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]   2.8304466   0.6383422   3.4615389   6.1504733   3.4152325
##   [6]   6.2263610   6.5477073   9.0993343   8.0418226   9.3717109
##  [11]  10.9756387  10.1283545  12.2468605  12.6780563  16.3155565
##  [16]  15.5912416  16.2977648  16.0640794  18.9211216  18.7762839
##  [21]  21.5907149  23.2462060  23.4015386  24.1061468  27.0100930
##  [26]  26.2023187  27.0883877  26.6310496  30.2142395  30.8753991
##  [31]  32.0275432  31.2226954  33.7821897  33.0819412  34.6630579
##  [36]  37.3655724  36.1177560  38.4240626  39.7277989  39.6909017
##  [41]  41.9016775  40.8365454  43.6544693  45.1888068  45.0414979
##  [46]  45.8185781  47.4888263  48.8236175  48.8453556  50.0763945
##  [51]  51.0970741  51.8023765  51.5922700  54.7125974  55.7241879
##  [56]  54.7663641  55.4921620  56.6276969  60.3829692  58.2453033
##  [61]  60.5776437  61.7128858  62.2392796  64.9088296  67.8037814
##  [66]  66.1063845  67.0834966  68.8881896  69.6538935  69.9908926
##  [71]  72.0177223  71.9256885  72.9552996  74.8237473  73.5271670
##  [76]  77.2170905  78.0646884  77.2505063  79.1991229  80.2406643
##  [81]  80.8950586  82.7432086  82.3374130  85.5897813  82.5047999
##  [86]  85.7040926  87.4424702  86.8896348  89.4160749  89.9235826
##  [91]  90.2057115  90.8331037  92.0501123  92.3223260  95.6368462
##  [96]  96.4862053  97.3810915  99.1371482 100.5826330 101.4176250
```

```r
print(k)
```

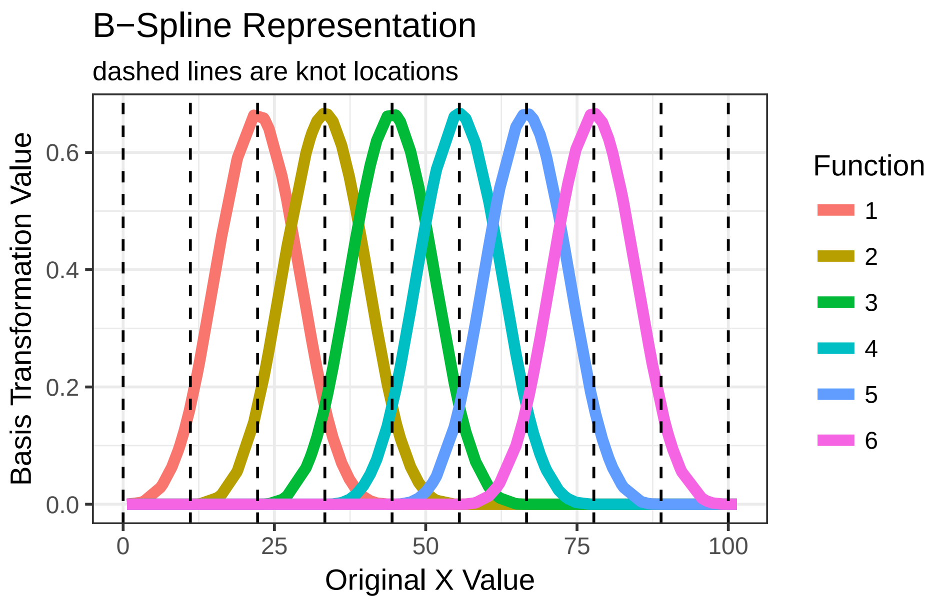
```
##  [1]   0.00000  11.11111  22.22222  33.33333  44.44444  55.55556  66.66667
##  [8]  77.77778  88.88889 100.00000
```

```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])
```

```r
  } 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.

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

1	2	3	4	5	6	X
0.0027551	0	0	0	0	0	2.8304466
0.0000316	0	0	0	0	0	0.6383422
0.0050395	0	0	0	0	0	3.4615389
0.0282684	0	0	0	0	0	6.1504733

1	2	3	4	5	6	X
0.0048399	0	0	0	0	0	3.4152325
0.0293278	0	0	0	0	0	6.2263610

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. We can do this by adding weights to the differences penalties and letting those weights vary smoothly with x.

From `mgcv` these can be called with `s(x, bs = "ad", k = 40, m = 4)`.

SCOP Splines

Can add shape constraints, aka monotonic functions.

5.4 Some Useful Smoother Theory

Identifiability Constraints

Since each smooth can't have it's own intercept so we force the smooth terms to sum to 0 over the observed values of x.

Effective Degrees of Freedom

The effective degrees of freedom from a smooth is

$$\sum_i (1 + \lambda D_{ii})^{-1}$$

where λ is the smoothing parameter and D is a diagonal matrix of eigen values of the $\mathbf{R^{-T}SR^{-1}}$. We can rewrite the degrees of freedom as the trace of $\mathbf{F}$ where

$$\mathbf{F} = (\mathbf{X^T X} + \lambda \mathbf{S})^{-1} \mathbf{X^T X}$$

So the degrees of freedom when there is no smoothing ($\lambda = 0$) is just the number of coefficients in the model and the number of zero eigenvalues of the penalty ($\lambda \to \infty$).

Null Space Penalties

Most smoothing penalties treat some null space of functions as completely smooth and therefore have zero penalty; A Cubic spline penalty ($\int f^{''}(x)^2 dx$) is zero for any straight line (2nd deriviate is 0, so there is nothing to sum to penalize). So when the penalty approaches infinity the smoother does not tend to 0 effect but tends to a straight line! So the penalty is not enough to remove a smooth term from the model altogether.

We can alleviate this by adding an extra penalty which only penalizes functions in the penalty null space (where a smoothing penatly has no effect).

The `select` argument in `gam` can be used to apply such penalties:

> If this is TRUE then gam can add an extra penalty to each term so that it can be penalized to zero. This means that the smoothing parameter estimation that is part of fitting can completely remove terms from the model. If the corresponding smoothing parameter is estimated as zero then the extra penalty has no effect.

5.5 Isotropic Smoothing

Isotropic smooths will produce identical predictions of the response variable under any rotations or reflections of covariates.

Thin Plate Regression Splines

So far each smoothing basis has the following characteristics:

1. You have to choose knot locations
2. Each basis can only incorporate one variable
3. It is not clear that they are better than any other basis

Thin Plate Splines

suppose we have a smooth function $g(x)$ we would like to estimate from n observations where $y_i = g(x_i) + \epsilon_i$. Thin plate splines estimate g by finding the function f that minimizes

$$\|\mathbf{y} - \mathbf{f}\|^2 + \lambda J_{md}(f)$$

where $J_{md}(f)$ is a penalty function measuring the "wiggliness" of f.

The functions making up the function space, $\mathbf{f}$, are linear independent polynomials spanning the space of the polynomials in $\Re^d$. Also the first couple functions (depending on the exact rank and dimension) span the space of functions for which $J_{md}(f)$ is 0, i.e., are in the null space of $J_{md}(f)$. For example if $m = d = 2$ then $\phi_1(x) = 1$, $\phi_2(x) = x_1$, and $\phi_3(x) = x_2$.

We do not have to define knots or select the basis functions for thin plate splines. Also we can use as many predictors as we like.

The problem is that these are very computationally costly; there are as many unkown parameters as data. So we want a low rank approximation of these splines

Thin Plate Regression Splines

Basically we want to truncate some of the wiggly components of the thin splate spline.

5.6 Tensor Product Smooth Interactions

Tensor products are scale invariant. A thin plate spline works similar to a flexible strip of and scales up to a flexible sheet. Tensor products just interlock multiple strips.

Tensor Product Bases

Tensor products, basically, set a smooth function of one covariate using a sequence of knots. Then we let each parameter of that smooth vary with z as well by defining them as a smooth function of z. The same tensor product would be found if we started with z instead of x.

These are scale invariate.

From `mgcv` these can be called with the `te` function, `te(x, y, z)`

ANOVA Decompositions of Smooths

Suppose we want to test a smooth interaction

$$f_1(x) + f_2(z) + f_3(x, z)$$

we can build tensor product interaction smooths with sum-to-zero constraints so that any main effects are removed.

In `mgcv` these can be called with the `ti` function.

5.7 Isotropy Versus Scale Invariance

Isotropic smooths are sensitive to linear rescaling of a single covariate. The reason is that the thin plate spline attempts to achieve the same smoothness per unit change in every covariate, so when a unit changes scale the function starts to fall apart. Tensor product smooths are not affected by any re-scaling.