BIOS 7720
Solutions to In-Class Exercises, Lecture 4
Andrew Leroux

First, load all of the packages we'll use for the in-class problems

```
library("fields");library("gridExtra");library("tidyverse");library("ggplot2");library("mgcv")
```

1. Plot a heatmaps of $\hat{f}_1(x_1, x_2) - \hat{f}_1(x_1, x_2)$ and $\hat{f}_2(x_1, x_2) - \hat{f}_2(x_1, x_2)$. Comment on any differences you see.

   Figure 1 plots the difference bteween the estimated coefficients adn the truth. The left panel plots the difference $f_1 - \hat{f}_1$ while the right panel plots the difference $f_2 - \hat{f}_2$. Blue regions correspond to areas where $\hat{f}$ underestimates the truth, while red regions are areas where $\hat{f}$ overestiamtes the truth.

   First consider $f_1 - \hat{f}_1$. The difference is a bit noisy, but if you look closely you'll see that the estimated function tends to overestimate the truth where the truth is more strongly negative (large negative peaks) and underestimate where the truth is more positive (large positive peaks). Essentially we're able to capture the overall shape well, but tend to miss the magnitude of the peaks slightly (except at the boundaries). Overall, across the surface, the average difference is very close to 0.

   Moving the $f_2 - \hat{f}_2$, we see that the estimated difference surface is very uniform in color, and right around 0.6. This constant shift can be explained by the identifiability constraints we impose to identify the surface. For the univariate case we require $\sum_{i=1}^{N} \hat{f}(x_i) = 0$. For bivariate surfaces we require $\sum_{i=1}^{N} \hat{f}(x_1, x_2) = 0$. However, this condition does not hold true over the simulated data. So, constraint imposed essentially shifts the function by a constant. This constant can be found by examining the "intercept" coefficient from the model fit (≈ 0.60). Once we account for that constant, our estimation is quite good.

```
## code below is copied from the lecture notes
set.seed(500)
N <- 5000
x1 <- runif(N, -3, 3); x2 <- runif(N, -3, 3)
f1 <- function(x1,x2) 2*cos(pi*x1/4)*sin(pi*x2/2)
f2 <- function(x1,x2) sin(pi/2 + x1) + cos(x2^2/4)
y_x1_x2_f1 <- f1(x1,x2) + rnorm(N)
y_x1_x2_f2 <- f2(x1,x2) + rnorm(N)
df_fit <- data.frame(y_x1_x2_f1,y_x1_x2_f2,x1,x2)
fit_te_x1_x2_f1 <- gam(y_x1_x2_f1 ~ te(x1, x2, k=c(10,10), bs=c("cr","cr")),
                       method="REML", data=df_fit)
fit_te_x1_x2_f2 <- gam(y_x1_x2_f2 ~ te(x1, x2, k=c(10,10), bs=c("cr","cr")),
                       method="REML", data=df_fit)
## get estimated coefficients
# grid of new x1, x2 values to predict on
```

```r
nx_pred <- 500
x1_pred <- seq(min(df_fit$x1),max(df_fit$x1),len=nx_pred)
x2_pred <- seq(min(x2),max(x2),len=nx_pred)
# get all combinations of x1 and x2 values
df_pred <- expand.grid(x1=x1_pred, x2=x2_pred)
# get actual coefficient estimates
f1hat_x1_x2 <- predict(fit_te_x1_x2_f1, newdata=df_pred, type='terms')
f2hat_x1_x2 <- predict(fit_te_x1_x2_f2, newdata=df_pred, type='terms')
# get predicted differences
df_plot <-
  data.frame(df_pred,
             f1hat=f1hat_x1_x2[,"te(x1,x2)"],
             f2hat=f2hat_x1_x2[,"te(x1,x2)"],
             f1 = f1(df_pred$x1, df_pred$x2),
             f2 = f2(df_pred$x1, df_pred$x2)) %>%
  mutate(f2_f2hat = f2 - f2hat,
         f1_f1hat = f1 - f1hat)
## make separate plots for \hat{f}_1 and \hat{f}_2
## (they're on very dffierent scales)
plt_f1_diff <-
    df_plot %>%
    ggplot() + theme_classic(base_size=18) +
    geom_raster(aes(x1,x2,fill=f1_f1hat)) +
    scale_fill_gradientn(colours=c("red","white","blue"),
                         limits=c(-1,1)*max(range(df_plot$f1_f1hat)))+
    ggtitle(expression(f1(x[1],x[2]) - hat(f1)(x[1],x[2]) ))+ labs(fill=NULL)
plt_f2_diff <-
    df_plot %>%
    ggplot() + theme_classic(base_size=18) +
    geom_raster(aes(x1,x2,fill=f2_f2hat)) +
    scale_fill_gradientn(colours=c("red","white","blue"),
                         limits=c(-1,1)*max(range(df_plot$f2_f2hat))) +
    ggtitle(expression(f2(x[1],x[2]) - hat(f2)(x[1],x[2]) )) + labs(fill=NULL)
```

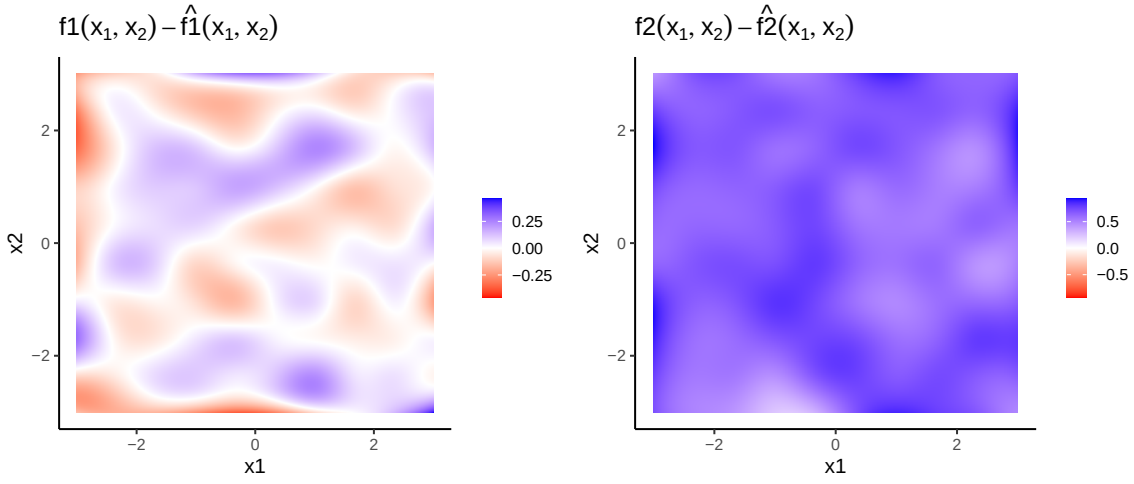

Figure 1: Difference bteween the true and estimated surfaces

2. This question relates to transformations of the predictors

   - Apply a monotonic transformation to x_1 (e.g. $\tilde{x}_1 = e^{x_1}$) simulated using the code above

```
df_fit <-
    df_fit %>%
    mutate(x1_tilde = exp(x1))
```

   - Estimate $\hat{f}_p(\tilde{x}_1, x_2)$ using the tensor product approach, plot the estimated coefficient on the transformed and original scale

     The estimated coefficient is plotted on the transformed and original scale in Figure 2 below.

```
fit_te_x1_tilde_x2_f1 <-
    gam(y_x1_x2_f1 ~ te(x1_tilde, x2, k=c(10,10), bs=c("cr","cr")),
        method="REML", data=df_fit)
x1_tilde_pred <- seq(min(df_fit$x1_tilde),max(df_fit$x1_tilde),len=nx_pred)
df_pred_tilde <- expand.grid(x1_tilde=x1_tilde_pred, x2=x2_pred)
f1_hat_x1_tilde_x2 <- predict(fit_te_x1_tilde_x2_f1, newdata=df_pred_tilde, type="response")
## get predicted values on the original scale on an evenly spaced grid
df_pred$x1_tilde <- exp(df_pred$x1)
df_pred$x1       <- NULL
f1_hat_x1_tilde_rs_x2 <- predict(fit_te_x1_tilde_x2_f1, newdata=df_pred, type="response")



df_plota <-
  data.frame(df_pred_tilde,
             f1hat=f1_hat_x1_tilde_x2)  %>%
    mutate(transform = "Transformed Scale")
df_plotb <-
  data.frame(df_pred,
             f1hat=f1_hat_x1_tilde_rs_x2) %>%
    mutate(transform = "Original Scale",
           x1_tilde = log(x1_tilde))
df_plot <- bind_rows(df_plota, df_plotb)

plot_f1_x1_tilde <-
    df_plot %>%
    ggplot() + theme_classic(base_size=18) +
    geom_raster(aes(x1_tilde,x2,fill=f1hat)) +
    scale_fill_gradientn(colours=c("red","white","blue"),
                         limits=c(-1,1)*max(range(df_plot$f1hat))) +
    facet_grid(~transform, scales="free_x")
```

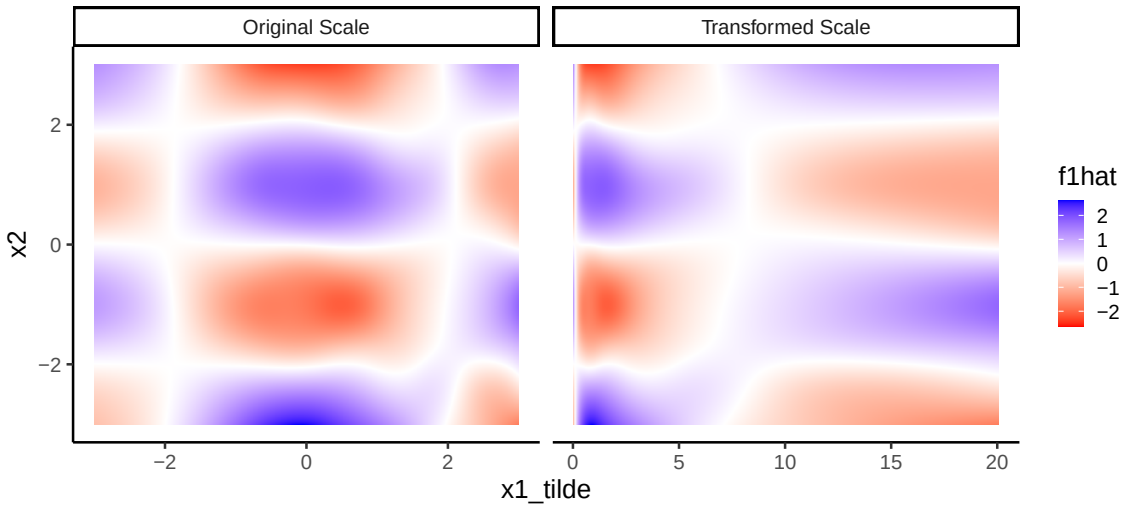

Figure 2: Estimated $\hat{f}(e^{x_1}, x_2)$ plotted on both the exponentiated scale used in model fitting (right panel) and on the original scale (left panel) for comparisons with the estimated coefficient fit directly on x_1

- Are the results the same? Why or Why not?

  The results are not the same as those shown in lecture (fitting the model on the original scale of x_1). This happens because the second derivative penalty is not the same since this is not just a linear re-scaling of the covariate. This can be readily seen by considering the univariate case (i.e. compare $f''(x)$ to $f''(g(x))$).

  Also of note, looking at thr right panel of Figure 2 you'll see that there is a very rapid change in the estimated coefficient surface near $e^{x_1} = 0$. This rapid change is only possible because by default *mgcv* places knots at the quantiles of the covariate space. If the knots were equally spaced over the range of the domain of e^{x_1}, we would be very unlikely to capture that rapid change in the surface.

3. This question relates to varying coefficient models for bivariate smooths. The *te()* function accepts "by" arguments which function the same way as univariate smooths.

   - Simulate data according the the model

     $$y_i = 2 + f_1(x_{i1}, x_{i2})(1 - x_{i3}) + f2(x_{i1}, x_{i2})x_{i3}$$

     where x_{i3} is a binary random variable
   - Fit this model using *mgcv::gam()* using the tensor product approach, plot the estimated $\hat{f}_1, \hat{f}_2$.

   This model incorporates two key ideas from the GAMs we've been discussing. The first is that of varying coefficients for bivariate surfaces (as stated in the model) and the second is identifiability. We can estimate this model in one of three ways. The first involves fitting the model as presented above.

4. This question uses the NHANES data from the course website.

   - Create the dataset:
     - Load the data, subset the data to "good" ("good_day" = 1) Mondays ("DoW" = "Monday")
       See code below. Note that some values of "Age" are missing in the dataset (NHANES 2003-2006 topcodes age at 85 for privacy reasons, these show up as missing for this age variable).

```
df_fit <- read_rds(here::here("data","data_processed","NHANES_AC_processed.rds"))
df_fit <-
    df_fit %>%
    filter(good_day %in% 1, DoW %in% c("Monday"), !is.na(Age))
```

     - Transform the data from wide to long format for the minute level activity counts. Specifically, each row in the long dataset should correspond to a subject-minute

```
df_fit <-
    df_fit %>%
    ## keep only ID, age, and activity count columns
    dplyr::select(SEQN, Age, MIN1:MIN1440) %>%
    ## transform to long format
    pivot_longer(cols=MIN1:MIN1440)
```

     - Add a (numeric) column for time of day (e.g. $1, \ldots, 1440$)

```
df_fit <-
    df_fit %>%
    mutate(time = as.numeric(str_extract(name, "[0-9]+")))
```

   - Estimate the following model:

$$E[\log(1 + \text{AC}_i(t))] = f(t, \text{Age}_i) + \epsilon_i(t) \qquad \epsilon_i(t) \sim N(0, \sigma^2_\epsilon)$$

     where $t = 1, \ldots, 1440$ denotes minute of the day and $\text{AC}_i(t)$ is the activity count for subject i at time t. Note that you'll need to transform the data from wide to long format before model fitting.

     When I wrote this problem I was working on a computing cluster and so I wasn't paying attention to memory requirements. Fitting this model on the size data that we have is not feasible using *mgcv::gam()* on a personal laptop. The *mgcv* package has a function which works (almost) exactly like *mgcv::gam()*, but is designed for very large datasets. This function, *mgcv::bam()*, performs computations in chunks, avoiding constructing the full design matrix which in our case is a massive $14,012,640 \times 210 = 2,942,654,400$ data elements. The *mgcv::bam()* function also has an option for performing a "fast" REML procedure for smooothing parameter selection (method="fREML") and an additional speed-up obtained by discretizing covariate values and doing some computations in parallel (discrete=TRUE). I use both of these options below to speed up computation time.

Also, here I use cyclic cubic regression splines in the "time" directions as time of day is cyclical. Note that you would get nearly identical results in this case if you had just used cubic regression splines.

```
## calculate log(1+AC(t))
df_fit <-
   df_fit %>%
   mutate(lAC = log(1+value))
## estimate the model
fit_Age_lAC <- bam(lAC ~ te(time, Age, bs=c("cc","cr"), k=c(15,15)),
                   method="fREML", data=df_fit, discrete=TRUE)
summary(fit_Age_lAC)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## lAC ~ te(time, Age, bs = c("cc", "cr"), k = c(15, 15))
##
## Parametric coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.1212896  0.0007486    2834   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                 edf Ref.df     F p-value
## te(time,Age) 207.1  208.9 28337  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.297   Deviance explained = 29.7%
## fREML = 3.1571e+07  Scale est. = 5.3023    n = 14012640
```

- Plot the estimated surface $f(t, \text{Age})$

  Figure 3 plots the estimated surface.

```
## get range of ages and time of day (minute) to predict on
na_pred <- 1000
nt_pred <- 1000
aind_pred <- seq(min(df_fit$Age), max(df_fit$Age), len=na_pred)
tind_pred <- seq(min(df_fit$time), max(df_fit$time), len=na_pred)
## get all combinations of time of day and age values
df_pred <- expand.grid(Age=aind_pred, time=tind_pred)
## obtain predictions. Note that here we predict on the resposne scale
## due to the identifiability constraints imposed by mgcv
fhat_time_age <- predict(fit_Age_lAC, newdata=df_pred, type="response")
## plot the data
plt_fhat <-
  data.frame(df_pred, "fhat" = fhat_time_age) %>%
  ggplot() + theme_classic(base_size=18) +
  geom_raster(aes(time,Age,fill=fhat)) +
  scale_fill_gradientn(colours=c("red","white","blue")) +
  ggtitle(expression(E ~ "[" ~ log(1+AC(t)) ~ "|" ~ Age ~ "]")) +
  scale_x_continuous(breaks=(c(1,6,12,18,23)*60+1),
                     labels=c("01:00","06:00","12:00","18:00","23:00")) +
  labs(fill=NULL)
```

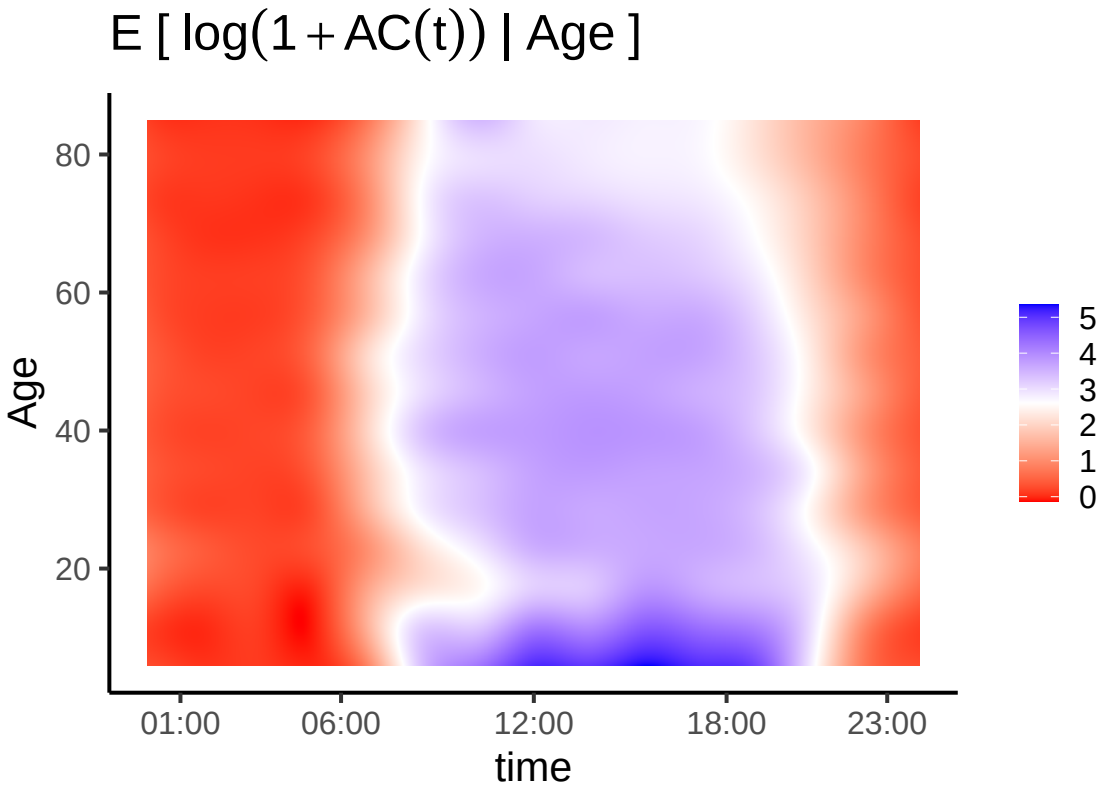

Figure 3: Estimated log activity counts as a function of time of day and age

- Are our model assumptions reasonable?

  We are assuming indepedent residuals, which in this case implies that observations within a person (time of day) are independent given the mean $E[\log(\text{AC}(t))|\text{Age}]$. This is unlikely to be a reasonable assumption. Indeed, we can get a Method of Moments estimate of the within person correlation and plot. Figure 4 plots the estimated within-person correlation of residuals by time of day (left panel) and a histogram of these correlations (right panel). We can see that, indeed, there appears to be within person correlation which is strongest (closest to 1) for times of day which are temporally "close". You will be investigating how ignoring correlated data can affect smoothing parameter selection in your homework.

```
resid_mat  <- matrix(fit_Age_lAC$residuals, nrow(df_fit)/1440, 1440, byrow=TRUE)
cov_resids <- cov(resid_mat)
cor_resids <- cov2cor(cov_resids)
df_plt_cor <-
        data.frame("corr" = as.vector(cor_resids),
                   "time1" = rep(1:1440, each=1440),
                   "time2" = rep(1:1440, 1440))
plt_cor <-
    df_plt_cor %>%
```

```
    ggplot() + theme_classic(base_size=18) +
    geom_raster(aes(x=time1,y=time2,fill=corr)) +
        scale_fill_gradientn(colours=c("red","white","blue")) +
    ggtitle(expression(rho(hat(e)(s), hat(e)(t)))) +
    scale_x_continuous(breaks=(c(1,6,12,18,23)*60+1),
                       labels=c("01:00","06:00","12:00","18:00","23:00")) +
    scale_y_continuous(breaks=(c(1,6,12,18,23)*60+1),
                       labels=c("01:00","06:00","12:00","18:00","23:00")) +
    labs(fill=NULL)
hist_cor <-
    df_plt_cor %>%
    ## take only one set of correlations
    ## drop time1=time2 since correlation is trivially 1
    filter(time1 > time2) %>%
    ggplot() + theme_classic(base_size=18) +
    geom_histogram(aes(x=corr)) +
    xlab(expression(rho(hat(e)(s), hat(e)(t)))) +
    geom_vline(xintercept=0, col='red',lwd=2,lty=2)
```

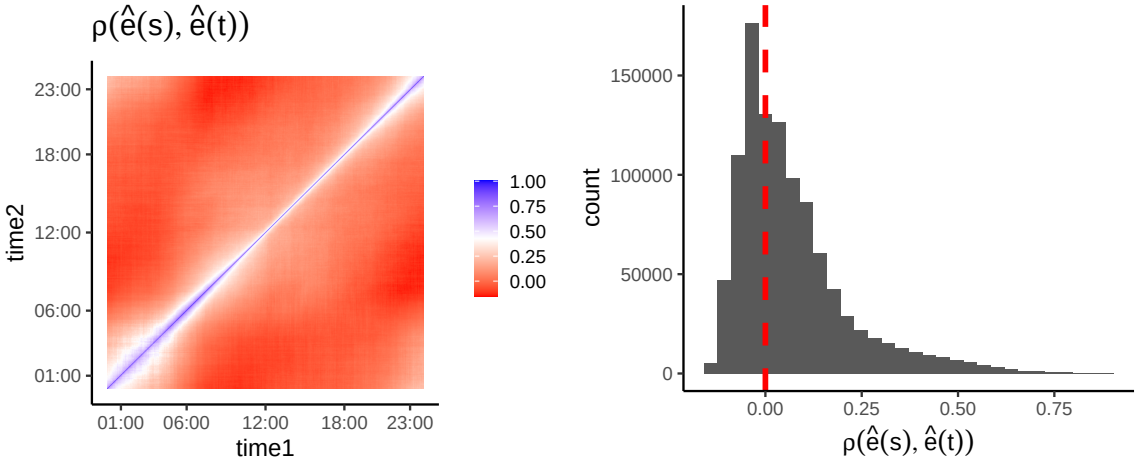

Figure 4: Method of Moments estimate of within-person correlation of residuals