Chapter 5: Splines

5.1 Smoothing Splines

Natural Cubic Splines

If we order our points in increasing value a natural cubic spline is a function made up of sections of cubic polynomials linking each successive pair of points. Of all the functions that are continous and interpolate the points the "smoothest" minimizes $\int_{x_1}^{x_n} f''(x)^2 dx$.

Cubic Smoothing Splines

Because most data is noisy and you don't have a full grasp of the data-generating mechanism you usually want to smooth the data rather than interpolate the points. So instead of fixing $g(x_i) = y_i$ we set them as parameters and attempt to minimize

$$\sum_{i=1}^{n}(y_i - g(x_i))^2 + \lambda \int g''(x)dx$$

with λ as the tuning parameter. We call $g(x)$ a *smoothing spline*.

While these are ideal smoothers (smoothing function that minimizes error) they have as many free parameters as there are data so can be problematic to estimate. Also because we are smoothing the function anyway we almost certainly won't need all n parameters.

5.2 Penalized Regression Splines

A Regression spline constructs a spline basis and then uses that basis to model the original data set.

5.3 Some One-Dimensional Smoothers

Cubic Splines

Already talked about these. If I get adventurous I'll try to fit one manually

From `mgcv` these can be called with `s(x, bs = "cr")`

Cyclic splines

These match the outermost knots so that the smooth function is equal there. Think of modeling smooth effects throughout the year, you wouldn't want there to be a discontinuity at the end of the year.

B-splines

A B-Spline (Basis Spline) are only non-zero between m + 3 adjacent knots (m + 1 is the order of the basis, so for cubic splines m = 2 and a B-spline would be non-zero for the nearest 5 knots). This makes them stable to compute. We can define them recursively by saying any m order spline is a weighted sum of lower order splines where the weights are defined by how close the x-values are to the series of knots. Wikipedia says:

(the first) ramps from zero to one as x goes from t_{i} to t_{i+k} and (the second) ramps from one to zero as x goes from t_{i+1} to t_{i+k+1}.

The final stage of recursion is just a binary indicating which knot span x is in.

So if we want a cubic B-spline with 6 knots we actually need to define 10 knots; 6 for the locations, 2 for the ends, and 2 for the cubic order.

```r
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(tidyr)
x <- 1:100 + rnorm(100)
k <- seq(0, 100, length.out = 10)

print(x)
```

```
##   [1]   1.841809   0.548938   2.349713   3.060230   3.817778   3.866378
##   [7]   6.162802   6.921074   8.509211  12.428942  11.356201  11.787058
##  [13]  13.570552  15.379946  15.946199  15.312689  18.362850  18.010886
##  [19]  17.994502  19.819417  20.480503  22.193271  21.876937  23.632351
##  [25]  24.944974  27.117899  27.280770  27.220376  28.566551  30.213022
##  [31]  31.427558  32.113449  33.433684  33.305177  34.664336  36.144619
##  [37]  34.255406  36.318085  40.300961  41.625635  41.727272  41.974533
##  [43]  42.722973  43.990880  44.551220  45.126460  45.813850  48.895778
##  [49]  48.620139  51.419232  52.414247  49.836873  54.172521  54.436788
##  [55]  57.140466  55.825733  55.679719  57.622955  59.938207  59.202995
##  [61]  61.920262  60.602770  63.250287  65.306476  67.069224  68.397588
##  [67]  67.288112  69.290198  66.322857  70.790852  70.791870  72.453601
##  [73]  71.873498  74.618931  75.081807  76.406003  78.245467  79.525852
##  [79]  79.104829  81.807645  80.899814  79.927038  82.746069  84.642821
##  [85]  83.414837  84.814427  87.587903  86.368012  90.214689  90.931328
##  [91]  90.902636  92.726540  93.615678  94.486305  93.009983  96.136112
##  [97]  95.968311  98.310290 100.353845 101.515123
```

```r
print(k)
```

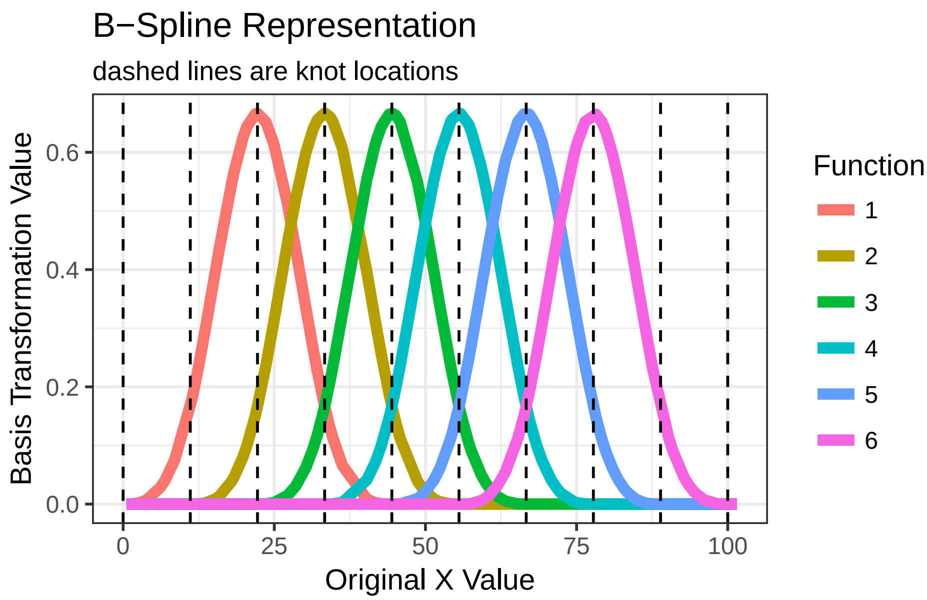
```
##  [1]   0.00000  11.11111  22.22222  33.33333  44.44444  55.55556  66.66667
##  [8]  77.77778  88.88889 100.00000
```

```r
bspline <- function(x, k, i, m = 2){
  ## Evaluate ith B-spline basis function of order m at the
  ## values in x, given knot locations in k
  if(m == -1){
    res <- as.numeric(x < k[i + 1] & x >= k[i])
  } else {
    z0 <- (x - k[i]) / (k[i + m + 2] - k[i + 1])
    z1 <- (k[i + m + 2] - x) / (k[i + m + 2] - k[i + 1])
```

```
    res <- z0 * bspline(x, k, i, m - 1) + z1 * bspline(x, k, i + 1, m - 1)
  }
  return(res)
}


spline_basis <- vapply(1:6, function(i, x_ = x, k_ = k) bspline(x = x_, k = k_, i = i), numeric(100))
spline_basis_df <- data.frame(spline_basis, stringsAsFactors = F)
names(spline_basis_df) <- as.character(seq_len(ncol(spline_basis)))
spline_basis_df$X <- x

spline_basis_df %>%
  gather(Function, Value, -X) %>%
  ggplot(aes(x = X, y = Value, color = Function)) +
  geom_line(size = 2) +
  geom_vline(aes(xintercept = loc), data = data_frame(loc = k), linetype = "dashed") +
  ggtitle("B-Spline Representation", "dashed lines are knot locations") +
  theme_bw() +
  xlab("Original X Value") +
  ylab("Basis Transformation Value")
```



So as you can see the outside 2 knots on both sides don't actual have full coverage, which is why we need to put knot locations outside the range of our data. We would then use this new basis as the variables in our regression model. We would then use these transformed values as inputs for our regression model.

```
knitr::kable(head(spline_basis_df), align = "c")
```

1	2	3	4	5	6	X
0.0007591	0	0	0	0	0	1.841809
0.0000201	0	0	0	0	0	0.548938
0.0015762	0	0	0	0	0	2.349713
0.0034821	0	0	0	0	0	3.060230
0.0067610	0	0	0	0	0	3.817778
0.0070225	0	0	0	0	0	3.866378

P-Splines

P-splines are low rank smoothers using a B-spline basis, but with a *difference penalty* applied to the parameters to control wiggliness. This means we are penalizing the squared differences between adjacent β_i values. We can represent this penalty as

$$\sum_{i=1}^{k-1} (\beta_{i+1} - \beta_i)^2 = \beta^{\mathbf{T}} \mathbf{P}^{\mathbf{T}} \mathbf{P} \beta$$

where P is just a diagonal difference matrix.

```
k <- 6
P <- diff(diag(k), differences = 1)
S <- t(P) %*% P
S
```

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

P-Splines do require evenly spaced knots, but other than that they are very flexible.

From `mgcv` these can be called with `s(x, bs = "ps", m = c(2, 3))` where m is the order for the basis and penalties, respectively.

Adaptive Smoothers

Sometimes we want the amount of smoothing to vary along with the x value.

From `mgcv` these can be called with `s(x, bs = "ad", m = c(2, 3))`