Smoothing Splines

Group #8 - Led Zeppelin

Joaquin Baquerizo, Bekalu Debelu, Leo DiPerna, Minji Kim, Daniel Sabanov

Introduction

Basically what Smoothing Splines does is it overfits the data by having a function $g(x) = (y_i - g_{xi})^2$, but then we add a "smoothing" parameter which is basically a penalization term.

- Penalization parameter: $\lambda \int g^{"}(t)^2 dt$
- The notation $g^{"}(t)$ indicates the second derivative of the function g.
- The first derivative of $g(t)$ measures the slope of a function at t, and the second derivative corresponds to the amount y which the slope is changing.
- Hence, broadly speaking, the second derivative is a measure of its *roughness*: it is large in absolute value if g(t) is very wiggly near t, and it is close to zero otherwise
- The second derivative of a straight line is zero; note that a line is perfectly smooth

Introduction

Examples of smoothing splines can be found all over, but I found them to be most prevalent in time series data.

- Severity trend for insurance losses typically estimated using loglinear regression of average loss amounts on historical period, which results in a single trend estimate for a block of years.
- If the trend can be assumed to be constant over time, that would be sufficient, but what happens if it is not?
- Ok, so lets use the actual year-to-year changes to produce an inflation trend index.
- The problem with this is that the year-to-year changes can be highly volatile and produce unusual patterns if used in pricing.
- This is when smoothing splines come in for the rescue, since it provides an easy tool or a compromise between a single trend for all years and different trend for each year.
- Of course this can be exemplified to a number of cases such as stocks, interest rates, and in our case sports data.

Introduction

- When $\lambda = 0$, then the penalty term ($\lambda \int g^{''}(t)^2 dt$) will have no effect, and so the function g will be very jumpy and will exactly interpolate the training observations.
- When $\lambda = \infty$, g will be perfectly smooth - it will just be a straight line that passes as closely as possible to the training points.
- In fact, g would just be the linear least squares line, since the loss function (*penalty function*) amounts to minimizing the residual sum of squares.
- For an intermediate value of λ we see that g will approximate the training observations but will be somewhat smooth.
- So, we see that λ controls the Bias-Variance trade off in Smoothing Spline

About Data

The data was collected using R library(MASS). It is a simulated motorcycle accident. Using to test crash helemts, it measures head acceleration in a simulated motorcycle accident.

```
library(MASS)
library(ggplot2)
data("mcycle")
head(mcycle)
```

```
  times accel
1   2.4   0.0
2   2.6  -1.3
3   3.2  -2.7
4   3.6   0.0
5   4.0  -2.7
6   6.2  -2.7
```

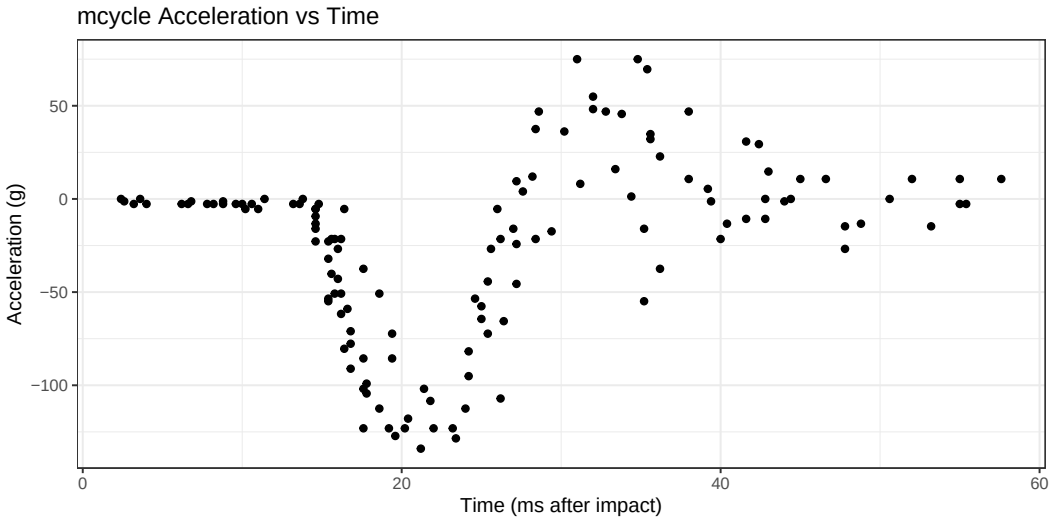
Comparison of Smoothing Spline to LOESS

```
library(stats)
library(rbenchmark)
players <- read.csv("data/NBA_players.csv")

benchmark("Smoothing Spline" = {
  ss.fit.1 <- smooth.spline(players$MP_PER_G, players$PF_PER_G)},

          "Loess" = {loess.fit <-loess(PF_PER_G ~ MP_PER_G, data = players)},
  replications = 100,
  columns = c("test", "replications", "elapsed","relative"))
```

```
              test replications elapsed relative
2            Loess          100    2.13     2.63
1 Smoothing Spline          100    0.81     1.00
```

This benchmark test shows the two functions runtime performance when fitting the same data 100 times. From the results we can see that smoothing spline is indeed the faster of the two methods with Loess being about three times slower.

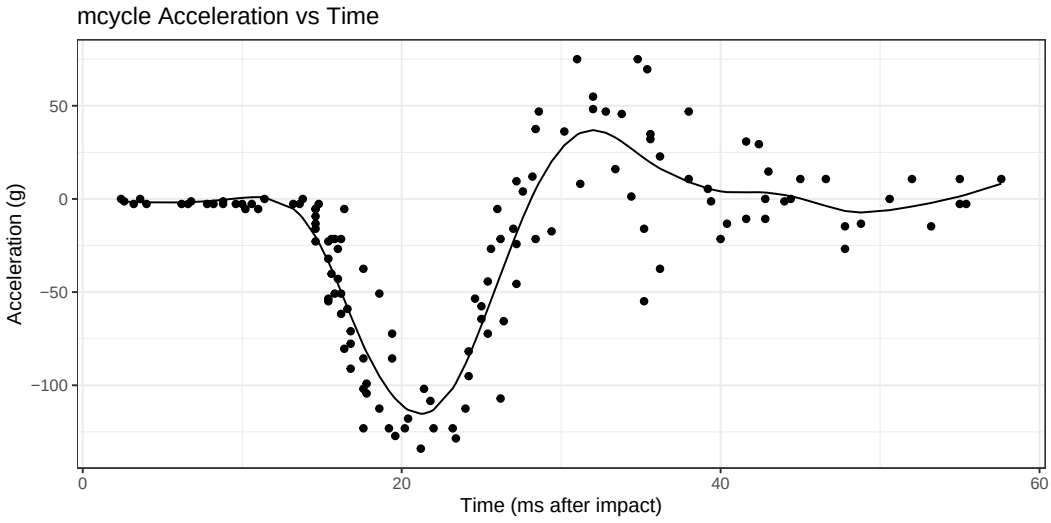
Plot


mcycle Acceleration vs Time
Acceleration (g)
50
0
−50
−100
0
20
40
60
Time (ms after impact)

Creating a Smoothing Spline

```
library(stats)
mfit.1 <- smooth.spline(mcycle$times, mcycle$accel)
```

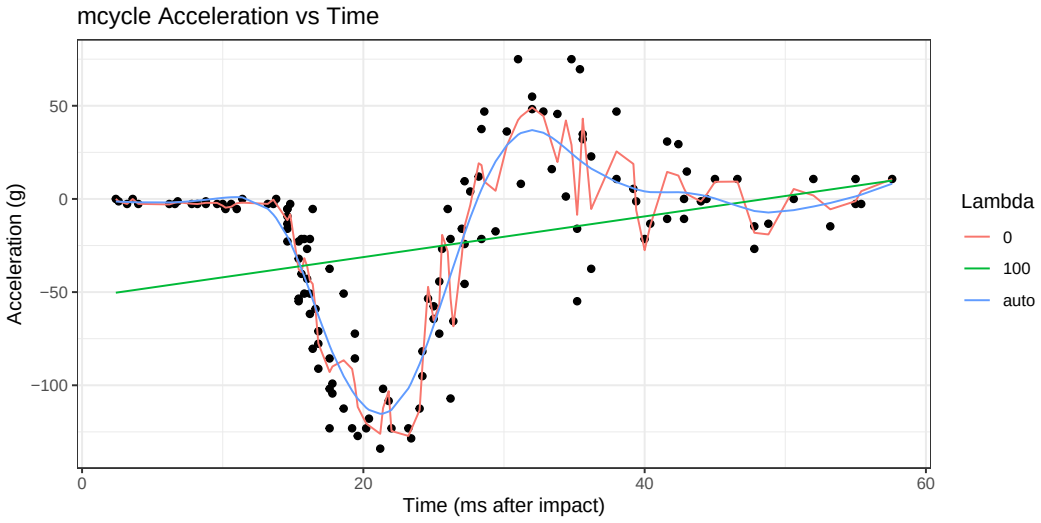
Smoothing Spline Plot



Changing Lambda

```
mfit.2 <- smooth.spline(mcycle$times, mcycle$accel, lambda = 0)

mfit.3 <- smooth.spline(mcycle$times, mcycle$accel, lambda = 100)
```

Comparing Lambdas



Best Fit Smoothing Spline

```
mfit.1$lambda
```

```
[1] 0.00011075
```

Lambda of this data is 0.00011075. This lambda is the best fit smoothing spline for this data. As lambda $= 0$, the smoothing spline curve is very noisy and overfitting. As lambda $= 100$, the smoothing spline curve is similar to the simple linear regression and underfitting.

Mathematical Background

A **spline** is a piecewise polynomial with pieces defined by a sequence of knots

$$\xi_1 < \xi_2 < ... < \xi_k$$

such that the pieces join smoothly at the knots.

The **knots** are the boundaries for each piecewise polynomial function that the spline is composed of. At each knot (other than the first and last), the polynomial function changes.

Penalized Sum of Squares

The goal of simple linear regression is to choose $\hat{\beta}$ such that $y_i = \hat{f}(x_i) = \hat{\beta}_0 + \hat{\beta}_1 x_i$ minimizes

$$\text{SSE} = \sum_{i=1}^{n} \left(y_i - \hat{f}(x_i)\right)^2$$

Similarly, when we fit a smoothing spline, we want to minimize the sum of squares error, but we also want to add a **smoothing penalty** to ensure a smooth fit. This penalty has the form

$$\text{P} = \lambda \int \hat{f}''(x)^2 dx$$

λ is a **tuning parameter** that can be adjusted to change the severity of the smoothing penalty.

Penalized Sum of Squares

This leads us to the **penalized sum of squares**, which is what we are trying to minimize when we fit a smoothing spline. It has the form

$$\text{PSS} = \text{SSE} + \text{P} = \sum_{i=1}^{n} \left(y_i - \hat{f}(x_i)\right)^2 + \lambda \int \hat{f}''(x)^2 dx$$

As $\lambda \to 0$, the smoothing penalty gets very small and the resulting curve becomes very noisy, following every detail in the data (**overfitting**).

As $\lambda \to \infty$, the smoothing penalty gets very large and the resulting curve looks like something you would get from simple linear regression (**underfitting**).

Minimizing the Penalized Sum of Squares

It turns out that the smoothing penalty can be written in the quadratic form as

$$\lambda \int \hat{f}''(x)^2 dx = \lambda \mu^\top K \mu$$

where

$$\mu = \begin{bmatrix} \hat{f}(x_1) \\ \vdots \\ \hat{f}(x_n) \end{bmatrix} \qquad K = \Delta^\top W^{-1} \Delta$$

Δ is an $n \times (n-2)$ matrix where
$\Delta_{i,i} = \frac{1}{h_i}$, $\Delta_{i,i+1} = -\frac{1}{h_i} - \frac{1}{h_i+1}$, $\Delta_{i,i+2} = \frac{1}{h_i+1}$

W is an $(n-2) \times (n-2)$ symmetric tridiagonal matrix where
$W_{i-1,i} = W_{i,i-1} = \frac{h_i}{6}$, $W_{i,i} = \frac{h_i + h_{i+1}}{3}$

$h_i = \xi_{i+1} - \xi_i$, the distance between consecutive knots

Minimizing the Penalized Sum of Squares

Now we can write the penalized sum of squares using matrices and vectors as

$$\text{PSS} = (y - \mu)^\top (y - \mu) + \lambda \mu^\top K \mu$$

We can take the partial derivative with respect to μ and set it equal to 0:

$$\frac{\partial \text{PSS}}{\partial \mu} = -2(y - \mu) + 2\lambda K \mu = 0$$

Then we can solve for μ:

$$\begin{aligned} \lambda K \mu &= y - \mu \\ \mu + \lambda K \mu &= y \\ (I + \lambda K)\mu &= y \\ \mu &= (I + \lambda K)^{-1} y \end{aligned}$$

We can use this formula to calculate the fitted values for smoothing splines.

Data Background

The data was collected using web scraping sourced from Sports Reference. The website includes data collected on different players and sports. Our data in particular was collected on basketball players from NBA. The data is accurate up-to the spring semester of 2022.

Link: https://drive.google.com/file/d/1Lx8WHBQITk9UgTlnnaD_WQ40KH-l-MNT

Dictionary

Column Name	Definition
Name	Player Name
G	Games Played
GS	Games Started
MP	Minutes Played
FG	Field Goals
FGA	Field Goals Attempted
FG2	2 Point Field Goals
FG2A	2 Point Field Goals Attempted
FG3	3 Point Field Goals
FG3A	3 Point Field Goal Attempted
FT	Field Throws
FTA	Field Throws Attempted
ORB	Offensive Rebounds
DRB	Defensive Rebounds

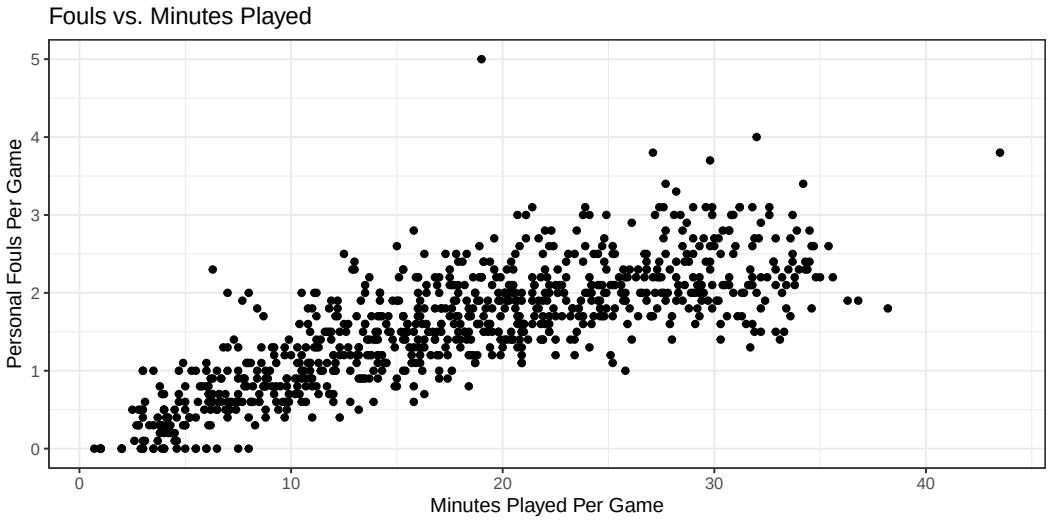
Dictionary

Column Name	Definition
TRB	Total Rebounds
AST	Assists Per Game
STL	Steals
BLK	Blocks
TOV	Turnovers
PF	Personal Fouls
PTS	Points
EFG_PCT	Effective Field Goal Percentage
X_PER_G	X Per Game
X_PCT	X Percentage

What Are We Focusing On?

As you can see there are a lot of variables in this data set and many different analyses can be performed. For now we would like to build a model to predict the number of fouls based on the time each player spends playing.

First Look


Fouls vs. Minutes Played
Personal Fouls Per Game
Minutes Played Per Game
0
1
2
3
4
5
0
10
20
30
40

Doing the Work by Hand

If we were doing it "by hand" it would look like this:

We first assign a lambda

```
lambda <- 10
```

Let's work with x and y for convenience.

```
x <- players$MP_PER_G
y <- players$PF_PER_G
```

Preparation

We need to remove duplicates to avoid creating a singularity.

First order all the points by x and initialize all our counting values.

```
o <- order(x)
x <- x[o]
y <- y[o]
c <- 0
x.unique <- rep(NA, length(x))
y.unique <- rep(NA, length(x))
prev <- x[1]
prev.count <- 1
prev.sum <- y[1]
```

Preparation

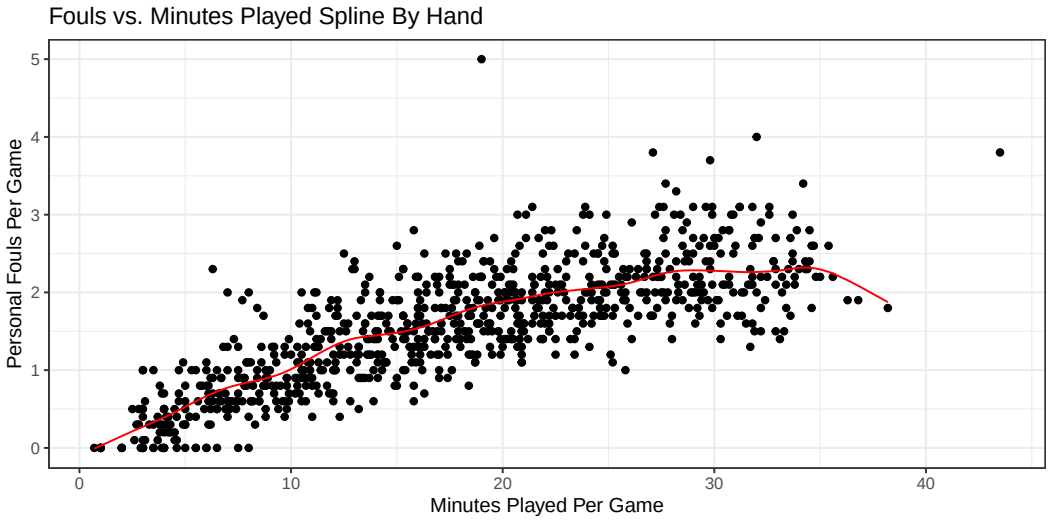
Average the y's for each repeating x.

```
for (i in 2:length(x)) {
  if (prev == x[i]) {
    prev.count <- prev.count + 1
    prev.sum <- prev.sum + y[i]
  } else {
    c <- c + 1
    x.unique[c] <- prev
    y.unique[c] <- prev.sum / prev.count
    prev <- x[i]
    prev.count <- 1
    prev.sum <- y[i]
  }
}
x <- x.unique[1:c]
y <- y.unique[1:c]
n <- length(x)
```

Performing the Calculation

```
# gets distance between each consecutive element of x
h = diff(x)

# create tridiagonal delta matrix
delta = matrix(0, n - 2, n)
for (i in 1:(n - 2)) {
  delta[i, i] <- 1 / h[i]
  delta[i, i + 1] <- 0 - (1 / h[i]) - (1 / h[i + 1])
  delta[i, i + 2] <- 1 / h[i + 1]
}

# create tridiagonal W matrix
W = matrix(0, n - 2, n - 2)
for (i in 1:(n - 2)) {
  W[i, i] = (h[i] + h[i + 1]) / 3
  W[i - 1, i] = h[i] / 6
  W[i, i - 1] = h[i] / 6
}

# calculate fitted values
K = t(delta) %*% solve(W) %*% delta
mu = solve(diag(n) + lambda * K) %*% y
```

Result

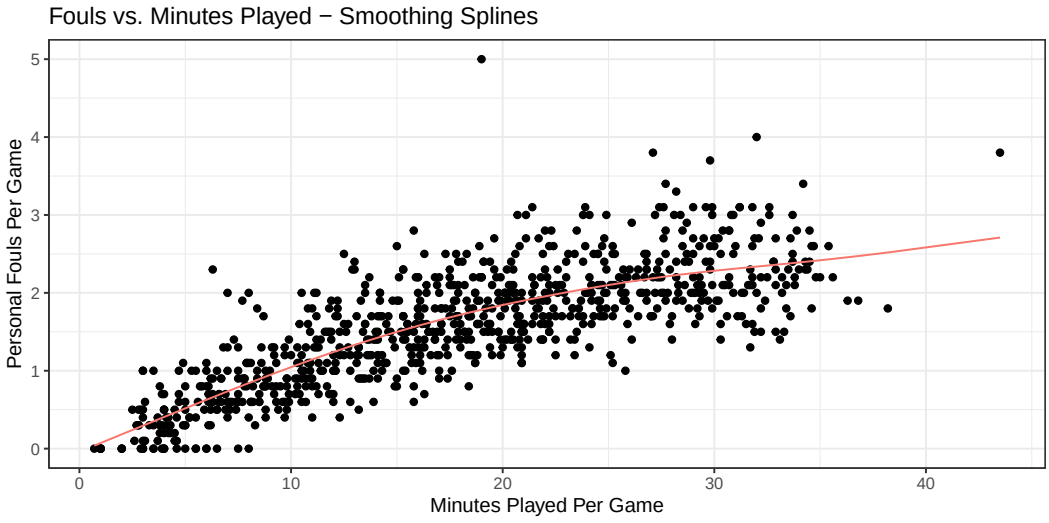

Fouls vs. Minutes Played Spline By Hand
Personal Fouls Per Game
Minutes Played Per Game
0
1
2
3
4
5
0
10
20
30
40

Creating a Model

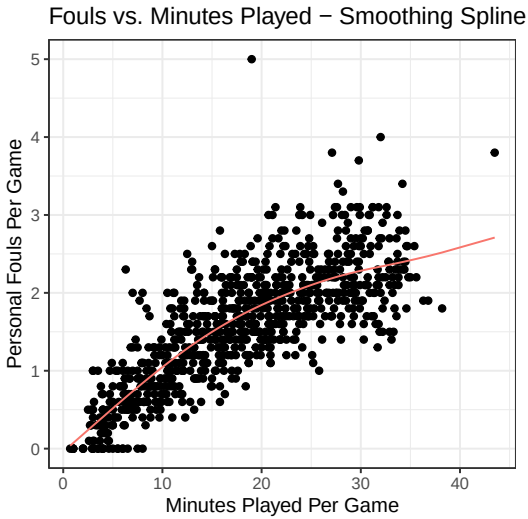
There is a built in function that performs smoothing splines in the 'stats' library. A nice thing about the built-in function is its ability to select the best lambda for us.

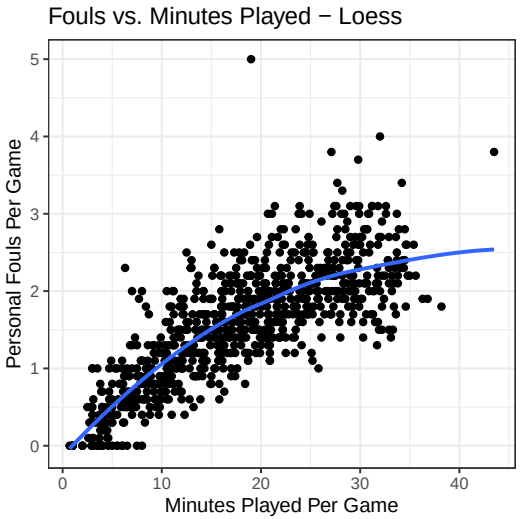
```
library(stats)
ss.fit.1 <-
  smooth.spline(players$MP_PER_G, players$PF_PER_G)
```

Smoothing Spline Plot



Comparing Smoothing Spline to Loess

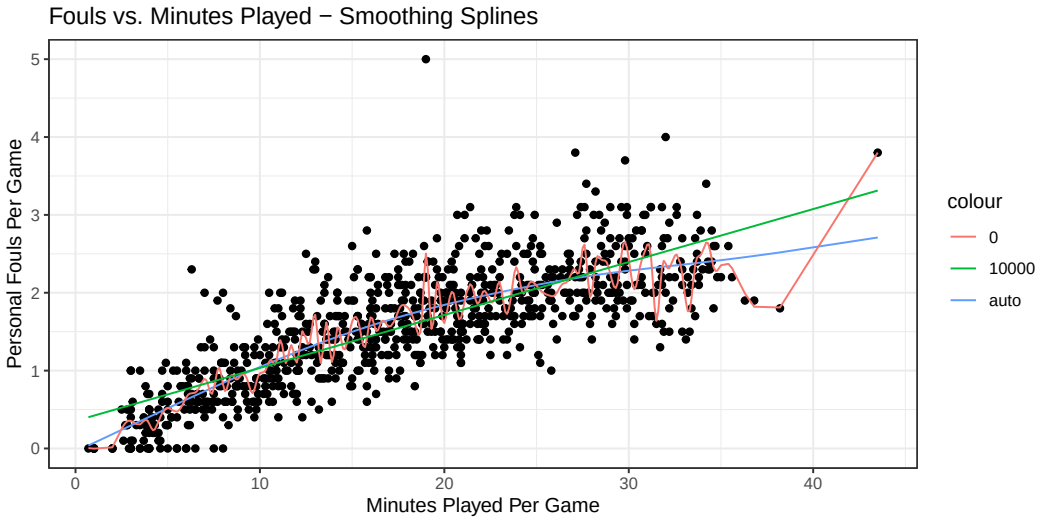




Changing our lambda

```
library(stats)
ss.fit.2 <-
  smooth.spline(players$MP_PER_G,
                players$PF_PER_G,
                lambda=0)

ss.fit.3 <-
  smooth.spline(players$MP_PER_G,
                players$PF_PER_G,
                lambda=10000)
```

Changing our lambda



References

https://grodri.github.io/demography/smoothing.pdf

https://static1.squarespace.com/static/5ff2adbe3fe4fe33db902812/t/6062a083acbfe82c7195b27d/1617076404560/ISLR%2BSeventh%2BPrinting.pdf#page=284

https://www.stat.cmu.edu/~ryantibs/advmethods/notes/smoothspline.pdf

http://users.stat.umn.edu/~helwig/notes/smooth-spline-notes.html

https://bookdown.org/tpinto_home/Beyond-Linearity/smoothing-splines.html

https://www.sports-reference.com/