Kernel smoothing

Kernel smoothing is essentially *weighted local averaging*. It balances bias and variability using a smoothing parameter that essentially controls how many points get high weight in the averaging.

```r
library(alr4)
# kernel regression
par(pch=20, lwd=2) # use filled circles and fat lines
oldpar <- par(mfrow=c(1,3)) # set array of plots
with(oldpar, {
  plot(Interval ~ Duration, main="bandwidth=5")
  lines(ksmooth(Duration, Interval, "normal", 5))
  plot(Interval ~ Duration, main="bandwidth=20")
  lines(ksmooth(Duration, Interval, "normal", 20))
  plot(Interval ~ Duration, main="bandwidth=40")
  lines(ksmooth(Duration, Interval, "normal", 40))
})
par(oldpar)
```

![Graphs showing kernel regression with different bandwidths.](image)
I used `with` to specify that all the functions inside the brackets refer to data in the `oldfaith` data frame. `plot` draws the plots, and `lines` draws the lines, with `ksmooth` determining the coordinates of the lines. "normal" is the name of the kernel function, and the last argument is the smoothing parameter.

The function `hcv` in the `sm` package uses a brute-force method with leave-one-out cross validation to get a smoothing parameter, and optionally draws a figure.

```
library(sm)
system.time(with(oldfaith, hcv(Duration, Interval)))

user  system elapsed
 0.42   0.02   0.43

with(oldfaith, hcv(Duration, Interval, display="lines"))

[1] 26.92
```

![Graph](image)

Newer (and recommended) code in `sm` computes cross-validation much more efficiently. While the above plot can’t be drawn, nearly the same solution is obtained:

```
system.time(with(oldfaith, print(h.select(Duration, Interval, method="cv"))))

[1] 23.91

user  system elapsed
 0.04   0.00   0.03
```
Unlike most of R, `sm` functions have some limited interactive capability if you install the `rpanel` package and its dependencies,

```r
install.packages("rpanel", dependencies=TRUE)
```

For a more interesting graph, which you should try, add the `panel` argument for interactive capability:

```r
library(rpanel)
with(oldfaith, sm.regression(Duration, Interval, method="cv", panel=TRUE))
```

This gives a graph with slidebars to see the effects of various choices of the smoothing parameter.

**loess (also known as lowess)**

Loess uses *local regression* rather than local averaging. The default in R is to fit using local quadratic polynomials with a nearest neighbor rule, that is “give the closest $f \times 100\%$ of the data positive weight” rather than a fixed bandwidth rule “give positive weight to all cases within $h$ of the point of interest”.

```r
(m1 <- loess(Interval ~ Duration, oldfaith))
```
Call:
loess(formula = Interval ~ Duration, data = oldfaith)

Number of Observations: 270
Equivalent Number of Parameters: 3.97
Residual Standard Error: 5.74

with(oldfaith, {
  plot(Interval ~ Duration, main="loess, default is f=.75, deg=2", type="n")
  grid(lty=1)
  points(Duration, Interval)
  lines(spline(Duration, fitted(m1)),lty=1)
  lines(spline(Duration, fitted(update(m1, span=.15))),lty=2, lwd=2)
  lines(spline(Duration, fitted(update(m1, span=.95))),lty=3, lwd=3)
  lines(spline(Duration, fitted(update(m1, degree=1))),lty=4, lwd=2, col="red")
})

legend("topleft", c("Default", "f=.15", "f=.95", "deg=1"), lty=1:4, inset=0.02,
col=c("black", "black", "black", "red"), lwd=c(1, 2, 3, 2))

loess has a predict method:

  predict(m1, newdata=data.frame(Duration=50*(1:8)))

  1   2   3   4   5   6   7   8
  NA 52.23 59.78 72.43 79.79 82.83  NA  NA

The NAs are returned because loess doesn't extrapolate. The spline function used above interpolated between observed values of Duration using a cubic spline. It also reordered the data according to the values of Duration.
Smoothing Splines

Smoothing splines are estimate the the regression function $f(x)$ to minimize a penalized MSE:

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

where $\lambda \geq 0$ is a smoothing parameter. Assuming all values of $x_i$ are unique and ordered, we draw a (curved) line between $x_i$ to $x_{i+1}$, where the curvature is controlled by a balance between small error at the data, which would require $\hat{f}(x_i) = y_i$, and smoothness of the function, so the function doesn’t fluctuate wildly between observed $x_i$.

```r
print(sm2 <- with(oldfaith, smooth.spline(Duration, Interval)))
```

Smoothing Parameter spar= 0.9068 lambda= 0.009558 (12 iterations)
Equivalent Degrees of Freedom (Df): 5.12
Penalized Criterion: 4496
GCV: 33.5

```r
with(oldfaith, {
  plot(Interval ~ Duration, main="Smoothing Splines")
  lines(sm2, lty=1, lwd=2)
  lines(update(sm2, spar=.1), lty=2, lwd=2)
  lines(update(sm2, spar=2.5), lty=3, lwd=2)
})

legend("topleft", c("Default", "spar=.1", "spar=2.5"), lwd=2, lty=1:3, inset=0.02)
```
Regression splines

Regression splines approximate $f(x)$ as a linear combination of basis functions with unknown weights. A simple expression of the idea is approximating $f(x)$ as a sum of powers of $x$. For this example, begin by standardizing $\text{Duration}$ to have mean zero and range from $-1$ to $+1$.

```r
mid <- with(oldfaith, (min(Duration) + max(Duration))/2)
StdD <- with(oldfaith, (Duration - mid)/max(Duration - mid))
x <- seq(-1, 1, length=201)
```

I'm going to draw two plots. The left-plot is just the polynomial basis functions on the interval $(-1,1)$. The right plot of of $\text{Interval}$ against the standardized $\text{Duration}$, showing the quartic regression fit. Also shown on this plot are the five components of the quartic fit, $\hat{\beta}_0 \times 1, \hat{\beta}_1 x, \ldots, \hat{\beta}_4 x^4$; the quartic fit is just the sum of these curves.

```r
par(mfrow=c(1, 2))
plot(x, x, type="l", lty=2, xlab="Std. X", ylab="y",
     main="Polynomial basis", ylim=c(-1, 1), xlim=c(-1, 1.4))
abline(h=0, lty=1)
lines(x,x^2, lty=3)
lines(x,x^3, lty=4)
lines(x,x^4, lty=5)
legend("bottomright", paste("Degree", 0:4), lty=1:5, cex=0.8, inset=0.02)

lm1 <- lm(Interval ~ poly(StdD, 4, raw=TRUE), oldfaith)
plot(Interval ~ StdD, oldfaith, main="Quartic fit",
     ylim=c(-20, 100), xlim=c(-1, 1.4))
lines(sort(StdD), predict(lm1, newdata=data.frame(StdD=sort(StdD))))

b <- coef(lm1)
#plot(x,b[2]*x,type="l",lty=2,xlab="Std. X",ylab="y",main="Polynomial basis",
#     ylim=c(-25,75))
lines(x,b[2]*x,lty=2)
abline(h=b[1],lty=1)
lines(x,b[3]*x^2, lty=3)
lines(x,b[4]*x^3, lty=4)
lines(x,b[5]*x^4, lty=5)
legend("bottomright", paste("Degree",0:4),lty=1:5, cex=.8)
```
The polynomial basis functions are defined on the whole interval \((-1, 1)\) and so they may not be very useful for modeling local features in a function. Other basis functions can be defined so that they are zero or nearly zero except for a neighborhood of \((-1, 1)\). For example, the B-splines are nonlinear functions of the range of \(x\); the number of knots (or the number of splines) and (3) the placement of the knots. The default behavior of the \texttt{bs} function is to select the knots to be equally spaced over the range of \(x\).

```r
library(splines)
matplot(x, bs(x, df=6), type="l", ylab="", col=1, lwd=2)
```

This first argument to \texttt{bs} is the value of the predictor, which is used to determine the range. The second default argument is the degrees of freedom. The result of the call to \texttt{bs} is a matrix with rows equal to the length of \(x\) and \(df\) columns. We will only work with
df to control the values of the splines. One could also work with the number of internal knots, a function of the df, and use non-equally spaced knots, which is the default in `bs`.

```r
bspline.predictors <- bs(StdD, 6)
print(bspline.predictors[1:5,], digits=3)
```

```
    1       2       3       4       5      6
[1,] 0.00606 0.1736 0.58185 0.238505 0.000000 0.00000
[2,] 0.65368 0.0733 0.00205 0.000000 0.000000 0.00000
[3,] 0.02806 0.3000 0.54331 0.128617 0.000000 0.00000
[4,] 0.47906 0.4396 0.08117 0.000129 0.000000 0.00000
[5,] 0.00000 0.0000 0.08846 0.642349 0.26800137
```

```r
sm1 <- lm(Interval ~ bs(Duration, 6), oldfaith)
plot(Interval ~ Duration, oldfaith, main="Regression splines")
or <- order(oldfaith$Duration)
lines(predict(sm1)[or]~Duration[or], oldfaith)
```
Here is another example, that uses the \texttt{Orthodont} data frame from the \texttt{nlme} package. The data consist of measures of distance between teeth as a function of age for a number of boys and girls.

```r
data(Orthodont, package="nlme") # get data from nlme package, don't load package
library(lattice)
print(xyplot(distance~age|Sex, data=Orthodont, group=Subject, type=c("g","l")))
```

Although the individual growth curves look fairly linear, we will use splines to fit as a function of age. Since the time series are so short, only 7 observations per subject, we can only set \texttt{df} to 3, which is the smallest value possible.

```r
library(lme4)
m1 <- lmer(distance ~ Sex*bs(age, 3) + (1 + bs(age, 3)|Subject), data=Orthodont, REML=FALSE)
m2 <- lmer(distance ~ Sex + bs(age, 3) + (1 + bs(age, 3)|Subject), data=Orthodont, REML=FALSE)
anova(m1, m2)
```

Data: Orthodont

Models:
m2: distance ~ Sex + bs(age, 3) + (1 + bs(age, 3) | Subject)
\texttt{m1: distance ~ Sex * bs(age, 3) + (1 + bs(age, 3) | Subject)}

\begin{tabular}{lrrrrr}
 Df & AIC & BIC & logLik & Chisq & Chi Df & Pr(>Chisq) \\
\hline
 m2 & 16 & 457 & 500 & -212 & \\
 m1 & 19 & 455 & 505 & -208 & 8.13 & 3 & 0.043 \\
\end{tabular}

\texttt{AIC(m1, m2)}

\begin{tabular}{lrr}
 df & AIC \\
 m1 & 19 & 454.5 \\
 m2 & 16 & 456.6 \\
\end{tabular}

\texttt{Anova(m1)}

\textbf{Analysis of Deviance Table (Type II Wald chisquare tests)}

\begin{tabular}{lrrr}
 Response: distance & Chisq & Df & Pr(>Chisq) \\
\hline
 Sex & 8.34 & 1 & 0.0039 \\
 bs(age, 3) & 127.80 & 3 & <2e-16 \\
 Sex:bs(age, 3) & 9.49 & 3 & 0.0234 \\
\end{tabular}

We fit mixed-models with the same random effects, separate intercepts and coefficients for the splines, for each subject. In \texttt{m1} each \texttt{SEX} has its own intercept and \texttt{age} effects; in \texttt{m2} the age effects are common for the two sexes. The \texttt{anova} produces a likelihood ratio test to compare the two models (I suppressed REML fitting for this test), and the \texttt{AIC} reinforces the improvement gained by fitting separate age effects for each sex. I’ve also shown a Wald test of the hypothesis of no \texttt{Sex:age} interaction using the \texttt{Anova} command in \texttt{car}. The \texttt{lme4} doesn’t provide such tests.

The \texttt{effects} package in R allows drawing a plot that summarizes the \texttt{Sex:bs(age, 3)} effect. Unfortunately there is a bug in effects that I introduced this summer, so to draw the effects plot you need to install the working version of \texttt{effects}:

\texttt{detach(package:effects)} # you can't update a package that is in use
\texttt{install.packages("effects", repos="http://r-force.r-project.org")}
\texttt{library(effects)} # loads the new version

\texttt{plot(effect("Sex:bs(age, 3)", m1, xlevels=list(age=20)), multiline=TRUE,}
\texttt{ key.args=list(x=.02, y=.98), grid=TRUE)}
Writing \( y \) for the response distance and \( x \) for age, \( f \) for the dummy variable for Sex, and \( b_j(x) \) for the \( j \) b-spline basis function, we have fit

\[
E(y|x, f) = \beta_0 + \beta_f f + \beta_1 b_1(x) + \beta_2 b_2(x) + \beta_3 b_3(x) + \beta_4 b_1(x)f + \beta_5 b_2(x)f + \beta_6 b_3(x)f
\]

Of interest in growth curves is the derivative of the fitted curve, giving the rate of change at each age:

\[
\frac{dE(y|x, f)}{dx} = \beta_1 b_1'(x) + \beta_2 b_2'(x) + \beta_3 b_3'(x) + \beta_4 b_1'(x)f + \beta_5 b_2'(x)f + \beta_6 b_3'(x)f
\]

where the primes indicate differentiation. If we can compute the derivatives of the b-spines we can get the rate function without much work. It turns out that \texttt{bs} contained code that computed the derivatives of the b-splines but did not return them; they were computed for a different purpose. I modified the function to return the derivatives, in a function I call \texttt{mybs}.

```r
# look at the derivative:
source("http://www.stat.umn.edu/~sandy/courses/8053/Data/mybs.R")
```

To get the first derivative, set up the derivative of the splines for the original data, and then evaluate them at the points that will be used for plotting. The \texttt{bs}, and hence \texttt{mybs}, have a \texttt{predict} method that evaluates the splines (or their derivatives) at arbitrary points.
deriv1 <- with(Orthodont, mybs(age, 3, deriv=1))
ages <- seq(8, 14, length=100)
dmale <- predict(deriv1, ages) %*% fixef(m1)[3:5]
dfemale <- predict(deriv1, ages)[, c(1, 2, 3, 1, 2, 3)] %*% fixef(m1)[3:8]
plot(ages, dmale, type="l", lwd=2, col="blue", ylim=c(min(dmale, dfemale),
max(dmale, dfemale)), xlab="Age", ylab="Deriv. of Fit distance",
main="Derivative of fitted distance")
lines(ages, dfemale, type="l", lwd=2, col="red", lty=2)
legend("topleft", inset=.01, c("Male", "female"), lty=c(1,2),
lwd=2, col=c("blue", "red"))