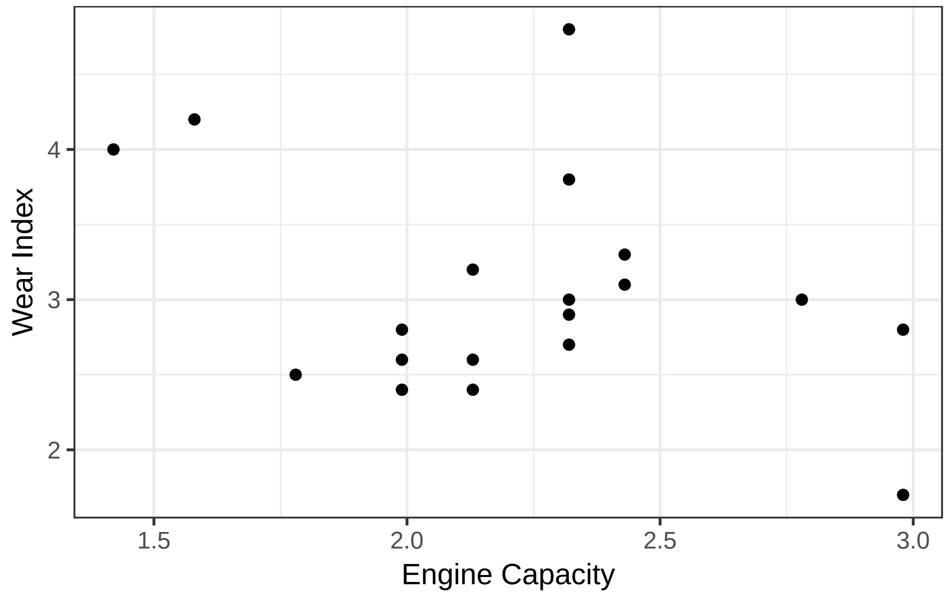
Chapter 4: Introducing GAMs

4.2 Univariate Smoothing

```r
library(gamair)
library(ggplot2)
library(tidyr)
suppressPackageStartupMessages(library(dplyr))
data(engine)
head(engine)
```

```
##   size wear
## 1 2.78  3.0
## 2 2.43  3.1
## 3 2.32  4.8
## 4 2.43  3.3
## 5 2.98  2.8
## 6 2.32  2.9
```

```r
ggplot(aes(x = size, y = wear), data = engine) +
  geom_point() +
  theme_bw() +
  xlab("Engine Capacity") +
  ylab("Wear Index")
```



We are now going to calculate the piecewise linear basis, $b_j(x)$, which as far as I can tell is just a linear interpolation between the two knots that surround a point.

First we are going to define the full function that takes in an array of x values and a sequence of knots and return the model matrix for the piecewise linear model. (Note this is the opposite order of how the book works and I've changed some variable names to help with readability.)

```r
tf.X <- function(x, xj){
  ## Tent function basis matrix given data x
  ## and knot sequence xj
  num_knots <- length(xj)
  num_data <- length(x)
  model_matrix <- matrix(NA, nrow = num_data, ncol = num_knots)
  for(tent in seq_len(num_knots)){
    model_matrix[, tent] <- tf(x, xj, tent)
  }
  return(model_matrix)
}
```

Now we need to define the helper function `tf` that will generate the tent functions, $b_j(x)$, from set defined by knots `xj`.

```
tf <- function(x, xj, j){
  dj <- xj * 0
  dj[j] <- 1
  approx(xj, dj, x)$y ## Return a list of points which linearly interpolate given data points
}
```

Let's work through this line by line

```
## Generate Knots
sj <- seq(min(engine$size), max(engine$size), length = 6)
print(sj)
```

```
## [1] 1.420 1.732 2.044 2.356 2.668 2.980
```

```
x <- engine$size
xj <- sj
num_knots <- length(xj)
num_data <- length(x)
model_matrix <- matrix(NA, nrow = num_data, ncol = num_knots)
tent <- 2
model_matrix[, tent] <- tf(x, xj, tent)
model_matrix[, tent]
```

```
##  [1] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [8] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.1730769
## [15] 0.1730769 0.1730769 0.8461538 0.5128205 0.0000000
```

Lets look at what actually goes on in the `tf` function.

```
dj <- xj * 0
dj[tent] <- 1
cbind(x, tent_function = approx(xj, dj, x)$y)
```

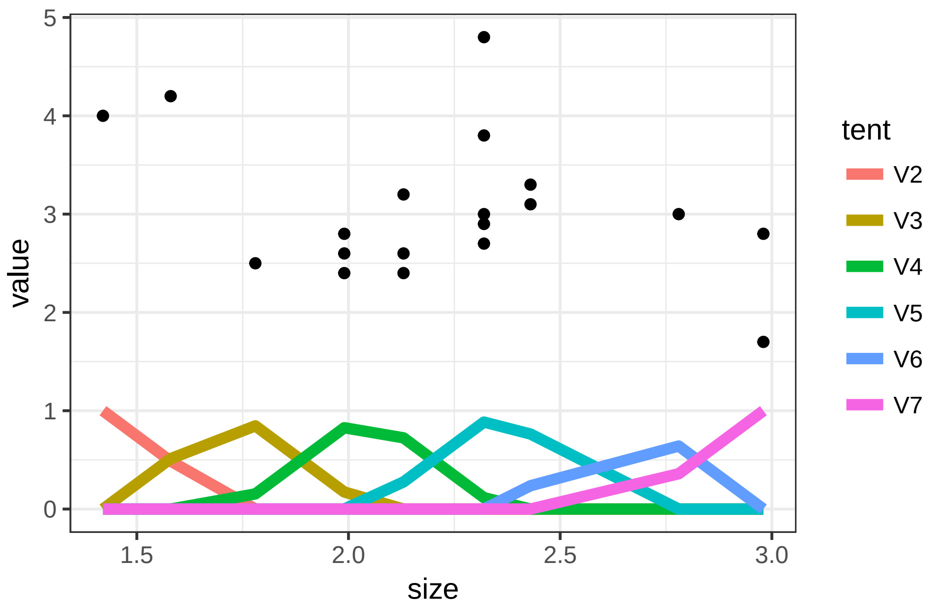
```
##          x tent_function
##  [1,] 2.78     0.0000000
##  [2,] 2.43     0.0000000
##  [3,] 2.32     0.0000000
##  [4,] 2.43     0.0000000
##  [5,] 2.98     0.0000000
##  [6,] 2.32     0.0000000
##  [7,] 2.32     0.0000000
##  [8,] 2.32     0.0000000
##  [9,] 2.32     0.0000000
## [10,] 2.13     0.0000000
## [11,] 2.13     0.0000000
## [12,] 2.13      0.0000000
## [13,] 2.98      0.0000000
## [14,] 1.99      0.1730769
## [15,] 1.99      0.1730769
## [16,] 1.99      0.1730769
## [17,] 1.78      0.8461538
## [18,] 1.58      0.5128205
## [19,] 1.42      0.0000000
```

So the tent function for the 2nd knot is 0 until the data get between the first (1.42) and third (2.044) knot and then just rises and falls as they get closer to the 2nd knot. Let's get them all now.

```r
X <- tf.X(engine$size, sj)
head(X)
```

```
##      [,1] [,2]      [,3]      [,4]      [,5]      [,6]
## [1,]    0    0 0.0000000 0.0000000 0.6410256 0.3589744
## [2,]    0    0 0.0000000 0.7628205 0.2371795 0.0000000
## [3,]    0    0 0.1153846 0.8846154 0.0000000 0.0000000
## [4,]    0    0 0.0000000 0.7628205 0.2371795 0.0000000
## [5,]    0    0 0.0000000 0.0000000 0.0000000 1.0000000
## [6,]    0    0 0.1153846 0.8846154 0.0000000 0.0000000
```

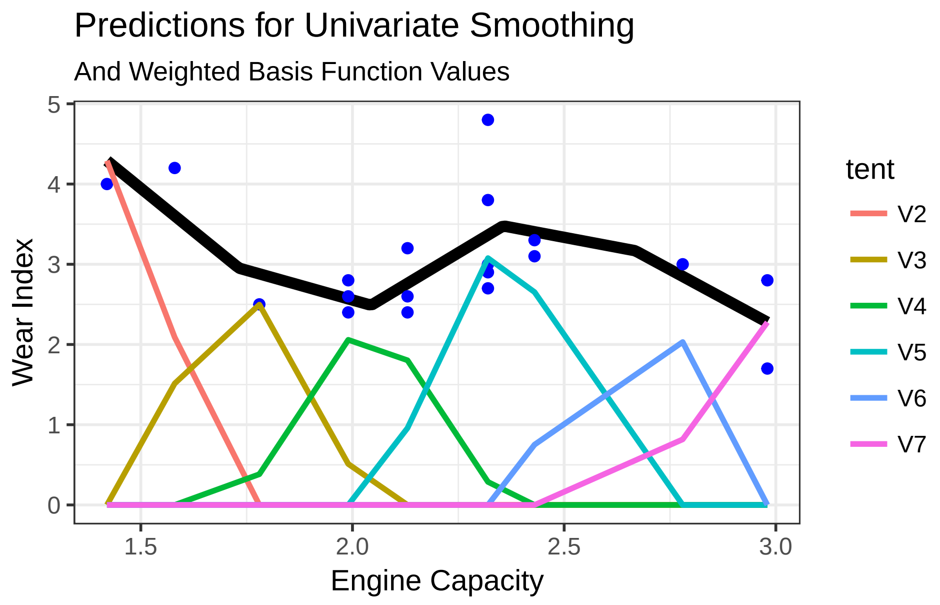
```r
data.frame(cbind(size = engine$size, X)) %>%
  gather(tent, value, -size) %>%
  ggplot(aes(x = size, y = value)) +
  geom_line(aes(color = tent), size = 2) +
  geom_point(aes(x = size, y = wear), data = engine) +
  theme_bw()
```



And now that we have our tent functions we can put them into a model to use them to predict wear.

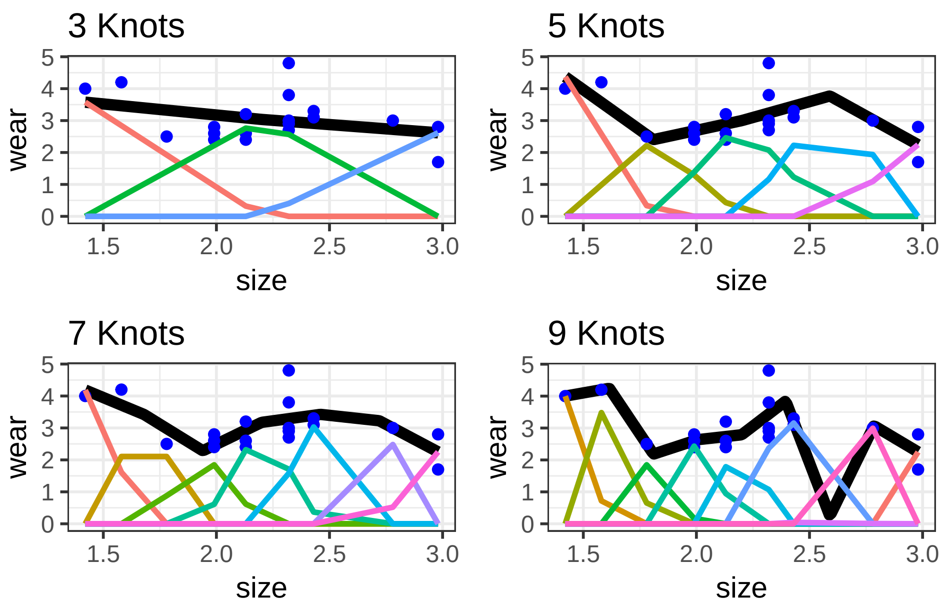
```r
b <- lm(engine$wear ~ X - 1)
bs <- coef(b)
X_weighted <- X
for(i in seq_len(ncol(X))) X_weighted[, i] <- X[, i] * bs[i]
pred_data <- seq(min(engine$size), max(engine$size), length = 200)
```

```
pred_matrix <- tf.X(pred_data, sj)
pred_wear <- drop(pred_matrix %*% coef(b))
data_frame(size = pred_data, wear = pred_wear) %>%
  ggplot(aes(x = size, y = wear)) +
  geom_line(size = 2) +
  geom_point(aes(x = size, y = wear), data = engine, color = "blue") +
  geom_line(aes(x = size, y = value, color = tent), size = 1, data = data.frame(cbind(size = engine$siz
  theme_bw() +
  xlab("Engine Capacity") +
  ylab("Wear Index") +
  ggtitle("Predictions for Univariate Smoothing", "And Weighted Basis Function Values")
```



The number of basis functions (6) was basically arbitrary, what if we re-ran the same code but now with different number of knots?

```
p1 <- fit_piecewise_linear_smooth(num_knots = 3)
p2 <- fit_piecewise_linear_smooth(num_knots = 5)
p3 <- fit_piecewise_linear_smooth(num_knots = 7)
p4 <- fit_piecewise_linear_smooth(num_knots = 9)
modeler::multiplot(p1, p3, p2, p4, cols = 2)
```



So as we increase the knots the function is able to fit the data better but also starts to experience some weird behavior and is most likely over-fitting.

Controlling Smoothness by Penalizing Wiggliness

For basic linear models we are attempting to minimize the fit according to mean square error

$$\|\mathbf{y} - \mathbf{X}\beta\|^2$$

But we can add in a penalty term to our optimization function that penalizes large coefficient values. For our tent function splines we get

$$\beta^{\mathbf{T}}\mathbf{S}\beta$$

where S is

$$\mathbf{S} = \mathbf{D}^{\mathbf{T}}\mathbf{D}$$

We add this by simply appending a penalized diagonal matrix to our data matrix.

$$\|\mathbf{y} - \mathbf{X}\beta\|^2 + \lambda\beta^{\mathbf{T}}\mathbf{S}\beta = \| \begin{bmatrix}\mathbf{y} \\ 0\end{bmatrix} - \begin{bmatrix}\mathbf{X} \\ \sqrt{\lambda}\mathbf{D}\end{bmatrix} \beta\|^2$$

So for each β we are basically saying that it should be pulled towards zero.

Let's start with $\lambda = 2$ and see how that changes things

```
sj <- seq(min(engine$size), max(engine$size), length = 20)
X <- tf.X(engine$size, sj)
D <- diff(diag(length(sj)), differences = 2)
D[1:6, 1:6]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1   -2    1    0    0    0
## [2,]    0    1   -2    1    0    0
## [3,]    0    0    1   -2    1    0
## [4,]    0    0    0    1   -2    1
## [5,]    0    0    0    0    1   -2
## [6,]    0    0    0    0    0    1
```

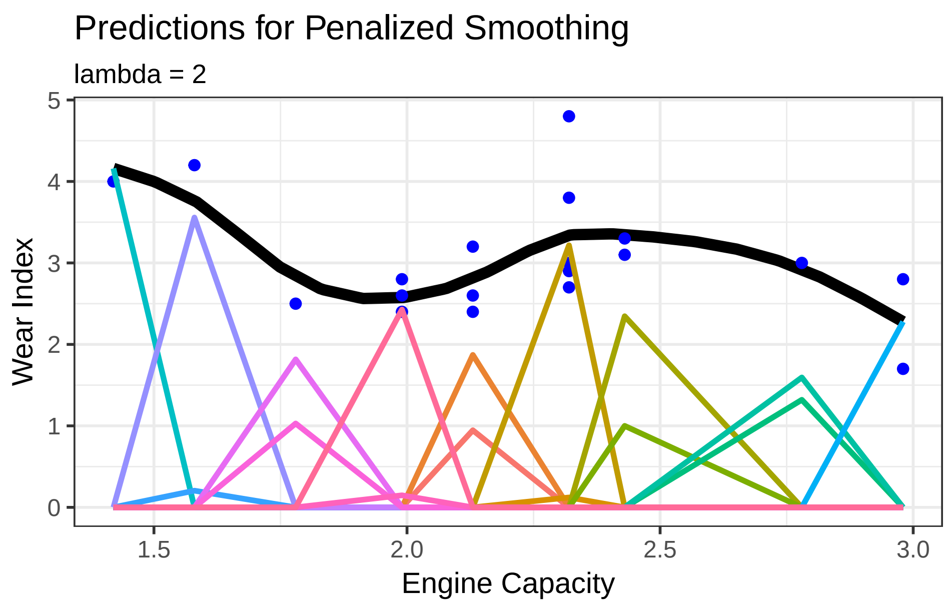
```
X_aug <- rbind(X, sqrt(2) * D)
y_aug <- c(engine$wear, rep(0, nrow(D)))
dim(X_aug)
```

```
## [1] 37 20
```

```
length(y_aug)
```

```
## [1] 37
```

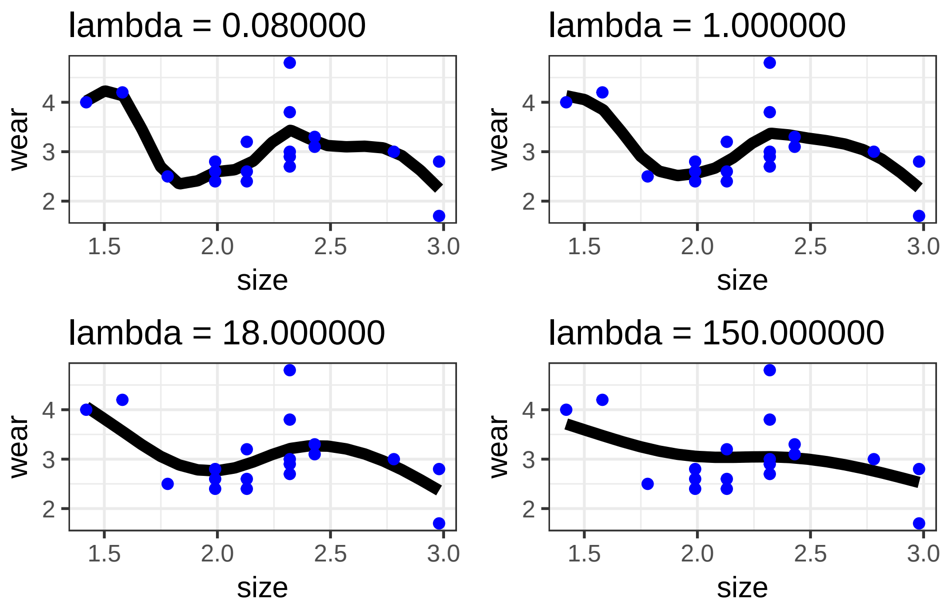
```
penalized_fit <- lm(y_aug ~ X_aug - 1)

bs <- coef(penalized_fit)
X_weighted <- X_aug
for(i in seq_len(ncol(X_aug))) X_weighted[, i] <- X_aug[, i] * bs[i]

pred_matrix <- tf.X(pred_data, sj)
pred_wear <- drop(pred_matrix %*% coef(penalized_fit))
data_frame(size = pred_data, wear = pred_wear) %>%
  ggplot(aes(x = size, y = wear)) +
  geom_line(size = 2) +
  geom_point(aes(x = size, y = wear), data = engine, color = "blue") +
  geom_line(aes(x = size, y = value, color = tent), size = 1, data = data.frame(cbind(size = engine$siz
  theme_bw() +
  xlab("Engine Capacity") +
  ylab("Wear Index") +
  ggtitle("Predictions for Penalized Smoothing", "lambda = 2") +
  scale_color_discrete(guide = F)
```



Well that is certianly more smooth than the original functions. And now we can throw this all into a function and visualize some different lambda values:

```r
prs.fit <- function(y, x, xj, sp){
  X <- tf.X(x, xj)
  D <- diff(diag(length(xj)), differences = 2)
  X <- rbind(X, sqrt(sp) * D)
  y <- c(y, rep(0, nrow(D)))
  lm(y ~ X - 1)
}
```

```r
p1 <- prs.fit_plot(lambda = .08)
p2 <- prs.fit_plot(lambda = 1)
p3 <- prs.fit_plot(lambda = 18)
p4 <- prs.fit_plot(lambda = 150)
modeler::multiplot(p1, p3, p2, p4, cols = 2)
```



The natrual question is how do we know which value of lambda generalizes the best? We can calculate the *generalized cross validation score* and find out.

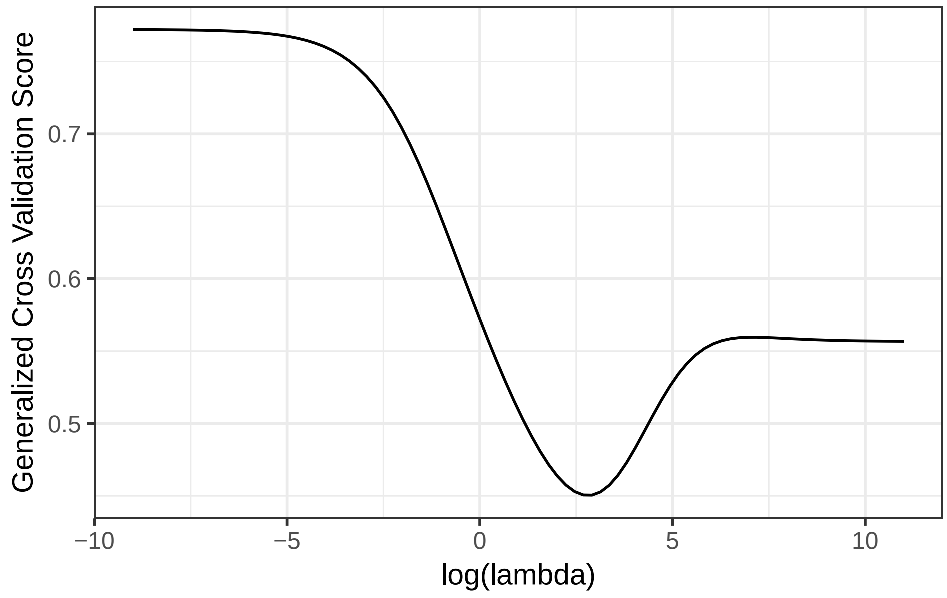
$$V_g = \frac{n * \sum_{i=1}^{n}(y_i - \hat{f}_i)^2}{[n - tr(A)]^2}$$

Where A is the hat matrix.

Lets find that for a bunch of lambda values

```r
rho <- seq(-9, 11, length = 90)
n <- nrow(engine)
V <- rep(NA, 90)
for(i in seq_along(V)){
  b <- prs.fit(y = engine$wear, x = engine$size, xj = sj, sp = exp(rho[i]))
  trF <- sum(influence(b)$hat[1:n])
  rss <- sum((engine$wear - fitted(b)[1:n])^2)
  V[i] <- n * rss / (n - trF)^2
}
```

```r
ggplot(aes(x = r, y = v), data = data_frame(r = rho, v = V)) +
  geom_line() +
  xlab("log(lambda)") +
  ylab("Generalized Cross Validation Score") +
  theme_bw()
```



So the best value of lambda, by generalized cross validation score, is 18.36

Bayesian Approach

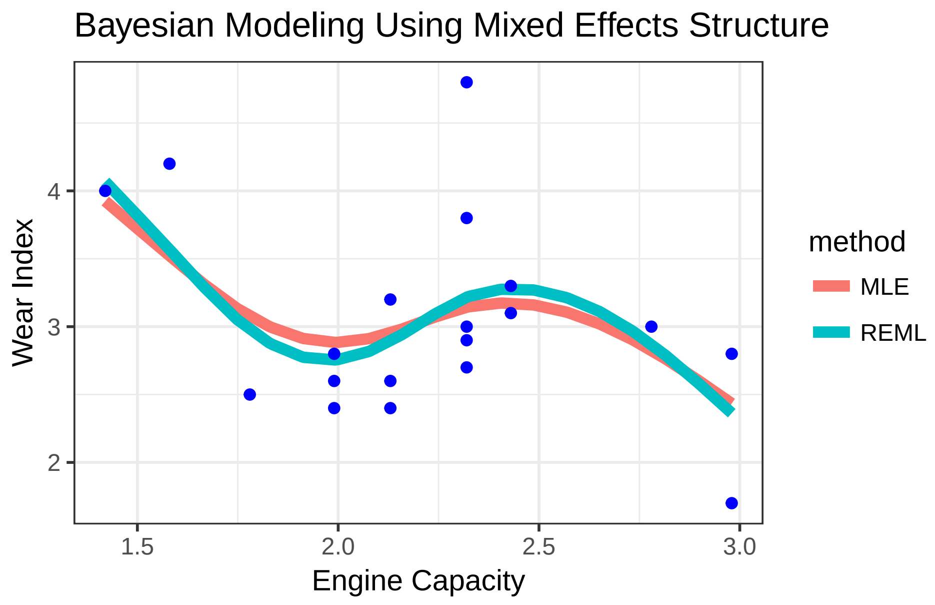
Basically instead of explicitly minimizing the size of the coefficients we set an equivalent prior on the β's and fit the model using the mixed effects model specification we used earlier.

```r
X0 <- tf.X(engine$size, sj)
D <- rbind(0, 0, diff(diag(20), differences = 2))
diag(D) <- 1
X <- t(backsolve(t(D), t(X0)))
Z <- X[, -c(1, 2)]
X <- X[, 1:2]
m <- optim(c(0, 0), llm, method = "BFGS", X = X, Z = Z, y = engine$wear)
b <- attr(llm(m$par, X, Z, engine$wear), "b")

X_pred <- t(backsolve(t(D), t(pred_matrix)))
pred_wear <- drop(X_pred %*% b)

library(nlme)
```

```
##
## Attaching package: 'nlme'

## The following object is masked from 'package:dplyr':
##
##     collapse
```

```r
g <- factor(rep(1, nrow(X)))
m <- lme(wear ~ X - 1, random = list(g = pdIdent(~ Z - 1)), data = engine)
```

```
pred_wear2 <- drop(X_pred %*% as.numeric(coef(m)))

data_frame(size = pred_data, MLE = pred_wear, REML = pred_wear2) %>%
  gather(method, predictions, -size) %>%
  ggplot(aes(x = size, y = predictions)) +
  geom_line(aes(color = method), size = 2) +
  geom_point(aes(x = size, y = wear), data = engine, color = "blue") +
  theme_bw() +
  xlab("Engine Capacity") +
  ylab("Wear Index") +
  ggtitle("Bayesian Modeling Using Mixed Effects Structure")
```


Bayesian Modeling Using Mixed Effects Structure
Wear Index
Engine Capacity
method
MLE
REML