Lecture 25—Wednesday, March 14, 2012

Topics

Local polynomial regression (continued)

We continue our discussion of local polynomial regression, also known as lowess. Lowess can provide a faithful local snapshot of the relationship between a response variable and a predictor. As we noted last time lowess is based on the following five ideas.

1. Binning
2. Local averaging
3. Kernel estimation
4. Local polynomial regression (in strictu)
5. Robust regression

So far we've discussed binning and local averaging.

- **Binning**. We use the predictor to construct a set of bins, a collection of intervals that form a partition of the x-axis. We average the values of the response variable of all observations assigned to a bin. The mean is then used as the predicted response at the x-value that corresponds to the midpoint of the bin.
- **Local averaging**. Local averaging is a modification of simple binning that allows observations to be used more than once. We replace the bin with a moving window that we move from observation to observation. Each time we average the observations contained in that window to

obtain the predicted value of an observation. Unlike simple binning local averaging yields a prediction for each observation.

Kernel estimation

Local averaging typically yields a curve that is rather choppy as observations enter and leave the window. Furthermore while points near x_0 are indeed informative about $f(x_0)$, they are not equally informative. We can obtain a smoother curve if we replace local averaging with locally weighted averaging, also known as kernel estimation. A kernel function is a function that weights observations that are close to the focal value x_0 more heavily than observations that are far away. If K is the kernel function then the weights at locations x_i near x_0 take the following form.

$$w_i = \frac{1}{h} \cdot K\left(\frac{x_i - x_0}{h}\right)$$

Here h is called the bandwidth. Functions typically used as kernel functions are the tricube kernel, the Gaussian kernel, and the rectangular kernel. The tricube function is defined as follows.

$$K(z) = \begin{cases} \left(1 - |z|^3\right)^3 & \text{if } |z| < 1 \\ 0 & \text{if } |z| \geq 1 \end{cases}$$

Each kernel function K satisfies the following properties.

1. $\int K(u)\,du = 1$
2. $K(u) = K(-u)$
3. $K\left(\frac{x_i - x_0}{h}\right) = 0 \text{ if } |x_i - x_0| > h$

With the weights calculated from the kernel we estimate $f(x_0)$ using the n observed values of y_i as follows.

$$\hat{f}(x_0) = \frac{\sum_{i=1}^{n} w_i y_i}{\sum_{i=1}^{n} w_i}$$

where $w_i = 0$ for observations whose distance from x_0 exceeds the band width. By adjusting the band width or equivalently the span, the proportion of data that are assigned nonzero weights, we can adjust the smoothness of the displayed estimate of *f*.

Local polynomial regression (in strictu)

To improve our estimate of $f(x_0)$ we can take advantage of the relationship between *y* and *x*. Currently we are using the *x*-values only to determine the weights in the kernel function that are then used in a weighted sum of the neighboring *y*-values. Instead we can carry out a weighted regression of *y* on *x* for the observations contained in each window with the weight given by the kernel function. So, in each window we fit the regression equation

$$y_i = a + b_1(x_i - x_0) + b_2(x_i - x_0)^2 + \cdots + b_p(x_i - x_0)^p + \varepsilon_i$$

where typically $p = 1$, 2, or 3. We estimate the parameters $a, b_1, \ldots, b_p$ by minimizing $\sum \varepsilon_i^2$ (ordinary least squares) or by minimizing $\sum w_i \varepsilon_i^2$ (weighted least squares). The regression equation then provides us with an estimate of $f(x_0)$.

Robustness

While local polynomial regression yields a perfectly legitimate estimate of *f*, the estimate can be sensitive to outliers. To minimize the effect of outlying values most local polynomial regression routines carry out an additional step. From the local polynomial regression we obtain the estimated residual, $e_i = y_i - \hat{y}_i$, for each observation. Using the residual we calculate a second weight, $W_i = W(e_i)$, where *W* is a kernel function. Typical choices for *W* are the bisquare function or the Huber weight function (in which the weight is constant in a neighborhood of zero outside of which it decreases with distance). The weight function *W* serves as a tuning constant so that the influence of an observation on the regression model varies inversely with how far its residual is removed from zero.

We refit the local polynomial regression model but this time we minimize the objective function $\sum w_i W_i \varepsilon_i^2$ where w_i is the kernel neighborhood weight and W_i is the robustness weight. This whole process is then repeated. Typically two bouts of reweighting are sufficient.

Local polynomial regression in R

Local polynomial regression can be carried out in R with the **lowess** and **loess** functions of base R. These functions differ in their defaults, syntax, and the organization of their return values but otherwise they do the same thing. Additive models with loess smoothers of multiple predictors can be fit using the **gam** package. The syntax takes the form: `y ~ gam(lo(x1) + lo(x2) + lo(x1, x2).`

Spline models

Spline models are piecewise regression functions in which the pieces are constrained to connect at their endpoints (yielding a continuous curve). Usually we require that both the first and second derivatives of the resulting curve be continuous at the join points (yielding a smooth curve). A popular example of a spline model is a cubic regression spline. Although cubic regression splines tend to yield results that are nearly indistinguishable from lowess they do so using a very different approach, one that readily generalizes to more complicated regression models such as those for structured and/or correlated data.

Global polynomial regression

Fig. 1 displays some data values along with the nonlinear model that was used to generate them. The observations are normally distributed with a mean given by the nonlinear model and a standard deviation of 0.3. Given the complicated pattern that is shown we might consider approximating the relationship between *y* and *x* with a polynomial. The graph of the true model has four critical points (places with horizontal tangents) over the range of the data suggesting that we would need at minimum a 5th degree polynomial to duplicate this. Fig. 1 shows a 5th degree and a 12th degree polynomial that were fit to these data using least squares.

```
set.seed(1)
myfunc <- function(x) sin(2*(4*x-2)) + 2*exp(-16^2*(x-0.5)^2)
x <- seq(0, 1, length=1001)
y <- myfunc(x) + rnorm(1001, mean=0, sd=0.3)
plot(y ~ x, col='grey80', cex=.7)
curve(myfunc, col='grey30', lwd=2, add=T)
# 5th degree polynomial
out.poly <- lm(y~x+ I(x^2)+ I(x^3)+ I(x^4)+ I(x^5))
# 12th degree polynomial
out.poly2 <- lm(y~x+ I(x^2)+ I(x^3)+ I(x^4)+ I(x^5)+ I(x^6)+ I(x^7)+ I(x^8)+ I(x^9)+ I(x^10)+
   I(x^11)+ I(x^12))
lines(x,fitted(out.poly), col=2, lty=2)
lines(x,fitted(out.poly2), col='dodgerblue', lty=2, lwd=2)
legend('topleft', c('true model', '5th degree polynomial', '12th degree polynomial'),
   col=c('grey30', 2, 'dodgerblue'), lty=c(1,2,2), lwd=c(2,1,2), cex=.8, bty='n')
```

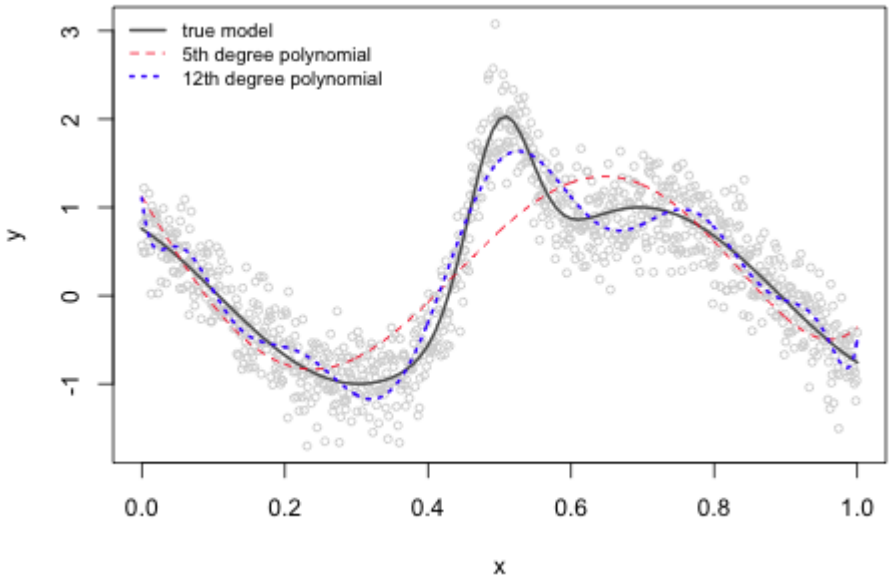

Fig. 1 Nonlinear function along with two different approximating polynomial regression models

The 5th degree polynomial has correctly detected only two of the four actual critical points and has completely missed the global maximum at $x \approx 0.5$. In addition it has found a completely spurious local minimum near the right hand endpoint of the data range. The 12th degree polynomial, on the other hand, detects all four critical points but not at the correct locations. It has also introduced a number of extra wiggles and inflection points that are not seen in the true model.

Higher order polynomial regression models tend to have numerical problems. Below is just a portion of the summary table output of the 5th degree polynomial model showing the correlations of the coefficient estimates. Over half the correlations exceed 0.90 in absolute value and none of the rest are small.

```
summary(out.poly, cor=T)
< snip >
Correlation of Coefficients:
       (Intercept) x     I(x^2) I(x^3) I(x^4)
x      -0.86
I(x^2)  0.74       -0.97
I(x^3) -0.66        0.92 -0.99
I(x^4)  0.60       -0.87  0.96  -0.99
I(x^5) -0.55        0.82 -0.93   0.97  -0.99
```

The situation is even worse with the 12th degree polynomial. In addition to having highly correlated coefficient estimates, the estimates are also highly unstable. Eight of the coefficient estimates exceed 1 million (in absolute value) even though the predictor lies in the interval (0, 1) and the

response variable does not exceed 3 in absolute value.

```
round(summary(out.poly2)$coefficients,2)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.11	0.14	7.98	0
x	-66.14	13.60	-4.86	0
I(x^2)	2715.77	438.03	6.20	0
I(x^3)	-51618.39	6610.70	-7.81	0
I(x^4)	512233.94	55903.10	9.16	0
I(x^5)	-3000123.89	291778.47	-10.28	0
I(x^6)	11095738.00	992749.67	11.18	0
I(x^7)	-26854130.42	2265140.83	-11.86	0
I(x^8)	43093813.27	3492640.24	12.34	0
I(x^9)	-45434744.54	3590409.63	-12.65	0
I(x^10)	30248270.34	2356911.60	12.83	0
I(x^11)	-11530274.86	893471.84	-12.91	0
I(x^12)	1918185.30	148782.41	12.89	0

The correlation of the coefficient estimates is so severe that if we try to fit a polynomial of a higher degree, the design matrix ends up being singular and we obtain missing values for some of the coefficient estimates.

```
out.poly3 <- lm(y~x+ I(x^2)+ I(x^3)+ I(x^4)+ I(x^5)+ I(x^6)+ I(x^7)+ I(x^8)+ I(x^9)+ I(x^10)+
   I(x^11)+ I(x^12)+ I(x^13))
coef(out.poly3)
```

(Intercept)	x	I(x^2)	I(x^3)	I(x^4)	I(x^5)	I(x^6)
1.108066e+00	-6.614309e+01	2.715772e+03	-5.161839e+04	5.122339e+05	-3.000124e+06	1.109574e+07
I(x^7)	I(x^8)	I(x^9)	I(x^10)	I(x^11)	I(x^12)	I(x^13)
-2.685413e+07	4.309381e+07	-4.543474e+07	3.024827e+07	-1.153027e+07	1.918185e+06	NA

Polynomial splines

With quadratic and cubic polynomial regression models correlations between coefficient estimates can be reduced by centering the predictor. This is typically accomplished by subtracting the mean of the predictor from each observed value and then carrying out regression on the centered values. For higher order polynomial models centering doesn't work and the correlations can be severe. From a modeling standpoint the basic problem with polynomials is that they are non-local: changing a single data value affects the fit of the model over the entire range of the data. We see this in Fig. 1 where in order to constrain the polynomials to have critical values at the correct places, additional critical values and inflection points had to be introduced in places where none should exist.

A solution to the intractability of global polynomial models is to parallel what we did with lowess. Instead of fitting a single high order polynomial model to all the data simultaneously we bin the data and fit separate lower order polynomials (cubic is the typical choice) to the data in each bin. We do this in such a way that the separate pieces link up smoothly. The boundaries of the bins are referred to as **knots**. This raises two questions.

1. How do we choose an appropriate number of knots, and
2. how do we choose the locations of the knots?

Once we've decided on the number of knots then two popular choices are to (1) place them at equally spaced locations over the range of the predictor, or (2) place them at quantiles of the predictor. To choose the number of knots we could fit a number of spline models with different numbers of knots and choose the one that visually provides an optimal degree of smoothness. This is analogous to fitting a sequence of lowess models and varying the span (or equivalently the fraction of observations used in each window). Certain kinds of splines manage to avoid the problem of knots altogether as will be discussed later.

There are three reasons why spline models are preferred over global polynomial models.

1. The analytical foundation of spline models is superior to global polynomial models.
2. Spline models can be fit in such a way so as to prevent overfitting.
3. Spline terms can readily be combined with parametric terms to yield what are called semiparametric models.

Truncated power series (TP) basis

The easiest polynomial spline model to understand is one that uses a truncated power series basis. In linear algebra a basis for a set V is defined to be a linearly independent set of elements of V such that any element of V can be written as a linear combination of the members of the basis. A linear combination of truncated power series basis elements can be used to represent functions.

The first few elements of a truncated power series basis are the individual terms of a non-local polynomial of order l, where typically we let $l = 2$ or 3. For $l = 2$ these terms would consist of a constant term, a linear term, and a quadratic term. For $l = 2$ the remaining elements of the truncated power series basis would be quadratic terms, $(x - k_i)_+^2$, one for each knot k_i, such that each quadratic term is identically zero for $x < k_i$ but nonzero for $x \geq k_i$. The individual quadratic terms are defined as follows.

$$(x-k_i)_+^2 = \begin{cases} 0, & \text{if } x < k_i \\ (x-k_i)^2, & \text{if } x \geq k_i \end{cases}$$

Each represents a parabola with vertex at $(k_i, 0)$ where we have only the right branch of the parabola.

For the data in Fig. 1 suppose we choose ten equally spaced knots starting at 0.05 and ending at 0.95, each 0.10 units apart. For $l = 2$ a polynomial spline regression model using a TP basis would be the following.

$$y_i = \underbrace{\gamma_1 + \gamma_2 x_i + \gamma_3 x_i^2}_{\text{global polynomial}} + \underbrace{\gamma_4 (x_i - 0.05)_+^2 + \gamma_5 (x_i - 0.15)_+^2 + \cdots + \gamma_{13} (x_i - 0.95)_+^2}_{\text{truncated polynomial terms}} + \varepsilon_i$$

The R code below fits this model and displays a graph of the individual terms of the TP basis scaled by their estimated γ values.

```
# truncated polynomial
f.c <- function(x,c) ifelse(x<c, 0, (x-c)^2)
# regression model
out.yuk <- lm(y~x+ I(x^2)+ f.c(x,.05)+ f.c(x,.15)+ f.c(x,.25)+ f.c(x,.35)+ f.c(x,.45)+ f.c(x,.55)+
   f.c(x,.65)+ f.c(x,.75)+ f.c(x,.85)+ f.c(x,.95))
plot(y ~ x, col='grey80', cex=.7)
curve(myfunc, col='grey30', lwd=2, add=T)
# global polynomial
curve(coef(out.yuk)[1]+ coef(out.yuk)[2]*x+ coef(out.yuk)[3]*x^2, add=T, lty=3, col=2, lwd=2)
# truncated polynomials scaled by regression coefficients
sapply(1:10, function(z) curve(coef(out.yuk)[z+3] * f.c(x, seq(.05,.95,.1)[z]), add=T,
   col=c(rep(c(1,3:4),3),1)[z], lty=2)) -> hick
legend('topright', c('true model', 'global polynomial term', 'truncated polynomials'),
   col=c('grey30',2,1), lty=c(1,3,2), lwd=c(2,2,1), cex=.8, bty='n')
```

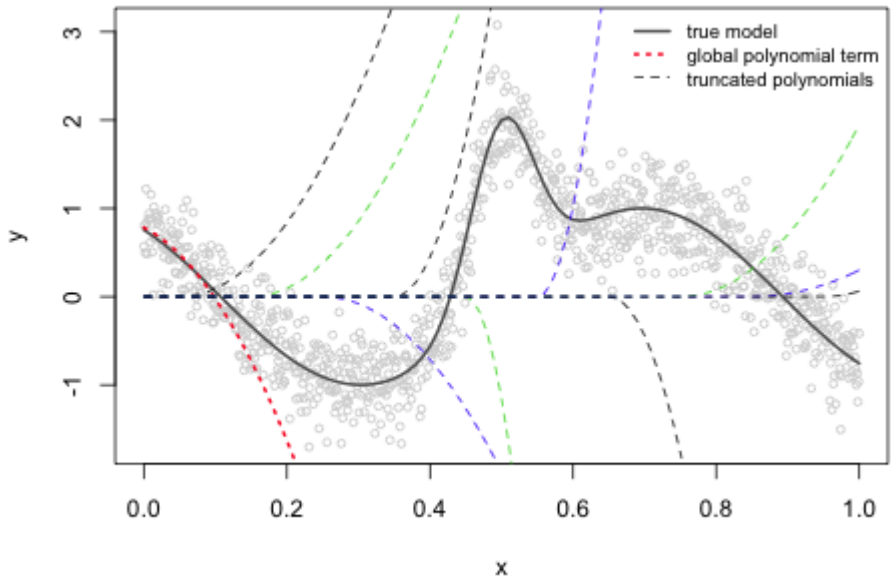

Fig. 2 Nonlinear function along with the estimated truncated power series basis terms from a quadratic polynomial spline model

If we add the global polynomial term to the individual truncated polynomials at each value of the predictor *x* we obtain the polynomial spline prediction of the function (Fig. 3). Observe that the fit is quite good and is better than what was obtained with the 12th degree global polynomial model we fit earlier even though 13 parameters are estimated in both cases.

```
plot(y ~ x, col='grey80', cex=.7)
curve(myfunc, col='grey60', lwd=3, add=T)
lines(x, fitted(out.yuk), col=2, lty=2)
legend('topright', c('true model', 'polynomial spline (TP) model'), col=c('grey60',2), lty=c(1,2),
   lwd=c(3,1), cex=.8, bty='n')
```

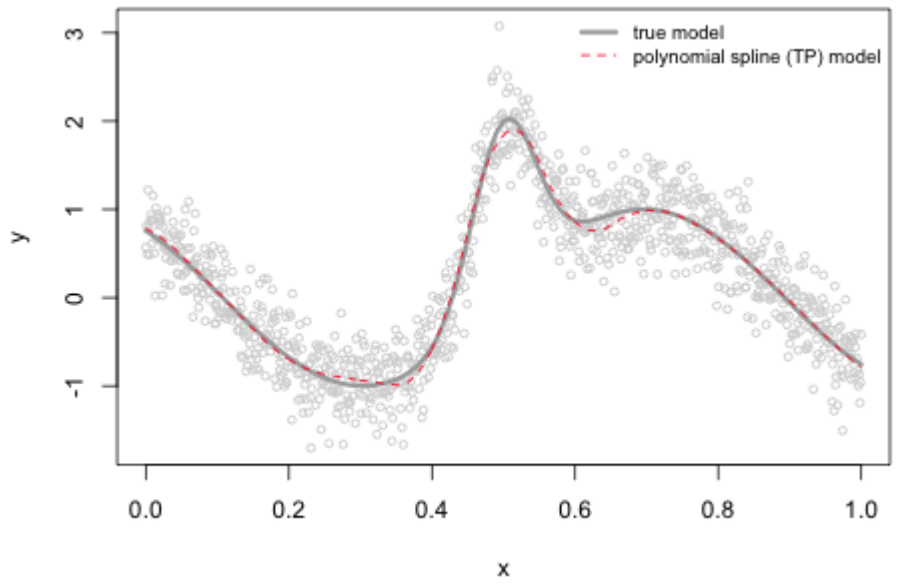

Fig. 3 Nonlinear function along with the estimated quadratic polynomial spline model using a TP basis and 10 knots.

B-spline basis

While the truncated polynomial basis avoids some of the problems associated with global polynomial regression models, it can still suffer from numerical instability because the elements of the TP basis are all unbounded above. Furthermore because each polynomial is nonzero for $x \geq k_1$ the domains of the different terms substantially overlap with each other. Because of this the TP basis is not the ideal choice for local polynomial regression. A better choice is to use a B-spline basis.

A B-spline basis is not especially interpretable; its attraction comes from its analytical tractability. The B-spline basis can be obtained from a recurrence relation. The j^{th} B-spline of order l is defined as follows.

$$B_j^l(x) = \frac{x - k_{j-l}}{k_j - k_{j-l}} B_{j-1}^{l-1}(x) + \frac{k_{j+1} - x}{k_{j+1} - k_{j+1-l}} B_j^{l-1}(x)$$

Here k_{j-l}, k_j, k_{j+1}, and k_{j+1-l} are knots. Notice that the order l B-spline basis is defined in terms of order $l-1$ B-spline basis elements. To get things started we need an explicit expression for the order $l = 0$ B-spline basis.

$$B_j^0(x) = \begin{cases} 1, & k_j \le x < k_{j+1} \\ 0, & \text{otherwise} \end{cases}, \quad j = 1, \ldots, d-1$$

where d is the number of knots. Each of these zero order B-splines is an example of a Haar function, a kind of step function. To obtain the first order B-spline basis we use the recurrence relation on the zero order B-spline basis elements. From the formula each term will be linear in the variable x. The second order B-spline basis is also defined using the recurrence relation but this time in terms of the first order B-spline basis. Since each of the first order B-spline basis elements is multiplied by the variable x in the formula we end up with quadratic terms. Fig. 4 shows the entire second order B-spline basis for the data of Fig. 1.

```
library(splines)
out.bs10 <- bs(x, knots = seq(0.05, 0.95, by = 0.1), degree=2)
plot(y ~ x, col='grey80', cex=.7)
curve(myfunc, col='grey30', lwd=2, add=T)
sapply(1:ncol(out.bs10), function(z) lines(x, out.bs10[,z], col=rep(1:4,3)[z], lty=2)) -> yuk
```

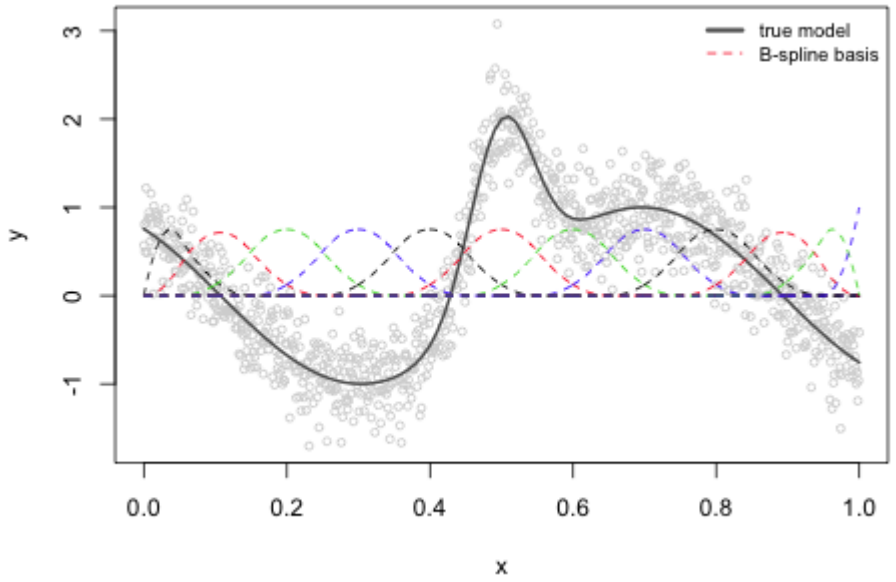

Fig. 4 Nonlinear function along with the 12 elements of a quadratic B-spline basis with knots at 0.05, 0.15, … , 0.95.

Using the second order B-spline basis as predictors in a linear regression model we can obtain the corresponding polynomial spline regression model (Fig. 5).

```
plot(y ~ x, col='grey80', cex=.7)
curve(myfunc, col='grey30', lwd=2, add=T)
mod3b <- lm(y ~ bs(x, knots = seq(0.05, 0.95, by = 0.1), degree=2))
lines(x, fitted(mod3b), col=2, lty=2)
legend('topright', c('true model', 'quadratic B-spline basis model'), col=c('grey60',2), lty=c(1,2),
   lwd=c(3,1), cex=.8, bty='n')
```

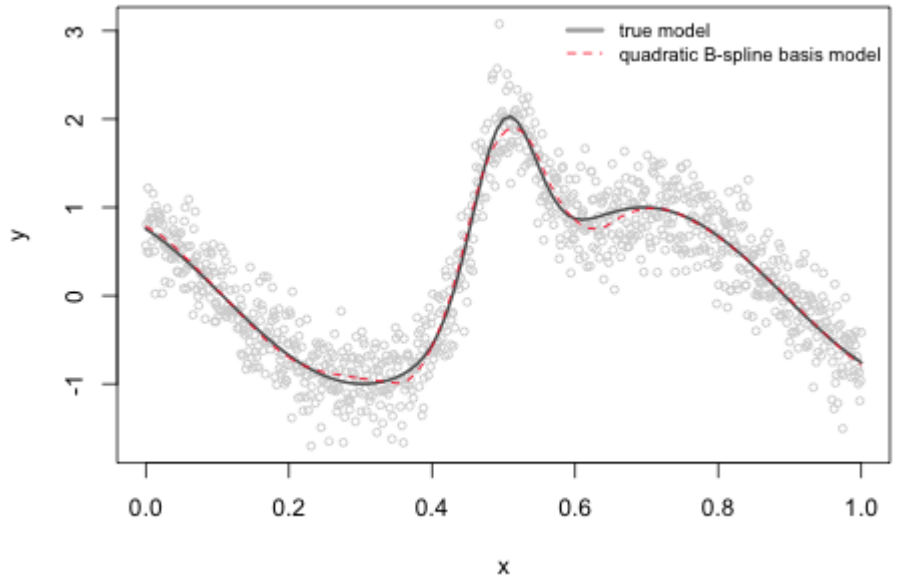

Fig. 5 Nonlinear function along with the estimated polynomial spline model using a quadratic B-spline basis and 10 knots.

Penalized splines

Penalized splines take a more systematic approach to the problem of knots. Consider the following more general version of the truncated power series basis that we considered earlier. This time the number and placement of the knots is left unspecified.

$$y_i = \underbrace{\gamma_1 + \gamma_2 x_i + \gamma_3 x_i^2}_{\text{global polynomial}} + \underbrace{\gamma_4 (x_i - k_1)_+^2 + \gamma_5 (x_i - k_2)_+^2 + \cdots + \gamma_{3+d} (x_i - k_d)_+^2}_{\text{truncated polynomial terms}} + \varepsilon_i$$

Written in matrix formulation this is the following.

$$\mathbf{y} = \mathbf{X}\boldsymbol{\gamma} + \boldsymbol{\varepsilon}$$

Here **X** is the design matrix whose columns consist of the TP basis elements and **γ** is a vector of regression coefficients. The number of knots is *d*. Coefficient estimates can be obtained with the method of least squares by minimizing the following sum of squares.

$$\begin{aligned} \mathrm{LS} = \sum_{i=1}^{n} \left(y_i - f(x_i) \right)^2 &= \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{d+3} \gamma_j B_j(x_i) \right)^2 \\ &= \left(\mathbf{y} - \mathbf{X}\boldsymbol{\gamma} \right)^T \left(\mathbf{y} - \mathbf{X}\boldsymbol{\gamma} \right) \end{aligned}$$

Here $B_j(x_i)$ are the elements of the TP basis and $\sum_{j=1}^{d+3} \gamma_j B_j(x_i)$ corresponds to the full truncated power series regression model shown above. The *d* terms of this expression that correspond to the truncated polynomial terms contribute a local roughness to the model to compensate for the smoothness imposed by the global polynomial portion. In matrix notation we can write the least squares solution as follows.

$$\hat{\boldsymbol{\gamma}} = \left(\mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{y}$$

With penalized splines we start with a large number of knots, so many knots that we're clearly overfitting the data. In the extreme case we would choose one knot for each unique value of *x* but in practice this causes numerical problems and so some large subset is chosen instead. We then find **γ** and λ that minimize the following penalized least squares objective function.

$$\mathrm{PLS}(\lambda) = \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{d+3} \gamma_j B_j(x_i) \right)^2 + \lambda \sum_{j=4}^{d+3} \gamma_j^2$$

λ is the smoothing parameter and is used to control the roughness of the polynomial spline fit. When λ = 0 the penalty term drops out and we obtain the ordinary least squares solution. For a fixed nonzero value of λ the penalty term contributes a positive amount to the objective function and the coefficients of the truncated polynomial terms have to be adjusted from their least squares values. The penalized least squares solution finds the values of $\gamma_1, \gamma_2, \ldots \gamma_{d+3}$, and λ that minimize the penalized least squares criterion. With penalized splines we replace the problem of choosing the optimal number of knots with choosing an optimal value for λ.

When the TP basis is replaced with a B-spline basis the terms in the penalized least squares objective function don't neatly divide into a smooth component and a roughness component, so we need to take a more general approach. We choose the following as the penalty term.

$$\lambda \int \left(f''(x) \right)^2 dz$$

The second derivative of a function is related to its curvature and the penalty term is the squared rate of change of the slope added up over the entire range of the estimates. So by using a penalty based on the second derivative we elect to penalize approximating functions that wiggle too much. For a function generated from a B-spline basis the second derivative can be approximated by taking second order differences of the coefficients of the B-spline basis. In general we can construct a penalized least squares criterion based on r^{th} order differences as follows.

$$\text{PLS}(\lambda) = \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{d} \gamma_j B_j(x_i) \right)^2 + \lambda \sum_{j=r+1}^{d} \left(\Delta^r \gamma_j \right)^2$$

where

$$\begin{aligned} \Delta^1 \gamma_j &= \gamma_j - \gamma_{j-1} \\ \Delta^2 \gamma_j &= \Delta^1 \left(\Delta^1 \gamma_j \right) = \Delta^1 \gamma_j - \Delta^1 \gamma_{j-1} = \gamma_j - 2\gamma_{j-1} + \gamma_{j-2} \\ &\vdots \end{aligned}$$

The penalized least squares criterion can also be expressed in matrix notation.

$$\text{PLS}(\lambda) = \left(\mathbf{y} - \mathbf{X}\boldsymbol{\gamma} \right)^T \left(\mathbf{y} - \mathbf{X}\boldsymbol{\gamma} \right) + \lambda \boldsymbol{\gamma}^T \mathbf{K} \boldsymbol{\gamma}$$

where **K** is called the penalty matrix. The PLS solution for a fixed λ takes the following form.

$$\hat{\boldsymbol{\gamma}} = \left(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{K} \right)^{-1} \mathbf{X}^T \mathbf{y}$$

and the spline model prediction of the vector of responses is given by

$$\hat{\mathbf{y}} = \mathbf{X} \hat{\boldsymbol{\gamma}} = \mathbf{X} \left(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{K} \right)^{-1} \mathbf{X}^T \mathbf{y} = \mathbf{S} \mathbf{y}$$

where **S** is called the smoothing matrix and is analogous to the hat matrix of ordinary linear regression.

It's worth noting that the PLS solution takes the same form as the solution to a regression model with random effects when the variance components are known. Consequently it is possible to represent a penalized spline model as a mixed model. We think of the data that lie between a set of knots as comprising a group and one random coefficient is assigned to each knot. In a penalized spline model the random effects are used to model the

nonlinearity rather than any unobserved heterogeneity. Continuing the analogy when $\lambda = \infty$ we have the common pooling model and when $\lambda = 0$ we have the separate slopes and intercepts model.

Smoothing splines

We can generalize penalized splines even further by leaving the form of the approximating function *f* unspecified. This leads to the following penalized least squares criterion.

$$\mathrm{SS} = \sum_{i=1}^{n} \left(y_i - f\left(x_i\right)\right)^2 + \lambda \int \left(f''(x)\right)^2 dz$$

It turns out that the optimal choice for *f* in this expression is a special class of functions known as natural cubic splines, which are a subset of ordinary polynomial splines. A function *f* is a natural cubic spline with knots $a \le k_1 < k_2 < \ldots < k_{d-1} < k_d \le b$ if

1. *f* is a cubic polynomial spline at the interior knots.
2. *f* satisfies the boundary conditions, $f''(a) = f''(b) = 0$. These boundary conditions constrains *f* to be linear on the intervals $[a, k_2]$ and $[k_{d-1}, b]$.

A natural spline basis can be obtained as only a slight modification of the B-spline basis we considered earlier.

With penalized and smoothing splines we avoid the problem of choosing the number and location of the knots by replacing it with the problem of estimating a single tuning parameter λ. We use cross validation to estimate λ as follows. Fix λ at some specific value. Remove one of the observations, call it x_i. Use the remaining $n - 1$ observations and the formula for the penalized least squares solution given above to estimate the natural spline function. Obtain the prediction of the value of the response for the deleted observation, $\hat{f}^{(-i)}\left(x_i\right)$. Repeat this for each of the *n* observations and then calculate the cross validation criterion shown below.

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

Choose λ that minimizes this quantity.

Fortunately it's unnecessary to carry out the regression *n* times for a given value of λ. Instead we can estimate the smoother only once using all the data and obtain the cross validation criterion as follows.

$$\mathrm{CV} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{f}(x_i)}{1 - s_{ii}}\right)^2$$

Here $\hat{f}(x_i)$ is the estimate of the smoother at x_i and s_{ii} is the ith diagonal element of the smoothing matrix **S** that was defined above. The matrix **S** can be difficult to calculate so the cross validation criterion is usually replaced with what's called the generalized cross validation criterion.

$$\mathrm{GCV} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{f}(x_i)}{1 - \mathrm{tr}(\mathbf{S})/n}\right)^2$$

In the GCV we replace s_{ii} with the average of the diagonal elements of **S**. This is a savings because while calculating **S** may be difficult it is possible to calculate tr(**S**) from the component matrices of **S** without calculating **S** explicitly.

The tr(**S**) is referred to as the equivalent degrees of freedom and is interpreted as the effective number of parameters of a smoother. It can be used to construct an AIC measure for comparing models. For an ordinary polynomial spline model the effective number of parameters is the number of elements of the basis. For penalized and smoothing splines the effective number of parameters decreases from this value as the smoothing parameter λ is increased.

Splines in R

The functions **bs** and **ns** from the **splines** package of R can be used to generate a basis of B-splines and natural splines respectively that can then be used in regression models. As we've seen to use these two functions we need to specify the knot locations. There are other packages that implement penalized and smoothing splines. Two packages that permit the incorporation of a wide class of smoothing terms directly in regression models are the **gam** and **mgcv** packages. Both of these packages fit what are called generalized additive models. The **gam** package uses a backfitting algorithm to estimate the models while the **mgcv** package (acronym for multiple generalized cross validation) uses iteratively reweighted least squares and maximum likelihood.

The **s** function of **mgcv** calculates various smooths and offers an array of choices for the spline basis. All of these are penalized splines so there is no need to specify knots. On the other hand there is an argument *k* that is sometimes needed to specify the dimension of the basis used to represent the smooth. The default value of this parameter can occasionally be too small yielding a less than optimal smooth.

To illustrate the use of the *k* argument I fit a smooth to the data of Fig. 1 using the **mgcv** package. The default spline choice for the **s** function is a thin plate spline. If we specify instead a cubic spline we see that there is very little difference in the two choices (Fig. 6). Notice that neither model has tracked the true model very well. That's because the default dimension of the basis used to represent the smooth term is set too low. When we increase it by specifying $k = 20$ the resulting spline curve provides a closer fit to the true model.

```
plot(y ~ x, col='grey90', cex=.7)
curve(myfunc, col='grey70', lwd=2, add=T)
```

```
library(mgcv)
# default is a thin plate spline
out.gam <- gam(y~s(x))
lines(x,predict(out.gam), col='pink', lwd=2)
# switch to a cubic spline
out.gam1 <- gam(y~s(x, bs='cr'))
lines(x,predict(out.gam1), col='dodgerblue1', lwd=2, lty=2)
# increase the dimension of the basis
out.gam2 <- gam(y~s(x, k=20))
lines(x,predict(out.gam2), col=2, lty=2)
legend('topleft',c('true model', 'thin plate spline (default k)', 'cubic spline (default k)', 'thin
   plate spline (k = 20)'), col=c('grey80', 'pink', 'dodgerblue1',2), lwd=c(2,2,2,1),
   lty=c(1,1,2,2), bty='n', cex=.8)
```

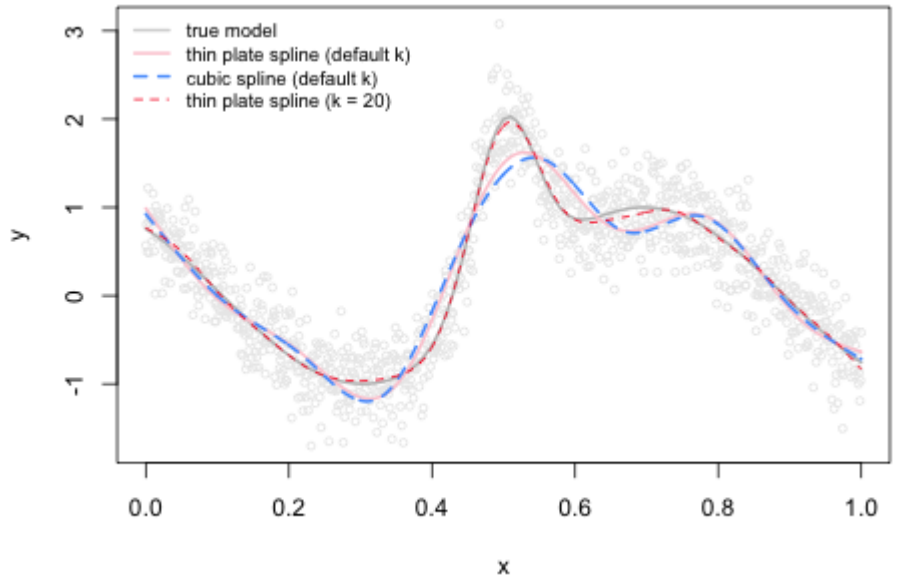

Fig. 6 Adjusting the basis dimension of the **s** function of the **mgcv** package

References

- Fahrmeir, L., T. Kneib, S. Lang, and B. Marx. 2013. *Regression: Models, Methods, and Applications*. New York: Springer. Chapter 8, Nonparametric regression, 413–533.

- Keele, Luke. 2008. *Semiparametric Regression for the Social Sciences*. New York: Wiley.

Course Home Page

Jack Weiss
Phone: (919) 962-5930
E-Mail: jack_weiss@unc.edu
Address: Curriculum for the Environment and Ecology, Box 3275, University of North Carolina, Chapel Hill, 27599
Copyright © 2012
Last Revised--August 11, 2013
URL: https://sakai.unc.edu/access/content/group/2842013b-58f5-4453-aa8d-3e01bacbfc3d/public/Ecol562_Spring2012/docs/lectures/lecture25.htm