STA 414/2104, Spring 2006 — Assignment #3

*Due at **start** of class on March 30. Note that this assignment is to be done by each student individually. You may discuss it in general terms with other students, but the work you hand in should be your own.*

In this assignment you will implement and try out additive regression models using natural linear splines. Natural linear splines are like the more common (and more useful) natural cubic splines, except that they are piecewise linear functions rather than piecewise cubic functions, and they are the solution of a penalized least squares problem in which the penalty involves the first derivative rather than the second derivative.

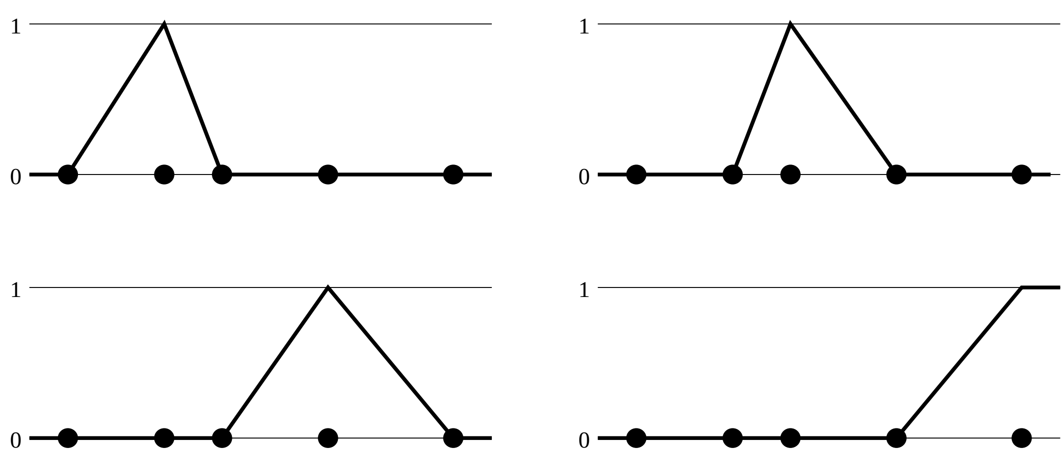
In detail, for a univariate problem, we would like to find the continuous function $f(x)$ that minimizes

$$\sum_{i=1}^{N} \Big[y_i - f(x_i)\Big]^2 \;+\; \lambda \int \Big[f'(x)\Big]^2 dx$$

Here, (x_i, y_i) are the inputs and responses for N training cases. One can show that the solution to this is a piecewise linear function with the boundaries of the pieces (the knots) at the distinct values of x_i. We will label these knots, in increasing order, as $\xi_1, \ldots, \xi_K$, where $K \leq N$ is the number of distinct values of x_i. The constant λ controls the amount of "smoothing" done. For this assignment, we will set λ manually, though in practice one would probably set it using cross validation.

Natural linear splines have zero derivatives before ξ_1 and after ξ_K (ie, they are constants in those regions), since in the regions beyond the data we can set the derivative to zero (minimizing the penalty) without any effect on how closely the data is fit. With this constraint, it's easy to see that the space of natural linear splines is spanned by a set of K basis functions. (Start with $2(K+1)$ basis functions for arbitrary piecewise linear functions, impose K continuity constraints at the knots, and then require the derivatives at the ends to be zero.)

We will use these splines in additive models, however, in which we will include a separate intercept term (with no penalty). We therefore don't want the equivalent of an intercept in the natural linear splines, reducing the number of basis functions to $K-1$. There are many possible sets of basis functions, but you should use the basis functions that look like this (for $K = 5$ knots, shown as dots on the horizontal axis):



Linear combinations of these can produce any continuous piecewise linear function, except that the functions produced always are zero for $x < \xi_1$, which can be changed using the intercept term.

You should write two functions in R or Matlab to implement additive models using natural linear splines. The `lspline.fit` function should take as arguments a matrix of inputs for training cases (with N rows and p columns), a vector of responses for training cases, and a value for λ. It should return a vector of coefficients that is the penalized least squares estimate. The first coefficient should be the intercept for the model. This should be followed by the coefficients of the spline basis functions for the first variable, of which there will be $N-1$, unless some values of the first variable are duplicates (in which case there will be fewer). The coefficients of the spline for the second variable follow, etc. The `lspline.pred` function should take as arguments the matrix of training inputs (as passed to `lspline.fit`), the vector of coefficents (as returned by `lspline.fit`), and a matrix of inputs for test cases. (The training inputs are needed to figure out what basis functions are being used.) It should return a vector of predicted responses for the test cases. Note that the test cases could be the same as the training cases, which is useful in seeing how well the spline estimate fits the training data.

To write these functions, you will need to write some other functions. The following functions are suggested (but not required). The `lspline.value.matrix` function should find the matrix of values for basis functions for a set of cases, given as arguments the matrix of training inputs and a matrix of inputs for the set of cases for which values are desired (which could be the same as the matrix of training inputs). The first column of this value matrix should be all 1s, providing for an intercept. The `lspline.penalty.matrix` function should return the matrix $\mathbf{P}$ such that $\beta^T \mathbf{P} \beta$ is the total penalty for all coefficients. It takes the matrix of training inputs as an argument. These two functions should in turn call functions `lspline.values` and `lspline.penalty` that do the analogous things for a single input variable.

To help you debug your program, here is the output for a simple example:

```
> print (x.train <- matrix (c(4.4, 3.3, 9.0, 3.2, 2.1, 8, 9, 8, 7, 9), 5, 2))
     [,1] [,2]
[1,]  4.4    8
[2,]  3.3    9
[3,]  9.0    8
[4,]  3.2    7
[5,]  2.1    9
> print (y.train <- c(8, 9, 4, 8, 6))
[1] 8 9 4 8 6
> print (x.test <- matrix (c(5.2, 4.3, 1.1, 9.9, 8, 6, 7.5, 7.4), 4, 2))
     [,1] [,2]
[1,]  5.2  8.0
[2,]  4.3  6.0
[3,]  1.1  7.5
[4,]  9.9  7.4
> print (beta <- lspline.fit (x.train, y.train, 0.1))
[1]  5.69263450  2.31644666  2.53611388  2.16615271 -1.69941209  0.09081161  0.51795156
> lspline.pred (x.train, beta, x.train)
[1] 7.949599 8.746700 4.084034 8.009081 6.210586
> lspline.pred (x.train, beta, x.test)
[1] 7.277327 7.892420 5.738040 4.029547
```

As is the case with natural cubic splines, it is possible to perform the computations required for natural linear splines very efficiently. However, for this assignment, you should just use the standard formulas for least squares regression with a quadratic penalty (as in the lecture notes).

You should try out your functions on the artificial data with two inputs that I have put on the web page, fitting to the training data, and then making predictions on the test data. You should do this for λ set to 0.0001, 0.001, 0.003, 0.01, 0.02, 0.03, 0.05, 0.1, and 0.2. For each setting of λ, find the average squared error of the predictions for the test cases by comparing with the actual responses that are also available from the web page.

The web page also has input values for a 51×51 grid of points. You should make predictions for these points as well, with λ set to 0.0001, 0.2, and whatever value produced the smallest average squared error on the test cases, and produce contour plots of these predictions. (Instructions for how to produce contour plots will be put on the web page soon.) Briefly discuss the results in terms of what the bias and variance appear to be with different values of λ.