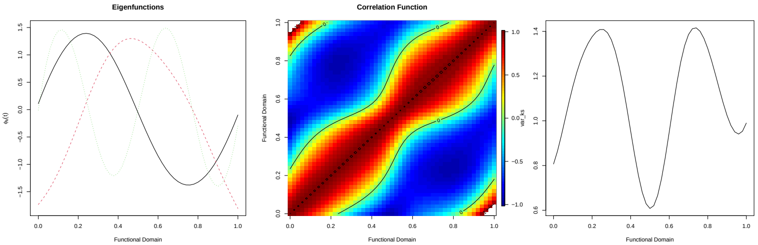
fPCA: Kernel Smoothing in R


```
str(fit_ks,max.level=1)

## List of 20
##  $ sigma2        : num 2.24
##  $ lambda        : num [1:3] 0.7118 0.3542 0.0645
##  $ phi           : num [1:51, 1:3] 0.109 0.296 0.472 0.637 0.789 ...
##  $ xiEst         : num [1:100, 1:3] 0.341 0.536 0.244 1.469 1.323 ...
##  $ xiVar         :List of 100
##  $ obsGrid       : num [1:50] 0 0.0204 0.0408 0.0612 0.0816 ...
##  $ workGrid      : num [1:51] 0 0.02 0.04 0.06 0.08 0.1 0.12 0.14 0.16 0.18 ...
##  $ mu            : num [1:51] -0.05963 -0.00631 0.03447 0.06401 0.08211 ...
##  $ smoothedCov   : num [1:51, 1:51] 0.807 0.824 0.825 0.809 0.775 ...
##  $ FVE           : num 96.2
##  $ cumFVE        : num [1:3] 60.5 90.7 96.2
##  $ fittedCov     : num [1:51, 1:51] 1.206 1.134 1.055 0.969 0.875 ...
##  $ optns         :List of 33
##  $ bwMu          : num 0.05
##  $ bwCov         : num 0.1
##  $ rho           : num 0.238
##  $ inputData     :List of 2
##  $ selectK       : int 3
##  $ criterionValue: num 96.2
##  $ timings       : 'difftime' Named num [1:4] 192.061 0.011 0.783 191.262
##   ..- attr(*, "units")= chr "secs"
##   ..- attr(*, "names")= chr [1:4] "total" "mu" "cov" "pace"
##  - attr(*, "class")= chr "FPCA"
```

fPCA: Kernel Smoothing in R

```r
## extract estimated (co)variance and correlation functions
cov_ks <- fit_ks$smoothedCov
var_ks <- diag(cov_ks)
cor_ks <- cov2cor(cov_ks)
sd_ks <- sqrt(var_ks)
cor_ks2 <- cov_ks/tcrossprod(sd_ks)
## extract estimated eigenfunctions
efuncs_ks <- fit_ks$phi
## get grid of the functional domain on which
## quantities are estimated
sind_ks <- fit_ks$workGrid
```

fPCA: Kernel Smoothing in R



fPCA: Penalized Splines

1. Estimate $\mu(t)$ using penalized splines under working independence
2. Smooth the covariance matrix using bivariate penalized splines
   - MoM estimate of $\Sigma(s, u)$ versus $(y(s) - \mu(s))(y(u) - \mu(s))$
   - Just the upper (or lower) triangular
   - Entire Covariance matrix
3. Ensure smoothed covariance matrix is a proper covariance matrix
4. Obtain estimated eigenfunctions
5. Estimate scores (ξ) given the estimated covariance
6. (Optional) obtain variance estimates
   - Variance conditional on estimated eigenfunctions
   - Incorporate variability in estimating eigenfunctions (using, e.g., bootstrap)

Estimating $\mu(t)$

- Suppose we have regularly observed data ($T_{ij} = T_j$)
- Fit the independence model, estimate $\mu(t)$ using penalized splines

$$
\begin{aligned}
y_i(t_j) &= \mu(t_j) + \epsilon_i(t_j) \\
&= \sum_{k=1}^{K^\mu} \xi_k^\mu + \epsilon_i(t_j) \\
\epsilon_i &\sim N(0, \sigma_\epsilon^2)
\end{aligned}
$$

- Center the functions so that $E[y(t)] = 0$ using the estimated mean function

$$\tilde{y}_i(t_j) = y_i(t_j) - \hat{\mu}(t_j)$$

Estimating $\Sigma(s, u)$

- Notation

$$\tilde{\boldsymbol{y}}_i = [\tilde{y}_{i1}, \ldots, \tilde{y}_{iJ}]^t$$
$$\tilde{\boldsymbol{y}} = [\tilde{\boldsymbol{y}}_1, \ldots, \tilde{\boldsymbol{y}}_N]$$

- Method of Moments estimate

$$\hat{\Sigma} = N^{-1} \tilde{\boldsymbol{y}} \tilde{\boldsymbol{y}}^t$$

- Call to mgcv of the form:

```
library("mgcv")
df_mu  <- data.frame(y=as.vector(y), t=rep(sind,each=N))
mu_fit <- gam(as.vector(y) ~ s(t, bs="cr",k=30),
              method="REML", data=df_mu)
mu_est <- predict(mu_fit, newdata=data.frame(t=sind),type='response')
ytilde <- y - matrix(mu_est, N, ns, byrow=TRUE)
```

Smoothing $\Sigma(s, u)$: Additive model

- Fit the model:

$$E[\tilde{y}(s)\tilde{y}(u)] = f(s, u)$$

- Where $f(s, u)$ is a bivariate function estimated using penalized splines
- Setting up the data

$$\begin{aligned} \boldsymbol{z}_y &= \mathrm{vec}(\hat{\Sigma}) \\ \boldsymbol{z}_{t_1} &= \boldsymbol{t} \otimes 1_{J\times 1} \\ \boldsymbol{z}_{t_2} &= 1_{J\times 1} \otimes \boldsymbol{t} \end{aligned}$$

- Call to mgcv of the form:

```
Sigma_hat <- crossprod(ytilde)/N
zy  <- as.vector(Sigma_hat)
zt1 <- rep(sind, each=ns)
zt2 <- rep(sind, ns)
fit_Sigma <- gam(zy ~ s(zt1, zt2, k=30), method="REML", weights=rep(N,ns*ns))
```

Smoothing $\Sigma(s, u)$: Additive model


```
## get predictions from the model fit at the range of observed values
df_pred  <- data.frame(zt1=zt1, zt2=zt2)
Sigma_sm <- predict(fit_Sigma, newdata=df_pred, type='response')
## transform into a matrix
## note this matrix is not symmetric!
## impose symmetry ourselves
Sigma_sm <- matrix(Sigma_sm, ns, ns)
Sigma_sm <- (Sigma_sm + t(Sigma_sm))/2
```

Choosing "K": Additive Model, PVE

- We can choose the number of eigenfunctions based on % variance explained
- Do eigen decomposition on the smoothed covariance matrix estimate
- Define $\text{PVE}(N_k) = \sum_{k=1}^{N_k} \lambda_k / \sum_{k=1}^{J} \lambda_k$
- Because smoothed $\hat{\Sigma}$ may not be positive semi-definite (negative eigenvalues), only consider positive eigenvalues
- Select K as the smallest K such that $\text{PVE}(K) > c$ where c is your pre-specified threshold (e.g. 95%)
- Note that by default the eigenvectors and values won't have the right scaling (homework question!)

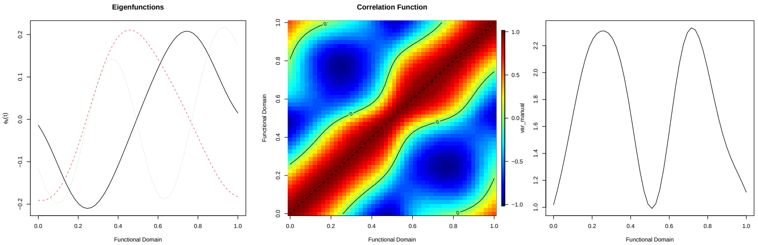
Choosing “K": Additive Model, PVE


```
## eigendecomposition of the smoothed covariance
eigen_Sigma_sm <- eigen(Sigma_sm, symmetric = TRUE)
evals_raw <- eigen_Sigma_sm$values
## choose K
evals_pos <- evals_raw[evals_raw >= 0]
c <- 0.95
(K <- min(which(cumsum(evals_pos)/sum(evals_pos) >= c)))

## [1] 3

## get eigenfucntions, eigenvalues, covariance
efuncs <- eigen_Sigma_sm$vectors[,1:K]
evals  <- eigen_Sigma_sm$values[1:K]
Sigma_manual <- efuncs %*% diag(evals) %*% t(efuncs)
cor_manual <- cov2cor(Sigma_manual)
var_manual <- diag(Sigma_manual)
```

Estimating Scores ξ_i

- Recall $y_i(s) = \mu(s) + \sum_{k=1}^{K} \xi_{ik}\phi_k(s) + \epsilon_i(s)$
- In order to get predictions for $\hat{y}$ we need to estimate ξ
- Two main methods for estimating ξ (homework)
  - Numeric integration
  - Best Linear Unbiased Predictor (BLUP)

Plotting Results



fPCA: Penalized Splines in *R refund::fpca.sc()*


```r
library("refund")
## fit fpca using refund::fpca.sc()
time_start_ps_sc <- Sys.time()
fpca_ps_sc <- fpca.sc(Y=y, argvals=sind, nbasis=10, pve=0.95)
time_end_ps_sc <- Sys.time()
difftime(time_end_ps_sc,time_start_ps_sc, units="mins")

## Time difference of 0.003729482 mins
```

fPCA: Penalized Splines in *R refund::fpca.sc()*


```
str(fpca_ps_sc)

## List of 8
##  $ Yhat      : num [1:100, 1:50] 1.47 1.99 -1.03 2.61 1.59 ...
##  $ Y         : num [1:100, 1:50] 4.2 3.75 -3.68 5.16 1.78 ...
##  $ scores    : num [1:100, 1:4] -0.313 -0.458 -0.265 -1.37 -1.224 ...
##  $ mu        : num [1:50(1d)] 0.0706 0.07 0.0693 0.0686 0.0678 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ : chr [1:50] "1" "2" "3" "4" ...
##  $ efunctions: num [1:50, 1:4] -0.00675 -0.19236 -0.37443 -0.54946 -0.71391 ...
##  $ evalues   : num [1:4] 1.018 0.434 0.234 0.143
##  $ npc       : int 4
##  $ argvals   : num [1:50] 0 0.0204 0.0408 0.0612 0.0816 ...
##  - attr(*, "class")= chr "fpca"
```

fPCA: Penalized Splines in *R refund::fpca.sc()*

```
## get estimated eigenfunctions
efuncs_sc <- fpca_ps_sc$efunctions
## get estimated covariance
cov_sc <- efuncs_sc %*% diag(fpca_ps_sc$evalues) %*% t(efuncs_sc)
cor_sc <- cov2cor(cov_sc)
## get estiamted variance function
var_sc <- diag(cov_sc)
```

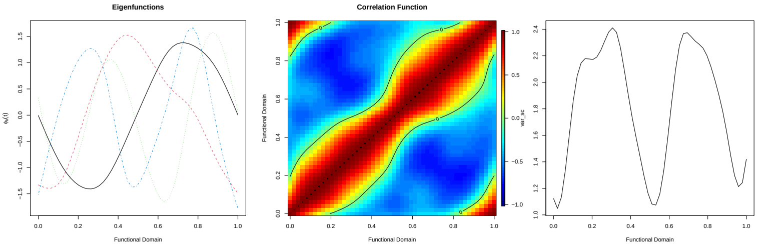
fPCA: Penalized Splines in *R refund::fpca.sc()*


```
## get estimated eigenfunctions
efuncs_sc <- fpca_ps_sc$efunctions
## get estimated covariance
cov_sc <- efuncs_sc %*% diag(fpca_ps_sc$evalues) %*% t(efuncs_sc)
cor_sc <- cov2cor(cov_sc)
## get estiamted variance function
var_sc <- diag(cov_sc)
```

fPCA: Penalized Splines in *R refund::fpca.sc()*


```
## get estimated eigenfunctions
efuncs_sc <- fpca_ps_sc$efunctions
## get estimated covariance
cov_sc <- efuncs_sc %*% diag(fpca_ps_sc$evalues) %*% t(efuncs_sc)
cor_sc <- cov2cor(cov_sc)
## get estiamted variance function
var_sc <- diag(cov_sc)
```

fPCA: Penalized Splines in *R refund::fpca.sc()*



fPCA: Penalized Splines Fast Covariance Smoothing

- Both Kernel and additive smooth (described above) can be quite slow, particularly for very large J
- Very fast method for smoothing the covariance matrix without ever having to construct it [Xiao et al., 2016]

fPCA: Penalized Splines in *R refund::fpca.face()*


```
library("refund")
## fit fpca using refund::fpca.face()
time_start_ps_face <- Sys.time()
## note: this number of
fpca_ps_face <- fpca.face(Y=y, argvals=sind,knots=35, pve=0.95)
time_end_ps_face <- Sys.time()
difftime(time_end_ps_face,time_start_ps_face, units="mins")

## Time difference of 0.0001729012 mins
```

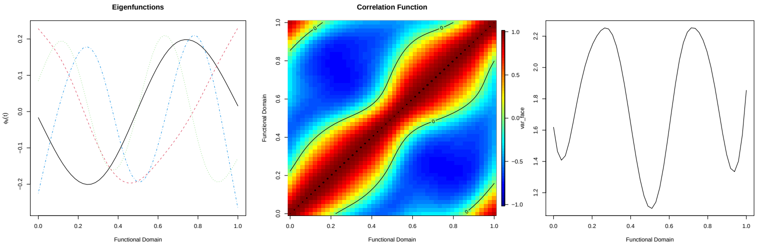
fPCA: Penalized Splines in *R refund::fpca.face()*


```
str(fpca_ps_face)

## List of 7
##  $ Yhat      : num [1:100, 1:50] 1.75 2.97 -1.58 3.49 2.59 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : NULL
##  $ Y         : num [1:100, 1:50] 4.15 3.7 -3.74 5.1 1.73 ...
##  $ scores    : num [1:100, 1:4] -2.27 -3.37 -1.73 -9.74 -8.68 ...
##  $ mu        : num [1:50] 0.0519 0.0533 0.0546 0.0558 0.0568 ...
##  $ efunctions: num [1:50, 1:4] -0.0166 -0.0393 -0.0618 -0.084 -0.1053 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : NULL
##  $ evalues   : num [1:4] 49.9 23.39 8.39 6.46
##  $ npc       : int 4
##  - attr(*, "class")= chr "fpca"
```

fPCA: Penalized Splines in *R refund::fpca.face()*


```
## get estimated eigenfunctions
efuncs_face <- fpca_ps_face$efunctions
## get estimated covariance
cov_face <- efuncs_face %*% diag(fpca_ps_face$evalues) %*% t(efuncs_face)
cor_face <- cov2cor(cov_face)
## get estiamted variance function
var_face <- diag(cov_face)
```

fPCA: Penalized Splines in *R refund::fpca.face()*



References I

Di, C.-Z., Crainiceanu, C. M., Caffo, B. S., and Punjabi, N. M. (2009).
Multilevel functional principal component analysis.
The Annals of Applied Statistics, 3(1):458 – 488.

Peng, J. and Paul, D. (2009).
A geometric approach to maximum likelihood estimation of the functional principal components from sparse longitudinal data.
Journal of Computational and Graphical Statistics, 18(4):995–1015.

Staniswalis, J. G. and Lee, J. J. (1998).
Nonparametric regression analysis of longitudinal data.
Journal of the American Statistical Association, 93(444):1403–1418.

Xiao, L., Zipunnikov, V., Ruppert, D., and Crainiceanu, C. (2016).
Fast Covariance Smoothing for High Dimensional Functional Data.
Statistics and Computing, 26(1):409 – 421.

Yao, F., Müller, H.-G., and Wang, J.-L. (2005).
Functional data analysis for sparse longitudinal data.
Journal of the American Statistical Association, 100(470):577–590.