BIOS 7720: Applied Functional Data Analysis

Lecture 3: Generalized Additive Models (GAMs) Part 1

Andrew Leroux

March 9, 2021

Logistics

- Homework 1 due date now 3/19
- Other homework 1 things
- Lecture recordings
- Best form of communication?
- Office hours start this week
  - One off this week: 5/12 at 10:00AM
  - Hereafter: Fridays at 9:00AM

Roadmap

1. Introduce GAMs and associated concepts
2. In-class exercises

Introduction

- General form

$$y_i \sim \text{EF}(\mu_i, \sigma)$$
$$g(E[y_i|\boldsymbol{x}_i]) = f(\boldsymbol{x}_i)$$

- An example with $g(x) = x$ and $\boldsymbol{x}_i = [x_{i1}, x_{i2}, x_{i3}, x_{i4}]^t$

$$\begin{aligned} E[y_i|\boldsymbol{x}_i] &= f(x_{i1}, x_{i2}, x_{i3}, x_{i4}) \\ &= f_1(x_{i1}) + f_2(x_{i2}) + f_3(x_{i3}, x_{i4}) \end{aligned}$$

Identifiability

- Up until now we've implicitly assumed the $c1_{N\times 1} \in \text{span}(\mathbf{\Phi})$ where $c \in \mathbb{R}^1$
- Suppose we want to fit the model

$$\begin{aligned} y_i &= \alpha_0 + f(x_i) \\ &= \alpha_0 + \mathbf{\Phi}\boldsymbol{\xi} \end{aligned}$$

- This model is not identifiable since

$$y_i = (\alpha_0 + b) + (\mathbf{\Phi}\boldsymbol{\xi} - b)$$

- Need to impose some constraint

Identifiability in *mgcv*

- Consider the constraint $\sum_{i=1}^{N} f(x_i) = 0$

$$\sum_{i=1}^{N} f(x_i) = 0$$

$$\mathbf{1}^t \mathbf{\Phi} \boldsymbol{\xi} = 0$$

- For this to be true for all $\boldsymbol{\xi}$, we require $\mathbf{1}^t \mathbf{\Phi} = 0$
- Easy solution, center $\mathbf{\Phi}$ columnwise

Identifiability in *mgcv*

- Column centered basis matrix: $\tilde{\mathbf{\Phi}} = (\boldsymbol{I}_{N\times N} - N^{-1}\mathbf{1}\mathbf{1}^t)\mathbf{\Phi}$
- Problem: $\tilde{\mathbf{\Phi}}$ is not full rank (homework problem)
- Solution:
  - Delete a column from $\tilde{\mathbf{\Phi}}$
  - Delete the corresponding row/column from $\boldsymbol{S}$

Identifiability in *mgcv*

$$y_i = \log(1 + x_i^2) + \epsilon_i$$
$$x_i \sim N(0, \sigma_x^2)$$
$$\epsilon_i \sim N(0, \sigma_\epsilon^2)$$

```
library("mgcv")
set.seed(-909)
N <- 1000
f <- function(x) log(1+x^2)
x <- rnorm(N, sd=3)
y <- f(x) + rnorm(N, mean=0, sd=0.5)
```

Identifiability in *mgcv*

What if we don't impose the identifiability constraint?

```
## number of basis functions to use in fitting
K  <- 10
## sequence of x values to predict on (for plotting)
xind <- seq(min(x), max(x), len=1000)
## set up the unconstrained smooth, apply identifiability constraints
sm <- smoothCon(s(x, bs="cr", k=K), data=data.frame(x=x))[[1]]
Phi<- sm$X
## fit using lm()
fit_1 <- lm(y~Phi)
## get the design matrix for our initial fit
lp_1 <- PredictMat(sm, data=data.frame(x=xind))
## get the estimated function at our x-values for predicting
fhat_1 <- lp_1 %*% coef(fit_1)

## Error in lp_1 %*% coef(fit_1):  non-conformable arguments
```

Identifiability in *mgcv*

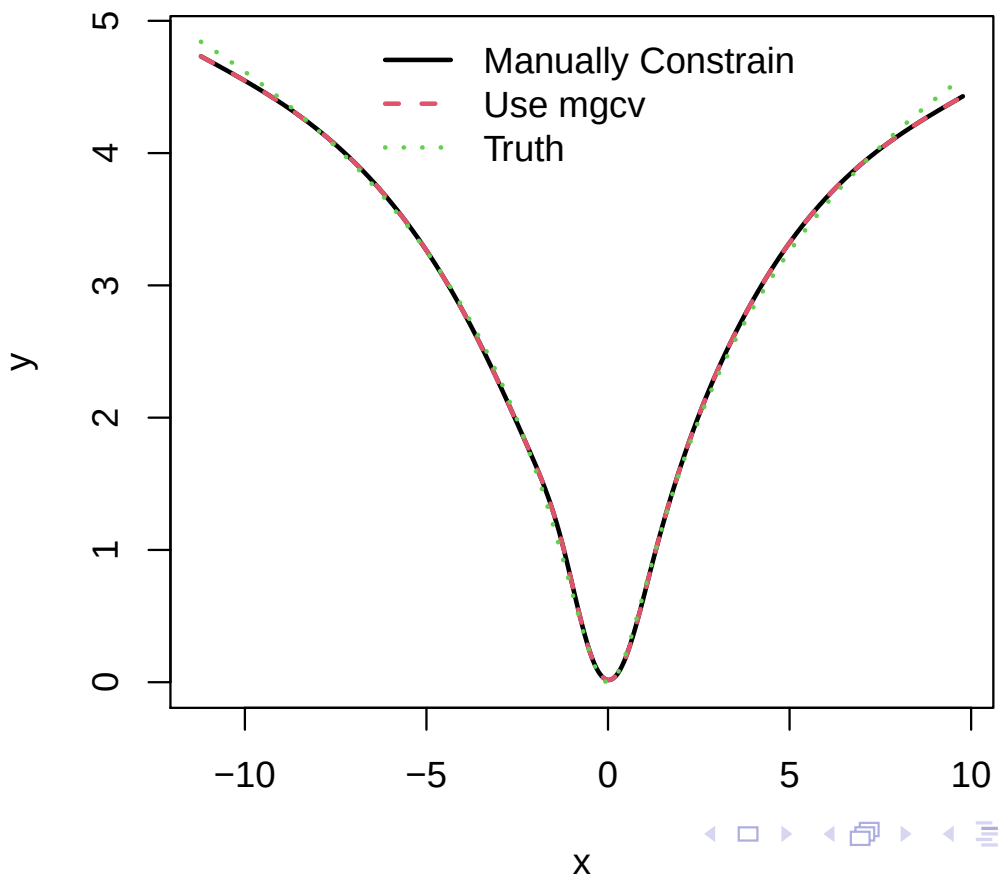

```
## number of basis functions to use in fitting
K  <- 10
## sequence of x values to predict on (for plotting)
xind <- seq(min(x), max(x), len=1000)
## set up the unconstrained smooth, apply identifiability constraints
sm <- smoothCon(s(x, bs="cr", k=K), data=data.frame(x=x))[[1]]
Phi<- sm$X
Phi_mn <- colMeans(Phi)
Phi_tilde <- sweep(Phi, 2, Phi_mn)
Phi_tilde <- Phi_tilde[,-K]
## fit using lm() -- no need to suppress the intercept
fit_1 <- lm(y~Phi_tilde)
## get the design matrix for our initial fit
lp_1 <- PredictMat(sm, data=data.frame(x=xind))
lp_1 <- cbind(1, sweep(lp_1, 2, Phi_mn)[,-K])
## get the estimated function at our x-values for predicting
fhat_1 <- lp_1 %*% coef(fit_1)
```

Identifiability in *mgcv*


```
## do the fit with the identifiability constraint absorbed
sm_cons <- smoothCon(s(x, bs="cr", k=K), data=data.frame(x=x),
                     absorb.cons = TRUE)[[1]]
fit_2 <- lm(y~sm_cons$X)
lp_2  <- cbind(1,PredictMat(sm_cons, data=data.frame(x=xind)))
fhat_2 <- lp_2 %*% coef(fit_2)
```

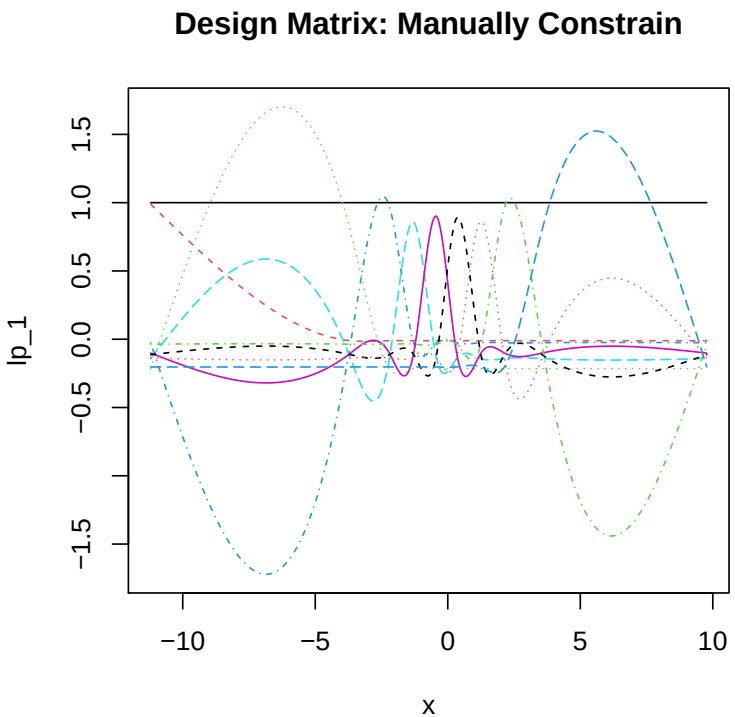
Identifiability in *mgcv*

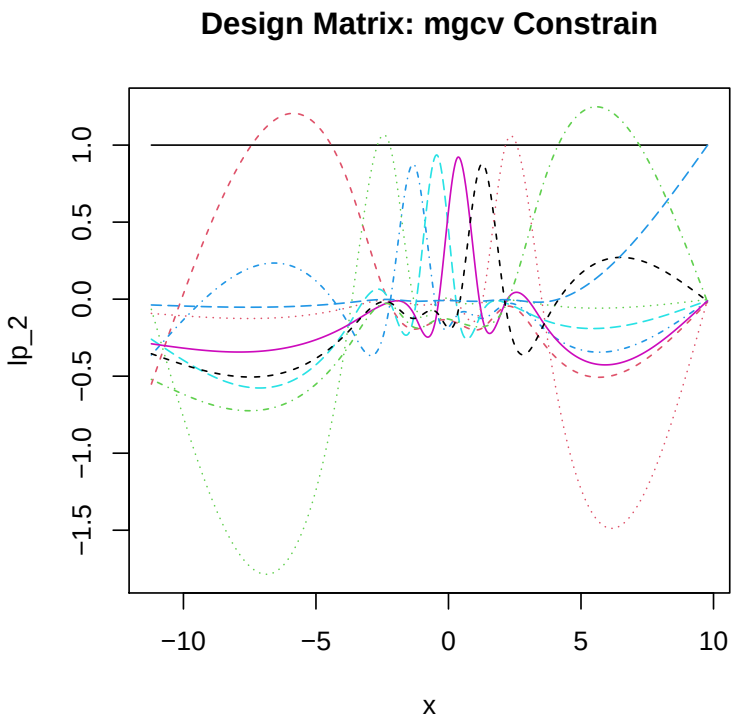
```
matplot(xind, cbind(fhat_1, fhat_2, f(xind)),type='l',xlab="x",ylab="y",lwd=2)
legend("top",c("Manually Constrain", "Use mgcv","Truth"), col=1:3,lty=1:3,bty='n',
```

Identifiability in *mgcv*


```
layout(matrix(c(1,2),1,2))
matplot(xind,lp_1,type='l',main="Design Matrix: Manually Constrain",xlab="x")
matplot(xind,lp_2,type='l',main="Design Matrix: mgcv Constrain",xlab="x")
```





Effective Degrees of Freedom

- Recall from linear regression $\hat{\sigma}^2 = ||\boldsymbol{y} - \hat{\boldsymbol{y}}||^2/(N-p)$
- With penalized splines, we want large p (flexibility)
- But what if the actual $\hat{f}(x)$ doesn't use all p df?
- as $\lambda \to \infty$, $\hat{f}(x) = \beta x$, so 1 df "used"

Effective Degrees of Freedom

- Recall from linear regression $\hat{\sigma}^2 = ||\boldsymbol{y} - \hat{\boldsymbol{y}}||^2/(N-p)$
- With penalized splines, we want large p (flexibility)
- But what if the actual $\hat{f}(x)$ doesn't use all p df?
- as $\lambda \to \infty$, $\hat{f}(x) = \beta x$, so 1 df "used"

Effective Degrees of Freedom

- Use $\boldsymbol{X}$ as general notation for our design matrix (e.g. $\boldsymbol{\Phi}$)
- Recall from linear regression $\hat{\boldsymbol{y}} = \boldsymbol{H}\boldsymbol{y}$
- $\boldsymbol{H} = \boldsymbol{X}(\boldsymbol{X}^t\boldsymbol{X})^{-1}\boldsymbol{X}^t$ is the matrix that projects $\boldsymbol{y}$ onto the column space of $\boldsymbol{X}$
- $\text{trace}(\boldsymbol{H}) = P$ (assuming $\boldsymbol{X}_{N\times P}$ is full column rank)
- Define for a fixed λ: $\text{EDF} = \text{trace}(\boldsymbol{X}(\boldsymbol{X}^t\boldsymbol{X} + \lambda\boldsymbol{S})^{-1}\boldsymbol{X}^t)$

Smoothing Parameter Selection

- Ordinary cross-validation
- Generalized cross-validation
- (Marginal) likelihood approaches

Smoothing Parameter Selection: OCV vs GCV

- Ordinary cross-validation is
  - Not defined for general exponential family outcomes
  - Not rotationally invariant
- Solution: generalized cross-validation
- For Gaussian data with identity link

$$\mathcal{V}_0 = N^{-1} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 / (1 - H_{ii})^2$$

$$\mathcal{V}_g = N \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 / (N - \text{trace}(H))^2$$

Smoothing Parameter Selection: OCV vs GCV


```
## simulate data y_i = 3*x_i + \epsilon_i
set.seed(-304); N <- 100; x <- rnorm(N); y <- 3*x + rnorm(N)
## rotate the response and design matrix using an orthgonal matrix
Q <- qr.Q(qr(cbind(1,x)),complete=TRUE)
Qy <- Q %*% y
Qx <- Q %*% cbind(1,x)
## fit the two models
fit_r <- lm(Qy ~ Qx - 1)
fit   <- lm(y ~ x)
cbind(coef(fit_r), coef(fit))

##           [,1]      [,2]
## Qx  0.1678198 0.1678198
## Qxx 2.9509341 2.9509341

## GCV and OCV for the two fits
OCV_r <- mean(residuals(fit_r)^2/(1-influence(fit_r)$hat))
OCV   <- mean(residuals(fit)^2/(1-influence(fit)$hat))
GCV_r <- N*sum(residuals(fit_r)^2)/(N-sum(influence(fit_r)$hat))^2
GCV   <- N*sum(residuals(fit)^2)/(N-sum(influence(fit)$hat))^2
data.frame("Model" = c("Original","Rotated"), GCV=c(GCV,GCV_r), OCV=c(OCV,OCV_r))

##      Model       GCV       OCV
## 1 Original 0.9206058 0.9023693
## 2  Rotated 0.9206058 0.8951349
```

Smoothing Parameter Selection: GCV

- Rotation by an orthogonal matrix $\boldsymbol{Q}$
  - Yields identical inference on regression coefficients
  - Different OCV criteria
  - Identical GCV criteria
- GCV exists for generalized outcomes using Deviance
- In practice GCV tends to undersmooth at realistic sample sizes
- Can introduce a tuning parameter $\gamma > 1$ to increase smoothness

$$\mathcal{V}_g = N \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 / (N - \gamma \text{trace}(H))^2$$

- Not clear how to choose γ in a principled way

Smoothing Parameter Selection: likelihood approaches

- As an alternative to GCV we can use likelihood based methods
- λ can be estimated as a variance parameter
  - Allows us to frame penalized splines as mixed models
  - More details next class
- Both ML and REML approaches for exponential family responses
- REML is considered the “default" by many for applied work

Simulate Gaussian and Binary Data

- Simulate data according to the model

$$\begin{aligned} g(E[y_i|\boldsymbol{x}_i]) &= \eta_i \\ &= 1 + f_1(x_{i1}) + f_2(x_{i2}) + f_3(x_{i3}) + f_4(x_{i4}) \\ &= 1 + \sin(\pi x_{i1}) + 2x_{i2} + (x_{i3} + x_{i3}^2) + \cos(\pi x_{i4})x_{i4} \end{aligned}$$

- Where y_i is either Gaussian or Binary (logit link)
- Goals
  - Estimate these models using GCV and REML
  - Quickly plot the results
  - Get point/variance estimates directly

Simulate Gaussian and Binary Data


```
library("mgcv"); set.seed(80487); N <- 200
P <- 4
X <- matrix(rnorm(N*P), N, P)
## set up the association structures
f1 <- function(x) sin(pi*x)
f2 <- function(x) 2*x
f3 <- function(x) 0.25*x^3
f4 <- function(x) cos(pi*x)*x
## get the linear predictor
eta  <- 1 + f1(X[,1]) + f2(X[,2]) + f3(X[,3]) + f4(X[,4])
## simulate gaussian outcomes
y_g <- eta + rnorm(N, sd=1)
## simulate binary outcomes
pr_y <- 1/(1+exp(-eta))
y_b  <- vapply(pr_y, function(x) sample(c(0,1), size=1, prob=c(1-x,x)),
               numeric(1))
```

Fit the Models using *mgcv::gam()*


```
## combine data into a dataframe
df_fit <- data.frame(y_g=y_g, y_b=y_b, X)
## set up basis type for all smooth terms
bs <- "cr"
## number of basis functions for all smooth terms
K  <- 20
## fit the models on the Gaussian data
fit_g_GCV  <- gam(y_g ~ s(X1, bs=bs, k=K) + s(X2, bs=bs, k=K) +
                        s(X3, bs=bs, k=K) + s(X4, bs=bs, k=K),
                  family=gaussian(), method="GCV.Cp", data=df_fit)
fit_g_REML <- gam(y_g ~ s(X1, bs=bs, k=K) + s(X2, bs=bs, k=K) +
                        s(X3, bs=bs, k=K) + s(X4, bs=bs, k=K),
                  family=gaussian(), method="REML", data=df_fit)
## fit the models on the binary data
fit_b_GCV  <- gam(y_b ~ s(X1, bs=bs, k=K) + s(X2, bs=bs, k=K) +
                        s(X3, bs=bs, k=K) + s(X4, bs=bs, k=K),
                  family=binomial(), method="GCV.Cp", data=df_fit)
fit_b_REML <- gam(y_b ~ s(X1, bs=bs, k=K) + s(X2, bs=bs, k=K) +
                        s(X3, bs=bs, k=K) + s(X4, bs=bs, k=K),
                  family=binomial(), method="REML", data=df_fit)
```

Plotting the results *mgcv::gam()*

- *mgcv* has default plotting methods
- just call `plot(fit)` on your fitted model. For example:

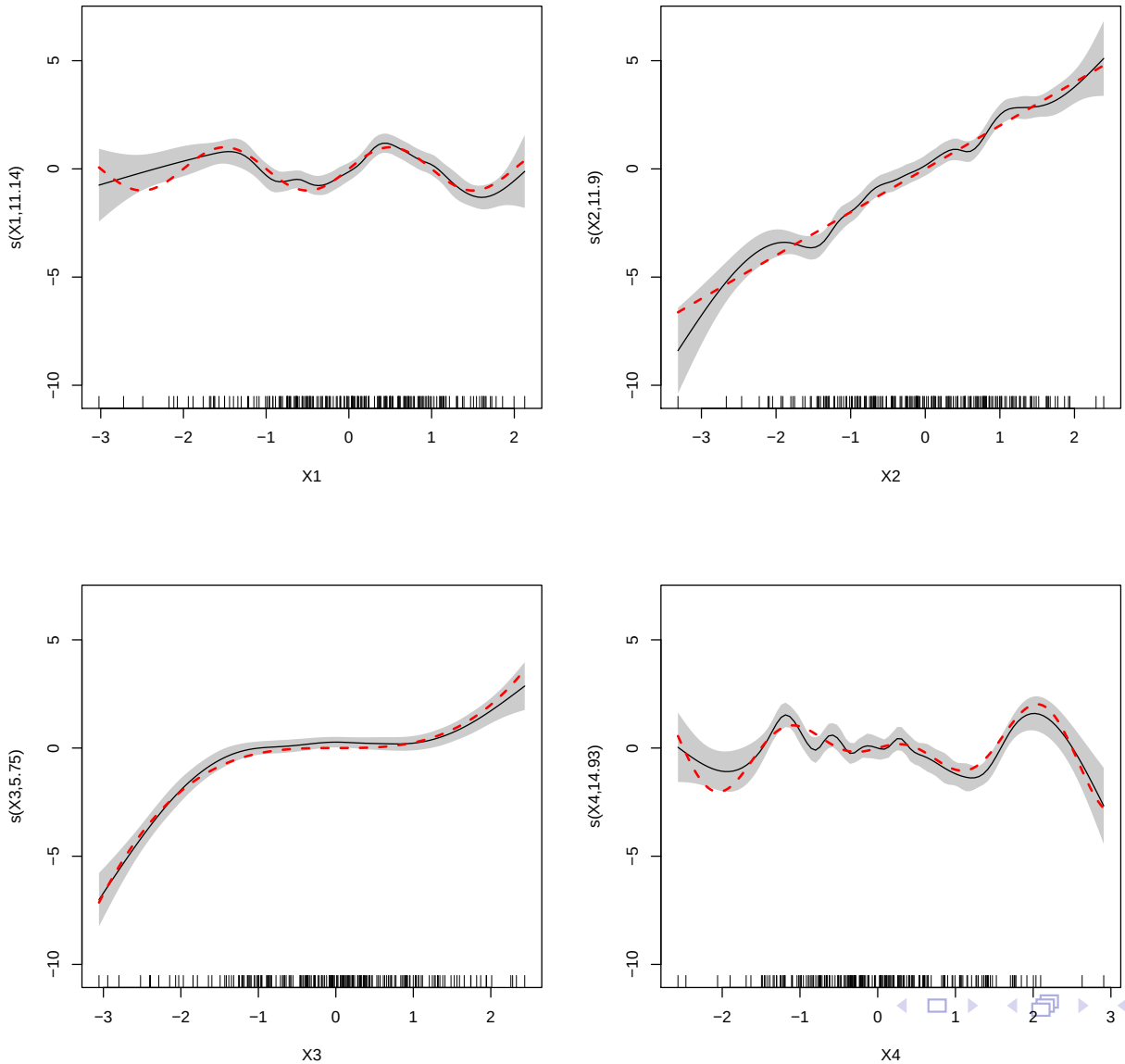
```
par(mfrow=c(2,2))
plot(fit_g_GCV, shade=TRUE)
```

- See ?plot.gam for additional options
- Can gain finer control by looping over coefficients

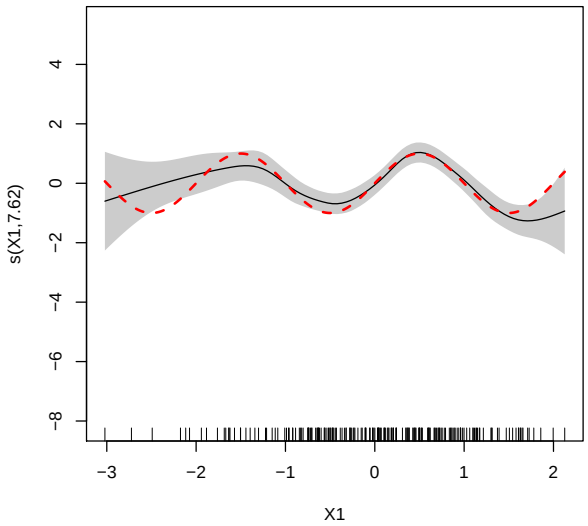
Plotting the results *mgcv::gam()*: Gaussian, GCV

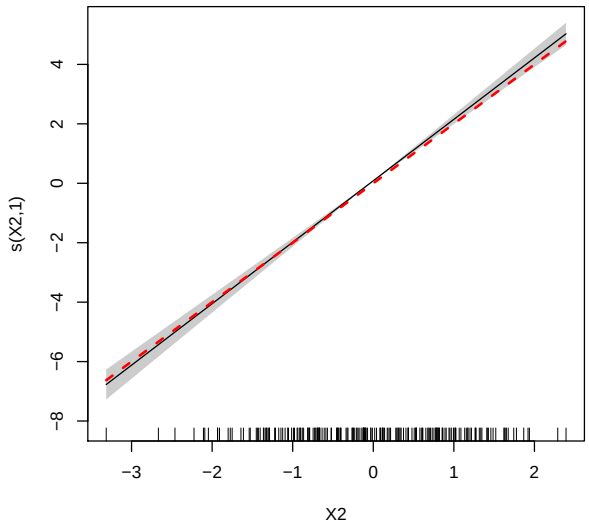

```
par(mfrow=c(2,2))
nx_pred <- 1000
xind_pred <-
  lapply(1:P, function(x){
    rn_x <- range(X[,x])
    seq(rn_x[1], rn_x[2], len=nx_pred)
  })
fn_ls <- list(f1,f2,f3,f4)
for(p in 1:P){
 plot(fit_g_GCV, select=p, shade=TRUE)
 lines(xind_pred[[p]], fn_ls[[p]](xind_pred[[p]]),col='red',lwd=2,lty=2)
}
```

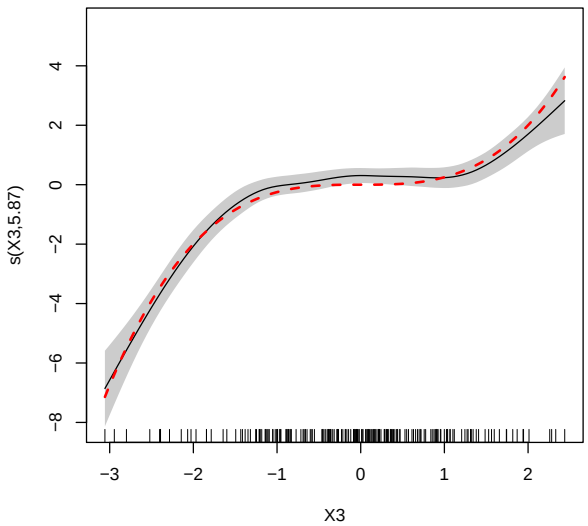
Plotting the results: Gaussian, GCV

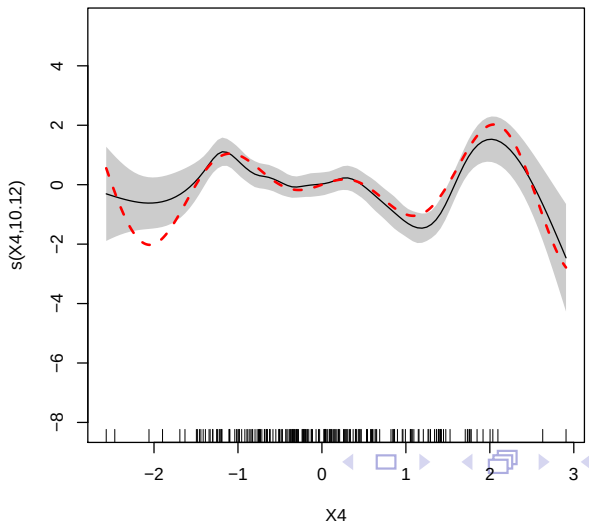


Plotting the results: Gaussian, REML

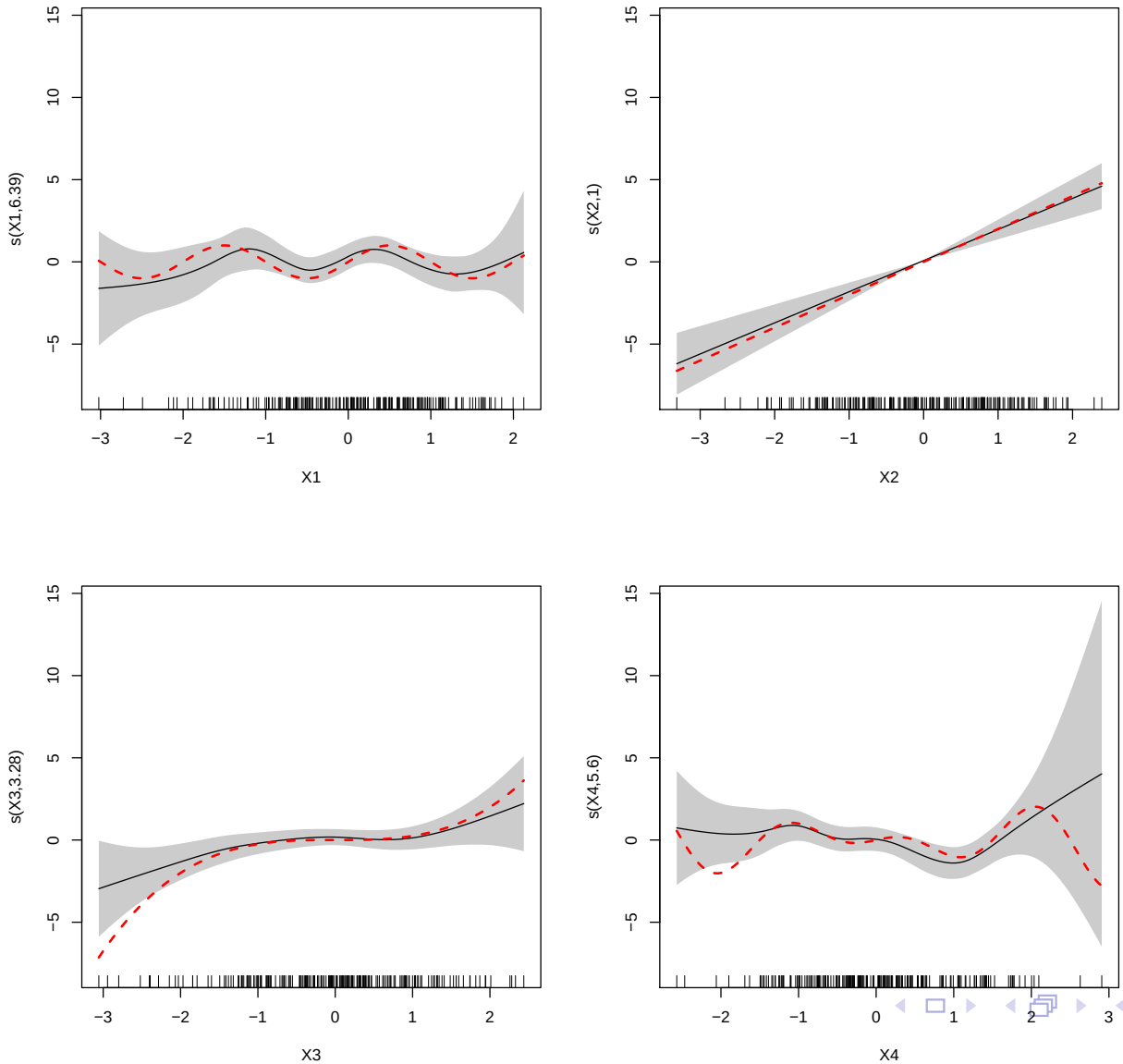




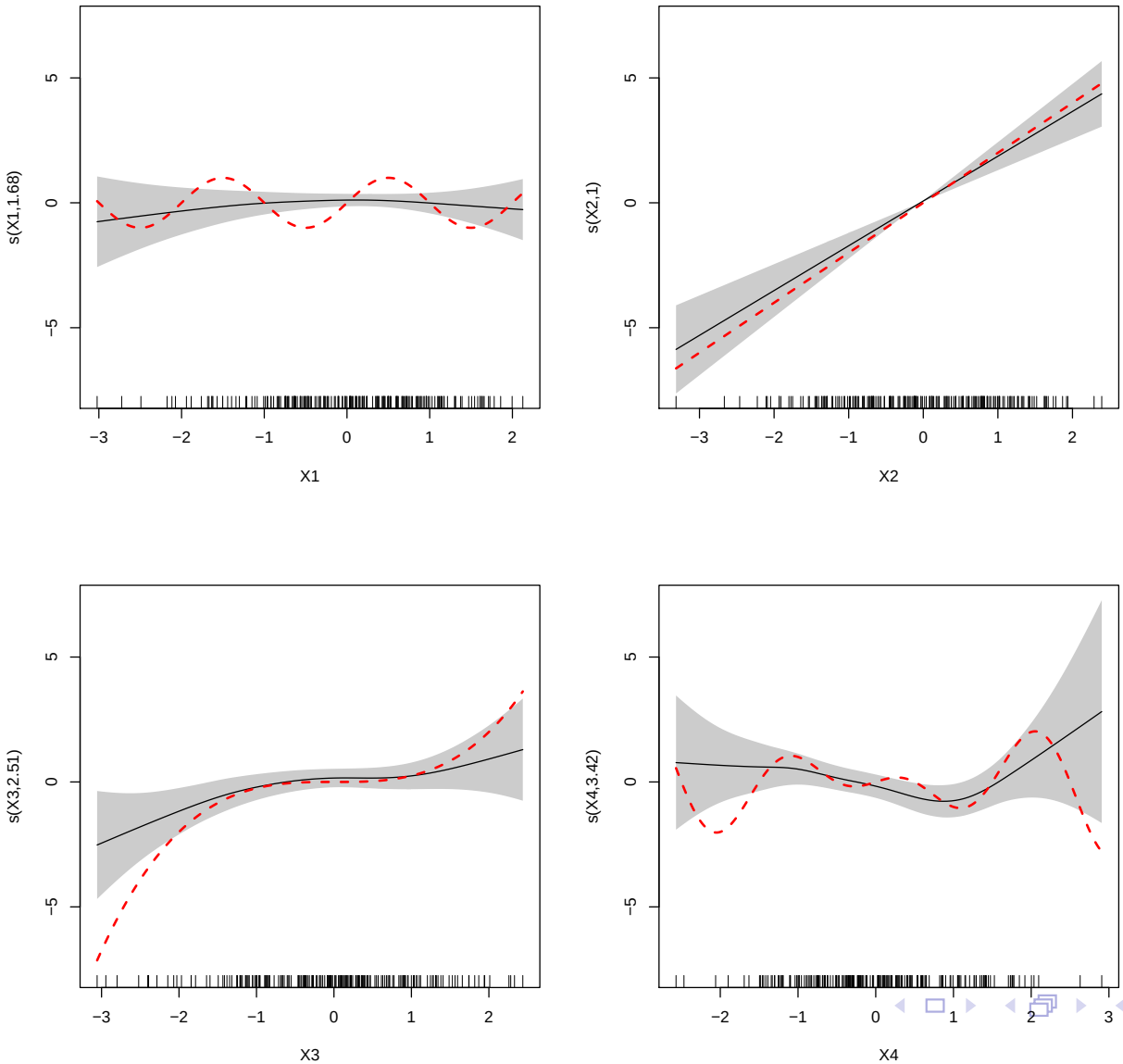




Plotting the results: Binary, GCV



Plotting the results: Binary, REML



Extracting These Quantities

- That's all helpful for a quick loop, but how can we get estimates manually?
- Use *predict.gam()*
- You can obtain predictions on
  - Response scale (type="response")
  - Linear predictor (type="link")
  - Component of the linear predictor (type="terms")
  - "design matrix" (type="lpmatrix")
- Set the argument `se.fit=TRUE` to get component-wise standard errors

Extracting These Quantities


```
## set up a new data frame with all "X" predictors at a new range of values
xind_pred <- seq(-3,3,len=1000)
df_pred <- data.frame(X1=xind_pred, X2=xind_pred, X3=xind_pred, X4=xind_pred)
head(df_pred)

##           X1        X2        X3        X4
## 1 -3.000000 -3.000000 -3.000000 -3.000000
## 2 -2.993994 -2.993994 -2.993994 -2.993994
## 3 -2.987988 -2.987988 -2.987988 -2.987988
## 4 -2.981982 -2.981982 -2.981982 -2.981982
## 5 -2.975976 -2.975976 -2.975976 -2.975976
## 6 -2.969970 -2.969970 -2.969970 -2.969970

## get the predicted values at these values for each "type"
yhat_g_REML   <- predict(fit_g_REML, newdata=df_pred, type="response", se.fit=TRUE
etahat_g_REML <- predict(fit_g_REML, newdata=df_pred, type="link", se.fit=TRUE)
smhat_g_REML  <- predict(fit_g_REML, newdata=df_pred, type="terms", se.fit=TRUE)
Phi_g_REML    <- predict(fit_g_REML, newdata=df_pred, type="lpmatrix", se.fit=TRUE
```

Extracting These Quantities: $\hat{y}_i, \hat{\eta}_i$


```
str(yhat_g_REML)

## List of 2
##  $ fit   : num [1:1000(1d)] -12.3 -12.2 -12.2 -12.1 -12.1 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ : chr [1:1000] "1" "2" "3" "4" ...
##  $ se.fit: num [1:1000(1d)] 1.71 1.7 1.69 1.68 1.67 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ : chr [1:1000] "1" "2" "3" "4" ...

str(etahat_g_REML)

## List of 2
##  $ fit   : num [1:1000(1d)] -12.3 -12.2 -12.2 -12.1 -12.1 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ : chr [1:1000] "1" "2" "3" "4" ...
##  $ se.fit: num [1:1000(1d)] 1.71 1.7 1.69 1.68 1.67 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ : chr [1:1000] "1" "2" "3" "4" ...
```

Are these quantities useful?

Extracting These Quantities: $\hat{f}$


```
str(smhat_g_REML)

## List of 2
##  $ fit   : num [1:1000, 1:4] -0.582 -0.576 -0.571 -0.565 -0.559 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:1000] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:4] "s(X1)" "s(X2)" "s(X3)" "s(X4)"
##  $ se.fit: num [1:1000, 1:4] 0.809 0.804 0.798 0.792 0.786 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:1000] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:4] "s(X1)" "s(X2)" "s(X3)" "s(X4)"
##  - attr(*, "constant")= Named num 0.936
##   ..- attr(*, "names")= chr "(Intercept)"

# matplot(smhat_g_REML,type='l',xlab="x",ylab=expression(hat(f)(x)))
```

- Returns a list with two elements
- The estimated smooths $\hat{f}_p$ and corresponding standard errors evaluated at each point in `df_pred`

Extracting These Quantities: Φ


```
str(Phi_g_REML)

##  num [1:1000, 1:77] 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:1000] "1" "2" "3" "4" ...
##   ..$ : chr [1:77] "(Intercept)" "s(X1).1" "s(X1).2" "s(X1).3" ...
##  - attr(*, "model.offset")= num 0
```

- Let $\boldsymbol{X}_{\text{new}} = [\boldsymbol{x}_{1,\text{new}}, \ldots, \boldsymbol{x}_{4,\text{new}}]$ be the matrix of "new" predictor values we want to evaluate at
- Then $\boldsymbol{\Phi}_{\text{new}} = [\boldsymbol{\Phi}_1(\boldsymbol{x}_{1,\text{new}}), \ldots, \boldsymbol{\Phi}_4(\boldsymbol{x}_{4,\text{new}})]$ is the associated design matrix
- $\hat{\boldsymbol{y}}_{\text{new}} = \boldsymbol{\Phi}_{\text{new}}\boldsymbol{\theta}$
- Where $\theta = [\beta_0, \hat{\boldsymbol{\xi}}_1, \ldots, \hat{\boldsymbol{\xi}}_4]^t$
- Why doesn't *mgcv* return standard errors?

Exercises

1. Simulate and additive model where the outcome is Poisson
2. Plot the estimated coefficients ($\hat{f}(x)$) with 95% CIs ($\hat{f}(x) \pm 2\text{SE}(\hat{f}(x))$)
3. Obtain estimates of $\hat{f}(\boldsymbol{x}_{\text{new}})$ for a new set of x values by:
   a. directly using *predict.gam()* with type="terms"
   b. post multiplying $\boldsymbol{\Phi}_{\text{new}}$ with $\boldsymbol{\theta}$ using the columns/entries associated with each smooth term

Next Class

- (Marginal) likelihood approach to smoothing parameter selection
- Smooths of more than one variable
- Linear functionals
- Interpreting summary output from *mgcv::gam()*