Module 06: Non-linear Regression

Live Session | SYS 6018 | Spring 2021

Contents

Required R Packages

We will be using the following R packages:

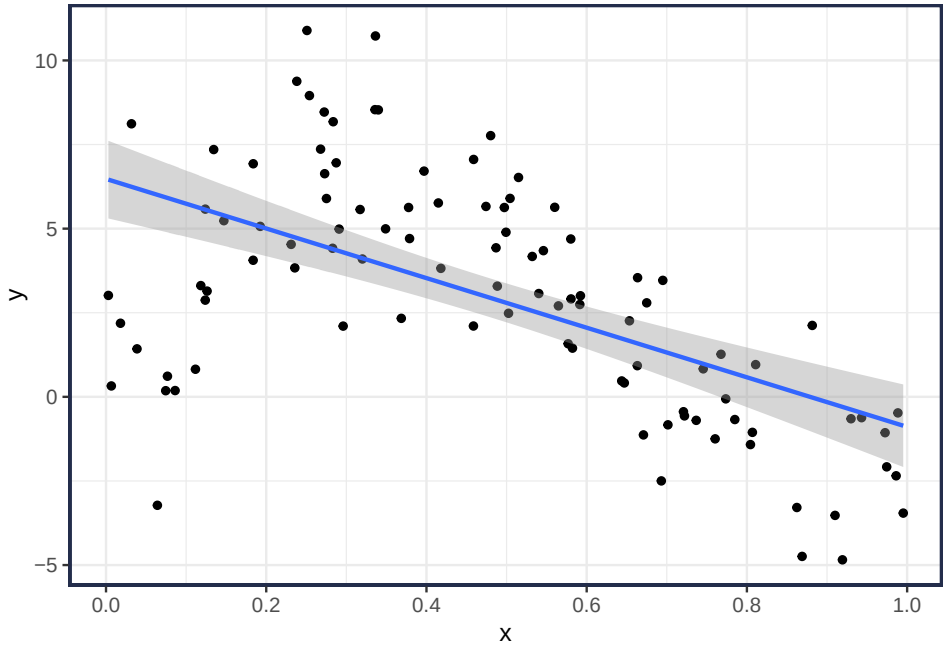
```
library(broom)       # tidy extraction of model components
library(splines)     # working with splines
library(mgcv)        # good GAM implementation
library(tidyverse)   # data manipulation and visualization
```

Simulated data set

```
#-- Data Generation
n = 100                                  # number of observations
generate_x <- function(n) runif(n)       # U[0,1]
f <- function(x) 1 + 2*x + 5*sin(5*x)    # true mean function
sd = 2                                   # stdev for error

set.seed(825)                            # set seed for reproducibility
x = generate_x(n)                        # get x values
y = f(x) + rnorm(n, sd=sd)               # get y values
data_train = tibble(x,y)
```

Linear fit not sufficient

```
ggplot(data_train, aes(x,y)) +
  geom_point() +
  geom_smooth(method="lm", formula='y~poly(x, 1)') +
  scale_x_continuous(breaks=seq(0, 1, by=.20))
```


10
5
0
−5
y
0.0
0.2
0.4
0.6
0.8
1.0
x

1 Basis Function Modeling

For a univariate x, a linear basis expansion is

$$\hat{f}(x) = \sum_j \hat{\theta}_j b_j(x)$$

where $b_j(x)$ is the value of the jth basis function at x and θ_j is the coefficient to be estimated.

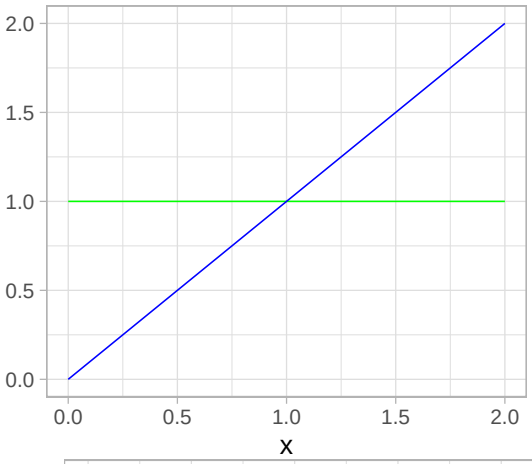
- The $b_j(x)$ are sometimes specified in advanced (i.e., not estimated). But other approaches use sample data to estimate (e.g., using quantiles for knot placement).
  - Be sure to estimate everything from the training data!

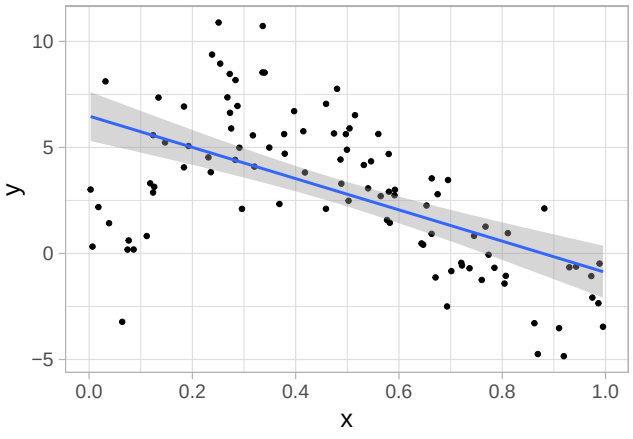
Examples:

- **Linear Regression**

$$\hat{f}(x) = \hat{\beta}_0 + \hat{\beta}_1 x$$

$$b_0(x) = 1$$
$$b_1(x) = x$$

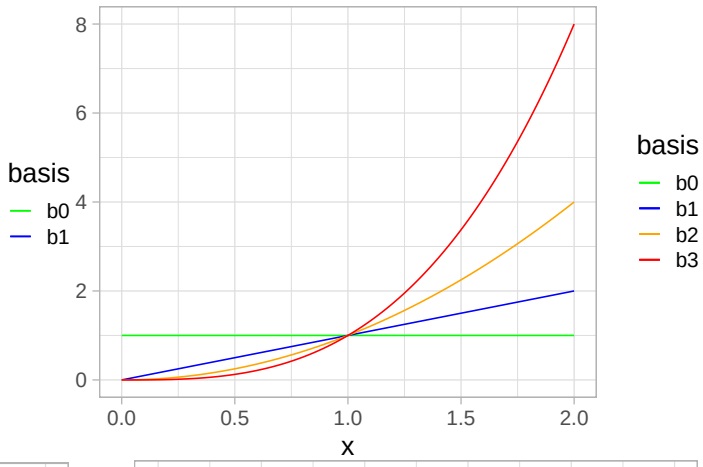


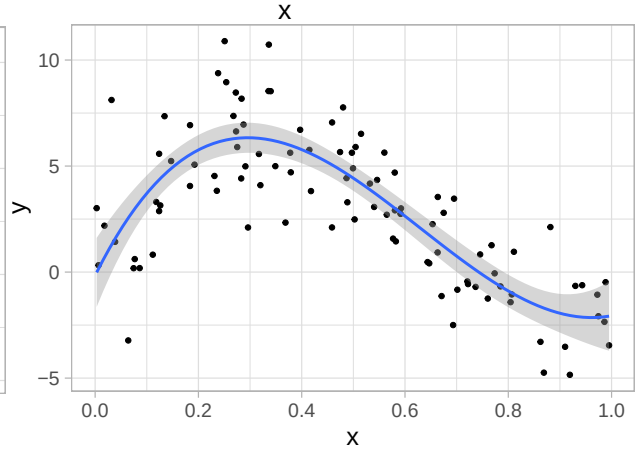


- **Polynomial Regression**

$$\hat{f}(x) = \sum_{j=0}^{d} \hat{\beta}_j x^j$$

$$b_j(x) = x^j$$

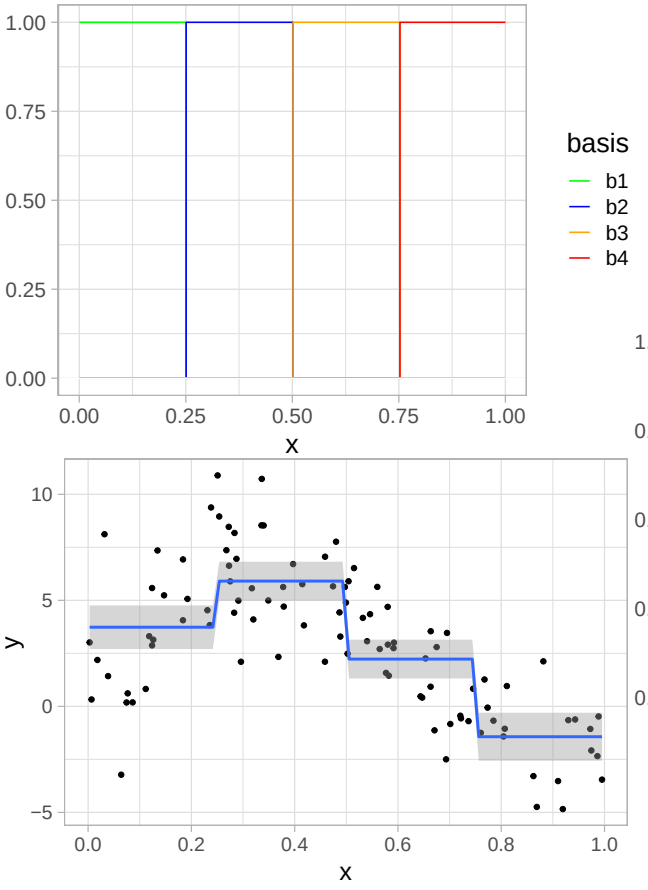




- **Piecewise Constant Regression (Regressogram)**

$$\hat{f}(x) = \sum_{j=1}^{p} \hat{\beta}_j \, \mathbb{1}(x \in R_j)$$

$$b_1(x) = \mathbb{1}(x \in R_1)$$
$$b_2(x) = \mathbb{1}(x \in R_2)$$
$$\vdots$$
$$b_p(x) = \mathbb{1}(x \in R_p)$$



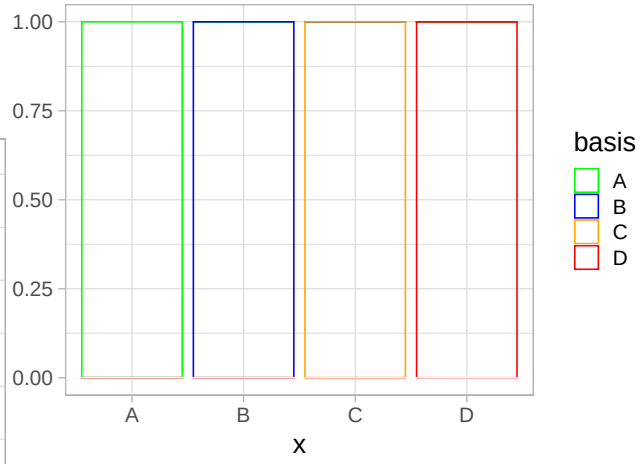
- **Categorical encoding (dummy, one-hot)**

$$x \in \{c_1, c_2, \ldots, c_p\}$$

$$\hat{f}(x) = \sum_{j=1}^{p} \hat{\beta}_j \, \mathbb{1}(x = c_j) \qquad \text{one-hot}$$

$$= \hat{\beta}_0 + \sum_{j=2}^{p} \hat{\beta}_j \, \mathbb{1}(x = c_j) \qquad \text{dummy}$$

$$b_0(x) = 1$$
$$b_1(x) = \mathbb{1}(x = c_1)$$
$$b_2(x) = \mathbb{1}(x = c_2)$$
$$\vdots$$
$$b_p(x) = \mathbb{1}(x = c_p)$$



1.1 Piecewise Polynomials

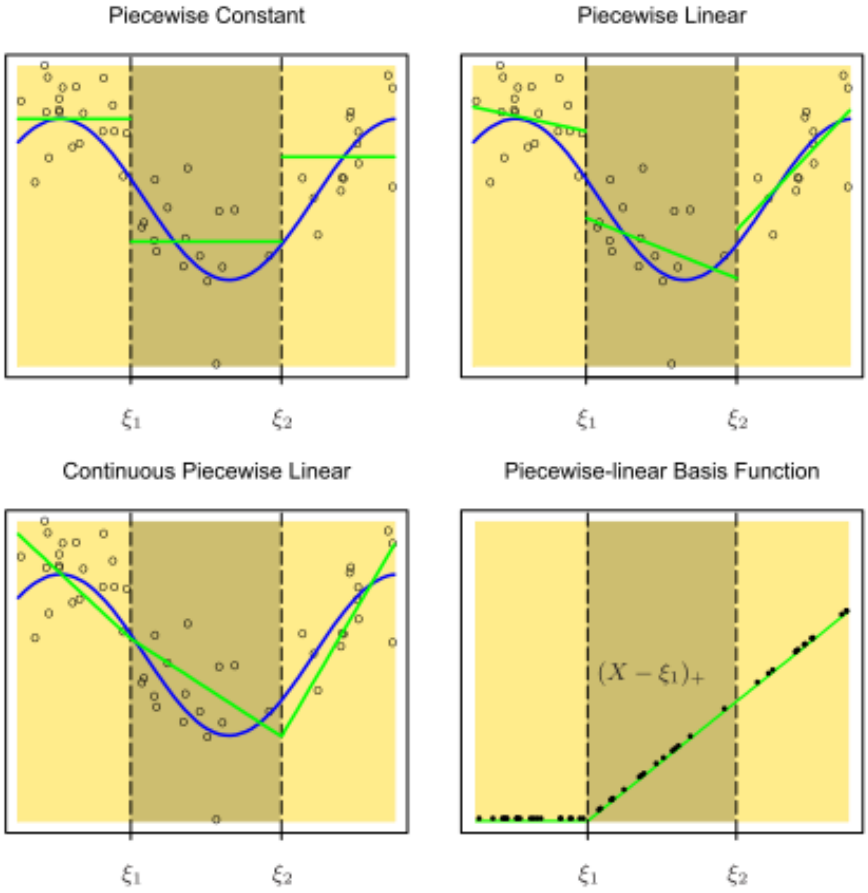
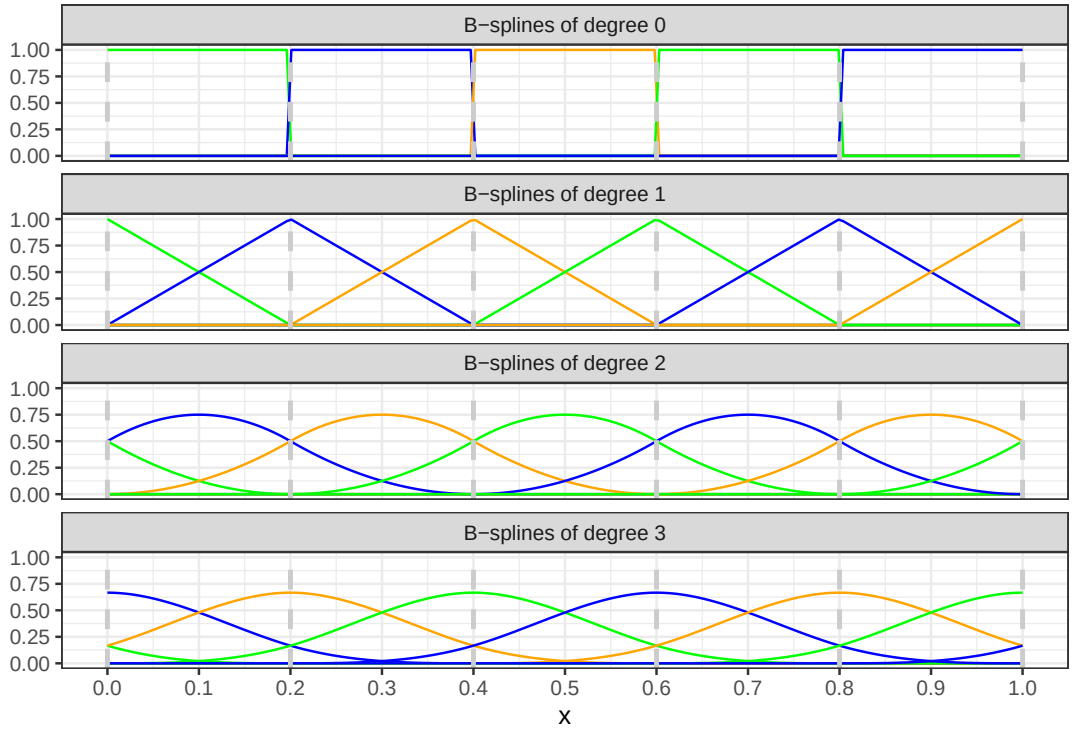


FIGURE 5.1. *The top left panel shows a piecewise constant function fit to some artificial data. The broken vertical lines indicate the positions of the two* knots *ξ_1 and ξ_2. The blue curve represents the true function, from which the data were generated with Gaussian noise. The remaining two panels show piecewise linear functions fit to the same data—the top right unrestricted, and the lower left restricted to be continuous at the knots. The lower right panel shows a piecewise–linear basis function, $h_3(X) = (X - \xi_1)_+$, continuous at ξ_1. The black points indicate the sample evaluations $h_3(x_i)$, $i = 1, \ldots, N$.*

1.2 B-Splines



Like ESL Fig 5.20, B-splines (knots shown by vertical dashed lines)

- A degree = 0 B-spline is a *regressogram* basis. Will lead to a piecewise constant fit.
- A degree = 3 B-spline (called *cubic* splines) is similar in shape to a Gaussian pdf. But the B-spline has finite support and facilitates quick computation (due to the induced sparseness).

1.2.1 Parameter Estimation

In matrix notation,

$$\begin{aligned}\hat{f}(x) &= \sum_j \hat{\theta}_j b_j(x) \\ &= B\hat{\theta}\end{aligned}$$

where B is the *basis matrix*.

- For example, a polynomial matrix is

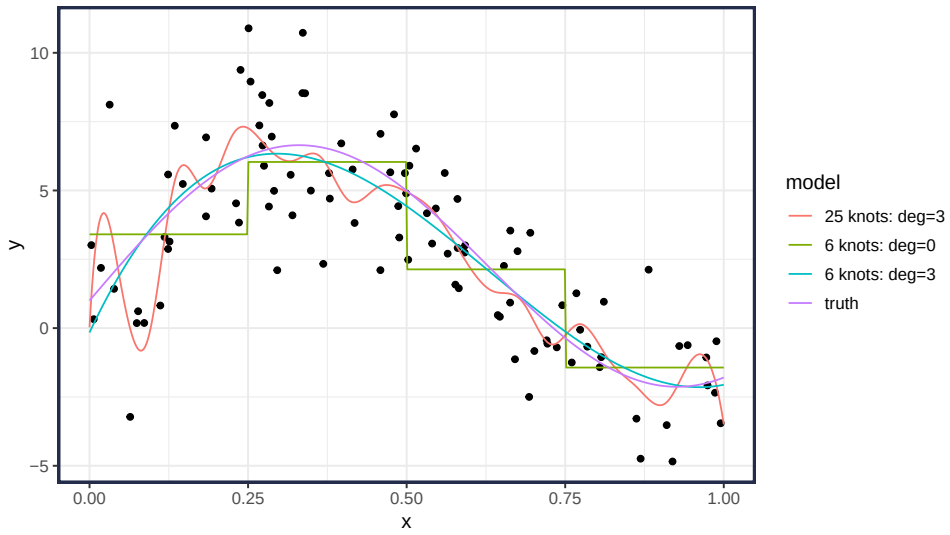
$$B = \begin{bmatrix} 1 & X_1 & \dots & X_1^J \\ 1 & X_2 & \dots & X_2^J \\ \vdots & \vdots & \vdots & \vdots \\ 1 & X_n & \dots & X_n^J \end{bmatrix}$$

- More generally,

$$B = \begin{bmatrix} b_1(x_1) & b_2(x_1) & \dots & b_J(x_1) \\ b_1(x_2) & b_2(x_2) & \dots & b_J(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ b_1(x_n) & b_2(x_n) & \dots & b_J(x_n) \end{bmatrix}$$

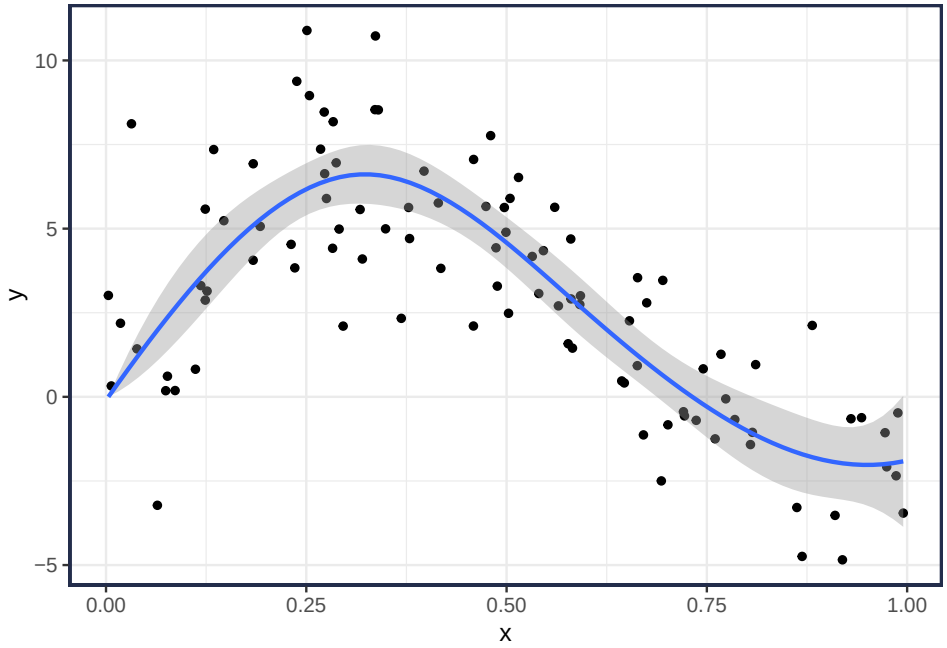
- Now, its in a form just like linear regression! Estimate with OLS

$$\hat{\theta} = (B^\mathsf{T} B)^{-1} B^\mathsf{T} Y$$



- It may be helpful to think of a basis expansion as similar to a dummy coding for categorical variables.
  - This expands the single variable x into df new variables.
- In R, the function `bs()` can be put directly in formula to make a B-spline.

```
#- fit a 5 df B-spline
# Note: don't need to include an intercept in the lm()
library(splines)
model_bs = lm(y~bs(x, df=5)-1, data=data_train)

#- examine results
broom::tidy(model_bs)
#> # A tibble: 5 x 5
#>   term           estimate std.error statistic  p.value
#>   <chr>             <dbl>     <dbl>     <dbl>    <dbl>
#> 1 bs(x, df = 5)1     3.53     1.18       2.99 3.51e- 3
#> 2 bs(x, df = 5)2     9.11     1.19       7.63 1.79e-11
#> 3 bs(x, df = 5)3     1.51     1.31       1.16 2.49e- 1
#> 4 bs(x, df = 5)4    -2.57     1.45      -1.77 7.97e- 2
#> 5 bs(x, df = 5)5    -1.92     0.980     -1.96 5.35e- 2

#- plot
ggplot(data_train, aes(x,y)) + geom_point() +
  geom_smooth(method='lm', formula='y~bs(x, df=5)-1')
```

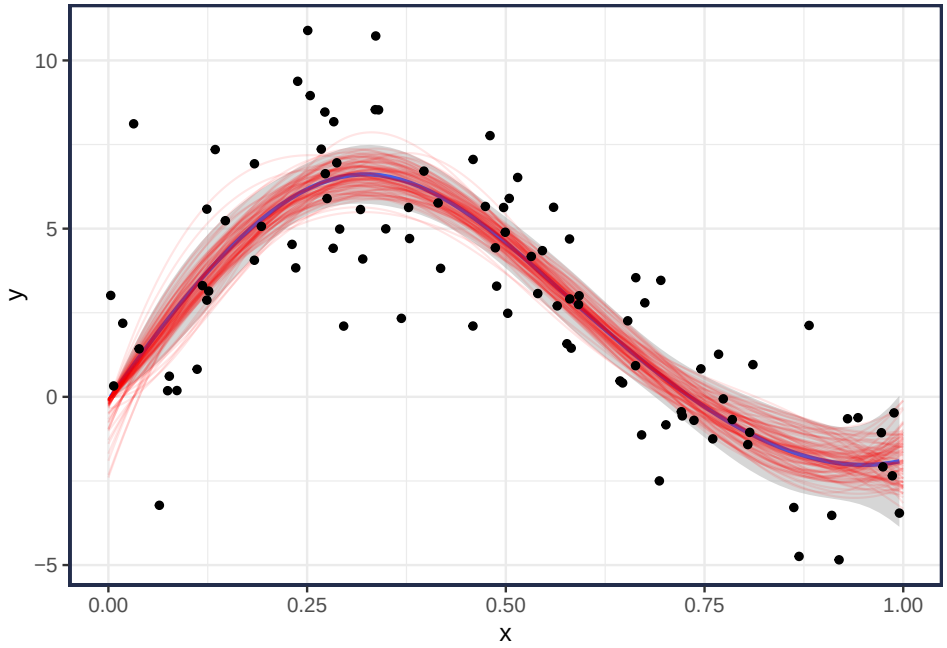


- Setting `df=5` will create a B-spline design matrix with 5 columns
  - So there are 5 basis functions
- The number of (internal) knots is equal to df-degree and at equally spaced quantiles of the data
  - With `df=5` and `deg=3`, there are 2 internal knots at the 33.33% and 66.66% percentiles of x

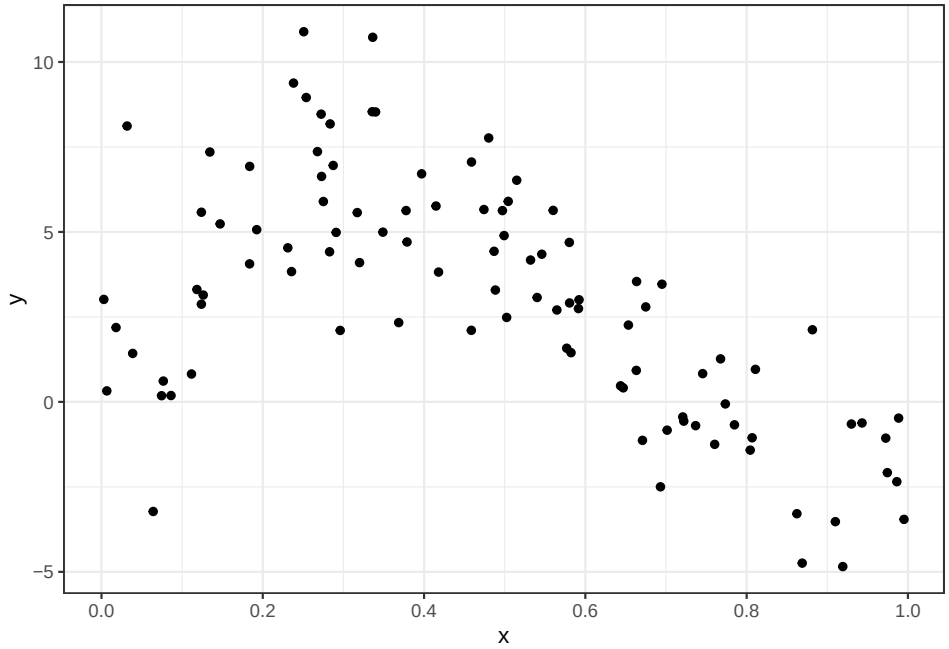
1.3 Bootstrap Confidence Interval for $f(x)$

Bootstrap can be used to understand the uncertainty in the fitted values

```
#-- Bootstrap CI (Percentile Method)
M = 100                                  # number of bootstrap samples
data_eval = tibble(x=seq(0, 1, length=300)) # evaluation points
YHAT = matrix(NA, nrow(data_eval), M)       # initialize matrix for fitted values

#-- Spline Settings
for(m in 1:M){
  #- sample from empirical distribution
  ind = sample(n, replace=TRUE)               # sample indices with replacement
  #- fit bspline model
  m_boot = lm(y~bs(x, df=5)-1,
              data=data_train[ind,])    # fit bootstrap data
  #- predict from bootstrap model
  YHAT[,m] = predict(m_boot, newdata=data_eval)
}

#-- Convert to tibble and plot
data_fit = as_tibble(YHAT) %>%
  bind_cols(data_eval) %>%      # add the eval points
  gather(simulation, y, -x)     # convert to long format

ggplot(data_train, aes(x,y)) +
  geom_smooth(method='lm',
              formula='y~bs(x, df=5)-1') +
  geom_line(data=data_fit, color="red", alpha=.10, aes(group=simulation)) +
  geom_point()
```



2 Optimal Complexity (and knots and edof)



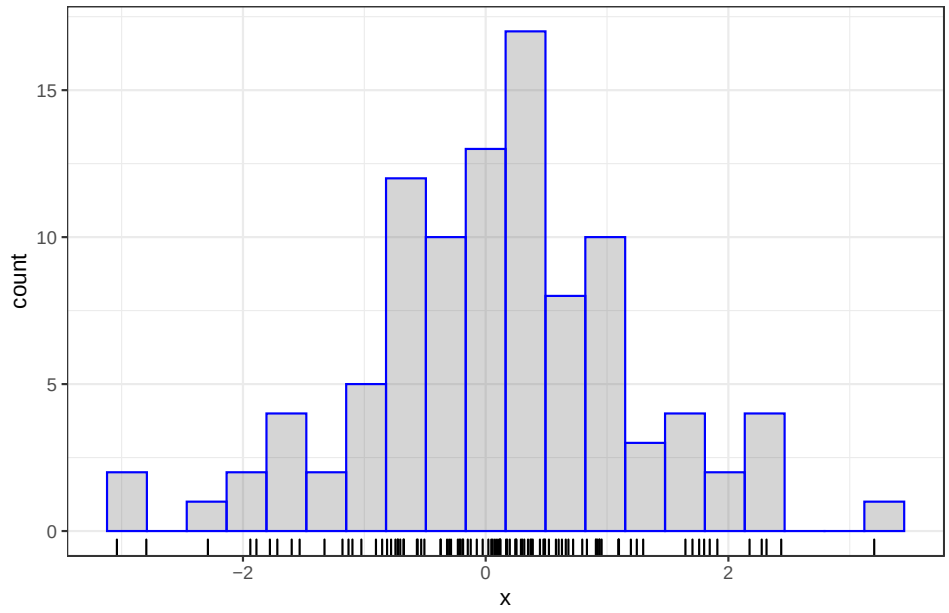
Your Turn #1

1. Where would you put the knots/breaks for a piecewise-constant (step-function) model?
2. How would you do this in R?
3. How would you evaluate the model?

2.1 Binning Continuous Data

- For regressograms and histograms

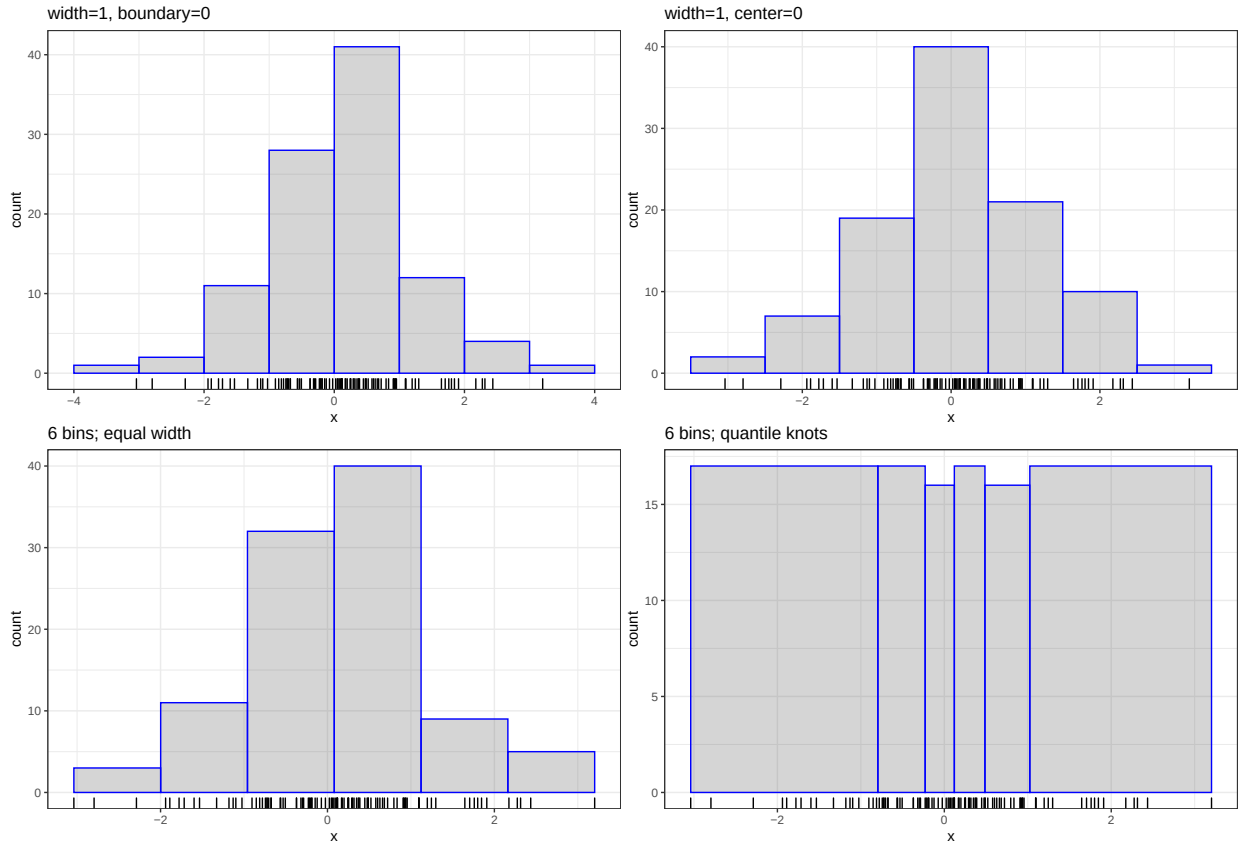
```
#- plot histogram

plot_hist <- function(..., title="") {
   ggplot(data=tibble(x), aes(x)) + geom_rug() +
    geom_histogram(...,
                   color="blue", alpha=.25) +
    labs(title=title) + theme_bw()
}

#-- Simulate data
set.seed(2020)
x = rnorm(100)  # N(0,1)

#-- distribution of x
summary(x)    # quartiles, range
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>  -3.039  -0.562   0.120   0.109   0.739   3.202
plot_hist(bins=20)
```



There are many ways to bin or discretize continuous data. Using equal width intervals or using varying width intervals that have about an equal number of points (quantile binning). The `cut_{interval, number, width}` functions from `ggplot2` package can perform this binning. The base R `cut()` is same as `ggplot2::cut_interval()`.

```
library(ggplot2)

#- width binning (equal bin size, specify width)
x_width = cut_width(x, width=1, boundary=0) # specify boundary
levels(x_width)
#> [1] "[-4,-3]" "(-3,-2]" "(-2,-1]" "(-1,0]"  "(0,1]"   "(1,2]"   "(2,3]"
#> [8] "(3,4]"
plot_hist(binwidth=1, boundary=0, title="width=1, boundary=0")

x_width = cut_width(x, width=1, center=0) # specify center of bin
levels(x_width)
#> [1] "[-3.5,-2.5]" "(-2.5,-1.5]" "(-1.5,-0.5]" "(-0.5,0.5]"  "(0.5,1.5]"
#> [6] "(1.5,2.5]"   "(2.5,3.5]"
plot_hist(binwidth=1, center=0, title="width=1, center=0")

#- range binning (equal bin size, specify number of bins)
x_interval = cut_interval(x, n=6)
levels(x_interval)
#> [1] "[-3.04,-2]"      "(-2,-0.959]"     "(-0.959,0.0814]" "(0.0814,1.12]"
#> [5] "(1.12,2.16]"     "(2.16,3.2]"
```

```
plot_hist(breaks=seq(min(x), max(x), length=7),
          title="6 bins; equal width")


#- quantile binning (equal number in each bin, specify number of bins)
x_quantile = cut_number(x, n=6)
levels(x_quantile)
#> [1] "[-3.04,-0.796]"  "(-0.796,-0.229]" "(-0.229,0.12]"   "(0.12,0.487]"
#> [5] "(0.487,1.02]"    "(1.02,3.2]"
dplyr::ntile(x, n=6)
#>   [1] 4 4 1 1 1 5 5 2 6 3 1 5 6 2 3 6 6 1 1 3 6 6 4 3 5 4 6 5 3 3 1 2 6 6 4 4 2
#>  [38] 3 2 4 5 2 2 2 1 4 2 3 5 4 3 5 2 4 3 3 3 1 2 5 6 4 1 6 5 4 5 3 3 4 5 6 1 2
#>  [75] 1 2 1 2 3 4 5 3 6 4 1 6 1 5 6 5 1 1 4 6 3 2 4 5 2 2
x_quantile %>% as.integer()
#>   [1] 4 4 1 1 1 5 5 2 6 3 1 5 6 2 3 6 6 1 1 3 6 6 4 3 5 4 6 5 3 3 1 2 6 6 4 4 2
#>  [38] 3 2 4 5 2 2 2 1 4 2 3 5 5 3 5 2 4 3 4 3 1 2 5 6 4 1 6 5 4 5 3 3 4 5 6 1 2
#>  [75] 1 2 1 2 3 4 5 3 6 4 1 6 1 6 6 5 1 1 4 6 3 2 4 5 2 2

plot_hist(breaks=quantile(x, probs=seq(0, 1, length=7)),
          title="6 bins; quantile knots")
```



Note: In R, the `cut()` functions will work in the formula, but won't work with `predict()`. A solution is to create a new categorical/factor column with the discretization.

3 Generalized Additive Modeling (GAM)

$$\hat{f}(x) = \sum_{j=1}^{p} f_j(x_j)$$

- $f_j(x)$ is *usually* a smooth function (e.g., B-splines) in x

- linear regression just specifies $f_j(x_j) = \beta_j x_j$ to be linear
- The R Package `mgcv` is the best package for implementing GAMs
  - allows interactions
  - very fast
  - finds optimal smoothing for each predictor

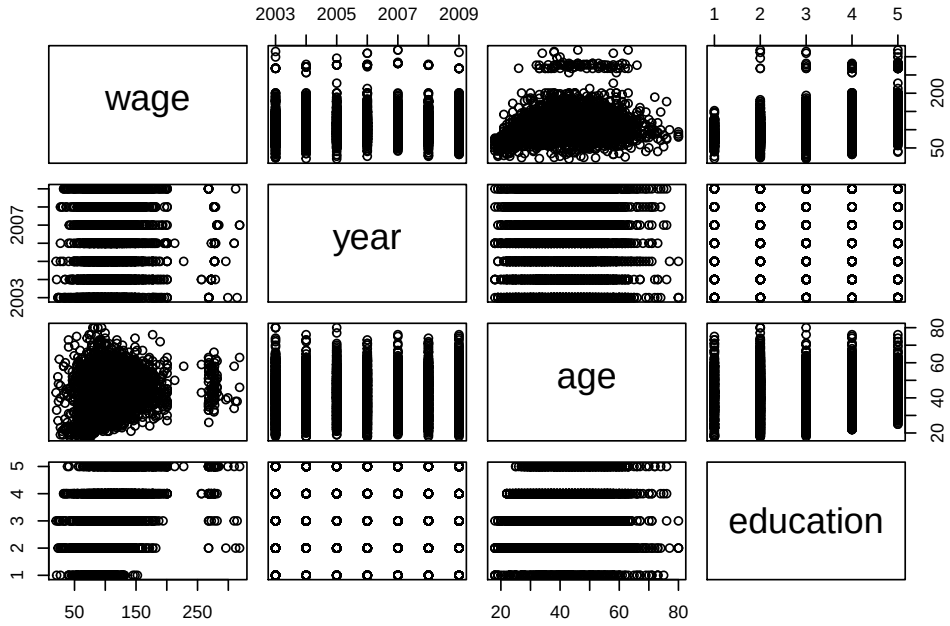
3.1 Using `mgcv`

Consider the Lab problem in section 7.8.3

```
library(ISLR)
data(Wage)
```

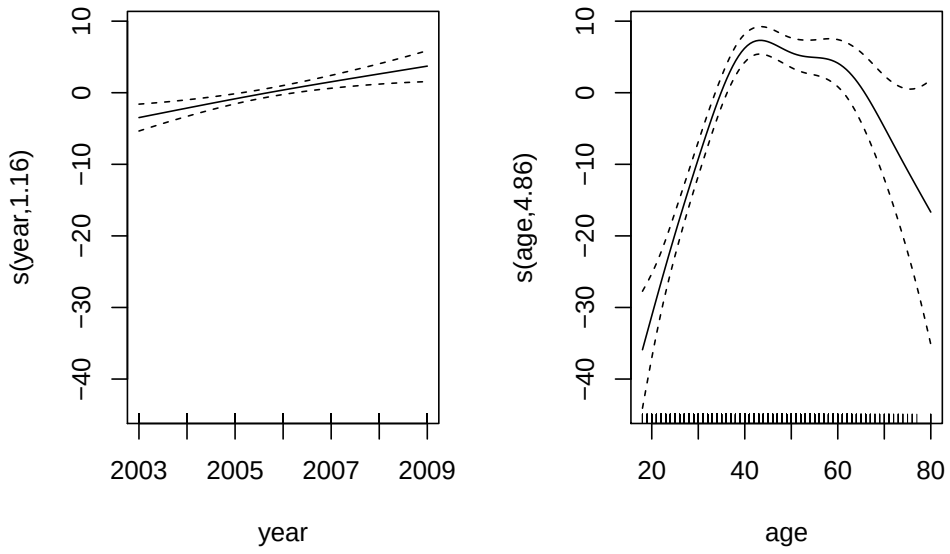
Predict `wage` using `year` (dbl), `age` (dbl), and `education` (fct).

```
#-- Pairs plot
Wage %>%
  select(wage, year, age, education) %>%
  pairs()
```




```
#-- Linear Model
fit_lm = lm(wage ~ year + age + education, data=Wage)
broom::tidy(fit_lm)
#> # A tibble: 7 x 5
#>   term                       estimate std.error statistic  p.value
#>   <chr>                         <dbl>     <dbl>     <dbl>    <dbl>
#> 1 (Intercept)               -2058.     649.         -3.17 1.54e- 3
#> 2 year                          1.06     0.324       3.26 1.12e- 3
#> 3 age                           0.562    0.0571      9.84 1.69e-22
#> 4 education2. HS Grad          11.4      2.48        4.60 4.34e- 6
#> 5 education3. Some College     24.2      2.61        9.30 2.61e-20
#> 6 education4. College Grad     39.7      2.59       15.4  2.38e-51
#> # ... with 1 more row

#-- Manual GAM using b-splines
fit_mgam = lm(wage ~ bs(year, df=4) + bs(age, df=5) + education,
              data=Wage)
broom::tidy(fit_mgam)
#> # A tibble: 14 x 5
#>   term               estimate std.error statistic  p.value
#>   <chr>                 <dbl>     <dbl>     <dbl>    <dbl>
#> 1 (Intercept)          48.1       6.02      7.99  1.96e-15
```

```
#> 2 bs(year, df = 4)1     0.559       3.99      0.140 8.89e- 1
#> 3 bs(year, df = 4)2   10.1         5.08      1.99  4.68e- 2
#> 4 bs(year, df = 4)3    3.58        3.99      0.898 3.69e- 1
#> 5 bs(year, df = 4)4    7.67        2.34      3.27  1.07e- 3
#> 6 bs(age, df = 5)1     5.64        9.48      0.595 5.52e- 1
#> # ... with 8 more rows

#-- GAM
library(mgcv)
gam(wage ~ year + age + education, data=Wage) %>%
  summary()  # 1290.8 GCV
#>
#> Family: gaussian
#> Link function: identity
#>
#> Formula:
#> wage ~ year + age + education
#>
#> Parametric coefficients:
#>                              Estimate Std. Error t value Pr(>|t|)
#> (Intercept)                 -2.06e+03   6.49e+02   -3.17   0.0015 **
#> year                         1.06e+00   3.24e-01    3.26   0.0011 **
#> age                          5.62e-01   5.71e-02    9.84  < 2e-16 ***
#> education2. HS Grad          1.14e+01   2.48e+00    4.60  4.3e-06 ***
#> education3. Some College     2.42e+01   2.61e+00    9.30  < 2e-16 ***
#> education4. College Grad     3.97e+01   2.59e+00   15.37  < 2e-16 ***
#> education5. Advanced Degree  6.49e+01   2.80e+00   23.13  < 2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#>
#> R-sq.(adj) =   0.26   Deviance explained = 26.2%
#> GCV = 1290.8  Scale est. = 1287.8    n = 3000

gam(wage ~ s(year, k = 5) + s(age) + education, data=Wage) %>%
  summary()  # 1240.2 GCV; s(year)=1.156; s(age)=4.857
#>
#> Family: gaussian
#> Link function: identity
#>
#> Formula:
#> wage ~ s(year, k = 5) + s(age) + education
#>
#> Parametric coefficients:
#>                             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)                    85.44       2.15   39.70  < 2e-16 ***
#> education2. HS Grad            10.98       2.43    4.52  6.3e-06 ***
#> education3. Some College       23.53       2.56    9.20  < 2e-16 ***
#> education4. College Grad       38.20       2.54   15.02  < 2e-16 ***
#> education5. Advanced Degree    62.58       2.76   22.69  < 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
#> s(year) 1.16   1.29 10.3 0.00045 ***
#> s(age)  4.86   5.93 38.1 < 2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> R-sq.(adj) =   0.29   Deviance explained = 29.3%
#> GCV = 1240.2  Scale est. = 1235.7    n = 3000

par(mfrow=c(1,2))
gam(wage ~ s(year, k = 5) + s(age) + education, data=Wage) %>%
  plot()
```



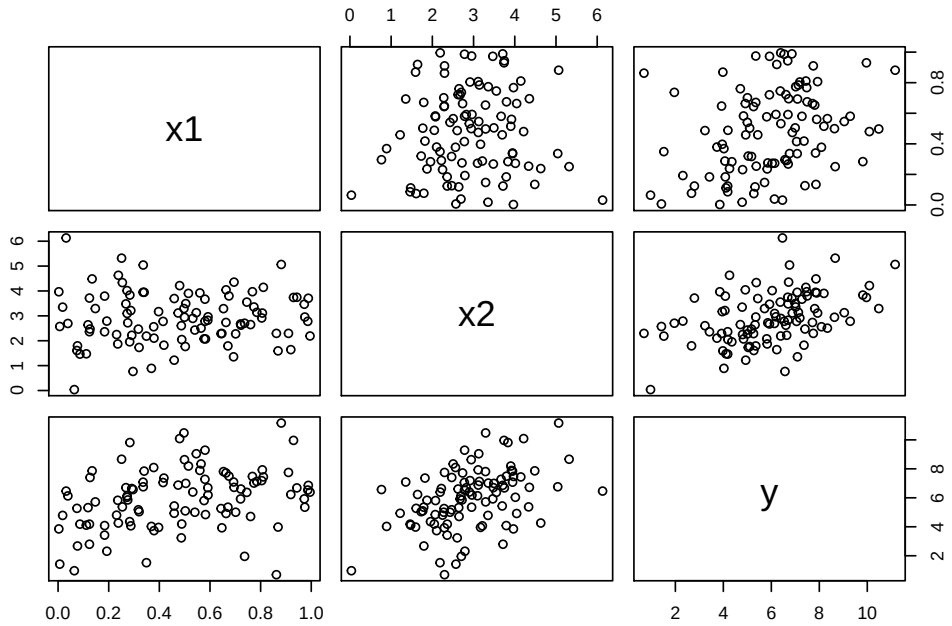
3.2 Backfitting (ISLR Problem 7.11)

In ISLR Section 7.7, it was mentioned that GAMs are generally fit using a backfitting approach. The idea behind backfitting is actually quite simple. We will now explore backfitting in the context of multiple linear regression.

Therefore, we take the following iterative approach: we repeatedly hold all but one coefficient estimate fixed at its current value, and update only that coefficient estimate using a simple linear regression. The process is continued until convergence—that is, until the coefficient estimates stop changing.

Variations of this type of iterative approach to estimation will show up in many courses.

```
#-- Data Generation
n = 100                                 # number of observations
generate_x <- function(n) tibble(x1=runif(n), x2=rnorm(n, 3, 1))
f <- function(x) 1.5 + 2*x$x1 + 1*x$x2    # true mean function
sd = 2                                  # stdev for error

set.seed(825)                           # set seed for reproducibility
x = generate_x(n)                       # get x values
x1=x$x1; x2=x$x2
y = f(x) + rnorm(n, sd=sd)              # get y values
data_train = tibble(x1=x$x1, x2=x$x2, y)
pairs(data_train)
lm(y~x1+x2, data=data_train)
#>
#> Call:
#> lm(formula = y ~ x1 + x2, data = data_train)
#>
#> Coefficients:
#> (Intercept)           x1           x2
#>       2.391        2.248        0.852
```



Model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

1. Initialize β_1

```
b1 = 0
```

2. Fit model to residuals without β_1 and estimate other coefficients.

$$y - \tilde{b}_1 x_1 = \beta_0 + \beta_2 x_2 + \epsilon$$

```
r = y - b1*x1
fit = lm(r~x2)
fit
#>
#> Call:
#> lm(formula = r ~ x2)
#>
#> Coefficients:
#> (Intercept)            x2
#>       3.392        0.878

b2 = coef(fit)[2]
```

3. Fit model to residuals without β_2 and estimate other coefficients.

$$y - \tilde{b}_2 x_2 = \beta_0 + \beta_1 x_1 + \epsilon$$

```
r = y - b2*x2
fit = lm(r~x1)
fit
#>
#> Call:
#> lm(formula = r ~ x1)
#>
#> Coefficients:
#> (Intercept)            x1
#>        2.32         2.24

b1 = coef(fit)[1]
```

4. Repeat until convergence

```
niters = 100
beta = matrix(0, niters, 3) # save results
colnames(beta) = c('b0', 'b1', 'b2')
b0 = b1 = b2 = 0            # initialize

for(i in 2:niters) {

  #-- Fix b1
  r = y - b1*x1
  fit = lm(r~x2)
  beta[i, c(1,3)] = coef(fit)
  b2 = coef(fit)[2]

  #-- Fix b2
  r = y - b2*x2
  fit = lm(r~x1)
  beta[i, c(1,2)] = coef(fit)
  b1 = coef(fit)[2]

  #-- Check for convergence
  if( sum(abs(beta[i,] - beta[i-1,])) < 1E-5){
    beta = beta[1:i,]
    break
  }
}
```

```
beta %>% as_tibble() %>%
  mutate(iter = row_number()) %>%
  pivot_longer(cols=-iter, names_to="coef") %>%
  ggplot(aes(iter, value, color=coef)) +
  geom_point() +  geom_line() +
  labs(y="coefficient") + theme_light(base_size=16)
```