Notes on Fixed Knot Regression with Cubic Splines

peter.t.swanson@gmail.com

June 4, 2017

1 Summary

In general, a **spline** is defined as a degree d piecewise polynomial whose function values and first $d-1$ derivatives agree at the points where they join (knots) [1, 2, 3, 4, 5]. Splines are used to fit non-linear relationships between X and Y and are:

1. Piecewise functions - divide the range of X into $K+1$ separate regions using knots ξ_j for $j = 1, \ldots, K$
2. Polynomial functions - within each of the $K+1$ regions, fit a degree d polynomial function
3. Constrained functions - constrain the function $f(X)$ to meet at each knot ξ_j and make the function smooth at the knots by requiring the first $d-1$ derivatives to be continuous

Cubic splines (degree $d = 3$) are often preferred because they are the lowest order spline where the human eye cannot identify the knot discontinuity. The splines described above are sometimes called **unrestricted cubic splines** to differentiate them from restricted or **natural cubic splines** which include the additional constraint that the function be linear before the first and after the last knots (called "'boundary knots") [1].

Splines have been used in agriculture, economics, pharmacokinetics, physics, meteorology, and nuclear materials research for over 30 years [6]. The use of cubic splines is also a common practice in default and prepayment modeling [7, 8, 9]. For survival analysis models, since the time dimension is typically non-linear, it is often modeled using cubic splines [10, 11, 12].

The methods described below are sometimes called **fixed knot regression splines** because we represent a fixed number of knots with a set of basis functions. These basis functions can be included as inputs to any generalized linear model (GLM) without violation of model assumptions. **Smoothing splines** are an alternative to fixed knot splines, but they cannot be included directly in a GLM [4]. Instead, smoothing splines must be fit with Generalized Additive Models (GAM) using backfitting algorithms [13]. A smoothing spline is a natural cubic spline with knots at every point x_i A tuning parameter λ is used to increase the smoothness of the function to avoid overfitting.

2 Motivation

These notes focus on simple linear regression with a single predictor X and a single response Y, however, what follows extends naturally to any GLM and to multiple predictors.

With linear regression, we fit the model

$$f(X) = X\beta = \beta_0 + \beta_1 X \tag{1}$$

where $f(X)$ is linear in the coefficient vector, β. This works well if Y is linear in X as in the left hand plot in Figure 1. This model works less well when Y is not linear in X as in the right plot in Figure 1. The plot in Figure 1 is an extreme example, however, in most cases $f(X)$ is nonlinear in X and linear regression is a sometimes necessary simplification [4]. This is especially true when the number of observations, n, is small or the number of predictors, p is large. If the relationship between X and Y is not linear, then the linear model in 1 will not be able to accurately capture that relationship.

```r
set.seed(8675309)
a = rnorm(100)
b = 1 + 2*a + rnorm(100)
fit = lm(b~a)
c = seq(5, 20, .1)
d = c + 0.2*c^2*(1+sin(c)) + rnorm(length(c))*20
fit2 = lm(d~c)
```

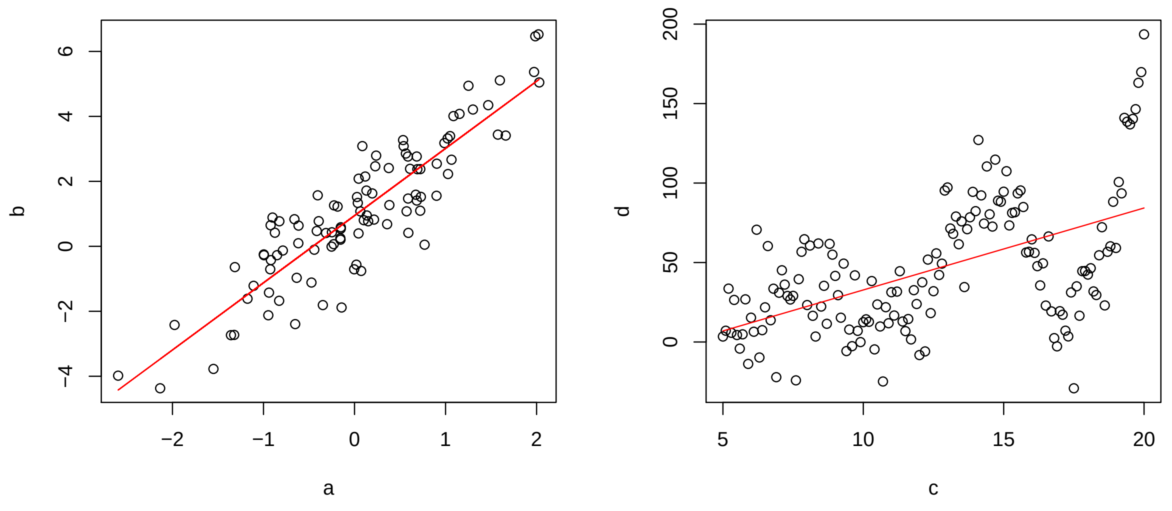


Figure 1: $Y = \beta_0 + \beta_1 X$ applied to a linear and a non-linear relationship

The important thing here is that the linear model in 1 imposes a linear relationship. It is linear in the coefficients, β, not necessarily in X. So if X does not have a linear relationship with Y, we would be violating a basic model assumption to use 1 to fit the relationship between X and Y.

In the case where Y is not linear in X, we have a few options to fit the observed data. There are more complicated non-linear models (ie GAM, MARS, Neural Networks, tree-based methods...), and it is sometimes possible to transform Y to make a model linear (ie $\log(Y)$)), however, in most cases, if we want to stick with standard generalized linear models, we have a few options:

1. Transform X ($\log(X)$, polynomials, interactions, ...)
2. Piecewise Constant (Step Functions of X)
3. Piecewise Polynomials (combination of 1 and 2)
4. Splines (piecewise polynomials with constraints)

2.1 Basis Functions

Options 1 and 2 are often done informally, however, all four options can be accomplished by extending the linear model using basis functions in X. For our purpose a **basis** is just a linear mapping (in a function space) of X to a set of basis functions $h_m(X), m = 1, \ldots, M$. Once this new set of basis inputs has been derived, we perform linear regression of Y onto the new set of inputs $h_m(X)$ instead of on the original variable X. So instead of fitting equation 1, we fit

$$f(X) = h_m(X)\beta = \beta_0 + \beta_1 h_1(X) + \cdots + \beta_m h_M(X) \tag{2}$$

The advantage of basis functions is that once they are established, we can use them to fit a standard linear model in the same way we would using a standard set of variables X [1, 5, 4]. We can do this because the linear model assumes $f(X)$ is linear in its coefficients, not in the original variables. So the final model will capture the linear effects of the basis functions but will be non-linear in the original feature space of X. A more formal definition of linear basis expansion can be found in [1]. Some commonly used basis functions include:

- $h_m(X) = X$ gives the original linear regression
- $h_m(X) = \log(X), \sqrt{X}$ gives some non-linear transformations of X
- $h_m(X) = X^m$, for $m = 0, 1, 2, \ldots, M$ gives a polynomial basis
- $h_m(X) = I(L_m \leq X < U_m)$ gives an indicator for X within a range. This is used in a piecewise constant basis representation

Other basis functions include wavelets, forier series, and regression splines. Note that these basis functions are fixed and known.

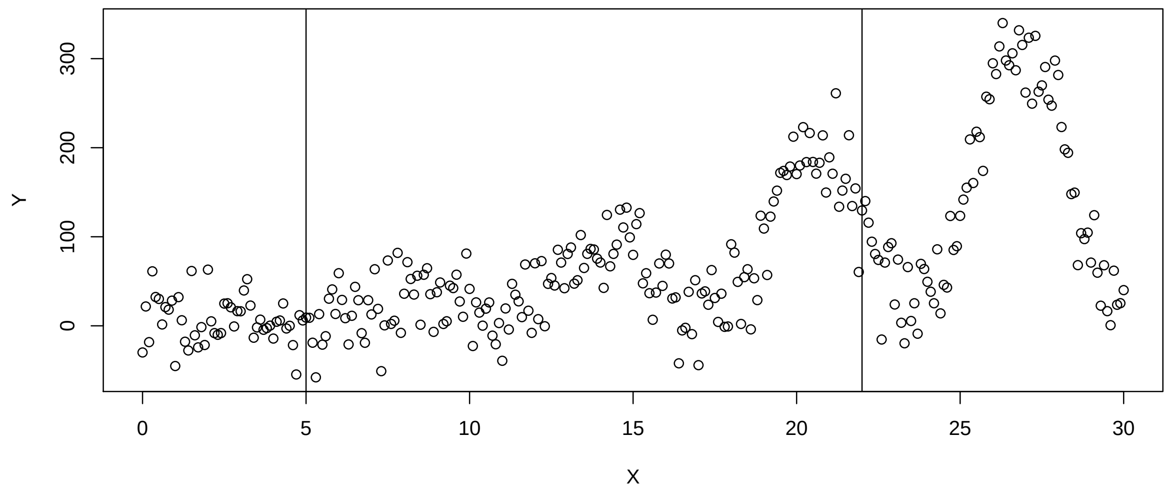
2.2 Truncated Power Series vs B-Spline

Bases can often be expressed in different ways. For example, a polynomial basis may be represented equally by an orthogonal polynomial basis. For splines, the two most popular options are truncated power series and B-spline bases. A B-spline basis is more computationally efficient and numerically stable than a Truncated Power basis [14, 1]. Unfortunately, B-splines do not allow for extrapolation beyond the boundary knots [5] and since B-splines depend on the observed values of x, care needs to be taken when applying functions in software packages [15]. On the other hand, an adjustment can be made to the truncated power basis to improve numerical stability and as long as modern statistical software is used (ie QR decomposition for matrix inversion), estimation is no longer a problem [5]. Finally, the truncated power series is much easier to understand, represent algebraically, and implement. The coefficient estimates will be different depending on the basis, but since they span the same space, the final model predictions will be the same. These notes stick to truncated power bases except to show how they are equivalent to B-splines for prediction.

3 Regression With Basis Functions

The next section demonstrates the basis function approach to fitting non-linear relationships. The code below simulates data from a non-linear model, x and y. We then take limited subset ($5 <= x <= 22$), so we can observe what happens when we extrapolate the model beyond the range of its training data.

```r
library(splines)
library(Hmisc)
rm(list=ls())
set.seed(8675309)

# simulate data (extended to show behavior in tails later)
xExt = seq(0, 30, .1)
yExt = xExt + 0.2*xExt^2*(1+sin(xExt)) + rnorm(length(xExt))*30

# select smaller range to model
limRange = (5<=xExt) & (xExt<=22)
x = xExt[limRange]
y = yExt[limRange]
```



3.1 Global Polynomials

A common method used to fit nonlinear data is to use polynomials. A degree d polynomial can be represented by

$$h_m(X) = X^m; m = 0, 1, 2, \ldots, M$$

concretely, a cubic polynomial basis can be represented as:

$$h_0(X) = 1, \quad h_1(X) = X, \quad h_2(X) = X^2, \quad h_3(X) = X^3 \tag{3}$$

and we can use this basis to build a model

$$\begin{aligned} f(X) &= \beta_0 h_0(X) + \beta_1 h_1(X) + \beta_2 h_2(X) + \beta_3 h_3(X) \\ &= \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 \end{aligned} \tag{4}$$

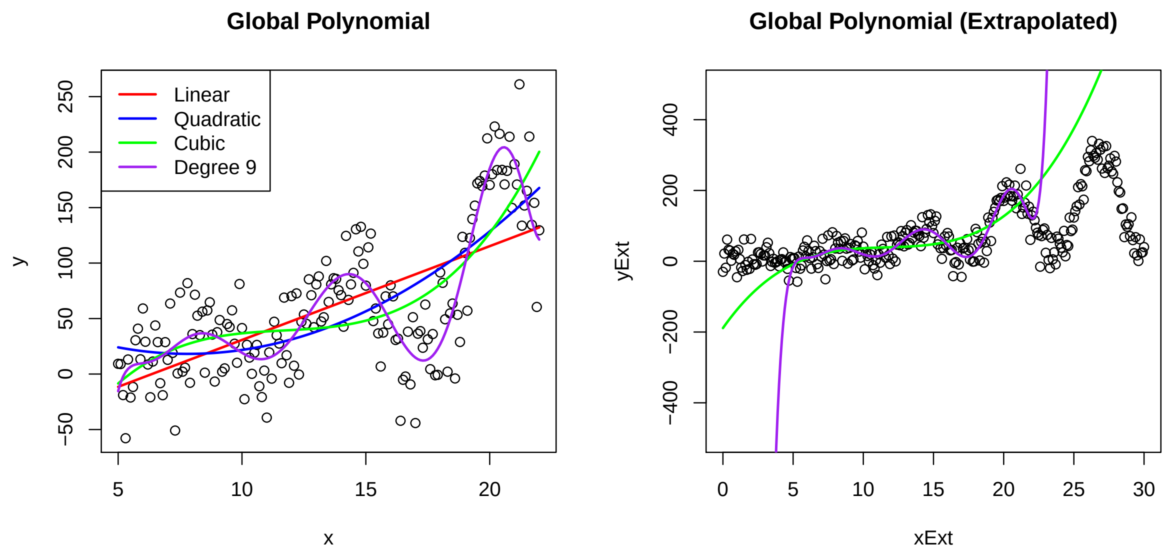
which is just standard polynomial regression. Its called a global polynomial here to differentiate it from local or piecewise polynomials used later. The code below fits orthogonal polynomials of degree 2, 3, and 9 below.

```r
fit1 = lm(y~x)                  # fit1 - linear
fit2 = lm(y~poly(x,2))          # fit2 - degree 2 global polynomial
fit3 = lm(y~poly(x,3))          # fit3 - degree 3 global polynomial
fit4 = lm(y~poly(x,9))          # fit4 - degree 9 global polynomial
```

Code below produces the plots of the polynomials. The plot on the left shows the fit in the training sample. We have to use a high-degree polynomial to fit this curve well. In general, we don't go beyond degree 3 or 4 because the curve can become too flexible and take on strange shapes [4]. Also keep in mind the bias-variance tradoff:

$$\uparrow \text{Flexibility} \quad = \quad \uparrow \text{Variance} \quad = \quad \downarrow \text{Bias} \tag{5}$$

It is possible to find an arbitrarily good fit to the training sample by increasing d. Of course, this will lead to overfitting and a poor estimate of the "true" model.



3.2 Piecewise Constant

Another approach would be to fit a piecewise constant model. This is equivalent to binning a continuous variable resulting in a categorical variable. We begin by specifying some knots ξ_j for $j = 1, \ldots, K$. Assume for now we have some a priori theory that suggests we place knots in this data at $X = 14$ and $X = 17$. Two knots breaks the domain into three regions which can be represented by three basis functions:

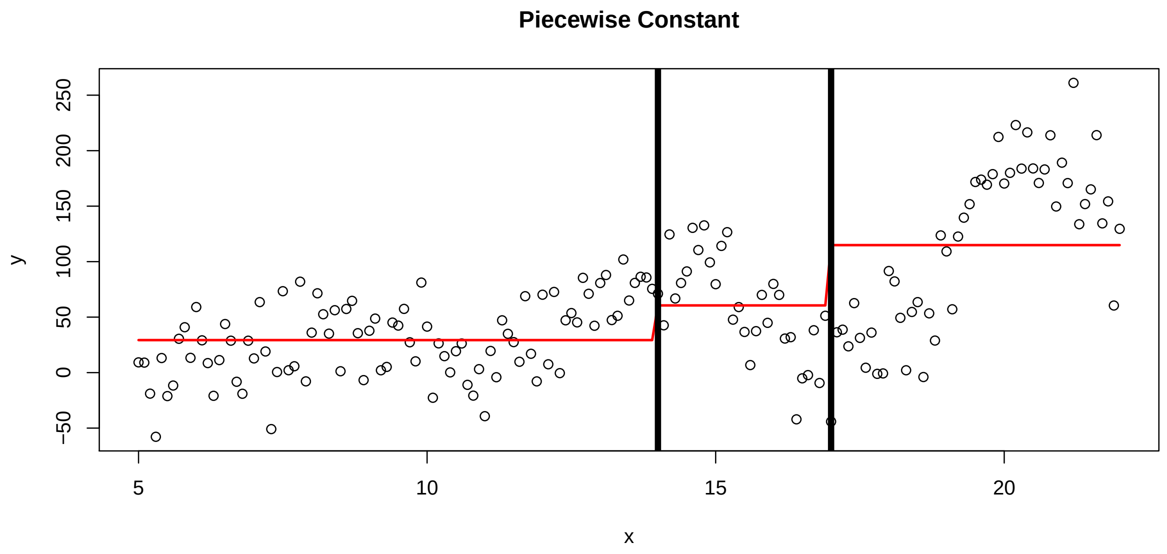
$$h_1(X) = I(X < \xi_1), \quad h_2(X) = I(\xi_1 \leq X < \xi_2), \quad h_3(X) = I(\xi_2 \leq X) \tag{6}$$

Fitting the model without an intercept

$$f(x) = \beta_1 h_1(X) + \beta_2 h_2(X) + \beta_3 h_3(X)$$

will give $\hat{\beta}_m = \bar{Y}_m$. The OLS coefficient estimates will be equal to the mean of y over each disjoin region. Equivalently, we could fit a model with an intercept and 2 of the 3 basis functions. This is a more traditional approach to fitting binned variables using dummy coding.

```
k = c(14,17)                          # knots

i1 = 1*(x<k[1])                       # indicators
i2 = 1*(k[1]<=x & x<k[2])
i3 = 1*(k[2]<=x)

fit5 = lm(y ~ 0 + i1 + i2 + i3)       # fit5 = piecewise constant
```



3.3 Piecewise Linear (+ constraints)

A natural extension of the piecewise constant model is to fit a linear term in each of the segments separated by the knots. This is accomplished by adding three new basis functions giving us the piecewise linear basis:

$$h_1(X) = I(X < \xi_1), \quad h_2(X) = I(\xi_1 \le X < \xi_2), \quad h_3(X) = I(\xi_2 \le X)$$
$$h_4(X) = I(X < \xi_1)X, \quad h_5(X) = I(\xi_1 \le X < \xi_2)X, \quad h_6(X) = I(\xi_2 \le X)X$$

```
fit6 = lm(y ~ 0 + i1 + i2 + i3 + I(i1*x) + I(i2*x) + I(i3*x))  # fit6 = piecewise linear
```

This can be seen in the left hand plot below. While an apparent improvement over the Piecewise Constant, we now have this strange situation where the prediction jumps at the knots. With the Piecewise Constant approach, there was nothing we could do, however, with the Piecewise Linear model, its possible to re-express the basis incorporating constraints. Here we want to constrain the function to meet at the knots. So we want a basis that will force

$$\lim_{X \to \xi+} f(x) = \lim_{X \to \xi-} f(x) = f(\xi)$$

and this can be accomplished by using truncated power series basis functions of the form

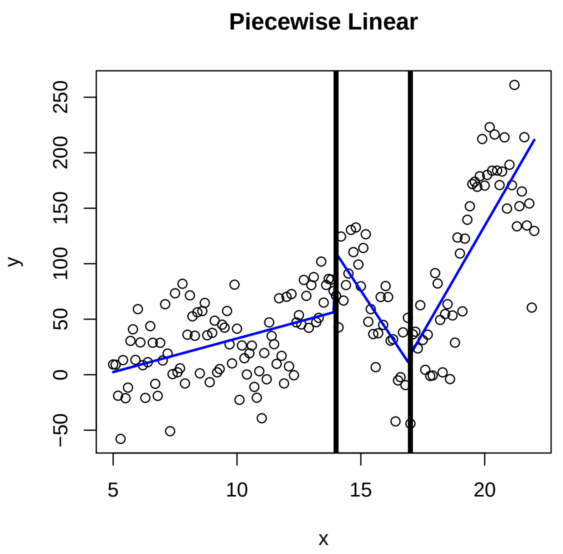
$$h_m(X, \xi) = (X - \xi)_+^d = \begin{cases} (X-\xi)_+^d & \text{if } \xi < X \\ 0 & \text{if } \xi \le X \end{cases}$$

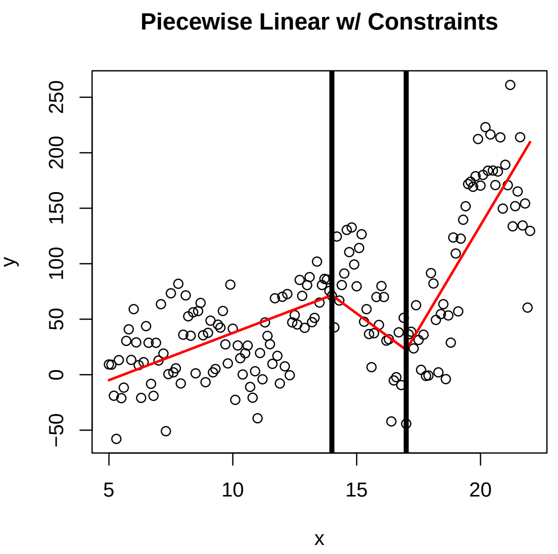
Note that a linear function is a degree $d = 1$ polynomial. So for a Piecewise Linear model with constraints with two knots we use

$$h_1(X) = 1, \quad h_2(X) = X, \quad h_3(X) = (X - \xi_1)_+, \quad h_4(X) = (X - \xi_2)_+ \tag{7}$$

Begin with a linear function of X and add another linear function of X at each knot with the requirement that each piece meet at the knots. This can be seen in the right hand plot below.

```
# use truncated power basis functions to constrain f(x) at knots
tpb1 = (x-k[1])*(k[1]<x)
tpb2 = (x-k[2])*(k[2]<x)

fit7 = lm(y ~ x + tpb1 + tpb2)       # fit7 = piecewise linear w/ constrained k
```





4 Cubic Spline Regression

Splines are generally defined as degree d piecewise polynomial functions where the first $d-1$ derivatives are constrained to be continuous [3, 2]. For a cubic spline this basically means we need constraints that ensure that the first and second derivatives are equal in the limits.

$$\lim_{x\to\xi-} f^{''}(x) = \lim_{x\to\xi+} f^{''}(x) = f(\xi)$$

4.1 Unrestricted Cubic Spline

The **unrestricted cubic spline** basis follows naturally from the constrained piecewise linear function in 7

$$h_1(X) = 1, \quad h_2(X) = X, \quad h_3(X) = X^2, \quad h_4(X) = X^3, \quad h_5(X) = (X-\xi_1)^3_+, \quad h_6(X) = (X-\xi_2)^3_+ \tag{8}$$

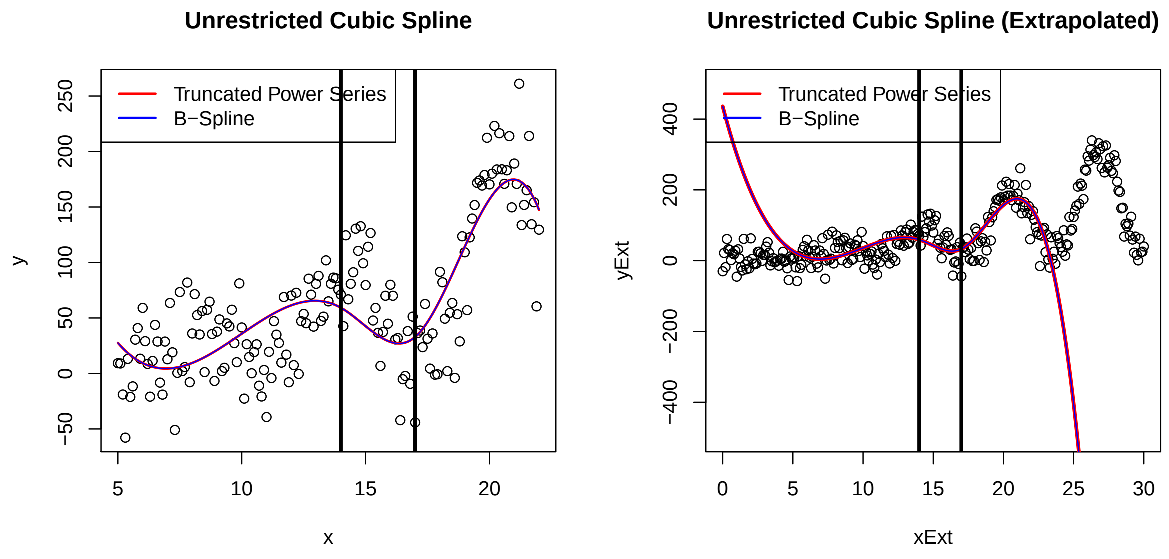
Cubic splines are said to be the lowest degree splines where the human eye cannot detect the discontinuities at the knots [1].

The code below fits the unrestricted cubic splines in two ways. `fit8` uses the truncated power basis described in 8, and `fit9` uses a B-spline basis representation. Note that both have the same number of parameters to estimate, but that the coefficients are vastly different. This isn't a problem, though, because the bases span the same space and this is evidenced by the fact that both the truncated power series and the B-spline basis yield the same predictions (see graphs below.)

```
ctpb1 = (k[1]<x)*(x-k[1])^3
ctpb2 = (k[2]<x)*(x-k[2])^3

fit8 = lm(y ~ poly(x,3) + ctpb1 + ctpb2)  # unrestricted cubic spline (truncated power basis)
summary(fit8)$coef

##                  Estimate  Std. Error   t value     Pr(>|t|)
## (Intercept)    -82.200924  23.0250242 -3.570069 4.677119e-04
## poly(x, 3)1 -1983.488473 409.8364136 -4.839708 2.968182e-06
## poly(x, 3)2 -1708.292288 318.7560423 -5.359247 2.776076e-07
## poly(x, 3)3  -694.193530 158.4053854 -4.382386 2.081823e-05
## ctpb1            3.253116   0.4993988  6.514064 8.463249e-10
## ctpb2           -6.016934   0.9378420 -6.415722 1.421777e-09

fit9 = lm(y~ bs(x, k=k))                      # unrestricted cubic spline (B-splines basis)
summary(fit9)$coef

##                  Estimate Std. Error   t value     Pr(>|t|)
## (Intercept)      27.56518   14.23959  1.935813 5.459997e-02
## bs(x, k = k)1 -77.92596   32.79455 -2.376186 1.863922e-02
## bs(x, k = k)2 118.22031   19.72509  5.993399 1.253194e-08
## bs(x, k = k)3 -68.84080   25.00718 -2.752841 6.569960e-03
## bs(x, k = k)4 213.97740   24.15416  8.858820 1.237323e-15
## bs(x, k = k)5 119.88316   23.24289  5.157842 7.088935e-07
```

This basis is "unrestricted" in the sense that it does not impose additional constraints after the first and last knots. This can be a limitation because just as polynomials can behave badly in the tails, a piecewise polynomial can also behave badly in the tails. This can be seen by the graph in the right below.

```
## Warning in bs(x, degree = 3L, knots = c(14, 17), Boundary.knots = c(5, 22:  some 'x' values beyond
boundary knots may cause ill-conditioned bases
```



4.1.1 Degrees of Freedom *df*

Note there are six basis functions corresponding to a model fit with six degrees of freedom. A piecewise cubic polynomial with two knots would have (4 parameters per region $(1, X, X^2, X^3)$) $\times$ (3 regions) $= 12df$. Cubic splines, however, place three constraints at each knot (continuity in $f, f^{'}$, and, $f^{''}$). This gives us (4 parameters per region) $\times$ (3 regions) $-$ (2 knots) $\times$ (3 constraints per knots) $= 6df$. So a spline uses less parameters and is less flexible than a piecewise polynomial.

4.2 Natural (Restricted) Cubic Splines

As seen above, the unrestricted cubic spline is poorly behaved in the tails. To address this, we can use an alternative basis for cubic splines which satisfies the "natural" boundary condition that the second derivative vanish at the first and last knot [16, 3] . In other words we fit a natural cubic spline basis which is linear before the first and after the last knots.

$$f(X) = \beta_0 + \beta_1 X + \sum_{j=1}^{K-2} \beta_{j+2} h_{j+1} \tag{9}$$

and for $j = 1, \ldots, K-2$,

$$h_{k+1} = (X - \xi_j)^3_+ - (X - \xi_{K-1})^3_+ (\xi_K - \xi_j)/(\xi_K - \xi_{K-1}) + (X - \xi_K)^3_+ (\xi_{K-1} - \xi_j)/(\xi_K - \xi_{K-1}) \tag{10}$$

For a more numerically stable basis, Harrell recommends dividing each term in 10 by

$$\tau = (\xi_K - \xi_1)^2 \tag{11}$$

This adjustment normalizes the nonlinear terms by scaling them to have units of X. After this adjustment, the truncated power basis will not provide any numerical issues assuming modern optimization software is used (ie QR decomposition for matrix inversion) [5].

Note that this is a truncated power basis, however, a B-spline basis could also be used to represent a natural cubic spline. The code below fits natural cubic splines in R. The first step is to assign **boundary knots**. Up to this point, we have been working with two knots, ξ_1 and ξ_2. We now add boundary knots near the beginning and the end of the data (at 5.1 and 19.9) . So now, we technically have four knots, and our former knots ξ_1 and ξ_2 are now ξ_2 and ξ_3 and are now called **internal knots**.

```
# add boundary knots
k = c(5.1, 10, 16, 19.9)
nk = length(k)
```

Next we fit the natural cubic spline in three ways. First, we manually define a truncated power basis in the code below. This method is general and can be used in SAS, Python, Matlab, or any other scientific computer language.

```
# MANUALLY CALCULATE NATURAL SPLINE BASIS (W/ EXPLICIT BOUNDARY KNOTS)
naturalSplineBasis = data.frame(x=x)
for (j in 1:(nk-2)){
z =  (k[j]<x)*(x-k[j])^3 - ((k[nk-1]<x)*(x-k[nk-1])^3)*((k[nk]-k[j])/(k[nk]-k[nk-1]))
+ ((x>k[nk])*(x-k[nk])^3)*((k[nk-1]-k[j])/(k[nk]-k[nk-1]))
z = z/(k[nk]-k[1])^2
naturalSplineBasis[j+1] = z
}
df = cbind(y,naturalSplineBasis)
```

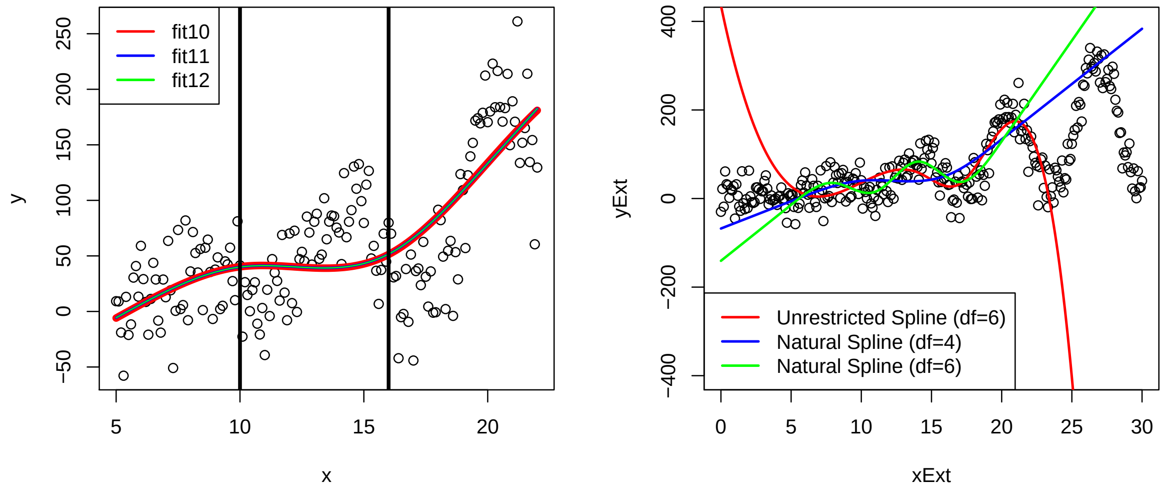
The manually calculated truncated power basis is used to estimate `fit10`. We also use the `rcspline.eval()` function to fit a truncated power basis in `fit11` and the `ns()` function to fit a B-Spline basis in `fit12`. As can be seen in the graphs below, all three bases yield the same prediction for $f(x)$.

```
# all 3 Natural Splines match
fit10 = lm(y ~ ., data=df)        # manual truncated power basis)
summary(fit10)$coef

##              Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) -67.73309  31.581842 -2.144685 0.0334239111
## x            12.35799   4.121662  2.998303 0.0031299243
## V2          -29.05193  10.970881 -2.648095 0.0088705321
## V3           94.39059  27.524249  3.429361 0.0007621962

fit11 = lm(y ~ rcspline.eval(x, knots=k, inclx=TRUE))  # truncated power basis
fit12 = lm(y ~ ns(x, knots=k[2:3], Boundary.knots = c(k[1], k[4])))  # B-Spline basis
```

In the right hand panel below, we extrapolate the natural (restricted) spline in `fit10` and compare it to the unrestricted spline from `fit9`. The blue line below shows the advantage of using a natural spline. While the unrestricted spline behaves badly in the tails, the natural spline is linear outside the boundary knots which generally gives some comfort when extrapolating beyond the range of the training dataset (of course, its always technically wrong to extrapolate but sometimes unavoidable).

The plots below also demonstrate another feature of natural splines. They are less flexible than unrestricted splines given the same number of knots. We can see from 10 that a natural cubic spline with $K = 4$ knots has 4 degrees of freedom. A natural spline with K knots (including boundary knots) will have K degrees of freedom.

In comparison the unrestricted spline with $K = 2$ knots has 6 degrees of freedom. This can actually be seen as an advantage for natural splines because we can use those extra degrees of freedom to better effect by including more interior knots [1]. In the plot below, we fit a final natural spline model using interior knots at $\xi = \{8, 11, 14, 17\}$ resulting in a natural cubic spline model with 6 df.



5 Knot Placement

In order to use fixed-knot splines, we need to know (1) how many knots to use, and (2) where to place them. Ideally there would be some theory to guide this process. For example, if we knew a function would change curvature at some point $X = a$, we could place a knot at a [17, 16]. This is sometimes known as **subjective knot placement** to differentiate it from automatic knot placement. However, in most cases pre-specification is not possible [5]. Fortunately, researchers have found that the placement of knots for restricted cubic spline models is not critical. Instead, the number of knots, K, has a much greater impact on the fit [18, 19]. Researchers generally recommend placing evenly spaced knots at fixed quantiles along the marginal distribution of X. This can help guarantee enough points in each interval and provides some protection against allowing outliers have too great of an effect on knot placement [5, 1, 4].

As to the number of knots, Stone suggests that natural cubic splines rarely require more than 5 knots and that when the sample size is "large" (ie > 100) and the response variable is uncensored that 5 knots is generally a good choice [18]. Increasing the number of knots increases the function's flexibility and with it the risk of overfitting. With this in mind, James, et al. suggest specifing a desired number of degrees of freedom, df, and using $df - 1$ equally spaced interior knots. In this way the bias/variance tradeoff is adjusted by treating df as a tuning parameter. Higher df results in greater flexibility. Standard cross-validation methods can be used to find an optimal df which is one option for an **automated knot selection** strategy [4]. Other authors recommend creating a larger number of basis functions and using backwards stepwise selection based on the AIC [1, 6, 20]. Potts proposed a BIC based stepwise selection method for censored survival models where the data is used to select an optimal significance level for entry and retention [21]. See Volinsky and Raferty for more details on this criteria [22].

5.1 Choice of Boundary Knots (Natural Splines)

For large, well behaved samples, Stone "tentatively" recommends placing boundary knots at the 5th smallest and 5th largest values [16]. In R, the `ns()` function from the `splines` package puts boundary knots at the smallest and largest values of the training data [23]. The `rcspline.eval()` function from the `Hmisc` package uses the following rules "For 3 knots, the outer quantiles used are 0.10 and 0.90. For 4-6 knots, the outer quantiles used are 0.05 and 0.95. For $K > 6$, the outer quantiles are 0.025 and 0.975" [24].

6 Extensions

Once a basis has been established for a variable, this basis replaces the original variable in the regression model. The basis can be combined with any other linear terms or with other splines. The spline equivalent of an interaction is given by a tensor product basis. See [1] for more details.

7 Note

This document was produced in RStudio using the knitr package [25] by `http://texblog.org`.

References

[1] Trevor J.. Hastie, Robert John Tibshirani, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction.* Springer, 2011.

[2] Patricia L Smith. "Splines as a useful and convenient statistical tool". In: *The American Statistician* 33.2 (1979), pp. 57–62.

[3] TF Devlin, BJ Weeks, et al. "Spline functions for logistic regression modeling". In: *Proceedings of the 11th Annual SAS Users Group International Conference.* Vol. 646. SAS Institute Inc., Cary, NC. 1986, p. 51.

[4] Gareth James et al. *An introduction to statistical learning.* Vol. 6. Springer, 2013.

[5] Frank Harrell. *Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis.* Springer, 2015.

[6] RL Eubank. "Approximate regression models and splines". In: *Communications in Statistics-Theory and Methods* 13.4 (1984), pp. 433–484.

[7] Patrick Bajari, Chenghuan Sean Chu, and Minjung Park. *An empirical model of subprime mortgage default from 2000 to 2007.* Tech. rep. National Bureau of Economic Research, 2008.

[8] Zhou X. Kane M. Chow M. Hu C. & Varrieur A. Dunsky R. M. *FHFA Mortgage Analytics Platform WhitePaper.* Federal Housing Finance Agency (FHFA), 2014.

[9] Robert M Dunsky and Thomas SY Ho. "Valuing fixed rate mortgage loans with default and prepayment options". In: *The Journal of Fixed Income* 16.4 (2007), p. 7.

[10] Paul D Allison. *Survival analysis using SAS: a practical guide.* SAS Institute, 2010.

[11] Harald Heinzl and Alexandra Kaider. "Gaining more flexibility in Cox proportional hazards regression models with cubic spline functions". In: *Computer methods and programs in biomedicine* 54.3 (1997), pp. 201–208.

[12] Mark J Rutherford, Michael J Crowther, and Paul C Lambert. "The use of restricted cubic splines to approximate complex hazard functions in the analysis of time-to-event data: a simulation study". In: *Journal of Statistical Computation and Simulation* 85.4 (2015), pp. 777–793.

[13] Trevor J Hastie and Robert J Tibshirani. *Generalized additive models.* Vol. 43. CRC Press, 1990.

[14] Lynn A Sleeper and David P Harrington. "Regression splines in the Cox model with application to covariate effects in liver disease". In: *Journal of the American Statistical Association* 85.412 (1990), pp. 941–949.

[15] William N Venables and Brian D Ripley. *Modern applied statistics with S-PLUS.* Springer Science & Business Media, 2013.

[16] Charles J Stone and Cha-Yong Koo. "Additive splines in statistics". In: *Proc Stat Comp Sect Am Statist Assoc* 27 (1985), pp. 45–48.

[17] Dale J Poirier. "Piecewise regression using cubic splines". In: *Journal of the American Statistical Association* 68.343 (1973), pp. 515–524.

[18] Charles J Stone. "[Generalized Additive Models]: Comment". In: *Statistical Science* 1.3 (1986), pp. 312–314.

[19] Sylvain Durrleman and Richard Simon. "Flexible regression models with cubic splines". In: *Statistics in medicine* 8.5 (1989), pp. 551–561.

[20] Patricia L Smith. "Curve fitting and modeling with splines using statistical variable selection techniques". In: (1982).

[21] William Potts. *Survival Data Mining: A Programming Approach.* SAS Institute, 2014.

[22] Chris T Volinsky and Adrian E Raftery. "Bayesian information criterion for censored survival models". In: *Biometrics* 56.1 (2000), pp. 256–262.

[23] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing. Vienna, Austria, 2015. URL: `https://www.R-project.org/`.

[24] Frank E Harrell Jr, with contributions from Charles Dupont, and many others. *Hmisc: Harrell Miscellaneous.* R package version 3.17-1. 2015. URL: `https://CRAN.R-project.org/package=Hmisc`.

[25] Yihui Xie. *knitr: A general-purpose package for dynamic report generation in R.* R package version 1.4.1. 2013. URL: `http://yihui.name/knitr/`.