BIOS 7720: Applied Functional Data Analysis

Lecture 10: Function on Scalar Regression (cont)

Andrew Leroux

April 13, 2021

Roadmap

1. Homework 2
2. Homework 1
3. Final Project Works in Progress Presentations
4. Continue FoSR

Homework 2

[switch content]

Distribution of Grades

- Mean: 94.04
- Median: 97.03
- SD: 8.93

Key Concepts

- Efficient programming
- Effective writing
- Visualization
- Proofs

Key Concepts: Efficient Programming

"Premature optimization is the root of all evil"
- Sir Tony Hoare

- Balance efficiency and time spent writing code:
- Some questions to ask yourself:
  - Will anyone besides myself use this code?
  - How often will I need to reuse this code in the current project?
  - Will I ever need this code again after the current project is complete?

Key Concepts: Efficient Programming

- Simulation studies are great places to look for inefficiencies
- Some low hanging fruit for efficiency gains
  - Avoid duplicate calculations
  - A little bit of linear algebra can go a long way
  - To the extent possible, set up storage containers in advance
  - Don't append using *rbind()*

Efficient Programming: Avoiding Duplicate Calculations

- Example using grid search for smoothing parameter selection
- Model: $y_i = f(x_i) + \epsilon_i$
- Apply spline basis $f(x_i) = \phi(x_i)^t \boldsymbol{\xi}$. In matrix form we have:

$$\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{\xi} + \boldsymbol{\epsilon}$$
$$\boldsymbol{\epsilon} \sim N(0, \sigma_\epsilon^2 \boldsymbol{I})$$

- We want to minimize the penalized least squares criteria

$$\text{PENSSE}_\lambda = (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\xi})^t(\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\xi}) + \lambda \boldsymbol{\xi}^t \boldsymbol{S} \boldsymbol{\xi}$$

- Where λ is selected using GCV via grid search

Efficient Programming: Duplicate Calculations/Linear Algebra


```
my_gcv1 <- function(x, y, bs="cr", k=50, loglambda=seq(-3,20,len=100)){
  sm <- smoothCon(s(x, bs=bs, k=k), data=data.frame(x=x))
  S  <- sm[[1]]$S[[1]]
  Phi <- sm[[1]]$X
  nlambda <- length(loglambda)
  gcv_vec <- rep(NA, nlambda)
  for(l in seq_along(1:nlambda)){
    H <- Phi %*% solve(crossprod(Phi) + exp(loglambda[l])*S) %*% t(Phi)
    trH <- sum(diag(H))
    xi_hat <- solve(crossprod(Phi) + exp(loglambda[l])*S) %*% t(Phi) %*% y
    y_hat  <- Phi %*% xi_hat
    gcv_vec[l] <- length(y)*sum( (y_hat-y)^2/(length(y)-trH)^2)
  }
  exp(loglambda[which.min(gcv_vec)])
}
```

Efficient Programming: Duplicate Calculations/Linear Algebra


```
my_gcv2 <- function(x, y, bs="cr", k=50, loglambda=seq(-3,20,len=100)){
  sm <- smoothCon(s(x, bs=bs, k=k), data=data.frame(x=x))
  S  <- sm[[1]]$S[[1]]
  Phi <- sm[[1]]$X
  Phi_sq <- crossprod(Phi)
  nlambda <- length(loglambda)
  gcv_vec <- rep(NA, nlambda)
  N       <- length(y)
  lambda <- exp(loglambda)
  for(l in seq_along(1:nlambda)){
    Phisq_S_inv <- chol2inv(chol(Phi_sq + lambda[l]*S))
    trH     <- sum(diag(Phi_sq %*% Phisq_S_inv))
    xi_hat <- Phisq_S_inv  %*% (t(Phi) %*% y)
    y_hat  <- Phi %*% xi_hat
    gcv_vec[l] <- N*sum( (y_hat-y)^2/(N-trH)^2)
  }
  lambda[which.min(gcv_vec)]
}
```

Efficient Programming: Duplicate Calculations/Linear Algebra


```
N <- 1000
x <- rnorm(N)
f <- function(x) sin(2*pi*x)
y <- f(x) + rnorm(N)

my_gcv1(x=x,y=y)

## [1] 428.6351

my_gcv2(x=x,y=y)

## [1] 428.6351

# library("microbenchmark")
microbenchmark(my_gcv2(x=x,y=y), my_gcv1(x=x,y=y), times=5)

## Unit: milliseconds
##                   expr        min         lq       mean     median         uq
##  my_gcv2(x = x, y = y)   38.80185   39.09607   40.26027   40.53394   40.73024
##  my_gcv1(x = x, y = y) 2992.57093 3042.20745 3048.03738 3054.88654 3066.42740
##         max neval
##    42.13927     5
##  3084.09457     5
```

Efficient Programming: Appending


```
append_ls1 <- function(K=1000, mat){
  ret <- vector(mode="list", length=K)
  names(ret) <- 1:K
  for(k in 1:K) ret[[k]] <- mat
  dplyr::bind_rows(ret)
}
append_c1 <- function(K=1000, mat){
  ret <- c()
  for(k in 1:K) {
    ret <- rbind(ret, mat)
  }
  ret
}
append_c2 <- function(K=1000,mat){
  dims <- dim(mat)
  ret <- matrix(NA, dims[1]*K, dims[2])
  inx <- 1:dims[1]
  for(k in 1:K){
    ret[inx,] <- mat
    inx <- inx+N
  }
  ret
}
```

Efficient Programming: Appending

Homework 2
Homework 1
Final Project
FoSR

```
N   <- 100; P <- 100
mat <- matrix(rnorm(N*P), N, P)
microbenchmark(append_ls1(K=100, mat=mat),
               append_c1(K=100, mat=mat),
               append_c2(K=100, mat=mat), times=5)

## Unit: microseconds
##                            expr        min         lq       mean     median
##  append_ls1(K = 100, mat = mat)    754.919    760.525   1823.322    841.001
##   append_c1(K = 100, mat = mat) 174979.095 175054.322 219486.354 184809.439
##   append_c2(K = 100, mat = mat)   2621.680   2728.427  23463.623   3040.107
##          uq        max neval
##     867.329   5892.838     5
##  266463.611 296125.303     5
##    4484.169 104443.733     5
```

Final Projects

- In-class works-in-progress presentations next week
- Total of 9 groups
- 15 minute presentation, 5 minutes for questions/feedback (20 minutes total per group)
- 10% of the final project grade, participation
- Each group member must speak at least once

Final Projects

- No requirements on format or content
- Example format
  - Description of the data and scientific question(s)
  - Some exploratory plots
  - Ideas on potential modelling approaches and how they would address the scientific question
  - Any initial results

FoSR

$$y_i(s) = f_0(s) + f_1(s)x_i + b_i(s) + \epsilon_i(s)$$

$$y_i(s) = f_0(s) + f_1(s)x_i + \sum_{k=1}^{K} \xi_{ik}\phi_k(s) + \epsilon_i(s)$$

$$b_i(t) \sim \mathrm{GP}(0, \boldsymbol{\Sigma}_b)$$

$$\epsilon_i(s) \overset{\text{iid}}{\sim} N(0, \sigma_\epsilon^2)$$

- How to estimate ϕ?
- Iterative procedure

FoSR

- In practice we also never observe $\phi_k(s)$
- How could we estimate them from the data?
- Iterative procedure:
  1. Estimate the working model under independence
  2. Fit fpca to the residuals
  3. Extract the eigenfunctions
  4. Fit the “weak" oracle model
  5. Repeat 1-4 as necessary

FoSR: Simulating Data


```
set.seed(19840)
# simulation settings
N  <- 200  # number of functions to simulate
ns <- 100   # 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 b_i(s) as \sum_k \xi_ik \phi_k(t)
bi <- xi %*% t(Phi)
```

FoSR: Simulating Data

```
## define f(s)
f <- function(s) sin(2*pi*s)
## get f(s) for all i, s
## fS is an N x ns matrix with rows repeated
fS <- kronecker(matrix(f(sind), 1, ns), matrix(1, N, 1))
x  <- rnorm(N)
## get f(s)*x for each individual
fX <- fS * kronecker(matrix(x, N, 1), matrix(1, 1, ns))
## simulate the outcome
y <- bi + fX + matrix(rnorm(N*ns, sd=2), N, ns)
```

FoSR: Ignore correlation


```
df_fit <-
  data.frame(
    id = factor(rep(1:N,each=ns)), # important that this is a factor variable!
    y = as.vector(t(y)),           # stack the vectors of Y^t (stacks individual f
    bi = as.vector(t(bi)),         # same thing with the random intercept
    x = rep(x, each=ns),           # repeat the fixed covariate for each function
    sind = rep(sind, N),           # include the functional domain, repeat for eac
    phi1 = rep(Phi[,1], N),        # incliude the true eigenfunctions, repeat for
    phi2 = rep(Phi[,2], N),
    phi3 = rep(Phi[,3], N),
    phi4 = rep(Phi[,4], N)
  )
head(df_fit)

##   id          y        bi         x       sind       phi1     phi2      phi3
## 1  1 -0.4223359 0.8028229 -2.192767 0.00000000 0.00000000 1.414214 0.0000000
## 2  1 -2.3254423 1.0359615 -2.192767 0.01010101 0.08969497 1.411366 0.1790288
## 3  1  2.9544945 1.2651368 -2.192767 0.02020202 0.17902876 1.402836 0.3551769
## 4  1  3.2506201 1.4872519 -2.192767 0.03030303 0.26764168 1.388657 0.5256101
## 5  1  3.4911148 1.6992696 -2.192767 0.04040404 0.35517689 1.368886 0.6875860
## 6  1  3.1149040 1.8982600 -2.192767 0.05050505 0.44128193 1.343603 0.8384984
##       phi4
## 1 1.414214
## 2 1.402836
## 3 1.368886
## 4 1.312911
## 5 1.235810
```

FoSR

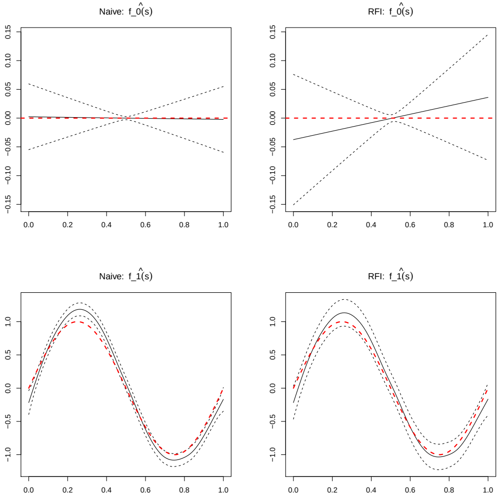

```
# library("mgcv"); library("refund")
## fit the indepedence model
fit_naive <- gam(y ~ s(sind, k=20, bs="cr") + s(sind, by=x, bs="cr", k=20),
                 data=df_fit, method="REML")
## extract residuals
resid_mat <- matrix(fit_naive$residuals, N, ns, byrow=TRUE)
## fit fpca
fpca_fit <- fpca.face(resid_mat,var=TRUE)
## get the estimated eigenfunctions
efuncs <- fpca_fit$efunctions
## add estimated eigenfunctions to the dataframe
for(k in 1:dim(efuncs)[2]){
  df_fit[[paste0("Phi",k,"_hat")]] <- efuncs[,k]
}
## fit the random functional intercept model --
## just use the first 4 estimated eigenfucntions
fit_rfi <- bam(y ~ s(sind, k=20, bs="cr") + s(sind, by=x, bs="cr", k=20) +
                 s(id,by=Phi1_hat, bs="re") + s(id,by=Phi2_hat, bs="re") +
                 s(id,by=Phi3_hat, bs="re") + s(id,by=Phi4_hat, bs="re"),
                 data=df_fit, method="fREML", discrete=TRUE)
```

FoSR


```
## get the estiamted coefficients + SEs
df_pred <- data.frame("sind"=sind, "x"=1,
                      ## we need to supply all columns of the
                      ## terms used in fitting the functional random
                      ## intercept model. Since we're only interested in the
                      ## fixed effects here, set to whatever you want
                      ## note that id must be included in the dataset used for
                      ## model fitting
                      Phi2_hat=0, Phi1_hat=0, Phi3_hat=0, Phi4_hat=0,
                      id=df_fit$id[1])
fhat_naive <- predict(fit_naive, newdata=df_pred, se.fit=TRUE,type='terms')
fhat_rfi   <- predict(fit_rfi, newdata=df_pred, se.fit=TRUE,type='terms')
head(fhat_rfi$fit)

##        s(sind)   s(sind):x s(id):Phi1_hat s(id):Phi2_hat s(id):Phi3_hat
## 1 -0.03740539 -0.21741888              0              0              0
## 2 -0.03665937 -0.12720559              0              0              0
## 3 -0.03591338 -0.03759143              0              0              0
## 4 -0.03516744  0.05082446              0              0              0
## 5 -0.03442159  0.13744295              0              0              0
## 6 -0.03367584  0.22166489              0              0              0
##   s(id):Phi4_hat
## 1              0
## 2              0
## 3              0
## 4              0
## 5              0
```

FoSR



FoSR

- *mgcv* Is best for independent random effects
- Alternative: combine *mgcv* with standard mixed model software
    - *mgcv::gamm()* function (syntax of *mgcv::gam()* and *nlme::lme()*)
    - *gamm4* package (syntax of *mgcv::gam()* and *lme4::lmer()*)
- Generally (numerically) prefer *gamm4* unless you need smooths of 2d+
- Computationally it's a bit more mixed

FoSR: Mixed Model Software


```
# library("gamm4")
system.time({
  fit_rfit_gamm4 <- gamm4(y ~
                              s(sind, bs="cr", k=20) +
                              s(sind, by=x, bs="cr", k=20),
                          random = ~(Phi1_hat + 0 | id) + (Phi2_hat + 0 | id) +
                                    (Phi3_hat + 0 | id)+ (Phi4_hat + 0 | id),
                          data=df_fit, REML=TRUE)

})

##    user  system elapsed
## 140.374   3.316 143.757

system.time({
fit_rfit_gamm <- gamm(y ~ s(sind, bs="cr", k=20) + s(sind, by=x, bs="cr", k=20),
                      random = list(id=pdDiag(~Phi1_hat + Phi2_hat +
                                              Phi3_hat + Phi4_hat + 0)),
                      data=df_fit, REML=TRUE)
})

##    user  system elapsed
##   9.545   0.162   9.714
```

FoSR: Mixed Model Software


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

## List of 2
##  $ lme:List of 18
##   ..- attr(*, "class")= chr "lme"
##  $ gam:List of 31
##   ..- attr(*, "class")= chr "gam"
##  - attr(*, "class")= chr [1:2] "gamm" "list"

str(fit_rfit_gamm4, max.level=1)

## List of 2
##  $ mer:Formal class 'lmerMod' [package "lme4"] with 13 slots
##  $ gam:List of 32
##   ..- attr(*, "class")= chr "gam"
```

FoSR: Mixed Model Software: *mgcv::gamm()*

```
summary(fit_rfit_gamm$gam)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(sind, bs = "cr", k = 20) + s(sind, by = x, bs = "cr", k = 20)
##
## Parametric coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.004282   0.014322  -0.299    0.765
##
## Approximate significance of smooth terms:
##              edf Ref.df     F p-value
## s(sind)     1.00   1.00  0.42   0.517
## s(sind):x 10.57  10.57 15.44  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.109
##   Scale est. = 4.067     n = 20000
```

FoSR: Mixed Model Software: *mgcv::gamm()*


```
summary(fit_rfit_gamm$lme)

## Linear mixed-effects model fit by maximum likelihood
##  Data: strip.offset(mf)
##        AIC      BIC    logLik
##   86635.22 86722.16 -43306.61
##
## Random effects:
##  Formula: ~Xr - 1 | g
##  Structure: pdIdnot
##                  Xr1          Xr2          Xr3          Xr4          Xr5
## StdDev: 1.528755e-05 1.528755e-05 1.528755e-05 1.528755e-05 1.528755e-05
##                  Xr6          Xr7          Xr8          Xr9         Xr10
## StdDev: 1.528755e-05 1.528755e-05 1.528755e-05 1.528755e-05 1.528755e-05
##                 Xr11         Xr12         Xr13         Xr14         Xr15
## StdDev: 1.528755e-05 1.528755e-05 1.528755e-05 1.528755e-05 1.528755e-05
##                 Xr16         Xr17         Xr18
## StdDev: 1.528755e-05 1.528755e-05 1.528755e-05
##
##  Formula: ~Xr.0 - 1 | g.0 %in% g
##  Structure: pdIdnot
##               Xr.01      Xr.02      Xr.03      Xr.04      Xr.05      Xr.06
## StdDev: 0.03025081 0.03025081 0.03025081 0.03025081 0.03025081 0.03025081
##               Xr.07      Xr.08      Xr.09     Xr.010     Xr.011     Xr.012
## StdDev: 0.03025081 0.03025081 0.03025081 0.03025081 0.03025081 0.03025081
##              Xr.013     Xr.014     Xr.015     Xr.016     Xr.017     Xr.018
## StdDev: 0.03025081 0.03025081 0.03025081 0.03025081 0.03025081 0.03025081
##
##  Formula: ~Phi1_hat + Phi2_hat + Phi3_hat + Phi4_hat + 0 | id %in% g.0 %in% g
##  Structure: Diagonal
##         Phi1_hat Phi2_hat Phi3_hat Phi4_hat Residual
## StdDev: 9.172615 7.073778 4.558037 3.539259  2.01669
##
## Fixed effects: y.0 ~ X - 1
##                      Value  Std.Error    DF   t-value p-value
## X(Intercept)    -0.0042819 0.01432291 19797 -0.298956  0.7650
```

FoSR: Mixed Model Software: *gamm4::gamm4()*


```
summary(fit_rfit_gamm4$gam)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(sind, bs = "cr", k = 20) + s(sind, by = x, bs = "cr", k = 20)
##
## Parametric coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.004277   0.014324  -0.299    0.765
##
## Approximate significance of smooth terms:
##             edf Ref.df      F p-value
## s(sind)    1.00   1.00  0.416   0.519
## s(sind):x 10.59  10.58 15.653  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.109
## lmer.REML =  86627  Scale est. = 4.0686    n = 20000
```

FoSR: Mixed Model Software: *gamm4::gamm4()*

```
summary(fit_rfit_gamm4$mer)

## Linear mixed model fit by REML ['lmerMod']
##
## REML criterion at convergence: 86626.6
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.9452 -0.6590  0.0015  0.6622  3.6997
##
## Random effects:
##  Groups   Name      Variance  Std.Dev.
##  id       Phi4_hat  1.266e+01 3.55844
##  id.1     Phi3_hat  2.063e+01 4.54149
##  id.2     Phi2_hat  4.984e+01 7.05946
##  id.3     Phi1_hat  8.280e+01 9.09925
##  Xr.0     s(sind):x 9.217e-04 0.03036
##  Xr       s(sind)   0.000e+00 0.00000
##  Residual           4.069e+00 2.01707
## Number of obs: 20000, groups:  id, 200; Xr.0, 18; Xr, 18
##
## Fixed effects:
##                  Estimate Std. Error t value
## X(Intercept)   -0.004277   0.014324  -0.299
## Xs(sind)Fx1     0.098904   0.153390   0.645
## Xs(sind):xFx1 -1.247590   0.125585  -9.934
## Xs(sind):xFx2  2.026394   0.193301  10.483
##
## Correlation of Fixed Effects:
##              X(Int) Xs()F1 X():F1
## Xs(sind)Fx1 -0.034
## Xs(snd):xF1 -0.016 -0.023
## Xs(snd):xF2 -0.007  0.026 -0.795
```

FoSR: NHANES Physical Activity vs Age

- Consider the model

$$\text{lAC}_i(s) = f_0(s) + f_1(s)\text{Age}_i + b_i(s) + \epsilon_i(s)$$

- Problem: data size is huge

FoSR: NHANES Physical Activity vs Age


```r
## library("tidyverse")
df <- read_rds(here::here("data","data_processed","NHANES_AC_processed.rds"))
## extract the PA data
lX <- log(1+as.matrix(df[,paste0("MIN",1:1440)]))
lX[is.na(lX)] <- 0
N  <- nrow(lX)
## bin the data into 60 minute intervals
tlen <- 60
nt   <- ceiling(1440/tlen)
inx_cols <- split(1:1440, rep(1:nt, each=tlen)[1:1440])
lX_bin <- vapply(inx_cols, function(x) rowMeans(lX[,x], na.rm=TRUE), numeric(N))
## get subject average curves
inx_rows <- split(1:N, factor(df$SEQN, levels=unique(df$SEQN)))
lX_bin_ind <- t(vapply(inx_rows, function(x) colMeans(lX_bin[x,], na.rm=TRUE), num
nid <- nrow(lX_bin_ind)
# get a data frame for model fitting
sind <- seq(0,1,len=nt)
df_fit <-
  data.frame(lAC=as.vector(t(lX_bin_ind)),
             sind = rep(sind, nid),
             SEQN = rep(unique(df$SEQN), each=nt)) %>%
  left_join(dplyr::select(df, SEQN, Age), by="SEQN") %>%
  mutate(id = factor(SEQN)) %>%
  filter(Age <= 30)
```

FoSR: NHANES Physical Activity vs Age


```r
## subset the data to just 500 participants
set.seed(10110)
nid_samp <- 500
id_samp  <- sample(unique(df_fit$id), size=nid_samp, replace=FALSE)
df_fit_sub <- subset(df_fit, id %in% id_samp)

## fit the naive model
fit_naive <- bam(lAC ~ s(sind, bs="cc",k=20) + s(sind, by=Age, bs="cc",k=20),
                 method="fREML",data=df_fit_sub, discrete=TRUE)
## extract the resiudals
resid_mat <- matrix(fit_naive$residuals,
                    nid_samp, nt,byrow=TRUE)
## fit fpca
fpca_fit  <- fpca.face(resid_mat, knots=15)
## add in eigenfunctiosn
for(k in 1:length(fpca_fit$evalues)){
    df_fit_sub[[paste0("Phi",k)]] <- rep(fpca_fit$efunctions[,k],nid_samp)
}
```

FoSR: NHANES Physical Activity vs Age


```
## fit the fri model using mgcv::bam()
system.time({
    fit_fri <- bam(lAC ~
                         s(sind, bs="cc",k=20) +
                         s(sind, by=Age, bs="cc",k=20) +
                         s(id, by=Phi1, bs="re") + s(id, by=Phi2, bs="re") +
                         s(id, by=Phi3, bs="re") + s(id, by=Phi4, bs="re"),
                       method="fREML",data=df_fit_sub, discrete=TRUE)
})

##    user  system elapsed
##  44.662   0.690  45.386
```

FoSR: NHANES Physical Activity vs Age

- Plot the estimated $\hat{f}_0(s)$ and $\hat{f}_1(s)$ from the naive and FRI fits
- Compare the shapes and CIs
- Do these results make sense?