Nonparametric Statistics - Epiphany term 2025

Jochen Einbeck

2025-03-20

1 Resampling methods

Michaelmas term material

2 Bayesian nonparametrics

Michaelmas term material

3 Density estimation

3.1 Introduction

Let X be a continuous random variable (r.v.) with cumulative distribution function (c.d.f.) $F(x) = P(X \leq x)$ and probability density function (p.d.f., “density”) $f = F'$. We are given data $x_1, \dots, x_n \in \mathbb{R}$, which is considered an independent and identically distributed (i.i.d.) sample from X.

Task: Find an estimate of f.

Starting point: For any (fixed) $x \in \mathbb{R}$,

$$f(x) = \frac{d}{dx}F(x) = \lim_{h \longrightarrow 0} \frac{F(x+h) - F(x-h)}{2h}.$$

Note that this is a functional of F, so we can estimate it by replacing F by its empirical distribution function (e.d.f.)

$$\hat{F}_n(x) = \frac{1}{n}\sum_{i=1}^{n} 1_{\{x_i \leq x\}} = \frac{\{\# x_i : x_i \leq x\}}{n}.$$

So, for small $h > 0$,

$$\hat{f}_n(x) = \frac{1}{2h}\frac{\{\# x_i : x_i \in (x-h, x+h]\}}{n}$$

Example 3.1

Climate data from Durham University Observatory (Burt & Burt, 2022).

Daily maximum temperature data for 2021 ($n = 363$).

```
temp.data0<- read.table("https://www.maths.dur.ac.uk/users/jochen.einbeck/Data/temp.dat")
```

```
dim(temp.data0)
```

```
## [1] 365   1
```

```
head(temp.data0)
```

```
##     x
## 1 5.5
## 2 2.4
## 3 5.3
## 4 3.9
## 5 3.8
## 6 2.3
```

```
sum(is.na(temp.data0))
```

```
## [1] 2
```

```
temp.data<-na.omit(temp.data0[,1])
head(temp.data)
```

```
## [1] 5.5 2.4 5.3 3.9 3.8 2.3
```

```
length(temp.data)
```

```
## [1] 363
```

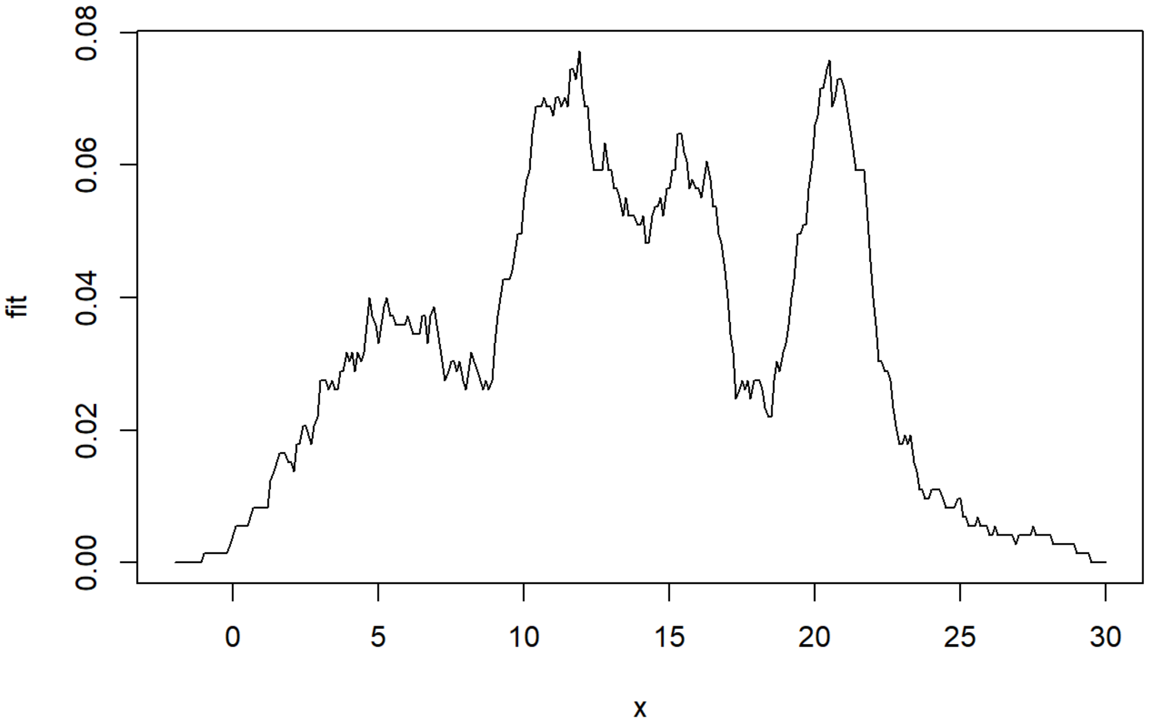
```
summary(temp.data)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   -0.10    9.65   13.50   13.62   19.00   28.50
```

Implement $\hat{f}_n(x)$ as inspired above:

```
my.density<- function(data,x,h){
  n<-length(data)
  denom<- 2*n*h
  num<- sum(data> x-h & data <= x+h)
  return(num/denom)
}
```

```
x<- seq(-2,30, by=0.1)
G<-length(x)
fit<- rep(0,G)
for (j in 1: G){
  fit[j]<- my.density(temp.data,x[j],1)
}
plot(x, fit, type="l" )
```



This looks like a reasonably try, as it gives an impression of the shape of the underlying, true density (f), of the data. However, it is very jagged. Usually "nature is smooth". Our density estimate should hence not appear as erratic.

There does exist a well-established graphical device, familiar to all of us, which can be used to obtain more "stable-looking" density estimates: the histogram. We'll look at this traditional tool first, before we consider more advanced, modern density estimators.

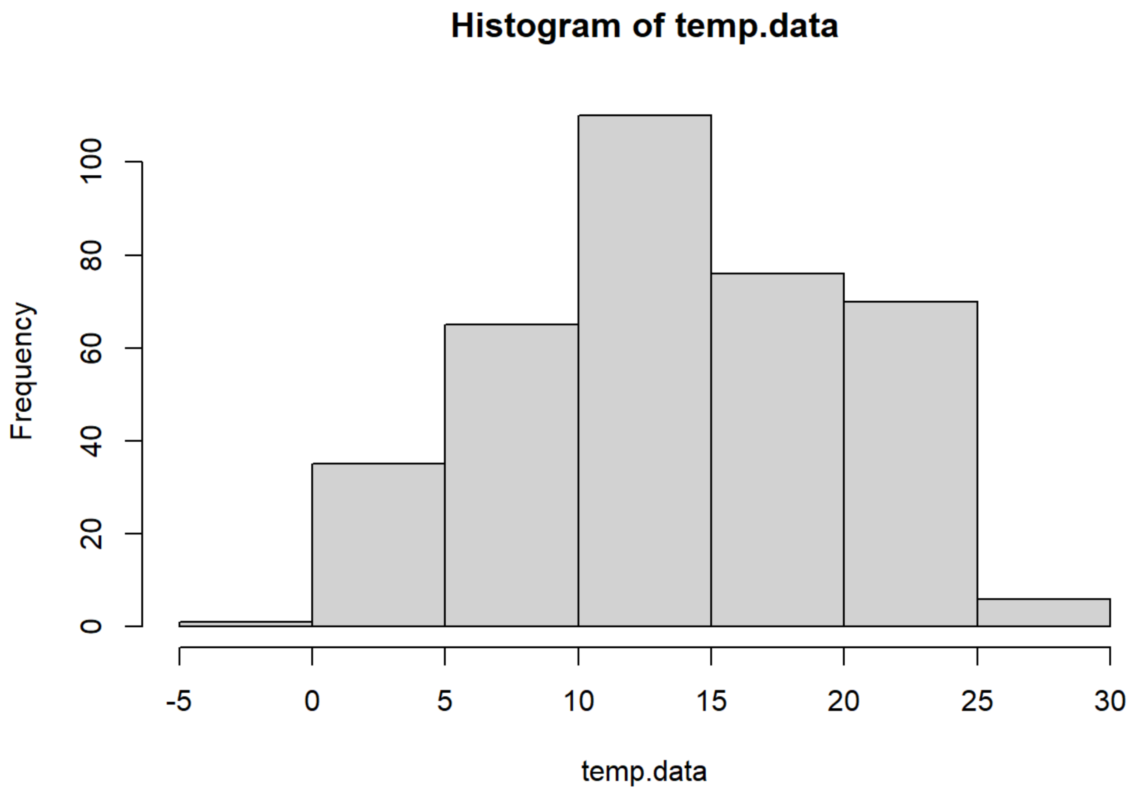
3.2 The histogram

The histogram is the oldest and most widely used density estimator.

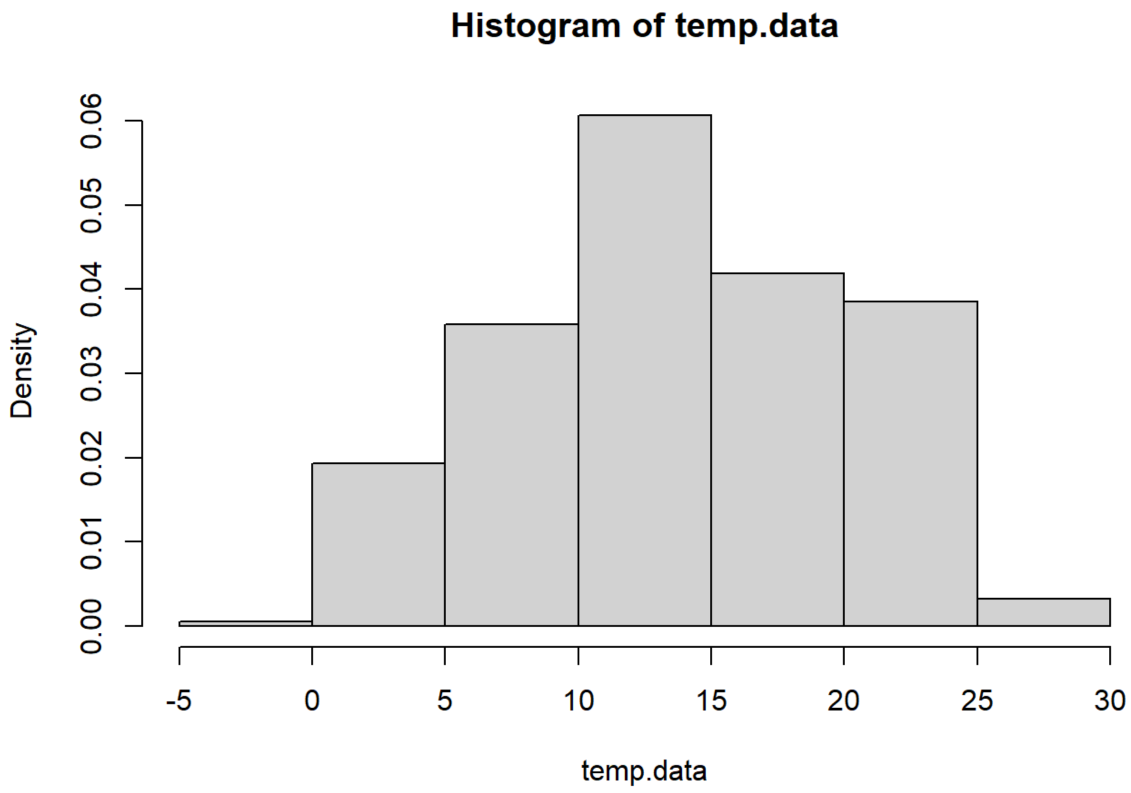
Basic idea: Rather than counting the data falling into $(x-h, x+h]$ for *all* possible values x, consider just a fixed set of bins of width $b(=2h)$.

Example 3.2

```
hist(temp.data)
```



```
hist(temp.data, freq=FALSE)
```



3.2.1 Constructing a histogram

- Divide the real line into equally sized intervals of fixed binwidth b.
- Count the number of observations falling into each bin, say n_j observations in the jth bin.
- Then the height of the jth bin is given by

$$f_j = \frac{n_j}{nb}$$

  (Why actually? Note that we would like the jth bin to represent the fraction n_j/n of the probability mass. Hence we equate $f_j \times b = n_j/n$, and solving this w.r.t f_j gives the equation above.)
- As a function of x,

$$\hat{f}_b(x) = \frac{\{\text{\# observations in bin containing } x\}}{nb}.$$

One can interpret this as stacking blocks of height $\frac{1}{nb}$ and width b on top of each other; that is one block for each observation.

For our developments we need a yet slightly more formal description of the histogram. For this, denote by x_0 the “origin” of the histogram, that is an arbitrary point on the real line where we put the left border of the “first” bin. The jth bin is then given by

$$B_j = (x_0 + (j-1)b, x_0 + jb]$$

for $j \in \mathbb{Z}$.

That is, for $x \in B_j$, we have

$$\hat{f}_b(x) = \frac{1}{nb} \sum_{i=1}^{n} 1_{B_j}(x_i)$$

where the indicator function is defined as

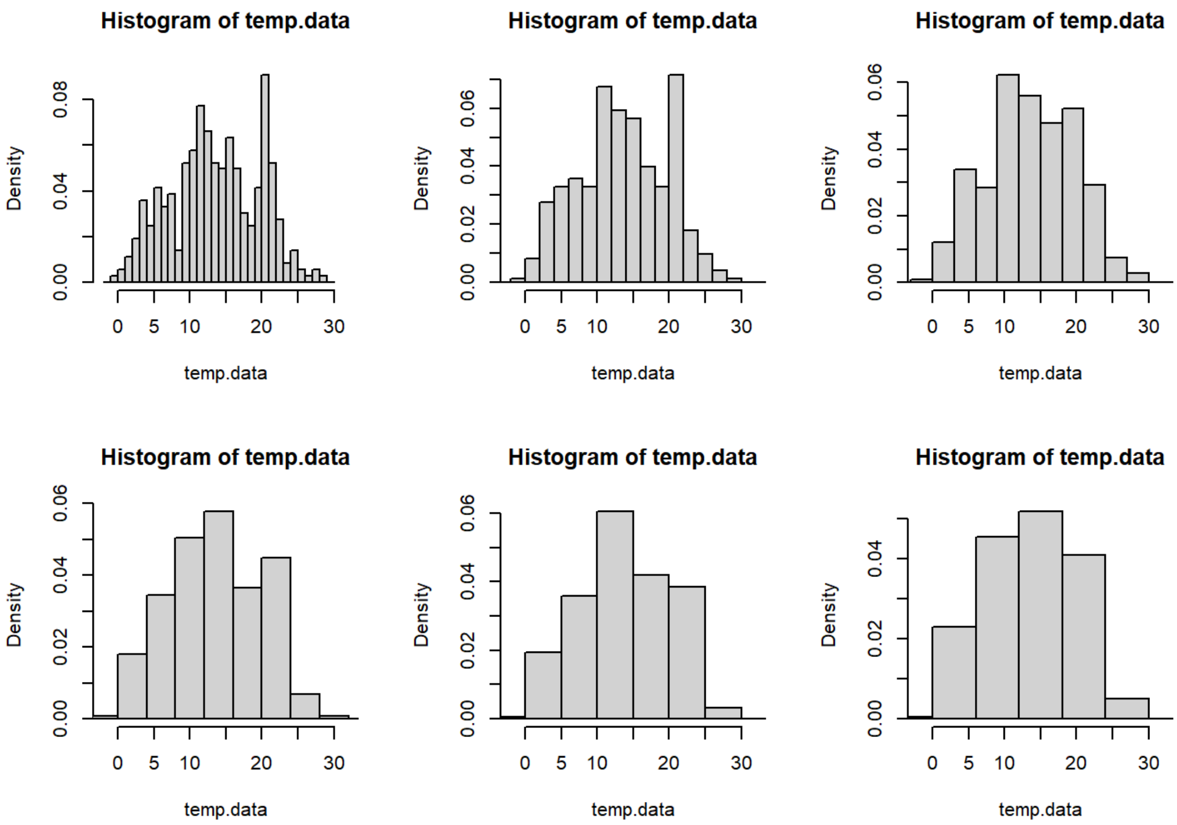
$$1_{B_j}(x_i) = \begin{cases} 1 & \text{if } x_i \in B_j \\ 0 & \text{otherwise} \end{cases}$$

3.2.2 Impact of b and x_0:

Example 3.3

- Keep $x_0 = 0$, vary $b = 1, \ldots 6$.

```r
par(mfrow=c(2,3))
x0<-0
bvec<-1:6
for (j in 1:6){
b<-bvec[j]
index<- -2:30
breaks= x0+ index*b
breaks
hist(temp.data, breaks=breaks, freq=FALSE, xlim=c(-2,32))
}
```



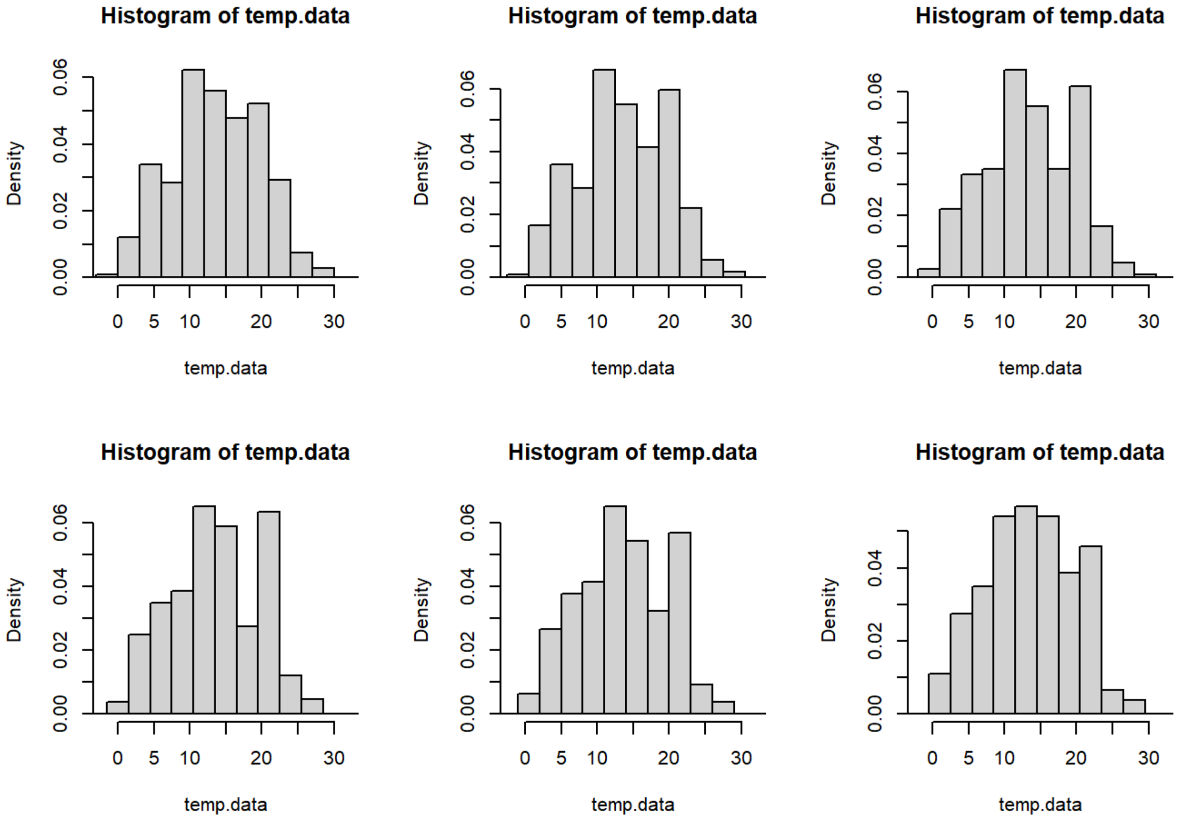
We see that

- for small values of b, the histogram appears very jagged.
- for high values of b, the histogram appears "smooth" (apart from the discontinuities due to the binning itself).

This is a typical case of a **bias-variance trade-off**: Small b corresponds to low bias/high variance, and large b corresponds to large bias/small variance. The binwidth b plays the role of a "smoothing parameter" for the histogram. As we will see, it is possible to optimize it by calibrating bias and variance in a principled way (Section 3.2.5).

- Now we keep $b = 3$, and vary $x_0 = 0, 0.5, \ldots, 2.5$.

```
par(mfrow=c(2,3))
b<-3
x0vec<-seq(0,2.5, by=0.5)
for (j in 1:6){
x0<-x0vec[j]
index<- -2:30
breaks= x0+ index*b
breaks
hist(temp.data, breaks=breaks, freq=FALSE, xlim=c(-2,32))
}
```



The quantity x_0 is a purely technical parameter which cannot be optimized. One can avoid the requirement to make this choice only by using an entirely different density estimation technique which does not require binning, and which is actually “smooth” (Section 3.3).

3.2.3 Bias and Variance of histogram

Without loss of generality (w.l.o.g.) say $x \in B_j$. We are interested in properties of $\hat{f}_b(x) = \frac{1}{nb}\sum_{i=1}^{n} 1_{B_j}(x_i)$ as an estimator of $f(x)$.

The key insight for this analysis is as follows: The random variable $1_{B_j}(x_i)$ is a Bernoulli variable with success probability

$$p_j = P(x_i \in B_j) = \int_{B_j} f(u)\,du = \int_{x_0+(j-1)b}^{x_0+jb} f(u)\,du$$

so that

$$E(1_{B_j}(x_i)) = p_j$$

and

$$\mathrm{Var}(1_{B_j}(x_i)) = p_j(1-p_j).$$

Hence, we have

$$\mathrm{Bias}(\hat{f}_b(x))) = E(\hat{f}_b(x)) - f(x) = E\left(\frac{1}{nb}\sum_{i=1}^{n} 1_{B_j}(x_i)\right) - f(x) =$$

$$= \frac{1}{nb}\sum_{i=1}^{n} E1_{B_j}(x_i) - f(x) = \frac{1}{nb}np_j - f(x) = \frac{p_j}{b} - f(x)$$

which can be interpreted as the difference between $f(x)$ and the average value of f within the interval. This is still not overly insightful. So let us try to look deeper into p_j by expanding $f(u)$ around the value x, for small binwidths $(b << 1)$. Recall that $x \in B_j$, $u \in B_j$. Then a Taylor expansion gives

$$f(u) \approx f(x) + f'(x)(u - x)$$

based on the reasoning that, for small b, the expression $(u - x)^2$ and all higher-order terms become “very small” and hence negligible. So,

$$\begin{aligned} p_j = \int_{x_0+(j-1)b}^{x_0+jb} f(u)\, du &\approx \int_{x_0+(j-1)b}^{x_0+jb} (f(x) + f'(x)(u-x))\, du \\ &= bf(x) + f'(x)\left[\frac{u^2}{2} - ux\right]_{x_0+(j-1)b}^{x_0+jb} = \ldots = \\ &= bf(x) + bf'(x)\left[(x_0 + (j-1/2)b)b - xb\right] = \\ &= bf(x) + bf'(x)(m_j - x) \end{aligned}$$

where we have identified that $m_j \equiv x_0 + (j - 1/2)b$ is just the mid-point of the interval B_j.

Hence,

$$\begin{aligned} \text{Bias}(\hat{f}_b(x))) &\approx \frac{1}{b}(bf(x) + bf'(x)(m_j - x)) - f(x) \\ &= f(x) + f'(x)(m_j - x) - f(x) \\ &= f'(x)(m_j - x) \end{aligned}$$

From this we can see that

- there is approxmately no bias at the mid-points m_j.
- the bias is the larger, the larger the gradient of the density.

A pragmatic approximation of this bias term, which makes the dependency on the binwidth b explicit, is given by

$$|\text{Bias}(\hat{f}_b(x))| \approx |f'(x)||m_j - x| \leq \frac{b}{2}|f'(x)|$$

Hence, this gives an approximate measure of the “worst possible” bias, which is attained around the end points of the bins. We observe that from this perspective, small binwidths appear preferable to large binwidths. However, this is unlikely to suggest that small binwidths are preferable *per se*, since, as motivated earlier, the variance is likely to show the opposite behavior. So, let’s work this variance out. This is now quite straightforward as we have done all the hard work already.

$$\begin{aligned} \text{Var}(\hat{f}_b(x)) &= \frac{1}{n^2b^2}\sum_{i=1}^{n}\text{Var}(1_{B_j}(x_i)) = \frac{1}{n^2b^2}np_j(1-p_j) = \frac{1}{nb^2}p_j(1-p_j) \\ &\approx \frac{1}{nb}(f(x) + bf'(x)(m_j - x))(1 - bf(x) - bf'(x)(m_j - x)) \end{aligned}$$

Hence, based on similar reasoning as above, we get an approximate quantification of the maximum variance as

$$\begin{aligned} \text{Var}(\hat{f}_b(x)) &\leq \frac{1}{nb}(f(x) + |f'(x)|\frac{b}{2})(1 - b(\ldots)) \\ &\approx \frac{1}{nb}f(x) \end{aligned}$$

We find that variance tends to zero if b tends to infinity (or n tends to infinity), indicating that a bias-variance trade-off between bias and variance is present. In the next section we aim to derive an optimal binwidth b by exploiting this trade-off.

This set of calculations was admittedly a bit adventurous, but it sets out the philosophy of the calculations which we will carry out repeatedly in this term: Under the assumption of small binwidth (or other smoothing parameters, such as bandwidth), approximate bias and variance terms through Taylor expansions around the target point (x), and then omit all terms except the leading term. The resulting expressions give approximations for the bias and variance under small b (and, sometimes, large n, but so far we did not do this).

3.2.4 MSE and consistency

From the approximate expression for (maximum) bias and variance, we obtain the (maximum) mean squared error (MSE) as

$$\begin{aligned}\text{MSE}(\hat{f}_b(x)) &= E\left[(\hat{f}_b(x) - f(x))^2\right] = \ldots = \\ &= \text{Bias}^2(\hat{f}_b(x)) + \text{Var}(\hat{f}_b(x)) \\ &\approx \frac{b^2}{4}(f'(x))^2 + \frac{1}{nb}f(x)\end{aligned}$$

So $\hat{f}_b(x)$ is **mean square consistent** for f(x) if $b = b_n$ is a sequence such that, if $n \longrightarrow \infty$,

i. $b_n \longrightarrow 0;$
ii. $nb_n \longrightarrow \infty.$

Does there exist such a sequence?

Let us define $M(b) = \frac{b^2}{4}(f'(x))^2 + \frac{1}{nb}f(x)$ and minimize this quantity w.r.t b.

So,

$$\begin{aligned}\frac{dM(b)}{db} &= \frac{b}{2}(f'(x))^2 - \frac{1}{nb^2}f(x) \overset{!}{=} 0 \\ \frac{b^3}{2}(f'(x))^2 &= \frac{1}{n}f(x) \\ b^3 &= \frac{2}{n}\frac{f(x)}{(f'(x))^2} \\ b &= \left[2\frac{f(x)}{(f'(x))^2}\right]^{1/3} n^{-1/3}\end{aligned}$$

So let's check. Call $c(x) \equiv \left[2\frac{f(x)}{(f'(x))^2}\right]$ for ease of notation.

i. $c(x)\frac{1}{n^{1/3}} \longrightarrow 0$ for $n \longrightarrow \infty$

ii. $n \times c(x)\frac{1}{n^{1/3}} = c(x)n^{2/3} \longrightarrow \infty$ for $n \longrightarrow \infty$

Hence, with $b = b_n = c(x)n^{-1/3}$, we have

$$\text{MSE}(\hat{f}_b(x)) \longrightarrow 0 \quad (n \longrightarrow \infty)$$

so $\hat{f}_b(x)$ is mean squared consistent for $f(x)$.

3.2.5 MISE and optimal binwidth

The result from Section 3.2.4 is still not useful: We want **one** optimal binwidth which does not depend on x.

To achieve this, we can consider the mean integrated squared error:

$$
\begin{aligned}
\text{MISE}(\hat{f}_b) = \int_x \text{MISE}(\hat{f}_b(x))\,dx &\approx \int \left(\frac{b^2}{4}(f'(x))^2 + \frac{1}{nb} f(x) \right)\,dx \\
&= \frac{b^2}{4} \int (f'(x))^2\,dx + \frac{1}{nb} \int f(x)\,dx \\
&= D_1 \frac{b^2}{4} + \frac{1}{nb}
\end{aligned}
$$

with constant $D_1 \equiv \int f'(x)^2\,dx$, and noting that $\int f(x)\,dx = 1$. Let us now denote $\tilde{M}(b) = D_1 \frac{b^2}{4} + \frac{1}{nb}$ and we minimize this expression w.r.t. b as before.

Then

$$
\begin{aligned}
\frac{d\tilde{M}(b)}{db} &= b\frac{D_1}{2} - \frac{1}{nb^2} \overset{!}{=} 0 \\
\frac{b^3 D_1}{2} &= \frac{1}{n} \\
b &= \left(\frac{2}{D_1} \right)^{1/3} n^{-1/3}
\end{aligned}
$$

Problem: We cannot compute b since f is unknown and hence $D_1 = \int f'^2$ is unknown too. A solution is known as the “normal reference” method, where one takes, only for the sake of computing D_1, a normal distribution as a proxy for f (Of course, this is not meaning that we assume f to have *really* a normal distribution — if we did so then the entire exercise of computing a histogram density would be a bit pointless!).

So, let us take (for this purpose only) $f \sim N(0, \sigma^2)$. Let $f^{(j)}(x)$ denote the jth derivative of f. Denote also by ϕ the density of $N(0,1)$ and by $\phi^{(j)}$ its j-th derivative. For our calculations, we make use of the result (without proof, but it is easy to proof), that

$$
\int (f^{(j)}(x))^2\,dx = \frac{1}{\sigma^{2j+1}} \int (\phi^{(j)}(x))^2\,dx
$$

With this, we use

$$
\begin{aligned}
\phi(x) &= \frac{1}{\sqrt{2\pi}} \exp(-\frac{x^2}{2}) \\
\phi^{'}(x) &= -\frac{1}{\sqrt{2\pi}} x \exp(-\frac{x^2}{2})
\end{aligned}
$$

and so can work out

$$
\begin{aligned}
\int (\phi^{'}(x))^2\,dx &= \int x^2 \frac{1}{2\pi} \exp(-x^2)\,dx \\
&= \frac{1}{\sqrt{2\pi}} \sqrt{\frac{1}{2}} \int x^2 \frac{1}{\sqrt{1/2}} \frac{1}{\sqrt{2\pi}} \exp\left(\frac{x^2}{2 \times \frac{1}{2}} \right)\,dx \\
&= \frac{1}{\sqrt{4\pi}} \times \frac{1}{2} = \frac{1}{4\sqrt{\pi}}
\end{aligned}
$$

where the equality sign leading to the last row makes use of the insight that the integral in the previous row is just the variance of a $N(0, 1/2)$ distribution.

Hence with the helper tool above we have

$$
\int (f'(x))^2\,dx = \frac{1}{\sigma^3} \frac{1}{4\sqrt{\pi}}
$$

so that we are able to derive the MISE-optimal binwidth

$$b_{opt} = \left(\frac{2}{\frac{1}{4\sqrt{\pi}} \frac{1}{\sigma^3}} \right)^{\frac{1}{3}} n^{-\frac{1}{3}} = (8\sqrt{\pi})^{1/3} \sigma n^{-1/3} = 2.42 \sigma n^{-1/3},$$

where σ is usually estimated by the sample standard deviation, s.

Note that Scott (2015) obtains an equation involving a slightly larger constant, $3.5\sigma n^{-1/3}$, by employing a notion of “average bias” rather than “maximum bias” as we have used. However, this is not really important: What is important is to understand how the optimal binwidth evolves with n.

Example 3.4

```
bopt<-function(vec){
  vec<-na.omit(vec)
  n<-length(vec)
  s<-sd(vec)
  b<- 2.42*s* n^(-1/3)
  return(b)
}
```

```
bopt(temp.data)
```

```
## [1] 2.074377
```

```
bscott<-function(vec){
  vec<-na.omit(vec)
  n<-length(vec)
  s<-sd(vec)
  b<- 3.5*s* n^(-1/3)
  return(b)
}
```

```
bscott(temp.data)
```

```
## [1] 3.000132
```

For further calculations let us denote

$$b_{opt} = c \sigma n^{-1/3}$$

with c chosen accordingly. The optimal binwidth is hence (approximately)

$$\begin{aligned}
\text{MISE}_{opt}(b) &= D_1 \frac{b_{opt}^2}{4} + \frac{1}{n b_{opt}} \\
&= D_1 \frac{c^2 \sigma^2 n^{-2/3}}{4} + \frac{1}{n c \sigma n^{-1/3}} \\
&= D_1 \frac{c^2 \sigma^2}{4} n^{-2/3} + \frac{1}{c\sigma} n^{-2/3} \\
&= O(n^{-2/3})
\end{aligned}$$

We say that the optimal MISE of histograms is of “order” $n^{-2/3}$, which mathematically means that the MISE, as a function of n, fulfils

$$\limsup_{n\longrightarrow\infty}\left|\frac{\mathrm{MISE}_{opt}(b)}{n^{-2/3}}\right|<\infty$$

In more simple words, $\mathrm{MISE}_{opt}(b)$ converges to 0 not slower than $n^{-2/3}$. We would want $\mathrm{MISE}_{opt}(b)$ to converge to 0 “quickly”, so orders n^{-r} for “large” r are preferred. We can expect that any nonparametric estimator will not converge quicker than parametric ones: Parametric estimators are usually unbiased and achieve n^{-1} rates by the Cramer-Rao bound. However, we will also see that there is quite some space between the rates $n^{-2/3}$ and n^{-1} and we will find efficient density estimators which will fill this space.

3.3 Kernel density estimation

3.3.1 Kernel functions

Recall, from Section 3.1,

$$\begin{aligned}\hat{f}_n(x)&=\frac{1}{2h}\frac{\{\#x_i : x_i\in(x-h,x+h]\}}{n}\\&=\frac{1}{nh}\sum_{i=1}^{n}\frac{1}{2}1_{\{(x-h,x+h)\}}(x_i)\\&=\frac{1}{nh}\sum_{i=1}^{n}K\left(\frac{x_i-x}{h}\right)\end{aligned}$$

where we have defined

$$K(u)=\frac{1}{2}1_{\{(-1,1)\}}(u)$$

which is a particular case of a **kernel** function (in this case the uniform kernel).

The uniform kernel is not a very good kernel. We can choose better ones which

- are (in some sense) smooth;
- give more weight to data points close to x;

for instance

- the Gaussian kernel

$$K(u)=\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{u^2}{2}\right)$$

- the Epanechnikov kernel

$$K(u)=\frac{3}{4}(1-u^2)1_{|u|\leq 1}$$

Formally, we call any function $K:\mathbb{R}\longrightarrow\mathbb{R}_0^+$ which is bounded, symmetric, unimodal, with $\int K(u)\,du=1$, a **kernel function** and

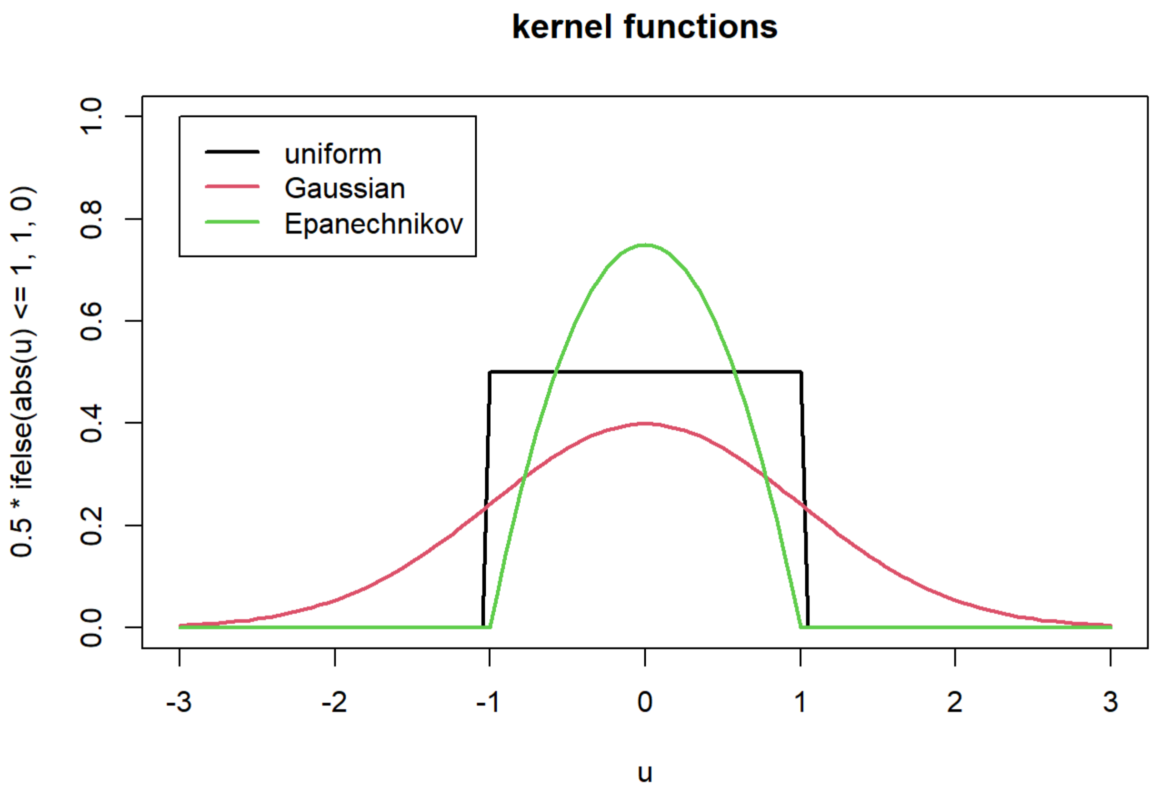
$$\hat{f}_h(x)=\frac{1}{nh}\sum_{i=1}^{n}K\left(\frac{x-x_i}{h}\right)$$

a **kernel density estimator** (KDE) (Note that, due to the symmetry of K, it is irrelevant whether we write $x-x_i$ or x_i-x in the numerator; the formulation $x-x_i$ is more common in the literature).

Example 3.5

We firstly visualize the three mentioned kernel functions.

```
u<-seq(-3,3, by=0.05)
plot(u, 0.5*ifelse(abs(u)<=1, 1,0), type="l", main="kernel functions", lwd=2, ylim=c(0,1))
lines(u, dnorm(u), lwd=2, col=2)
lines(u, 3/4*(1-u^2)*ifelse(abs(u)<=1,1,0), lwd=2, col=3)
legend(-3, 1, c("uniform", "Gaussian", "Epanechnikov"), lwd=c(2,2,2), col=c(1,2,3))
```



Then we implement a general version `my.density` of our kernel density estimator $\hat{f}_h(x)$, which can used in conjunction with any kernel `my.kernel`.

```
my.kernel<-function(u){
  #u<- 0.5*ifelse(abs(u)<=1, 1,0)
  u<- dnorm(u)
  #u<- 3/4*(1-u^2)*ifelse(abs(u)<=1,1,0)
}
```

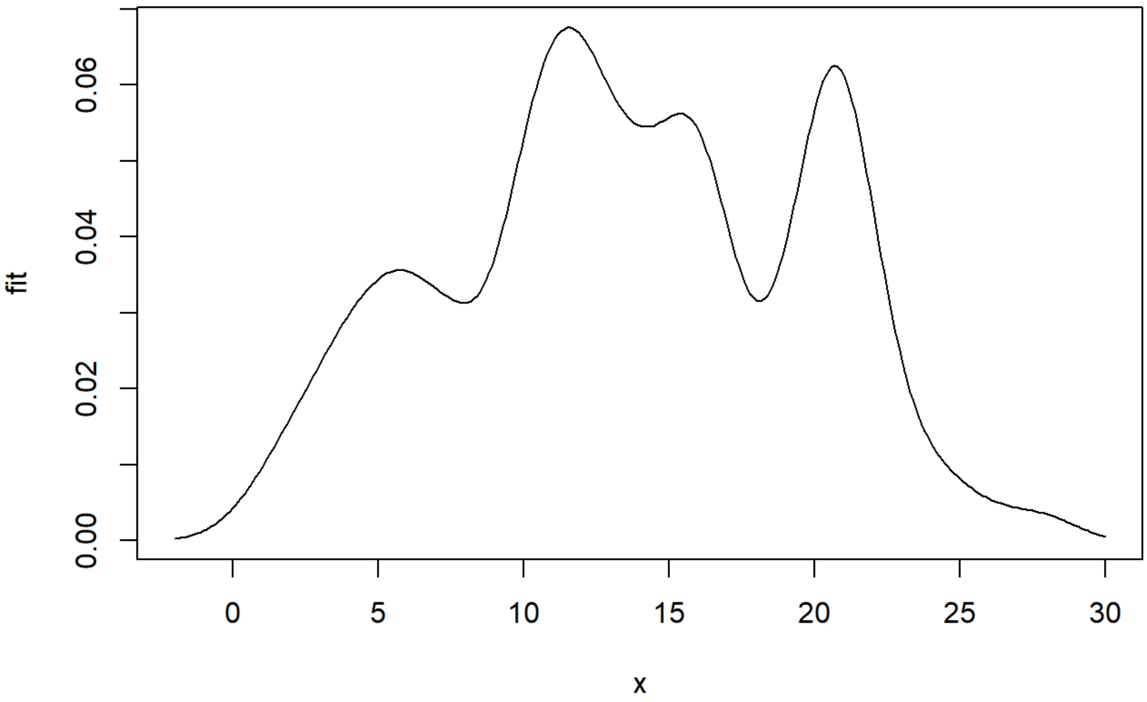
```
my.density<- function(data,x,h){
  data<-na.omit(data)
  n<-length(data)
  denom<- n*h
  num<- sum(my.kernel((data-x)/h))
  return(num/denom)
}
```

We apply the KDE now on the Durham temperature data, for each value of grid ranging from -2 to 30, spaced by 0.1.

For the Gaussian kernel, the result is:

```
x<- seq(-2,30, by=0.1)
G<-length(x)
fit<- rep(0,G)
for (j in 1: G){
  fit[j]<- my.density(temp.data,x[j],1) # uses h=1
}

plot(x, fit, type="l" )
```



This can be repeated for the other kernel functions by uncommenting the appropriate lines in the function `my.kernel`. For the uniform kernel, this will result in exactly the same estimator as previously shown in Example 3.1.

3.3.2 Basic features of KDEs

$\hat{f}_h(x)$ can be rewritten as

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}K\left(\frac{x-x_i}{h}\right) \equiv \frac{1}{n}\sum_{i=1}^{n}K_h(x-x_i)$$

where $K_h(\cdot) = \frac{1}{h}K\left(\frac{\cdot}{h}\right)$ is a centered (at x_i) and rescaled kernel with

$$\begin{aligned}\int K_h(x-x_i)\,dx &= \frac{1}{h}\int_{-\infty}^{\infty} K\left(\frac{x-x_i}{h}\right)\,dx \\ &= \frac{1}{h}\int_{-\infty}^{\infty} K(u)hdu = 1.\end{aligned}$$

With this we see

$$\int \hat{f}_h(x)\,dx = \int_{-\infty}^{\infty} \frac{1}{n}\sum_{i=1}^{n} K_h(x-x_i)\,dx$$
$$= \frac{1}{n}\sum_{i=1}^{n}\int K_h(x-x_i)dx = \frac{1}{n} \times n \times 1 = 1$$

so that the estimated density always integrates to 1.

Further, we see that $\hat{f}_h(x)$ is a sum of n ‘bumps’, positioned at the x_i, of size

$$\frac{1}{n}K_h(x-x_i),$$

each of these contributing a probability mass $1/n$. One can say that the KDE redistributes the point masses $\frac{1}{n}$ smoothly to the vicinity of each data point.

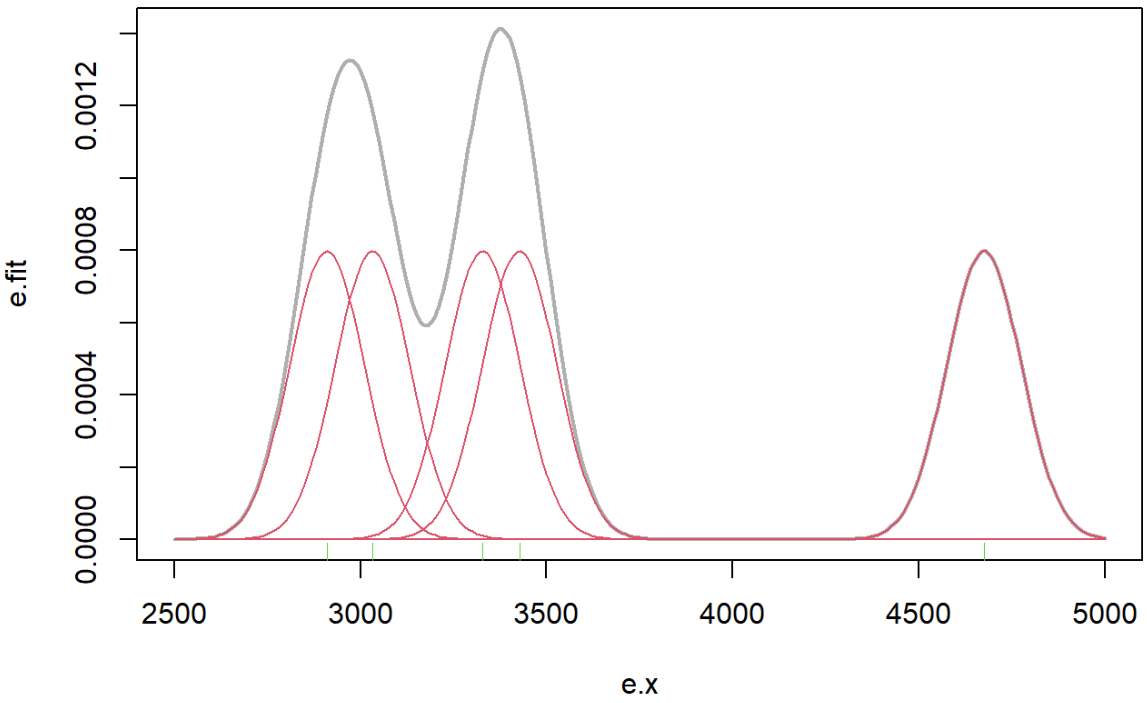
Example 3.6 (Basu’s elephants)

Five elephants were famously weighed in Basu’s (1971) elephant fable: Combo (4675kg), Flimbo (3032kg), Linda (3328kg), Pamela (3427kg), and Sara (2910kg). Let’s produce a KDE of these:

```r
elephants<- c(4675, 3032, 3328, 3427, 2910)
e.x <- seq(2500, 5000, by=10)
e.G    <- length(e.x)
e.fit  <- rep(0,e.G)
e.h    <- 100 # uses h=100 (arbitrary at this point)
for (j in 1: e.G){
  e.fit[j]<- my.density(elephants,e.x[j],e.h)
}

plot(e.x, e.fit, type="l", lwd=2, col="grey70" )
rug(elephants, col=3)

for (j in 1:5){
  lines(e.x, 1/(5*e.h)*my.kernel((e.x-elephants[j])/100), col=2)
}
```



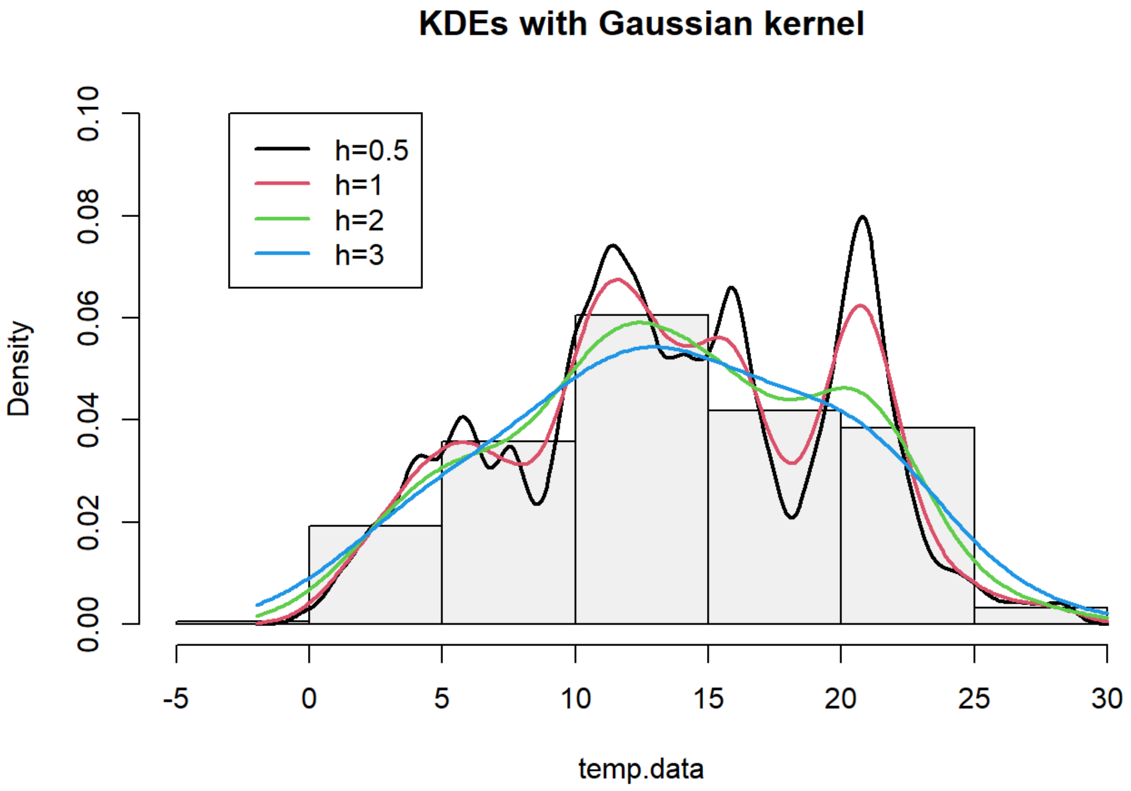
3.3.3 Effect of bandwidth

Example 3.7 (Back to Durham temperature data)

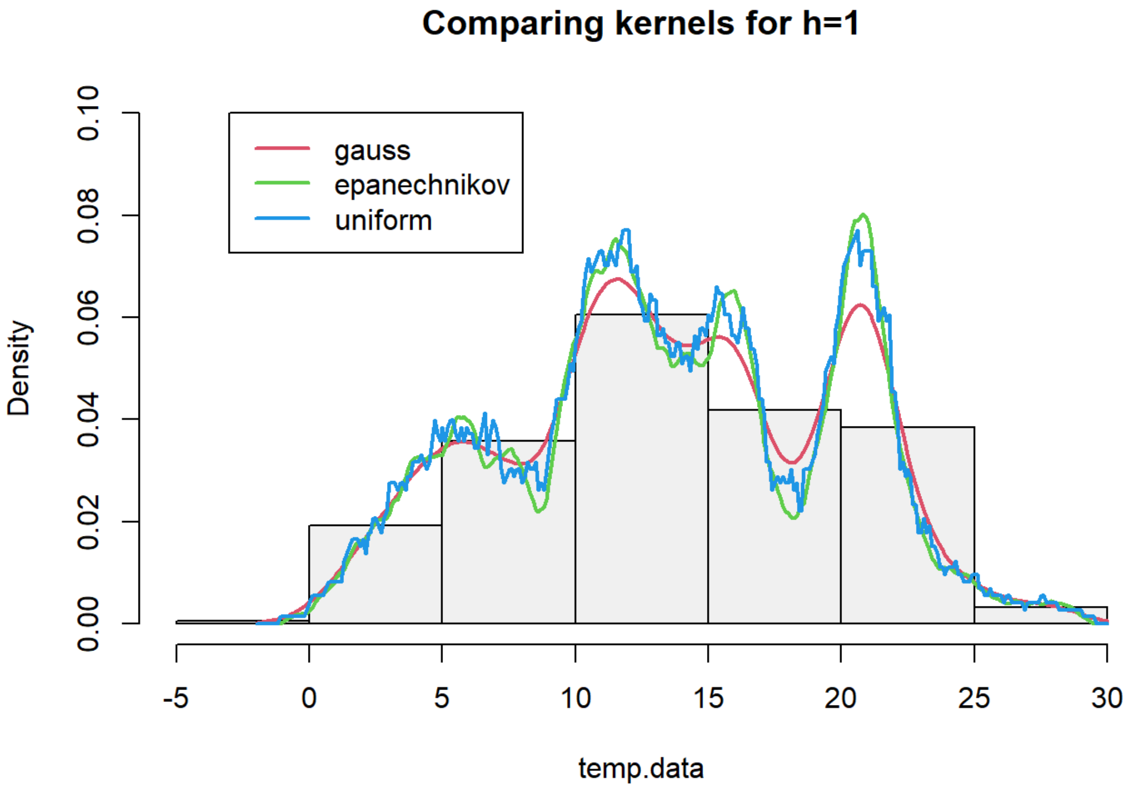
Firstly, let us put all our estimation components together, to have an easily usable function which does not require running iterations:

```
est.density<- function(data, xgrid=data, h=1, kernel="gauss"){
  my.kernel<- switch(kernel,
                     "gauss"= function(x){dnorm(x)},
                     "epa"=function(x){x<- 3/4*(1-x^2)*ifelse(abs(x)<=1,1,0)},
                     "uni"=function(x){x<- 1/2*ifelse(abs(x)<=1,1,0)}
  )
  data<-na.omit(data)
  n<-length(data)
  g<-length(xgrid)
  est<-rep(0, g)
  denom<- n*h
  for (j in 1:g){
    num<- sum(my.kernel((data-xgrid[j])/h))
    est[j]<- num/denom
  }
  return(est)
}
```

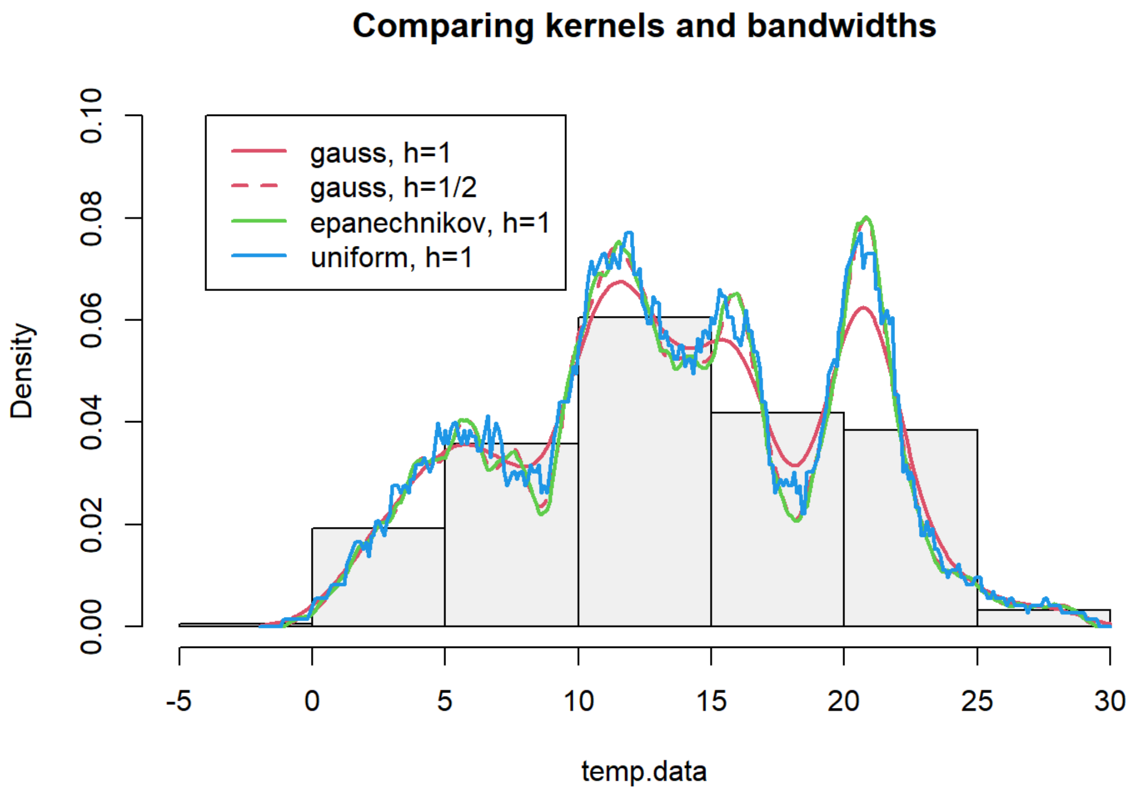
```
x<- seq(-2,30, by=0.1)
hist(temp.data, freq=FALSE, ylim=c(0,0.1), main="KDEs with Gaussian kernel",  col="grey95")
lines(x,est.density(temp.data, x, h=0.5, "gauss"), col=1, lwd=2)
lines(x,est.density(temp.data, x, h=1, "gauss"), col=2, lwd=2)
lines(x,est.density(temp.data, x, h=2, "gauss"), col=3, lwd=2)
lines(x,est.density(temp.data, x, h=3, "gauss"), col=4, lwd=2)
legend(-3, 0.1, c("h=0.5","h=1", "h=2", "h=3"), lty=c(1,1,1,1), lwd=c(2,2,2,2), col=c(1,2,3,4))
```



```
x<- seq(-2,30, by=0.1)
hist(temp.data, freq=FALSE, ylim=c(0,0.1), main="Comparing kernels for h=1", , col="grey95")
lines(x,est.density(temp.data, x, h=1, "gauss"), col=2, lwd=2)
lines(x,est.density(temp.data, x, h=1, "epa"), col=3, lwd=2)
lines(x,est.density(temp.data, x, h=1, "uni"), col=4, lwd=2)
legend(-3, 0.1, c("gauss", "epanechnikov", "uniform"), lty=c(1,1,1), lwd=c(2,2,2), col=c(2,3,4))
```



```
x<- seq(-2,30, by=0.1)
hist(temp.data, freq=FALSE, ylim=c(0,0.1), main="Comparing kernels and bandwidths", col="grey95")
lines(x,est.density(temp.data, x, h=1, "gauss"), col=2, lwd=2)
lines(x,est.density(temp.data, x, h=1/2, "gauss"), col=2, lwd=2, lty=2)
lines(x,est.density(temp.data, x, h=1, "epa"), col=3, lwd=2)
lines(x,est.density(temp.data, x, h=1, "uni"), col=4, lwd=2)
legend(-4, 0.1, c("gauss, h=1", "gauss, h=1/2", "epanechnikov, h=1", "uniform, h=1"), lty=c(1,2,
1,1), lwd=c(2,2,2,2), col=c(2,2,3,4))
```



The bandwidth, h, takes the role of a smoothing parameter, with

- h small $\longrightarrow$ low bias, high variance
- h large $\longrightarrow$ high bias, low variance

Something more subtle is going on when comparing different kernels: The same bandwidth does seem to imply a different degree of smoothness for different kernels. For instance, $h = 0.5$ for the Gaussian kernel seems to give a similar degree of smoothness as $h = 1$ for an Epanechnikov kernel.

We investigate the reason for this effect in the next subsection.

3.3.4 Comparing bandwidths across kernels

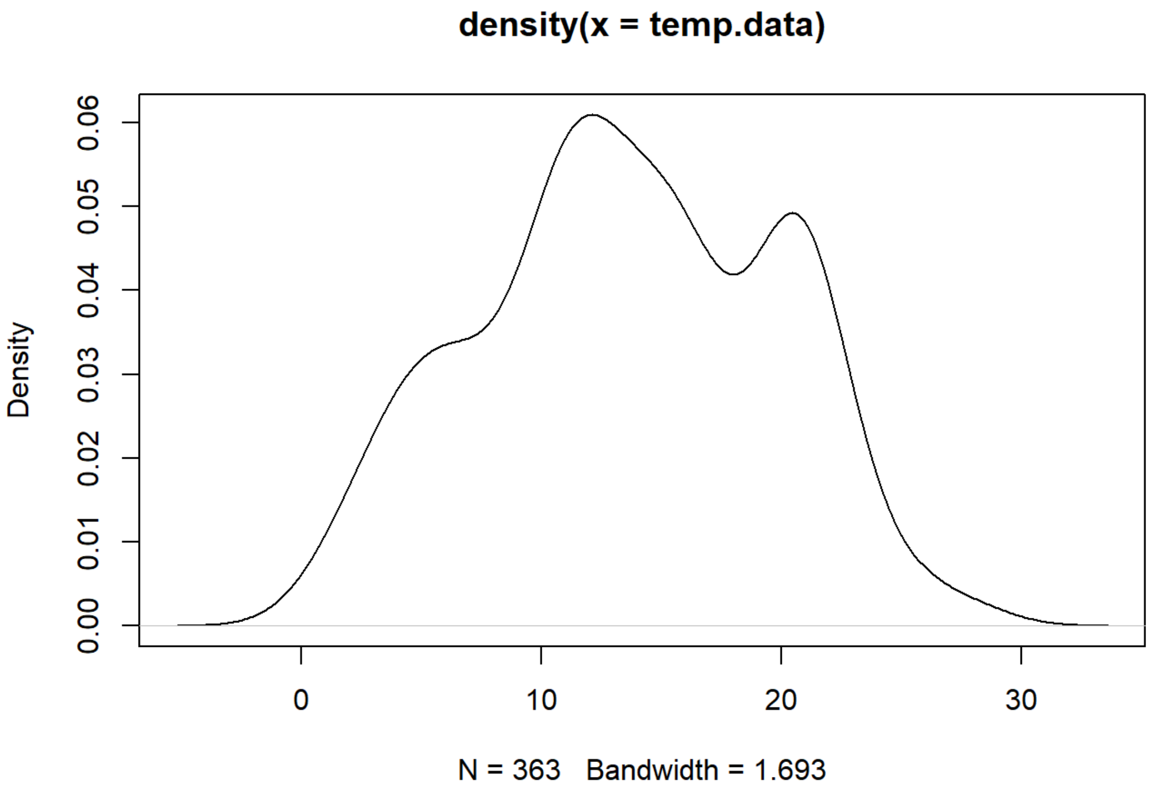
Example 3.8

Consider built-in R function `density` .

```
temp.dens<- density(temp.data)
temp.dens
```

```
##
## Call:
##  density.default(x = temp.data)
##
## Data: temp.data (363 obs.);  Bandwidth 'bw' = 1.693
##
##        x                 y
##  Min.   :-5.179   Min.   :0.0000106
##  1st Qu.: 4.511   1st Qu.:0.0030456
##  Median :14.200   Median :0.0267690
##  Mean   :14.200   Mean   :0.0257760
##  3rd Qu.:23.889   3rd Qu.:0.0452311
##  Max.   :33.579   Max.   :0.0609223
```

```
# note bw=h=1.693 (for Gauss kernel)
plot(temp.dens)
```



The function `density` uses the Gaussian kernel is used by default. We also see that the output returns a result `Bandwidth 'bw' = 1.693`. How does this relate to “our” bandwidth h?

To answer this question, from the help file, we find that “The kernels are scaled such that this is the standard deviation of the smoothing kernel.”

Recall that

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i)$$

is a sum of n ‘bumps’, positioned at the x_i. Here, the $K_h(x_i - x)$ correspond to what is meant with “smoothing kernel”.

These smoothing kernels are positioned at each x_i, and distribute the point mass of x_i in the vicinity of x_i. We are interested in the standard deviation of this “redistribution”. So, for fixed x_i, consider (for a moment) $K_h(x - x_i)$ as the density of a random variable $\tilde{X}$, evaluated at $\tilde{X} = x$. So, the task is to work out $SD(\tilde{X})$ for the various kernel choices.

Therefore, we find firstly that

$$\begin{aligned}
\mathrm{Var}(\tilde{X}) &= E\left[(\tilde{X} - E\tilde{X}))^2\right] \\
&= \int (x - x_i)^2 \frac{1}{h} K\left(\frac{x - x_i}{h}\right)\, dx \\
&= h \int \left(\frac{x - x_i}{h}\right)^2 K\left(\frac{x - x_i}{h}\right)\, dx \\
&= h^2 \int u^2 K(u)\, du
\end{aligned}$$

where we have used in the first row that $E(\tilde{X}) = x_i$ (obvious, as the bumps are symmetric and centered at the x_i), and later used the transformation $u = (x - x_i)/h$, which then implies $dx = h\, du$.

In the resulting expression, $\int u^2 K(u)\, du$ is a “second kernel moment” which is often tabulated, for instance in Table B.2 of Wand and Jones (1995). For ease of access displayed here:

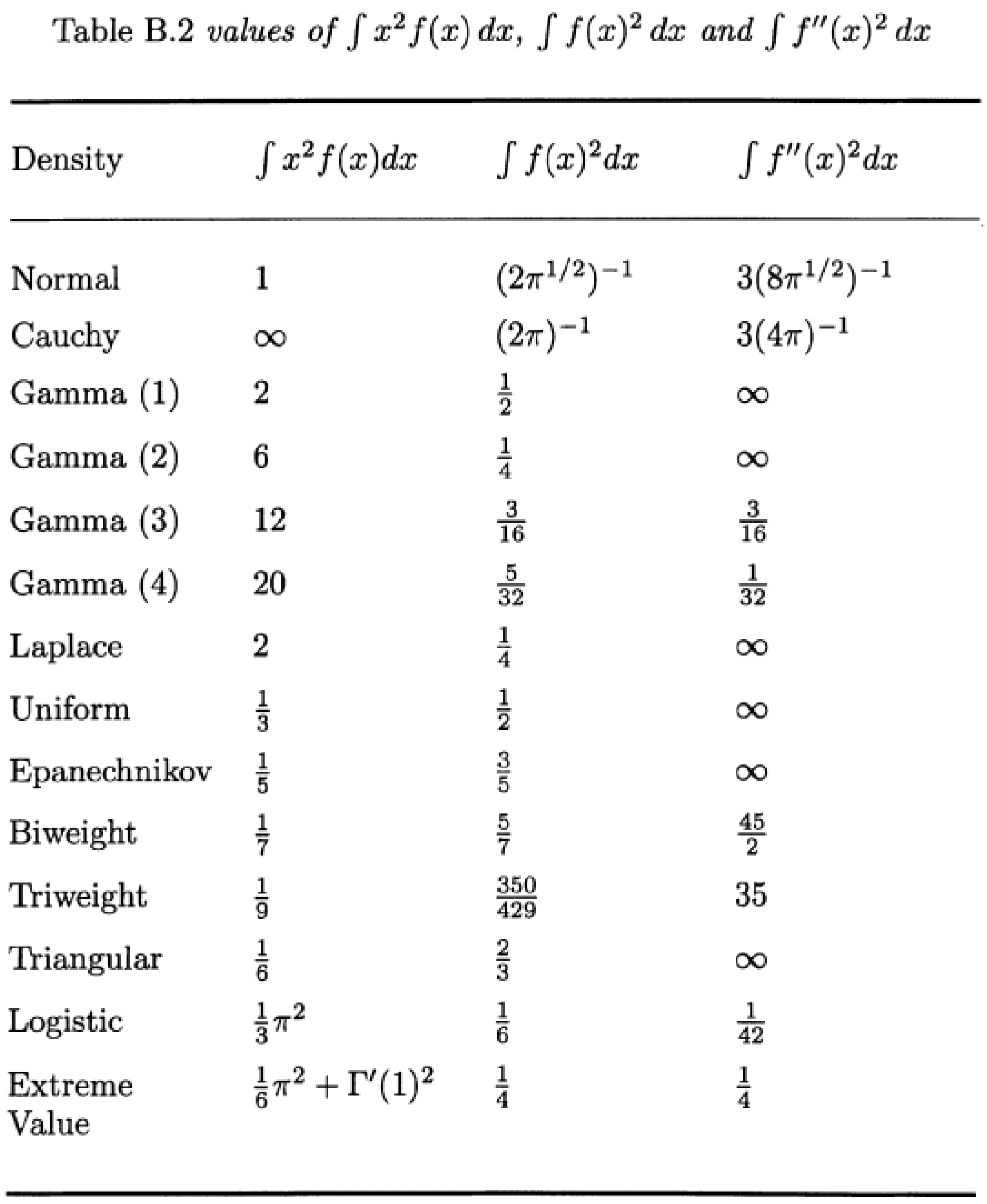


Table B.2 *values of* $\int x^2 f(x)\,dx$, $\int f(x)^2\,dx$ *and* $\int f''(x)^2\,dx$

Density	$\int x^2 f(x)dx$	$\int f(x)^2 dx$	$\int f''(x)^2 dx$
Normal	1	$(2\pi^{1/2})^{-1}$	$3(8\pi^{1/2})^{-1}$
Cauchy	∞	$(2\pi)^{-1}$	$3(4\pi)^{-1}$
Gamma (1)	2	$\frac{1}{2}$	∞
Gamma (2)	6	$\frac{1}{4}$	∞
Gamma (3)	12	$\frac{3}{16}$	$\frac{3}{16}$
Gamma (4)	20	$\frac{5}{32}$	$\frac{1}{32}$
Laplace	2	$\frac{1}{4}$	∞
Uniform	$\frac{1}{3}$	$\frac{1}{2}$	∞
Epanechnikov	$\frac{1}{5}$	$\frac{3}{5}$	∞
Biweight	$\frac{1}{7}$	$\frac{5}{7}$	$\frac{45}{2}$
Triweight	$\frac{1}{9}$	$\frac{350}{429}$	35
Triangular	$\frac{1}{6}$	$\frac{2}{3}$	∞
Logistic	$\frac{1}{3}\pi^2$	$\frac{1}{6}$	$\frac{1}{42}$
Extreme Value	$\frac{1}{6}\pi^2 + \Gamma'(1)^2$	$\frac{1}{4}$	$\frac{1}{4}$

- For the Gaussian kernel, where $K(u) = \frac{1}{\sqrt{2\pi}} e^{-u^2/2}$, one has

  $$\int u^2 K(u)\,du = 1$$

  (we could have known this without the table too), and so

  $$SD(\tilde{X}) = \sqrt{h^2 \times 1} = h$$

  That is for the Gaussian kernel, one has indeed just $\text{bw} = h$.
- For the uniform kernel, $K(u) = \frac{1}{2} 1_{\{-1 \le u \le 1\}}$,

  $$\int u^2 K(u)\,du = \frac{1}{3}$$

  and so

$$SD(\tilde{X}) = \sqrt{h^2 \times \frac{1}{3}} = \frac{1}{\sqrt{3}}h$$

Hence $\mathrm{bw} = \frac{1}{\sqrt{3}}h = 0.578\, h$ and

$$h = 1.73\, \mathrm{bw}.$$

- Finally, for the Epanechniknov kernel, $K(u) = \frac{3}{4}(1-u^2)1_{\{|u|\leq 1\}}$,

$$\int u^2 K(u)\, du = \frac{1}{5}$$

and so

$$SD(\tilde{X}) = \sqrt{h^2 \times \frac{1}{5}} = \frac{1}{\sqrt{5}}h$$

Hence $\mathrm{bw} = \frac{1}{\sqrt{5}}h = 0.447\, h$ and

$$h = 2.236\, \mathrm{bw}.$$

This result corresponds to the intuition from example 3.7 that, in order to attain the same degree of smoothness as the Epanechnikov kernel with bandwidth h, one needs a bandwidth of about

$$\frac{1}{2.236}h \approx \frac{1}{2}h$$

for the Gaussian kernel.

3.3.5 Statistical properties of the KDE

We consider initially a single point x and investigate the properties (bias, variance, MSE) of $\hat{f}\,(x)$ as an estimator of $f(x)$.

In our calculations, we will assume that, for sufficiently small h, terms of order h^{p+1} are negligible in comparison to h^p. That means, for instance,

$$ah^p + bh^{p+1} + ch^{p+2} = ah^p + O(h^{p+1}) = ah^p + hO(h^p) = ah^p + o(h^p) = h^p(a + o(1)).$$

Here, $O(h^p)$ signifies a term which grows at most proportional to h^p for sufficiently small h, and $o(h^p)$ denotes a term that becomes asymptotically smaller than h^p as $h \longrightarrow 0$.

(*Note*: On a more formal level, which is albeit not required for this course, the properties $o(\cdot)$ and $O(\cdot)$ are defined as follows (e.g. Wand & Jones, 1995). Let a_n and b_n be two real-valued deterministic sequences. Then

- $a_n = o(b_n)$ as $n \longrightarrow \infty$, if and only if $\lim_{n\longrightarrow\infty} \left|\frac{a_n}{b_n}\right| = 0$
- $a_n = O(b_n)$ as $n \longrightarrow \infty$, if and only if $\mathrm{limsup}_{n\longrightarrow\infty} \left|\frac{a_n}{b_n}\right| < \infty$

To reconcile these definitions with our setup, think of $h = h_n$ as a sequence such that $h_n \longrightarrow 0$ as $n \longrightarrow \infty$.)

We will make use of these notations systematically from now on. Hence, we will avoid using the $\approx$ symbol as we did in the histogram section, but will use equality symbols with appropriate order terms specified.

We also use the general notations $\mu_j = \int u^j K(u)\, du$ and $\nu_j = \int u^j K^2(u)\, du$.

Equipped with this, we calculate,

$$
\begin{aligned}
E(\hat{f}_h(x)) &= E\left(\frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right)\right) = \\
&= \frac{1}{nh}\sum_{i=1}^{n} E\left(K\left(\frac{x-x_i}{h}\right)\right) \\
&\overset{x_i \text{ iid}}{=} \frac{1}{h} E\left(K\left(\frac{x-x_i}{h}\right)\right) \\
&= \frac{1}{h}\int K\left(\frac{x-z}{h}\right) f(z)\,dz = \frac{1}{h}\int K\left(\frac{z-x}{h}\right) f(z)\,dz \\
&\overset{u=(z-x)/h}{=} \frac{1}{h}\int K(u) f(x+hu) h\,du \\
&= \int K(u)\left(f(x) + (hu)f'(x) + \frac{1}{2}(hu)^2 f''(x) + \frac{1}{6}(hu)^3 f'''(x) + \dots\right) \\
&= f(x)\underbrace{\int K(u)\,du}_{=1} + hf'(x)\underbrace{\int uK(u)\,du}_{=0} + \frac{h^2}{2}f''(x)\underbrace{\int u^2K(u)\,du}_{\equiv\mu_2} + O(h^3) \\
&= f(x) + \frac{h^2}{2}f''(x)\mu_2 + \underbrace{O(h^3)}_{=hO(h^2)=o(h^2)}
\end{aligned}
$$

Hence, we get the expression for the bias as

$$
\text{Bias}(\hat{f}(x)) = E\hat{f}_h(x) - f(x) = \frac{h^2}{2}f''(x)\mu_2 + o(h^2)
$$

Therefore, the bias will increase when h is larger, or when the curvature of the density is larger.

For the variance, we calculate

$$
\begin{aligned}
\mathrm{Var}(\hat{f}_b(x)) &= \mathrm{Var}\left(\frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right)\right) \\
&= \frac{1}{n^2h^2}\sum_{i=1}^{n}\mathrm{Var}\left(K\left(\frac{x-x_i}{h}\right)\right) \\
&\overset{\text{iid}}{=} \frac{1}{nh^2}\mathrm{Var}\left(K\left(\frac{x-x_i}{h}\right)\right) \\
&= \frac{1}{nh^2}\left[E\left(K^2\left(\frac{x-x_i}{h}\right)\right) - \left[E\left(K\left(\frac{x-x_i}{h}\right)\right)\right]^2\right] \\
&= \frac{1}{nh^2}\left[\int K^2\left(\frac{x-z}{h}\right) f(z)\,dz - \left[h\underbrace{\left(f(x)+\frac{h^2}{2}f''(x)\mu_2)+o(h^2)\right)}_{\text{from bias calculation}}\right]^2\right] \\
&= \frac{1}{nh^2}\left[\int K^2(u) f(x+hu)h\,du - h^2O(1)\right] \\
&= \frac{1}{nh^2}\left[\int K^2(u)(f(x)+huf'(x)+O(h^2))h\,du - h^2O(1)\right] \\
&= \frac{1}{nh}\left[f(x)\underbrace{\int K^2(u)\,du}_{\nu_0} + \underbrace{O(h)}_{=h\times O(1)=o(1)}\right] \\
&= \frac{1}{nh}f(x)\nu_0 + o\left(\frac{1}{nh}\right)
\end{aligned}
$$

This is a similar formula as for the histogram. The variance gets larger if nh is small or if $f(x)$ is large.

In summary we have

$$
\mathrm{MSE}(\hat{f}_h(x)) = \frac{h^4}{4}(f''(x))^2\mu_2^2 + \frac{1}{nh}f(x)\nu_0 + o(h^4) + o\left(\frac{1}{nh}\right)
$$

So, we again find that $\hat{f}_h(x)$ is mean square consistent if $h \equiv h_n$ is a sequence such that, if $n \longrightarrow \infty$

i. $h_n \longrightarrow 0$
ii. $nh_n \longrightarrow \infty$

To find such a sequence, let us proceed to the MISE directly (noting our experience with the histogram!)

3.3.6 MISE and optimal bandwidth

Consider the integrated MSE

$$
\begin{aligned}
\text{MISE}(\hat{f}_h) &= \int \text{MSE}\left(\hat{f}_h(x)\right)\, dx \\
&= \frac{h^4}{4}\mu_2^2 \int f''(x)^2\, dx + \frac{1}{nh}\int f(x)\, dx + o(h^4) + o\left(\frac{1}{nh}\right) \\
&\approx \frac{h^4}{4}\mu_2^2 \int f''(x)^2\, dx + \frac{1}{nh}\nu_0 \\
&\equiv \text{AMISE}(\hat{f}_h)
\end{aligned}
$$

where the notation AMISE stands for “Approximated” MISE.

Taking derivatives,

$$
\frac{d\text{AMISE}(\hat{f}_h)}{dh} = h^3\mu_2^2 \int f''(x)^2\, dx - \frac{1}{nh^2}\nu_0 \overset{!}{=} 0
$$

Hence

$$
\begin{aligned}
h^5\mu_2^2 \int f''(x)^2\, dx &= \frac{\nu_0}{n} \\
h^5 &= \frac{1}{n}\frac{\nu_0}{\mu_2^2}\frac{1}{\int f''(x)^2\, dx} \\
h_{opt} &= \left(\frac{\nu_0}{\mu_2^2 \int f''(x)^2\, dx}\right)^{1/5} n^{-1/5} \equiv cn^{-1/5}
\end{aligned}
$$

We verify

- Clearly $h_{opt} \longrightarrow 0$ (as $n \longrightarrow \infty$)
- $n \times h_{opt} = cnn^{-1/5} = cn^{4/5} \longrightarrow \infty$ (as $n \longrightarrow \infty$)

so using this optimal bandwidth, $\hat{f}_h(x)$ is mean squared consistent as an estimator of $f(x)$ (see end of last subsection). Furthermore,

$$
\text{AMISE} = \frac{c^4}{4}n^{-4/5}\mu_2^2 \int f''(x)^2\, dx + \frac{\nu_0}{cn^{4/5}} = O(n^{-4/5}).
$$

Since the remainder terms in the MISE are of a negligible order compared to the AMISE terms, this actually implies

$$
\text{MISE} = O(n^{-4/5})
$$

We conclude that the optimal bandwidth leads to a MISE of order $n^{-4/5}$ which is *better* than the convergence order of $n^{-2/3}$ observed for the histogram estimators.

3.3.7 Normal reference bandwidth for KDEs

The equation for h_{opt} derived in the previous subsection is still not useful as it depends on the unknown quantity

$$
\int f''(x)^2\, dx.
$$

Again we use the concept of “normal reference” to find a proxy for this quantity. So, take for a moment $N(0, \sigma^2)$.

We know from previous considerations (Sec 3.2.5) that

$$
\int f''(x)^2\, dx = \frac{1}{\sigma^5}\int \phi''(x)^2\, dx
$$

and also (Sec 3.3.4, Table B.2) that

$$\int \phi^{''}(x)^2\,dx = \frac{3}{8\sqrt{\pi}}.$$

Hence, we can put things together as

$$h_{opt} = \left(\frac{8\sqrt{\pi}\nu_0}{3\mu_2^2}\right)\sigma \times n^{-1/5}.$$

The constants μ_2^2 and ν_0 are “kernel moments” which can be easily computed for specific kernels. Specifically, for the Gaussian kernel $K(u) = \frac{1}{\sqrt{2\pi}}e^{-u^2/2}$, one finds, invoking again Table B.2, that

$$\mu_2 = \int u^2 K(u)\,du = 1$$
$$\nu_0 = \int K^2(u)\,du = \frac{1}{2\sqrt{\pi}}$$

So we have

$$h_{opt} = \left(\frac{8\sqrt{\pi}/(2\sqrt{\pi})}{3 \times 1^2}\right)\sigma \times n^{-1/5} = \left(\frac{4}{3}\right)^{1/5}\sigma n^{-1/5}.$$

Several choices have been suggested for the estimation of σ:

- The sample standard deviation,

$$\hat{\sigma} = s = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

- A rescaled version of the IQR, robust to outliers (see also Lab 5):

$$\hat{\sigma} = \frac{IQR}{1.34}$$

- The minimum of the two, robust to multimodulalities:

$$\hat{\sigma} = \min\left\{s, \frac{IQR}{1.34}\right\}$$

Using the latter option leads to the “rule of thumb estimator”

$$h_{ROT} = 1.06 \times \min\left\{s, \frac{IQR}{1.34}\right\}n^{-1/5},$$

Due to likely underestimation of $\int f^{''}(x)^2\,dx$, Silverman (1986) proposed an alternative constant, 0.9, yielding

$$h_{Sil} = 0.9 \times \min\left\{s, \frac{IQR}{1.34}\right\}n^{-1/5}$$

Example 3.9

```r
h.rot<- function(data){
  data<-na.omit(data)
  n<-length(data)
  sigma<-min(sd(data), IQR(data)/1.34)
  hopt<- 1.06*sigma*n^(-1/5)
  return(hopt)
}
```

```
h.rot(temp.data)
```

```
## [1] 1.99388
```

```
h.sil<- function(data){
  data<-na.omit(data)
  n<-length(data)
  sigma<-min(sd(data), IQR(data)/1.34)
  hopt<- 0.9*sigma*n^(-1/5)
  return(hopt)
}
```

```
h.sil(temp.data)
```

```
## [1] 1.692917
```

Note we have seen this value before!

```
density(temp.data)
```

```
## 
## Call:
##  density.default(x = temp.data)
## 
## Data: temp.data (363 obs.);  Bandwidth 'bw' = 1.693
## 
##        x                y            
##  Min.   :-5.179   Min.   :0.0000106  
##  1st Qu.: 4.511   1st Qu.:0.0030456  
##  Median :14.200   Median :0.0267690  
##  Mean   :14.200   Mean   :0.0257760  
##  3rd Qu.:23.889   3rd Qu.:0.0452311  
##  Max.   :33.579   Max.   :0.0609223
```

```
density(temp.data, bw="nrd0")
```

```
## 
## Call:
##  density.default(x = temp.data, bw = "nrd0")
## 
## Data: temp.data (363 obs.);  Bandwidth 'bw' = 1.693
## 
##        x                y            
##  Min.   :-5.179   Min.   :0.0000106  
##  1st Qu.: 4.511   1st Qu.:0.0030456  
##  Median :14.200   Median :0.0267690  
##  Mean   :14.200   Mean   :0.0257760  
##  3rd Qu.:23.889   3rd Qu.:0.0452311  
##  Max.   :33.579   Max.   :0.0609223
```

The bandwidth h_{ROT} can also be selected via

```
density(temp.data, bw="nrd")
```

```
##
## Call:
##  density.default(x = temp.data, bw = "nrd")
##
## Data: temp.data (363 obs.);  Bandwidth 'bw' = 1.994
##
##        x                 y
##  Min.   :-6.082   Min.   :1.006e-05
##  1st Qu.: 4.059   1st Qu.:2.358e-03
##  Median :14.200   Median :2.360e-02
##  Mean   :14.200   Mean   :2.463e-02
##  3rd Qu.:24.341   3rd Qu.:4.484e-02
##  Max.   :34.482   Max.   :5.914e-02
```

3.3.8 Cross-validation

Recall

$$\begin{aligned}\text{MISE}(\hat{f}_h) &= \int \text{MSE}(\hat{f}_h(x))\,dx \\ &= \int E\Big(\hat{f}_h(x) - f(x)\Big)^2\,dx \\ &= E\int \Big(\hat{f}_h(x) - f(x)\Big)^2\,dx\end{aligned}$$

where the interchanging of the integral and the expectation operator is possible by Fubini's theorem since the argument of the integral can be assumed bounded (the density is bounded and the kernel is bounded). [We do not strongly worry about such technicalities in this module!].

Rewriting this expression, we have

$$\text{MISE}(\hat{f}_h) = E\left(\int \hat{f}_h(x)^2\,dx - 2\int \hat{f}_h(x)f(x)\,dx + \int f(x)^2\,dx\right)$$

It is clear that the last term of this does not depend on h at all, so it is irrelevant for the question of minimizing the MISE. Hence we can focus on minimizing the expression

$$E\int \hat{f}_h(x)^2\,dx - 2E\int \hat{f}_h(x)f(x)\,dx$$

with respect to h. A problem is here that we still don't know f so we cannot directly evaluate the right-hand side expression. However, intuitively one would suggest that a Monte-Carlo version of $E\int \hat{f}_h(x)f(x)\,dx$ could be used here, that is to approximate this expectation by its empirical average,

$$\frac{1}{n}\sum_{i=1}^{n}\hat{f}_h(x_i)$$

This approximation turns out to be biased upwards, since the x_i are firstly used to estimate the density $\hat{f}_h$, and then to evaluate it at exactly these points. The solution to this problem is, for evaluation of the ith data point x_i, to omit the value x_i

from the estimation of the density. This gives rise to the expression

$$\frac{1}{n}\sum_{i=1}^{n}\hat{f}_h^{(-i)}(x_i)$$

which, with the **leave-one-out** (LOO) estimator

$$\hat{f}^{(-i)}(x)=\frac{1}{n-1}\sum_{j=1,j\neq i}^{n}K_h(x-x_j),$$

is now an *unbiased* approximation of $E\int\hat{f}_h(x)f(x)\,dx$.

Proof:

We work out the expectation of the empirical average of the LOO estimators.

$$\begin{aligned}E\left(\frac{1}{n}\sum_{i=1}^{n}\hat{f}_h^{(-i)}(x_i)\right)&=E(\hat{f}_h^{(-i)}(x_i))\\&=E\left(\frac{1}{n-1}\sum_{j=1,j\neq i}^{n}K_h(x_i-x_j)\right)\\&\overset{iid}{=}E\left(K_h(x_i-x_j)\right)=\\&=\int\int K_h(x-z)f(x,z)\,dx\,dz=\\&\overset{indep.}{=}\int\int K_h(x-z)f(x)f(z)\,dx\,dz\end{aligned}$$

We also see that

$$\begin{aligned}E\left(\int\hat{f}_h(x)f(x)\,dx\right)&=E\left(\int\frac{1}{n}\sum_{i=1}^{n}K_h(x-x_i)f(x)\,dx\right)\\&=\frac{1}{n}\sum_{i=1}^{n}\int EK_h(x-x_i)f(x)\,dx\\&=\int\left(\int K_h(x-z)f(z)\,dz\right)f(x)\,dx\\&=\int\int K_h(x-z)f(x)f(z)\,dz\end{aligned}$$

so that the two expectations correspond to each other.

We will refer to the resulting criterion as “cross-validation” criterion, $CV(h)$. In the literature it is also known as “least-squares cross-validation” (LSCV) or as “cross-validation estimator of risk”. We have

$$CV(h)=\int\hat{f}_h(x)^2\,dx-\frac{2}{n}\sum_{i=1}^{n}\hat{f}_h^{-i}(x_i)$$

Of course, the first integral needs to be computed here as well, this can be done through integration techniques based on the estimated density; we do not delve deeper into this but leave this one for software to sort out.

The CV selector of the bandwidth h is then given by

$$h_{CV}=\arg\min_h CV(h)$$

Example 3.10

The `density()` function supports the cross-validation technique via the option `ucv` (unbiased cross-validation).

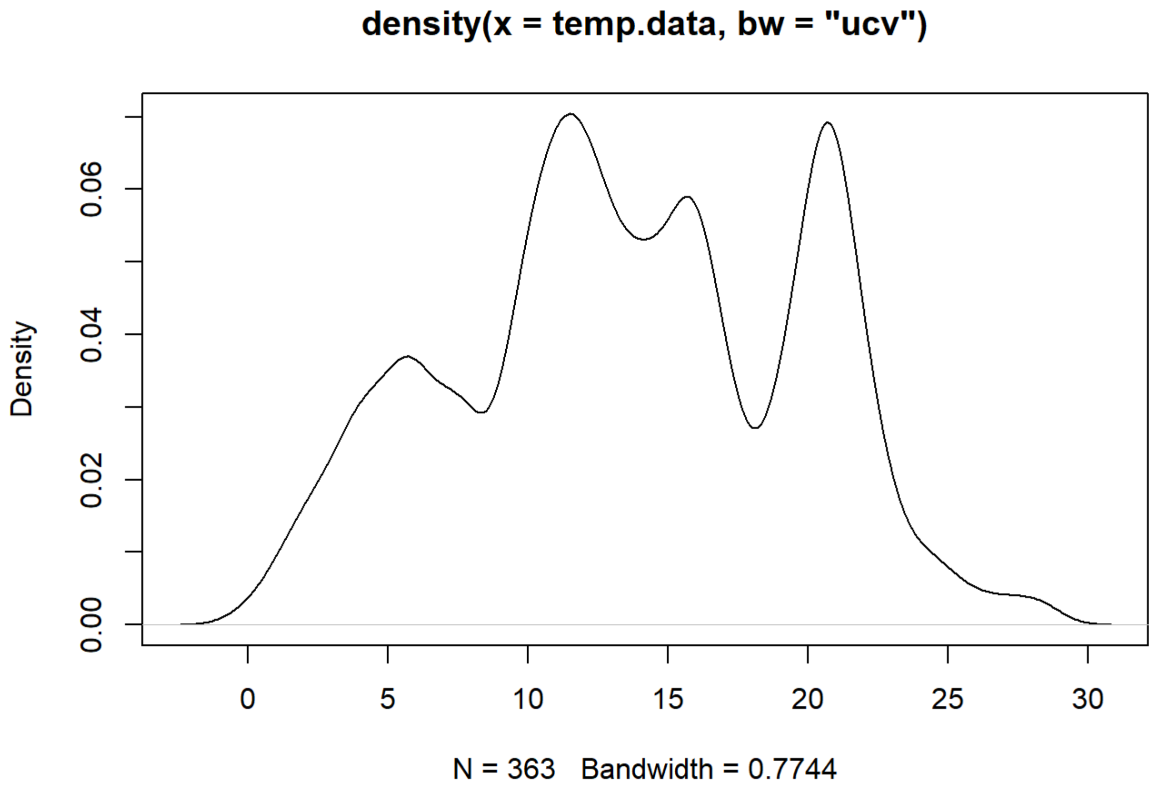
```
density(temp.data, bw="ucv")
```

```
##
## Call:
##  density.default(x = temp.data, bw = "ucv")
##
## Data: temp.data (363 obs.);  Bandwidth 'bw' = 0.7744
##
##        x                 y
##  Min.   :-2.423   Min.   :1.637e-05
##  1st Qu.: 5.888   1st Qu.:6.354e-03
##  Median :14.200   Median :3.050e-02
##  Mean   :14.200   Mean   :3.005e-02
##  3rd Qu.:22.512   3rd Qu.:5.309e-02
##  Max.   :30.823   Max.   :7.040e-02
```

```
bw.ucv(temp.data)
```

```
## [1] 0.7743859
```

```
plot(density(temp.data, bw="ucv"))
```



The resulting bandwidth of $h = 0.7744$ looks very small in comparison to previously computed bandwidths for this data set. Have we perhaps got trapped in a local minimum of $CV(h)$? We hence investigate a bit further, and look for a function in R which allows us to produce the complete cross-validation curve $CV(h)$. We find such a function in R package `locfit`.

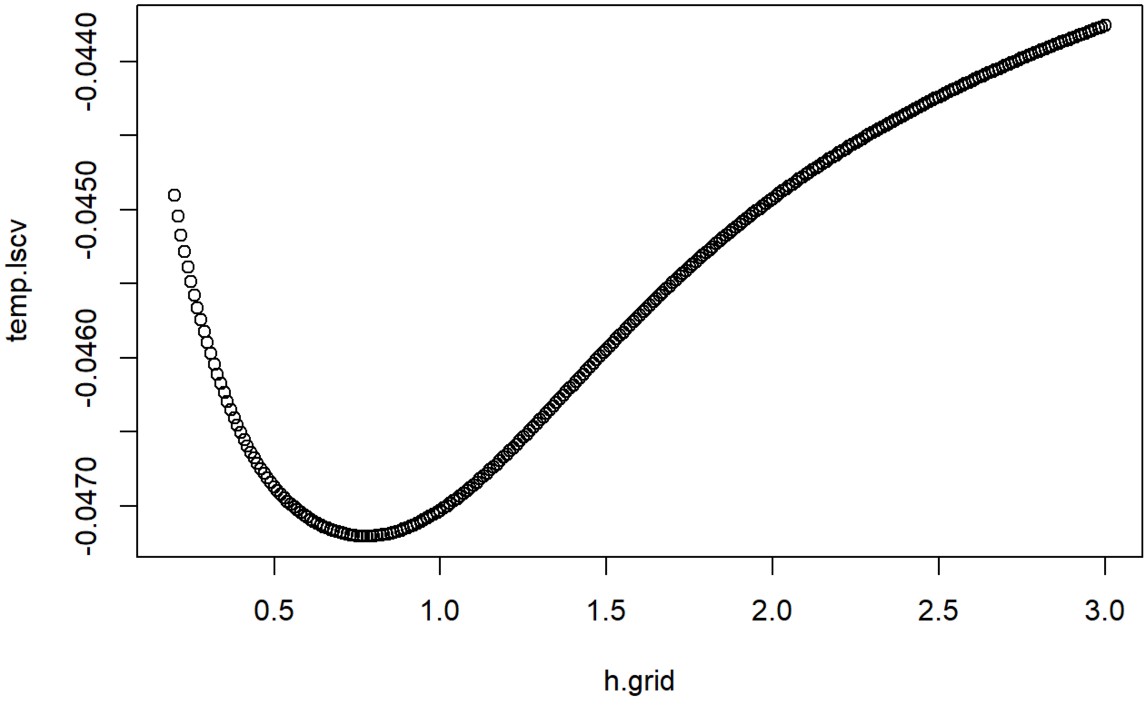
Please note the unusual bandwidth scaling in locfit (`?lscv.exact`); we need to multiply “our” bandwidth by 2.5 to put the estimates on the same scale.

```
require(locfit)
```

```
## Loading required package: locfit
```

```
## locfit 1.5-9.11   2025-01-27
```

```
h.grid<-seq(0.2,3, by=0.01)
h.length<-length(h.grid)
temp.lscv<- rep(0, h.length)
for (j in 1:h.length){
temp.lscv[j]<- lscv.exact(temp.data,h=2.5*h.grid[j])[1]
}
plot(h.grid, temp.lscv)
```



```
h.grid[which.min(temp.lscv)]
```

```
## [1] 0.78
```

The resulting minimum at 0.78 is very close to 0.7744 so that there is not any doubt on the correctness of this value! The cross-validation technique settles on a very small bandwidth here. Looking at the density estimate above, it may be that the technique tries somewhat “too hard” too maintain the various modes in the data set.

3.3.9 Asymptotic normality and confidence intervals

We consider the usual KDE and rewrite

$$\hat{f}_h(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right) = \frac{1}{n}\sum_{i=1}^{n} K_h(x-x_i) \equiv \frac{1}{n}\sum_{i=1}^{n} y_i$$

where

$$y_i = K_h(x-x_i) = \frac{1}{h}K\left(\frac{x-x_i}{h}\right)$$

can be considered as i.i.d. random variables. With this notation,

$$\hat{f}_h(x) = \frac{y_1+\ldots+y_n}{n} \equiv \bar{y}$$

is just the sample mean of the y_i, and gence the Lindeberg-Feller CLT applies

$$\frac{\bar{y}-E(\bar{y})}{\mathrm{Var}(\bar{y})} \xrightarrow{d} N(0,1),$$

where the notation $\xrightarrow{d}$ means “convergence in distribution”, that is, when considered as a random variable, the CDF of the left hand side would converge to the CDF of the right hand side (at each point where that CDF is continuous), as $n \longrightarrow \infty$.

In the above,

$$\begin{aligned} \bar{y} &= \hat{f}_h(x) \\ E(\bar{y}) &= f(x) + \frac{h^2}{2}f''(x)\mu_2 + O(h^3) \\ \mathrm{Var}(\bar{y}) &= \frac{1}{nh}f(x)\nu_0 + o\left(\frac{1}{nh}\right) \end{aligned}$$

From this we have

$$\frac{\hat{f}_h(x) - f(x) - \frac{h^2}{2}f''(x)\mu_2 + O(h^3)}{\sqrt{\frac{1}{nh}f(x)\nu_0 + o\left(\frac{1}{nh}\right)}} \xrightarrow{d} N(0,1).$$

We can rewrite this as

$$\sqrt{nh}\,\frac{\hat{f}_h(x) - f(x) - \frac{h^2}{2}f''(x)\mu_2 + O(h^3)}{\sqrt{f(x)\nu_0 + o(1)}} \xrightarrow{d} N(0,1)$$

Now, it is clear that the term $o(1)$ in the denominator vanishes if n tends to infinity, so it becomes irrelevant for the distributional statement being made. Hence, the property remains true if it is omitted, and so we can say

$$\sqrt{nh}\left(\hat{f}_h(x) - f(x) - \frac{h^2}{2}f''(x)\mu_2 + O(h^3)\right) \xrightarrow{d} N(0, f(x)\nu_0).$$

Based on this, we would *like to say* that

$$\sqrt{nh}\left(\hat{f}_h(x) - f(x)\right) \xrightarrow{d} N(0, f(x)\nu_0),$$

as this would imply that, for large n,

$$P\left(\hat{f}_h(x) - z_{1-\alpha/2}\sqrt{\frac{f(x)\nu_0}{nh}} \leq f(x) \leq \hat{f}_h(x) + z_{1-\alpha/2}\sqrt{\frac{f(x)\nu_0}{nh}}\right)$$

and hence

$$\hat{f}_h(x) \mp z_{1-\alpha/2}\sqrt{\frac{\hat{f}_h(x)\nu_0}{nh}}$$

could be used as an approximate $(1-\alpha)$ confidence interval for $f(x)$.

However, this would require the bias terms $\frac{h^2}{2}f''(x)\mu_2 + O(h^3)$ to get asymptotically erased when mutiplying with $\sqrt{nh}$. Is this the case?

- For the $O(h^3)$ term, $\sqrt{nh}O(h^3) = O(n^{1/2}h^{7/2})$. With $h_{opt} = cn^{-1/5}$,

$$O\left(n^{1/2}(cn^{-1/5})^{7/2}\right) = O\left(n^{1/2}n^{-7/10}\right) = n^{-1/5} \longrightarrow 0 \;(n \longrightarrow \infty)$$

so, yes, this term disappears, when working with the optimal bandwidth.

- For the $O(h^2)$ term, $\sqrt{nh}O(h^2) = O(n^{1/2}h^{5/2})$. With $h_{opt} = cn^{-1/5}$,

$$O\left(n^{1/2}(cn^{-1/5})^{5/2}\right) = O\left(n^{1/2}n^{-1/2}\right) = O(1)$$

which does *not* vanish as n tends to infinity. So, the term $\frac{h^2}{2}f''(x)\mu_2$ does not vanish unless

$$\begin{aligned}\sqrt{nh}\,h^2 &= o(1)\\ nh\,h^4 &= o(1)\\ h &= o(n^{-1/5})\end{aligned}$$

so the bandwidth would need to vanish faster than the optimal bandwidth (we would refer to this as “undersmoothing”).

The above formula for the CI is still commonly used in practice as it correctly reflects the variability of $\hat{f}_h(x)$. But it would be centered at the biased mean $f(x) + \frac{h^2}{2}f''(x)\mu_2$, unless h is small enough (in relation to n) to erase the bias term.

Note: The CI constructed here has *pointwise* coverage in the sense that, for each x, one has

$$P(f(x) \in CI) \approx 1-\alpha$$

(the probability statement referring to the randomness of the interval). Different methodology needs to be used for *confidence bands* which cover the entire function f with probability $1-\alpha$.

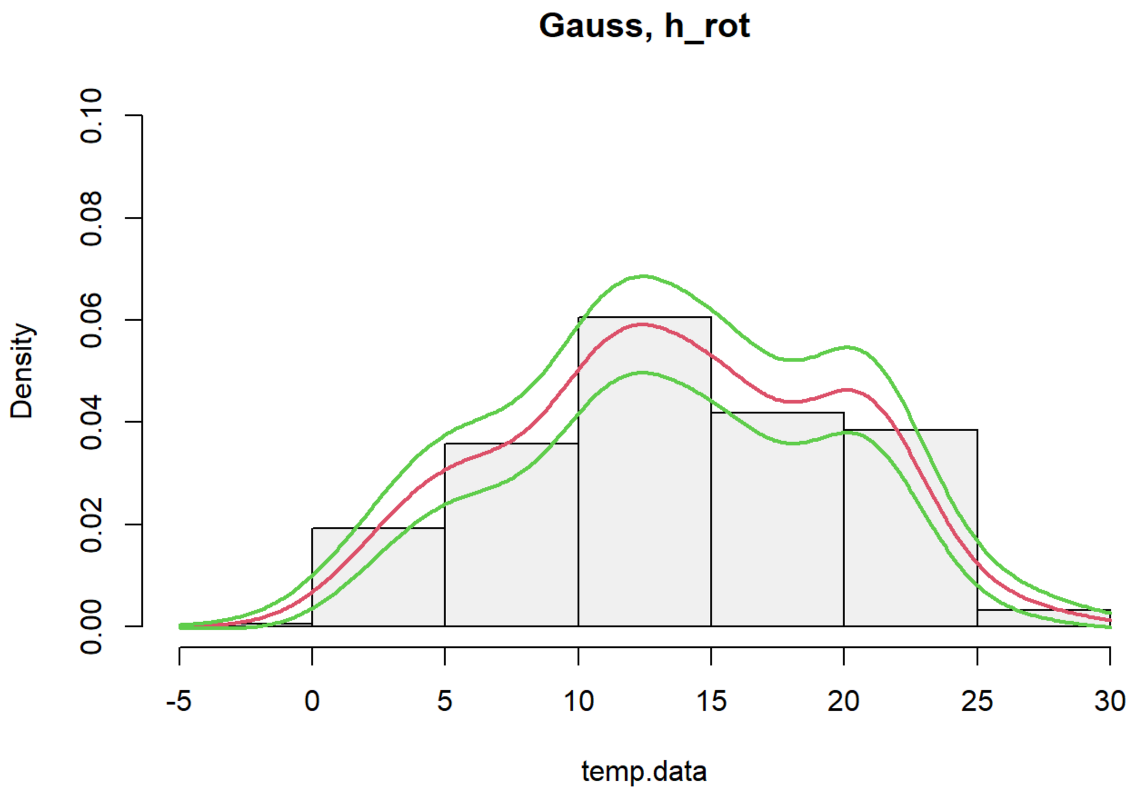
Example 3.11

We implement the expression for the confidence intervals by integrating them into our earlier function `est.density()` (Version from lectures, not lab).

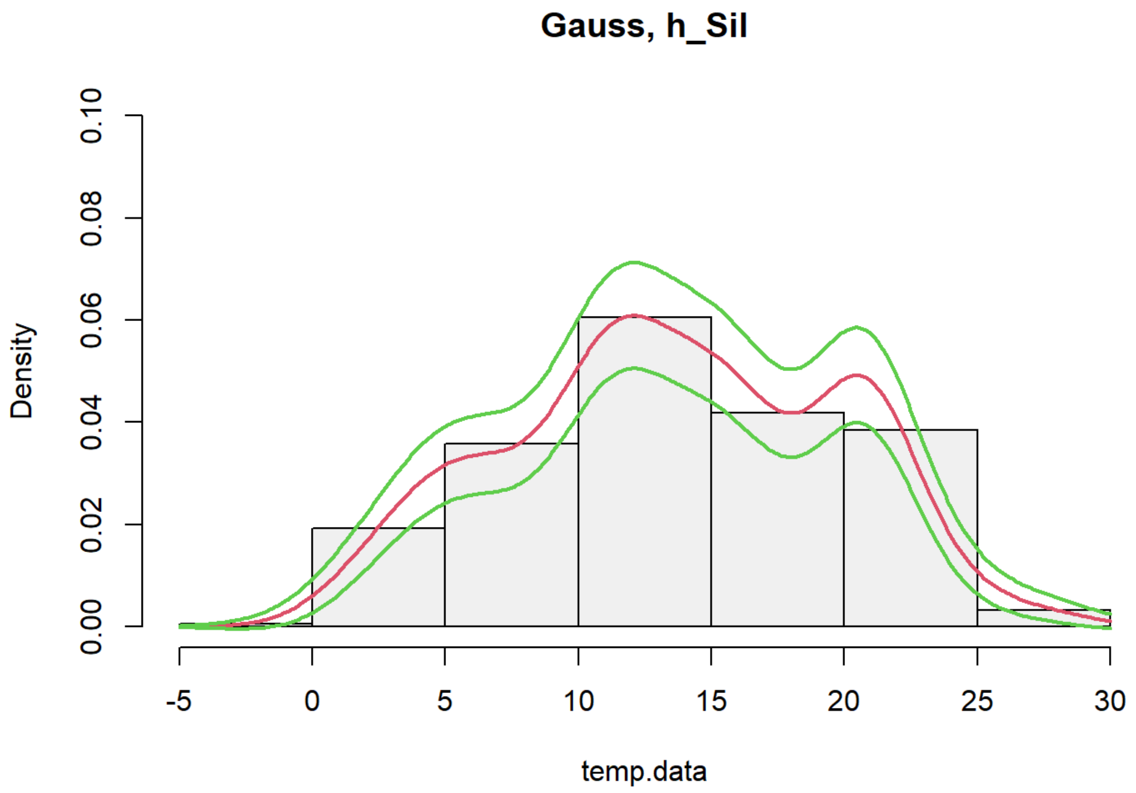
```
est.density<- function(data, xgrid=dat, h=1, kernel="gauss", alpha=0.05){
  my.kernel<- switch(kernel,
                     "gauss"= function(x){dnorm(x)},
                     "epa"=function(x){x<- 3/4*(1-x^2)*ifelse(abs(x)<=1,1,0)},
                     "uni"=function(x){x<- 1/2*ifelse(abs(x)<=1,1,0)}
  )
  nu0<- switch(kernel,
                     "gauss"= 1/(2*sqrt(pi)),
                     "epa"=3/5,
                     "uni"=1/2
  )
  data<-na.omit(data)
  n<-length(data)
  g<-length(xgrid)
  est<-lower<-upper <- rep(0, g)
  denom<- n*h
  for (j in 1:g){
    num<- sum(my.kernel((data-xgrid[j])/h))
    est[j]<- num/denom
    upper[j]<- est[j]+qnorm(1-alpha/2)*sqrt(est[j]*nu0/(n*h))
    lower[j]<- est[j]-qnorm(1-alpha/2)*sqrt(est[j]*nu0/(n*h))
  }
  return(c("est"=list(est), "lower"=list(lower), "upper"=list(upper)))
}
```

Then we apply this function on density esimates for a few kernels and bandwidths. We begin with Gaussian kernel and the bandwidth choices that we have identified so far, h_{ROT}, h_{Sil}, and h_{CV}:

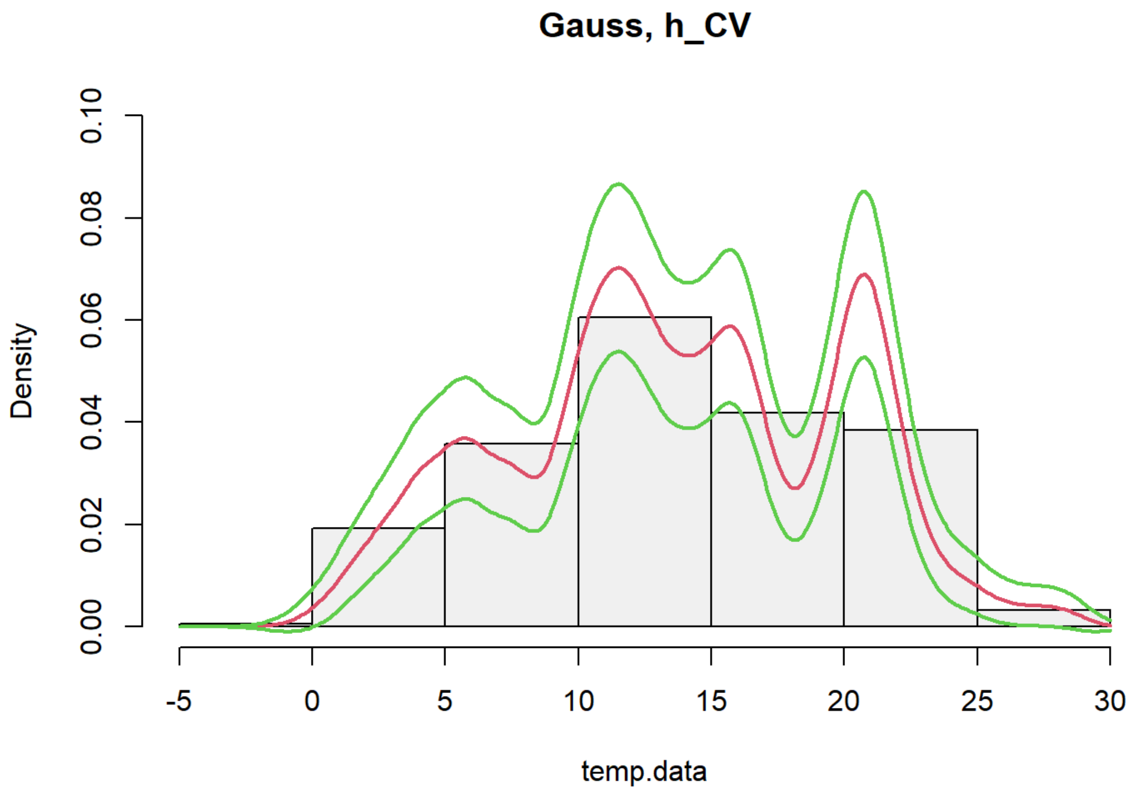
```
x<- seq(-5,30, by=0.1)
hist(temp.data, freq=FALSE, ylim=c(0,0.1), col="grey95", main="Gauss, h_rot")
estimate<- est.density(temp.data, x, h=1.99, "gauss")
lines(x,estimate$est, col=2, lwd=2)
lines(x,estimate$lower, col=3, lwd=2)
lines(x,estimate$upper, col=3, lwd=2)
```



```
x<- seq(-5,30, by=0.1)
hist(temp.data, freq=FALSE, ylim=c(0,0.1), col="grey95", main="Gauss, h_Sil")
estimate<- est.density(temp.data, x, h=1.69, "gauss")
lines(x,estimate$est, col=2, lwd=2)
lines(x,estimate$lower, col=3, lwd=2)
lines(x,estimate$upper, col=3, lwd=2)
```



```
x<- seq(-5,30, by=0.1)
hist(temp.data, freq=FALSE, ylim=c(0,0.1), col="grey95", main="Gauss, h_CV")
estimate<- est.density(temp.data, x, h=0.78, "gauss")
lines(x,estimate$est, col=2, lwd=2)
lines(x,estimate$lower, col=3, lwd=2)
lines(x,estimate$upper, col=3, lwd=2)
```

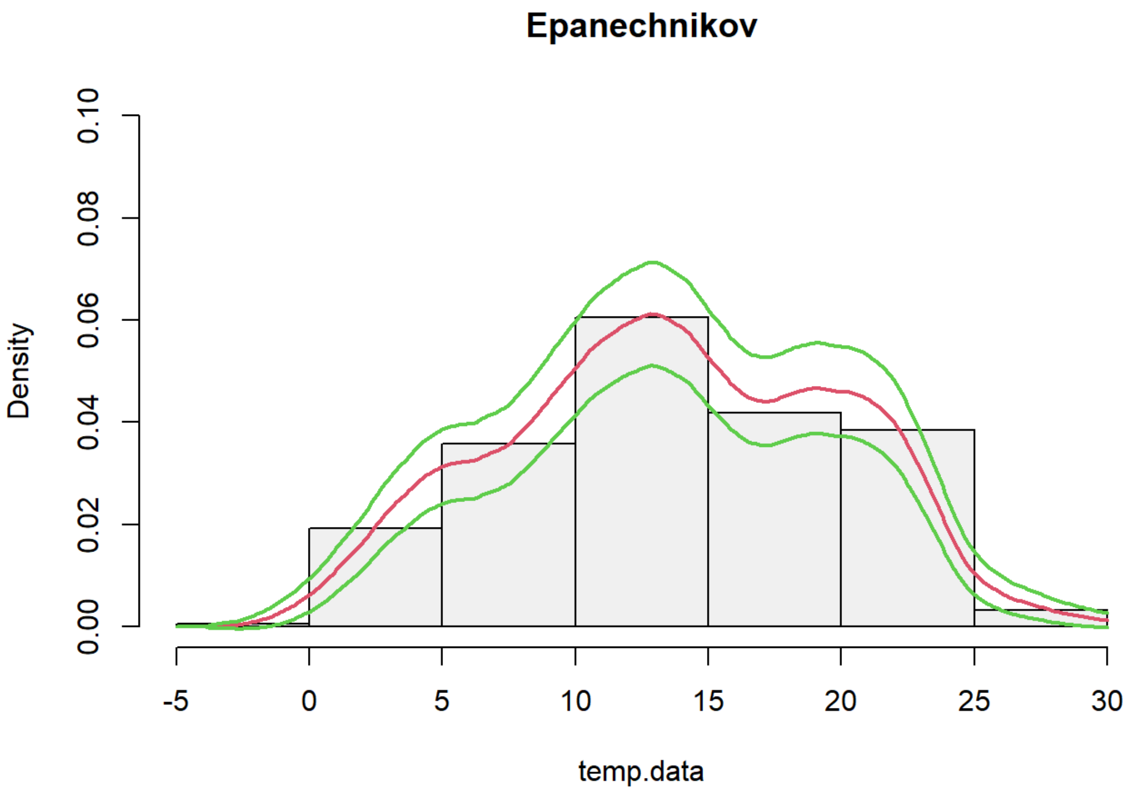


It is noted that the cross-validated bandwidth is considerably smaller than the asymptotically optimal bandwidths, so this bandwidth choice has plausibly done a better job at removing the bias term and hence would likely lead to better coverage properties of this confidence interval. However, the available theory does not allow us to quantify or verify this notion more precisely, for a real data set with finite sample size.

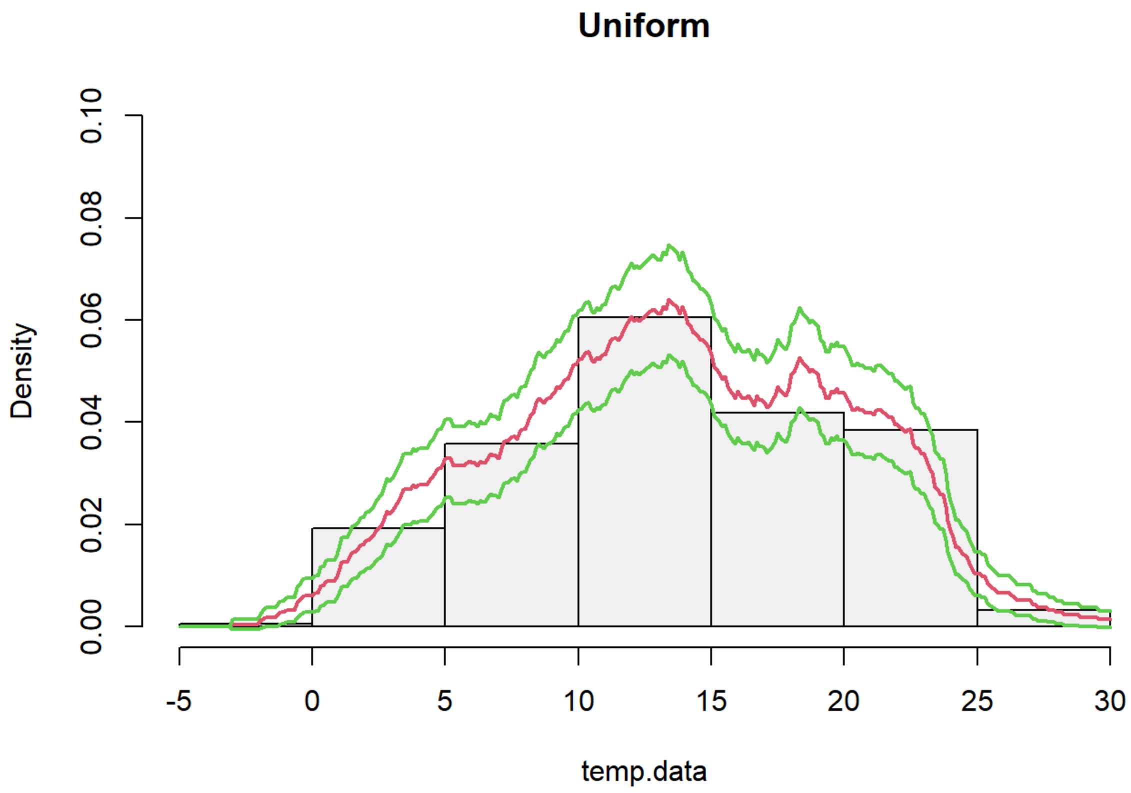
Next we also compare with the results for the Epanechnikov and Uniform kernels. We take Silverman's bandwidth as reference value and recompute the bandwidths according to Sec 3.3.4 as

```
h.epa<- 1.69*2.236
h.uni<- 1.69*1.73
```

```
x<- seq(-5,30, by=0.1)
hist(temp.data, freq=FALSE, ylim=c(0,0.1), col="grey95", main="Epanechnikov")
estimate<- est.density(temp.data, x, h=h.epa, "epa")
lines(x,estimate$est, col=2, lwd=2)
lines(x,estimate$lower, col=3, lwd=2)
lines(x,estimate$upper, col=3, lwd=2)
```



```
x<- seq(-5,30, by=0.1)
hist(temp.data, freq=FALSE, ylim=c(0,0.1), col="grey95", main="Uniform")
estimate<- est.density(temp.data, x, h=h.uni, "uni")
lines(x,estimate$est, col=2, lwd=2)
lines(x,estimate$lower, col=3, lwd=2)
lines(x,estimate$upper, col=3, lwd=2)
```



3.4 Multivariate density estimation

We are given d- variate random samples $x_i = (x_{i1}, \ldots, x_{id})^T \in \mathbb{R}^d$, $i = 1, \ldots, n$, from a density $f : \mathbb{R}^d \longrightarrow \mathbb{R}_0^+$.

Task: For a given $x \in \mathbb{R}^d$, estimate $f(x)$.

3.4.1 The multivariate KDE

Recall the 1-dimensional KDE in the formulation

$$\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^{n} K_{h_1}(x_i - x),$$

is a sum of n “bumps”, centered at the x_i, each with $\int K_h(x - x_i)\, dx = 1$.

The idea for d-dimensional KDE is simply to use d-variate bumps, centered at the $x_i \in \mathbb{R}^d$, $i = 1, \ldots, n$. For instance, one can use densities of multivariate normal $N(x_i, H)$ distributions for some bandwidth matrix

$$H = \begin{pmatrix} h_1^2 & & * \\ & \ddots & \\ * & & h_d^2 \end{pmatrix}$$

where h_j is the bandwidth in direction $j = 1, \ldots, d$, and the off-diagonal elements $*$ are usually taken to be zero (but could also be something else).

So (again in the normal case), we can write

$$K_H(x-x_i) = \frac{1}{(2\pi)^{d/2}} \frac{1}{|H|^{1/2}} \exp\left\{-\frac{1}{2}(x-x_i)^T H^{-1}(x-x_i)\right\}$$

so that

$$\hat{f}_H(x) = \frac{1}{n}\sum_{i=1}^{n} K_H(x-x_i)$$

is a d-variate KDE for $f(x)$.

Example 3.12

We are given a data set on characteristics of $n = 48$ soil samples. We initially only consider $d = 2$ variables, namely

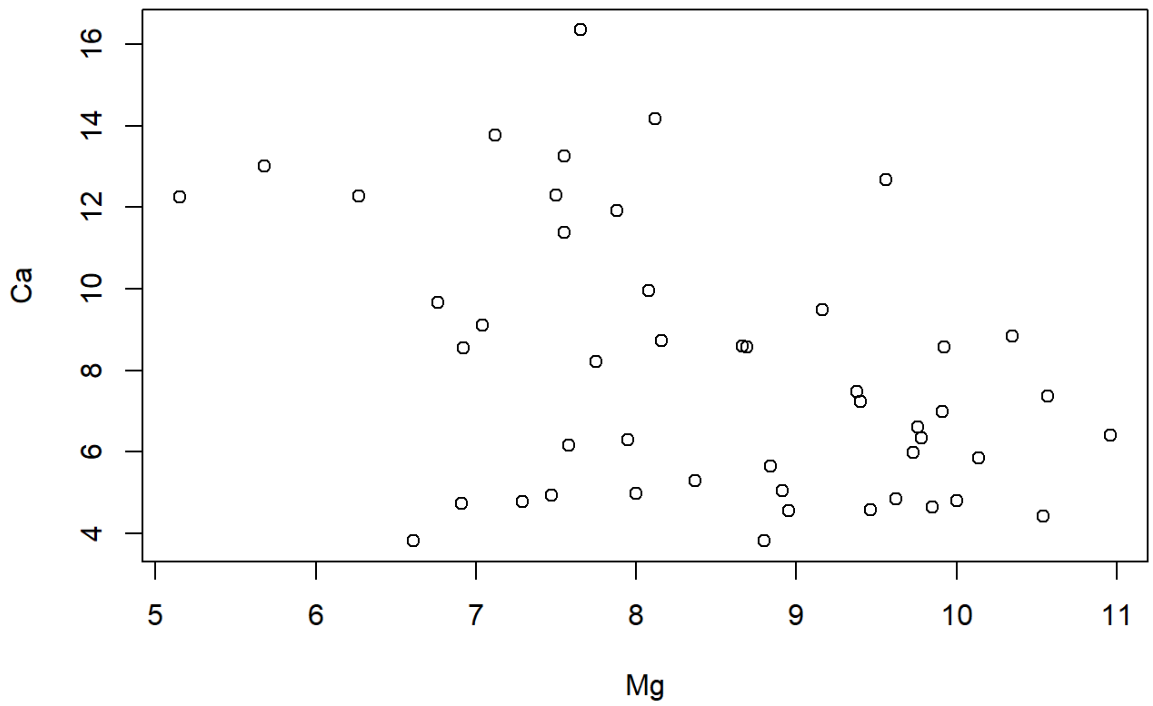
- Mg (magnesium in me/100 gm)
- Ca (calcium in me/100 gm)

(Here, me/100 gm denotes “milliequivalents per 100 grams of soil”, a measure the amount of a substance based on its chemical charge, which is a standard unit in soil chemistry).

```
require(carData)
```

```
## Loading required package: carData
```

```
data(Soils)
soils<- Soils[,c("Mg", "Ca")]
plot(soils)
```



We would like to produce a kernel density estimate for this bivariate data set. Therefore we would like to adapt the above introduced, generic formula for the d-variate KDE, to the bivariate case. For the sake of notation let us say that the bivariate data take the form

$$x_i = \begin{pmatrix} y_i \\ z_i \end{pmatrix} \in \mathbb{R}^2, i = 1, \dots, n.$$

We also assume that we work with a diagonal bandwidth matrix of shape

$$H = \begin{pmatrix} h_1^2 & 0 \\ 0 & h_2^2 \end{pmatrix}.$$

Then we can write

$$\begin{aligned}
\hat{f}_H(x) &= \frac{1}{n}\sum_{i=1}^{n} K_H(x - x_i) \\
&= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2\pi}\frac{1}{h_1 h_2}\exp\left\{-\frac{1}{2}(y - y_i, z - z_i)\begin{pmatrix} 1/h_1^2 & 0 \\ 0 & 1/h_2^2 \end{pmatrix}\begin{pmatrix} y - y_i \\ z - z_i \end{pmatrix}\right\} \\
&= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2\pi}\frac{1}{h_1 h_2}\exp\left\{-\frac{1}{2}\left[\left(\frac{y - y_i}{h_1}\right)^2 + \left(\frac{z - z_i}{h_2}\right)^2\right]\right\} \\
&= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{\sqrt{2\pi}h_1} e^{-\frac{1}{2}\left(\frac{y-y_i}{h_1}\right)^2} \times \frac{1}{\sqrt{2\pi}h_2} e^{-\frac{1}{2}\left(\frac{z-z_i}{h_2}\right)^2} \\
&= \frac{1}{n}\sum_{i=1}^{n} K_{h_1}(y - y_i) K_{h_2}(z - z_i) \\
&= \frac{1}{n h_1 h_2}\sum_{i=1}^{n} K\left(\frac{y - y_i}{h_1}\right) K\left(\frac{z - z_i}{h_2}\right)
\end{aligned}$$

where $K(u) = \frac{1}{\sqrt{2\pi}} e^{-u^2/2}$ is the 1-dim Gaussian kernel. The conclusion from this derivation is that an easy way to implement multivariate kernels is by simply multiplying the univariate versions. (This is best appreciated by reading the equations from bottom to top!)

We now produce an implementation of the 2-dimensional case, which makes use of the product representation just derived.

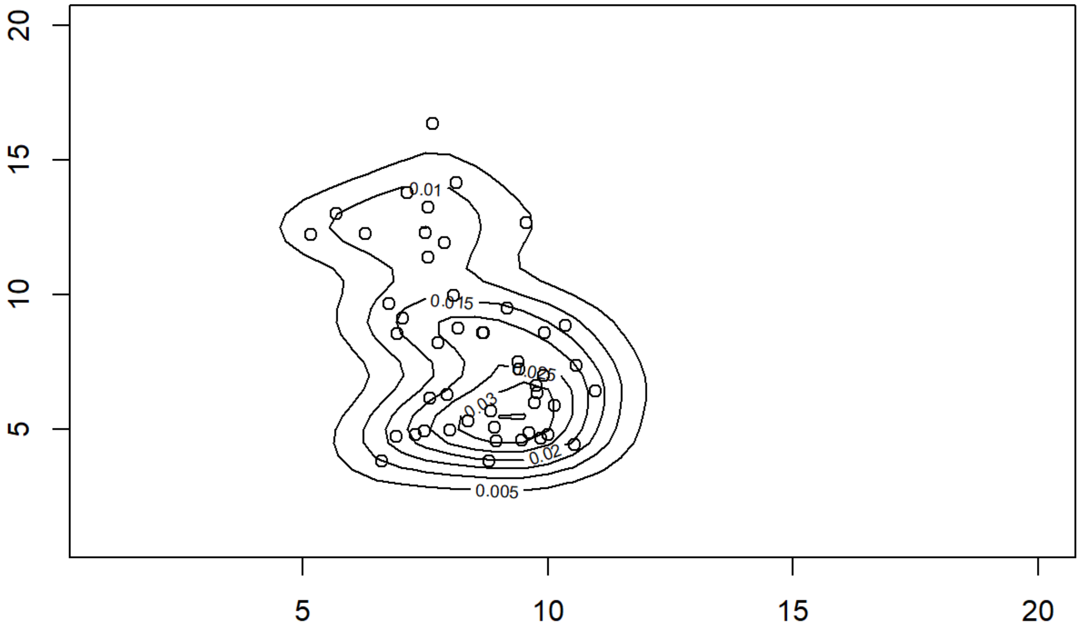
```r
my.2d.density<- function(data,x,h){
  # note that `data` is now a matrix, and x, h are vectors
  data <- na.omit(data)
  n <- dim(data)[1]
  d <- dim(data)[2]
  if (d!=2){stop("Incorrect data dimension: d=2 required")}
  denom<- n*h[1]*h[2]
  num<- sum(my.kernel((data[,1]-x[1])/h[1])*my.kernel((data[,2]-x[2])/h[2]) )
  return(num/denom)
}
```

Assuming that `my.kernel` is set to be the Gaussian kernel, the result is:

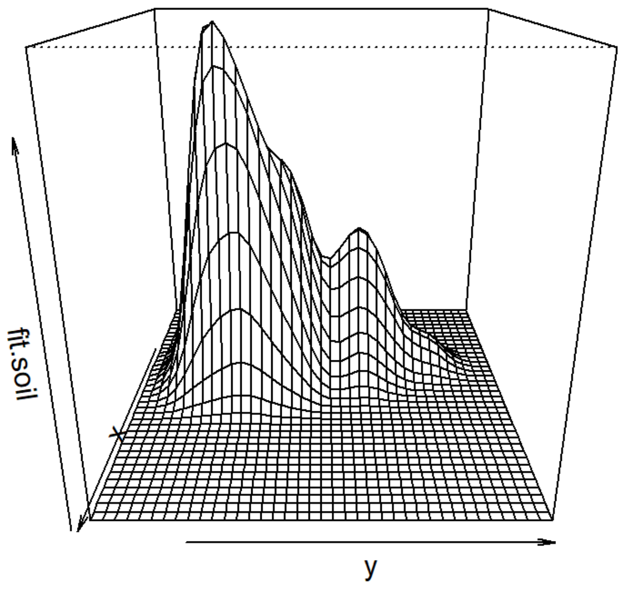
```r
x<- seq(1,20, by=0.5)
y<-  seq(1,20, by=0.5)
G<-length(x)
fit.soil<- matrix(0, nrow=G, ncol=G)
for (j in 1:G){
  for (k in 1:G){
  fit.soil[j,k]<- my.2d.density(soils,c(x[j], y[k]),c(1,1))
  # uses h=1 in both directions
  }
  }
```

```r
contour(x,y,fit.soil)

points(soils)
```



```r
persp(x,y, fit.soil, phi=20, theta=90)
```



Similarly to the 1-dim notation

$$K_h(x - x_i) = \frac{1}{h} K\left(\frac{x - x_i}{h}\right)$$

we can write more generally in d dimensions

$$K_H(x - x_i) = \frac{1}{|H|^{1/2}} K\left(H^{-1/2}(x - x_i)\right)$$

where $K : \mathbb{R}^d \longrightarrow \mathbb{R}_0^+$ is a d-variate kernel, i.e. a symmetric, bounded, unimodal function such that

$$\int_{u_d} \cdots \int_{u_1} K(u_1, \dots, u_d) d_{u_1} \dots d_{u_d} \equiv \int K(u)\, du = 1.$$

For instance, the Gauss kernel in d dimensions is given by $K(u) = \frac{1}{(2\pi)^{d/2}} \exp\{-\frac{1}{2} u^T u\}$, and $H \in \mathbb{R}^{d \times d}$ is a symmetric and positive definite bandwidth matrix.

3.4.2 Optimal bandwidth matrices

Based on similar calculations as in Sections 3.3.5 to 3.3.7, and then extending the idea of “normal reference” to the d-variate case, one obtains he optimal bandwidth matrix for $d > 1$ as

$$H_{opt} = \left(\frac{4}{d+2}\right)^{2/(d+4)} n^{-2/(d+4)} \hat{\Sigma}$$

where $\hat{\Sigma} = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(x_i - \bar{x})^T$ is the estimated variance matrix (Wand & Jones, Sec 4.9). If one uses

$$\hat{\Sigma} = \begin{pmatrix} \hat{\sigma}_1^2 & & \\ & \ddots & \\ & & \hat{\sigma_d}^2 \end{pmatrix}$$

where $\hat{\sigma}_j$ is the standard deviation of the jth variable, then

$$H_{opt} = \begin{pmatrix} h_{1,opt}^2 & & \\ & \ddots & \\ & & h_{d,opt}^2 \end{pmatrix}$$

with

$$h_{j,opt} = \left(\frac{4}{d+2}\right)^{1/d+4} n^{-1/(d+4)}\hat{\sigma}_j$$

for $j = 1, \dots, d$. Noting that the leading multiplicative term is ≈ 1 (see example below), this leads to the surprisingly simple rule-of-thumb formula

$$h_{j,ROT} = \hat{\sigma}_j n^{-1/(d+4)}$$

This is known as **Scott's rule in** $\mathbb{R}^d$.

Example 3.13

Check that the multiplicative leading term of h_{opt} is approximately 1:

```
d<- 1:20
(4/(d+2))^(1/(d+4))
```

```
##  [1] 1.0592238 1.0000000 0.9686251 0.9505798 0.9397142 0.9330330 0.9289309
##  [8] 0.9264849 0.9251351 0.9245277 0.9244309 0.9246891 0.9251954 0.9258747
## [15] 0.9266738 0.9275544 0.9284884 0.9294556 0.9304408 0.9314329
```

Then implement h_{ROT}:

```
hd.rot<- function(data){
  data<-na.omit(data)
  n<-dim(data)[1]
  d<-dim(data)[2]
  hopt<-rep(0,d)
  for (j in 1:d){
    sigmaj<-sd(data[,j])
    hopt[j]<- sigmaj*n^{-1/(d+4)}
  }
return(hopt)
}
```

Apply on 2-dim soils data

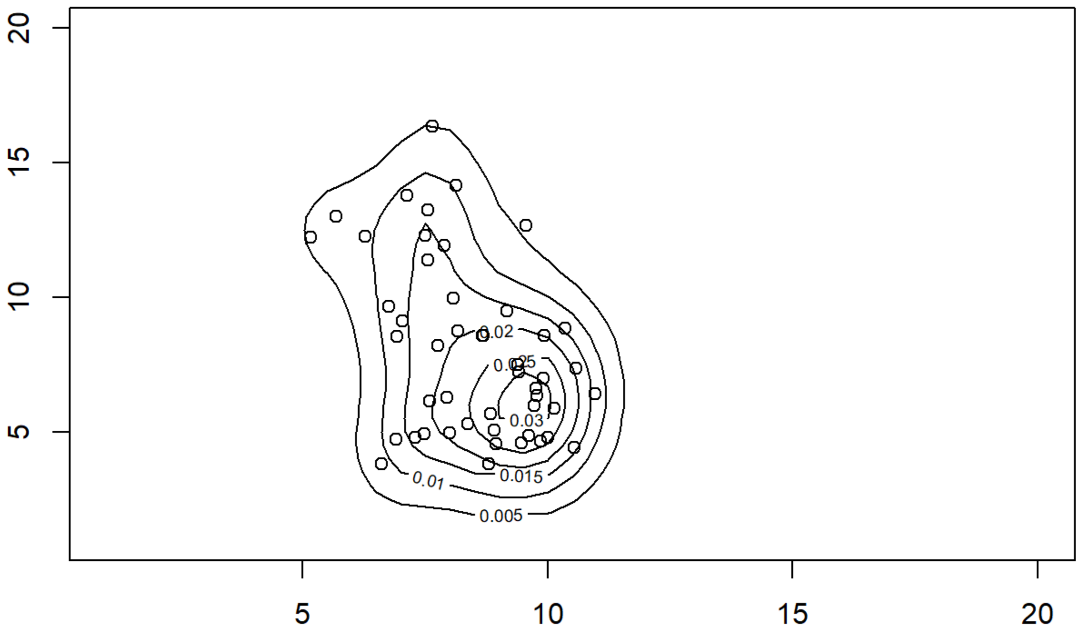
```
h.soil2<-hd.rot(soils)
h.soil2
```

```
## [1] 0.717701 1.707931
```

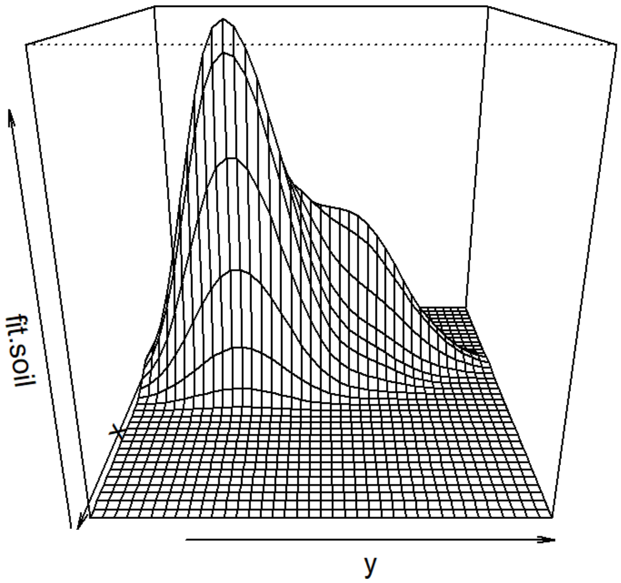
Estimate the KDE with these bandwidths:

```
x<- seq(1,20, by=0.5)
y<-  seq(1,20, by=0.5)
G<-length(x)
fit.soil<- matrix(0, nrow=G, ncol=G)
for (j in 1:G){
  for (k in 1:G){
  fit.soil[j,k]<- my.2d.density(soils,c(x[j], y[k]), h.soil2)
  }
  }
```

```
contour(x,y,fit.soil)
points(soils)
```



```
persp(x,y,fit.soil, phi=20, theta=90)
```

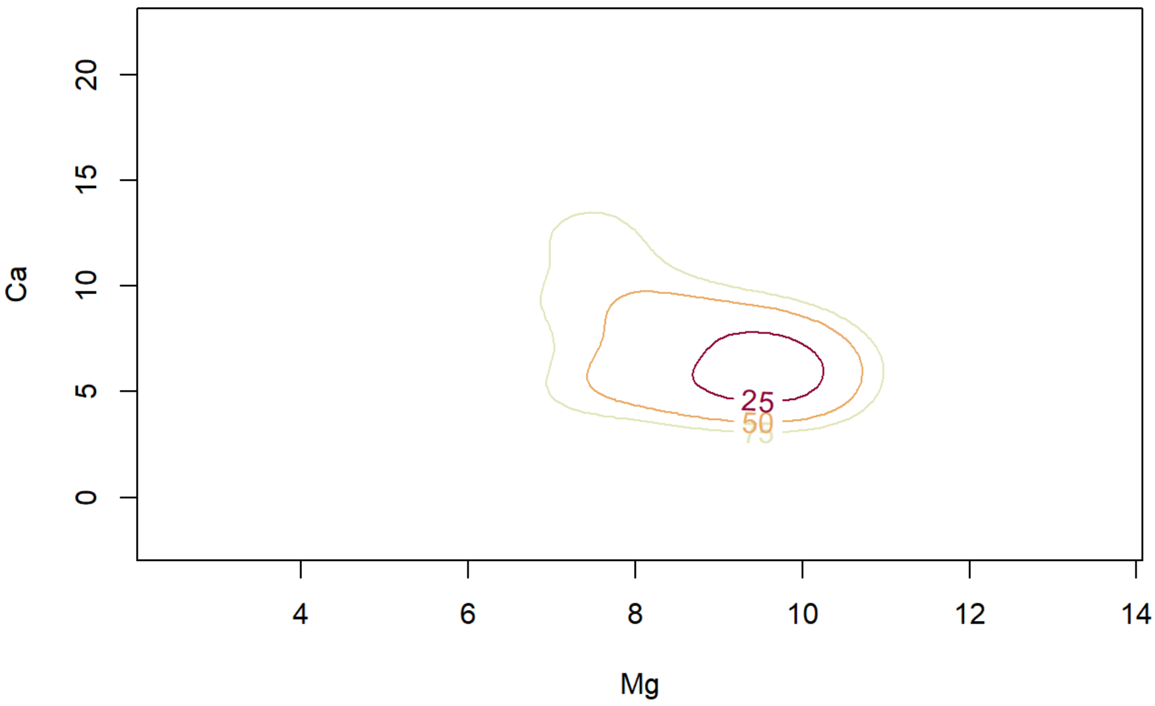


We can use also use some ready-built R functions, such as `kde` in package `ks` :

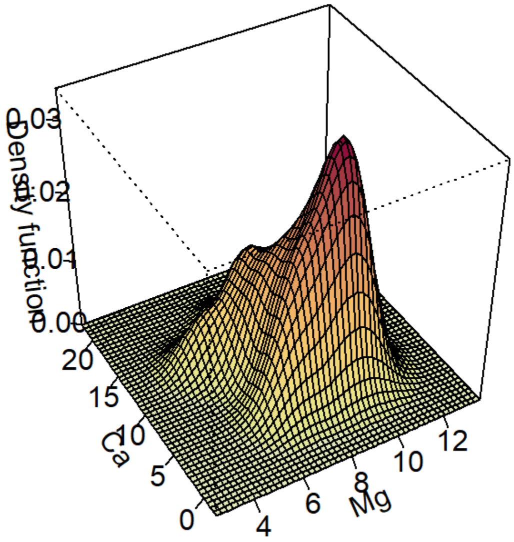
```
require(ks)
```

```
## Loading required package: ks
```

```
fit.soil1<-kde(soils)
plot(fit.soil1)
```



```
plot(fit.soil1, display="persp")
```



Which bandwidths have been used here? Let's inspect:

```
names(fit.soil1)
```

```
##  [1] "x"           "eval.points" "estimate"    "H"           "gridtype"
##  [6] "gridded"     "binned"      "names"       "w"           "type"
## [11] "cont"
```

```
sqrt(fit.soil1$H)
```

```
## Warning in sqrt(fit.soil1$H): NaNs produced
```

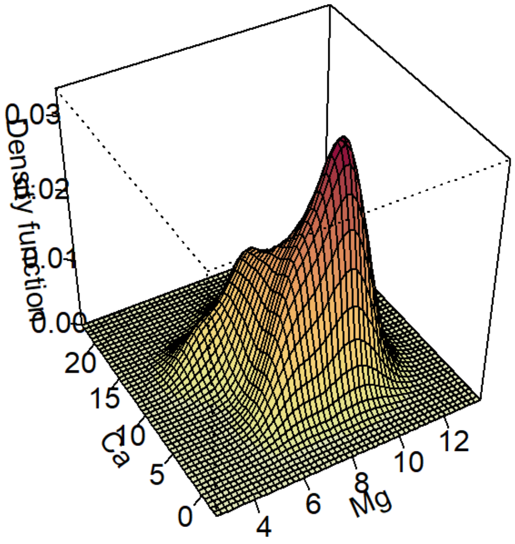
```
##           [,1]     [,2]
## [1,] 0.7314344      NaN
## [2,]       NaN 1.578264
```

The result is of this is slightly different than from our ROT since the `Hpi` bandwidth used in `kde` does not use normal reference, but instead estimates a pilot density first, from which the integrated second derivatives are estimated. We can still use Scott's bandwidth by plugging it manually into `kde` :

```
fit.soil2 <- kde(soils, H=diag(h.soil2^2))
sqrt(fit.soil2$H)
```

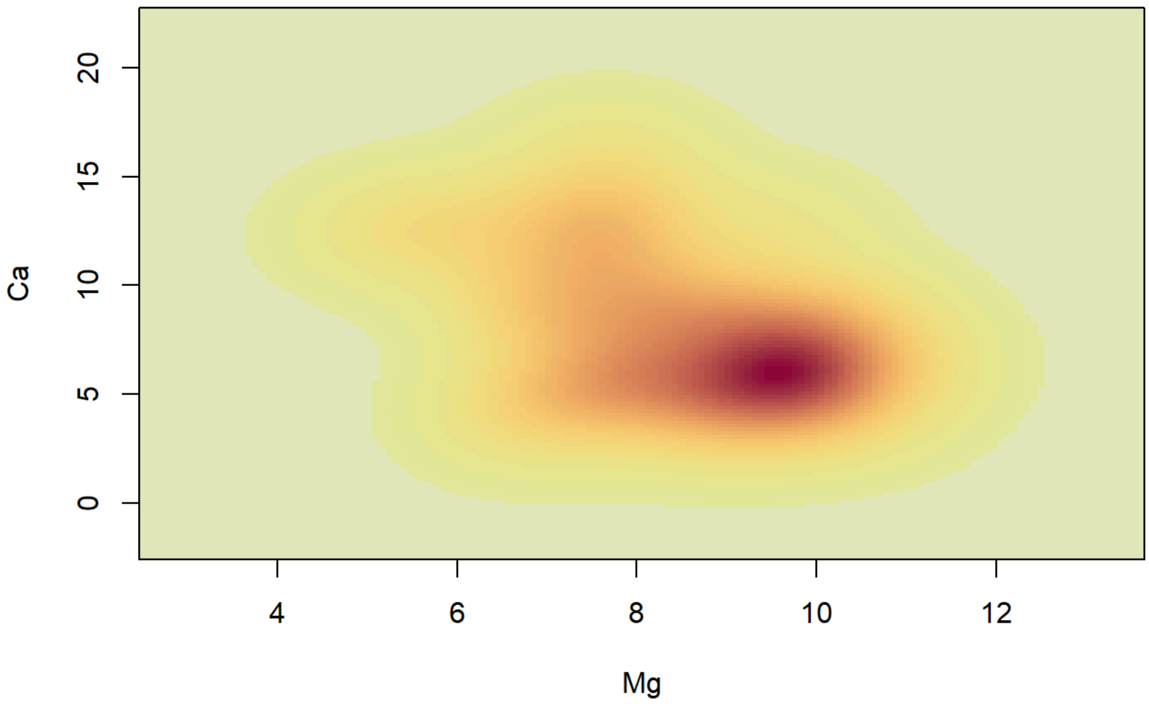
```
##          [,1]     [,2]
## [1,] 0.717701 0.000000
## [2,] 0.000000 1.707931
```

```
plot(fit.soil2, display="persp")
```

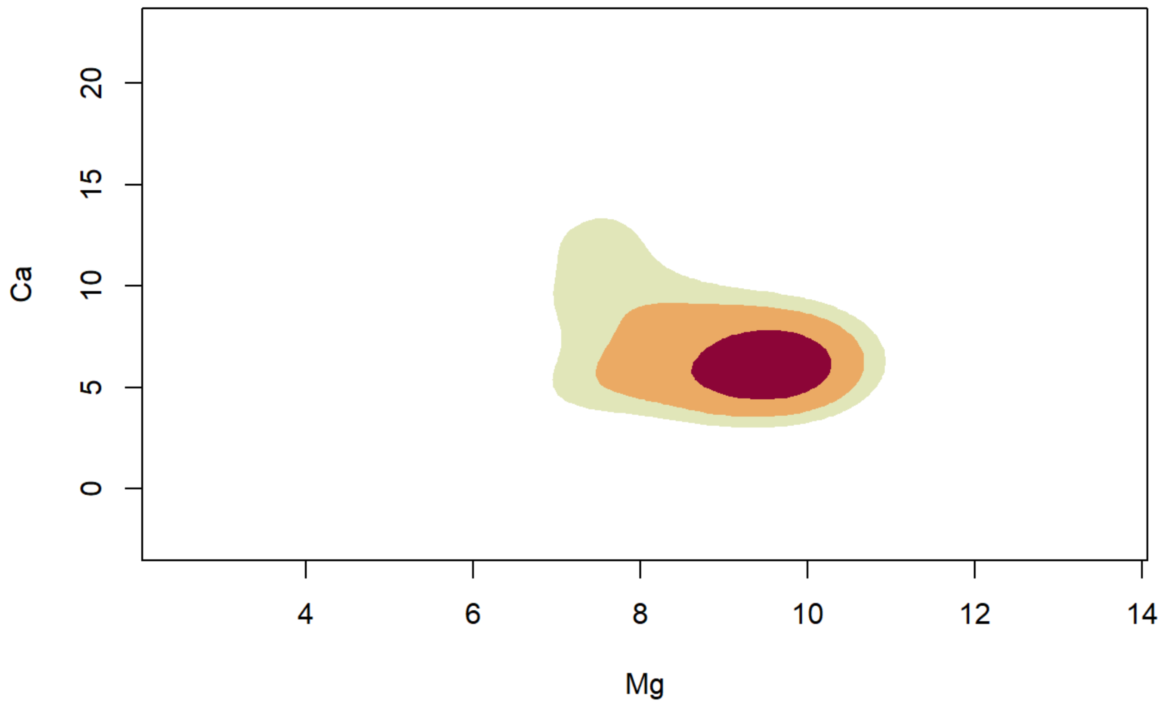


There are some further graphical options which one may find neat…

```
# ?plot.kde
plot(fit.soil2, display="image")
```



```
plot(fit.soil2, display="filled.contour")
```



3.4.3 The curse of dimensionality

Modern data, such as gene expression (microarray) data, may have dimensions $d > 10000$! Can we do KDE with aribtrary dimension d?

To answer this question, let us begin with a thought experiment. Let's assume data $x_1, \ldots, x_n$ are sampled from a d-variate uniform distribution $\Omega = (U[0,1])^d$, with say $n = 1000$. Now let's take the bandwidth in each direction as

$$h_j \equiv h = 1/4$$

and assume we are interested in estimating $f(x)$ at a point x which sits sufficiently in the interior of the unit cube so that $x \pm h$ does not overlap with any boundary. The question that we ask is:

How many data points can be expected to be situated in the "minicube" centered at x, $\Omega_x = [x \pm 1/4]^d$?

We can easily work this out:

d	**# points in Ω_x**
1	$(2 \times 1/4) \times 1000 = \frac{1}{2} \times 1000 = 500$
2	$(1/2^2) \times 1000 = 250$
3	$(1/2)^3 \times 1000 = 125$
4	$(1/2)^4 \times 1000 = 62.5$
$\vdots$	$\vdots$
10	$(1/2)^{10} \times 1000 < 1$

This illustrates the **curse of dimensionality**: Local neighborhoods in high dimensions tend to be sparse (almost empty). To counter this effect, one would need to hold $h^d n$ approximately constant. So if we assume $h^d n = c$ and $h < 1$, this would mean

$$n = ch^{-d} = c\left(\frac{1}{h}\right)^d$$

where then $1/h > 1$ and hence this means an exponential growth of n is required. Or alternatively, one could try to increase the bandwidth accordingly, so that we would need

$$h \propto n^{-\frac{1}{d}}$$

This does actually look similar to our optimal bandwidth $h_{j,ROT} \propto n^{-1/(d+4)}$, which corresponds to a gentle increase of h with increasing d, so perhaps there is a glimmer of hope? Let's try!

Example 3.14

This example requires the availability of the following R packages

```
require(plot3D)
```

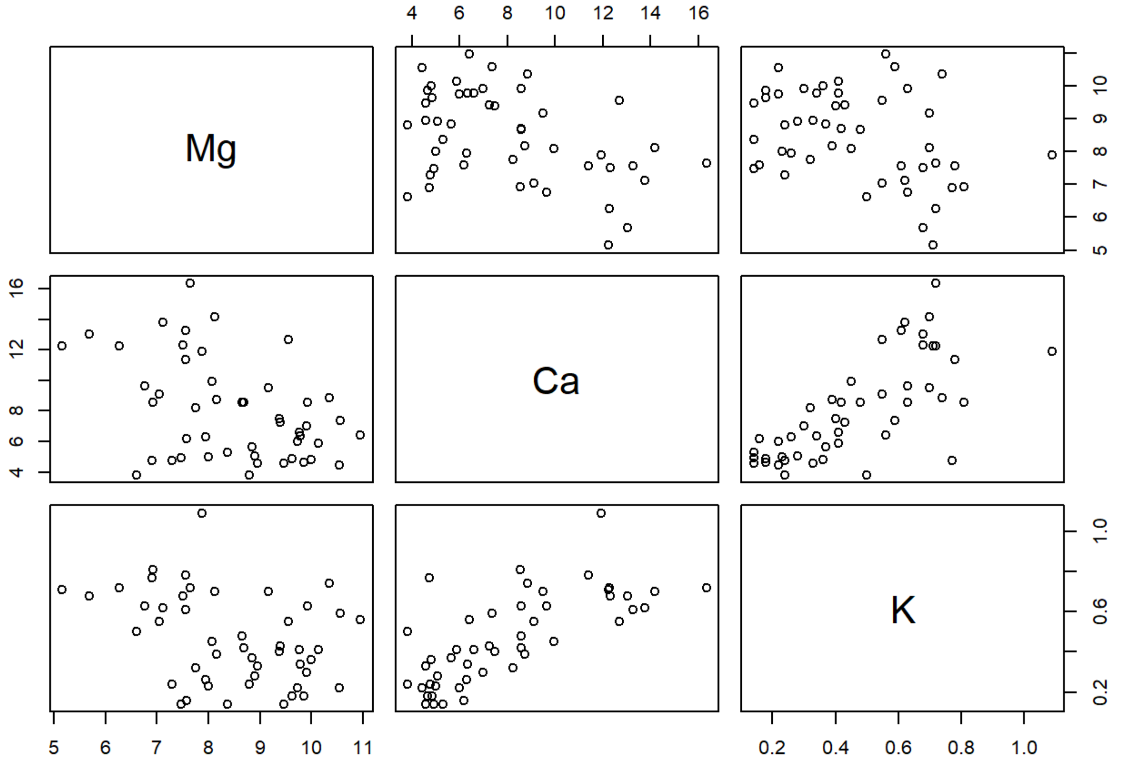
```
## Loading required package: plot3D
```

```
require(rgl)
```

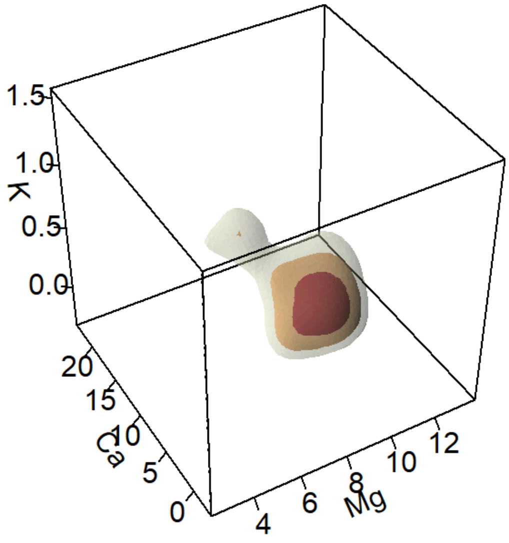
```
## Loading required package: rgl
```

We consider again the Soils data, which features a range of further predictor variables. Let us successively increase the number of variables, and observe the behavior of the resulting KDE when using the ROT optimal bandwidths.

```
soils3<- Soils[,c("Mg", "Ca", "K")]
plot(soils3)
```



```
h.soil3 <- hd.rot(soils3)
fit.soil3<-kde(soils3, H=diag(h.soil3^2))
plot(fit.soil3)
```

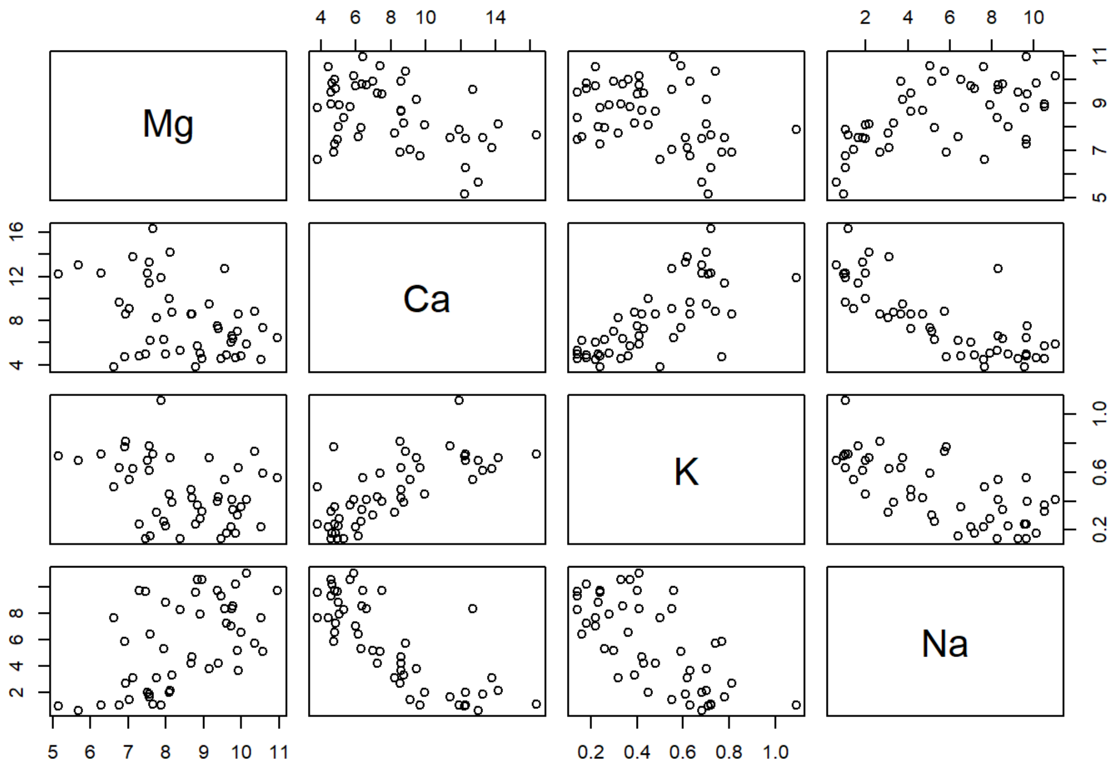


```
plot(fit.soil3, display="rgl")
```

(Note the external window with the interactive RGL 3D-display which pops up when you run the chunk above!)

The scenario $d = 3$ encounters some challenges in visualization, but by and large still works well.

```
soils4<- Soils[,c("Mg", "Ca", "K", "Na")]
plot(soils4)
```



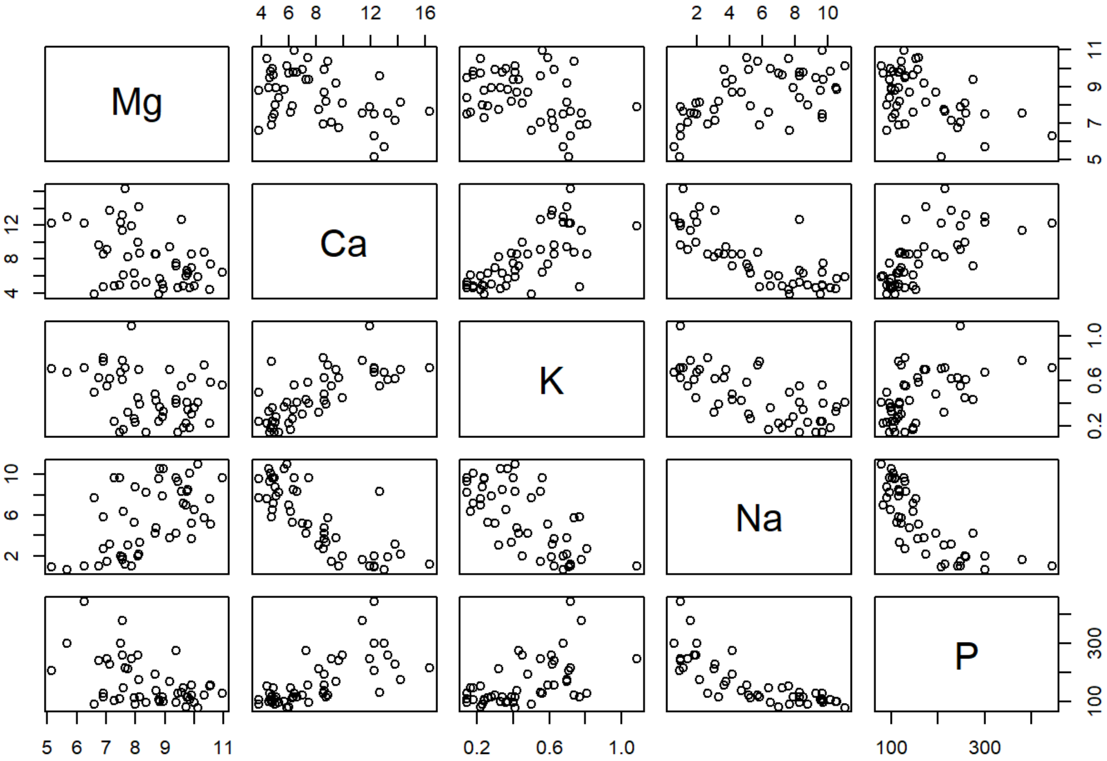
```
h.soil4 <- hd.rot(soils4)
fit.soil4<-kde(soils4, H=diag(h.soil4^2))
# plot(fit.soil4)     # commented out as it gives an error
names(fit.soil4)
```

```
##  [1] "x"           "eval.points" "estimate"    "H"           "gridtype"
##  [6] "gridded"     "binned"      "names"       "w"           "type"
## [11] "cont"
```

```
# fit.soil4$estimate  # commented out as it needs lots of space
```

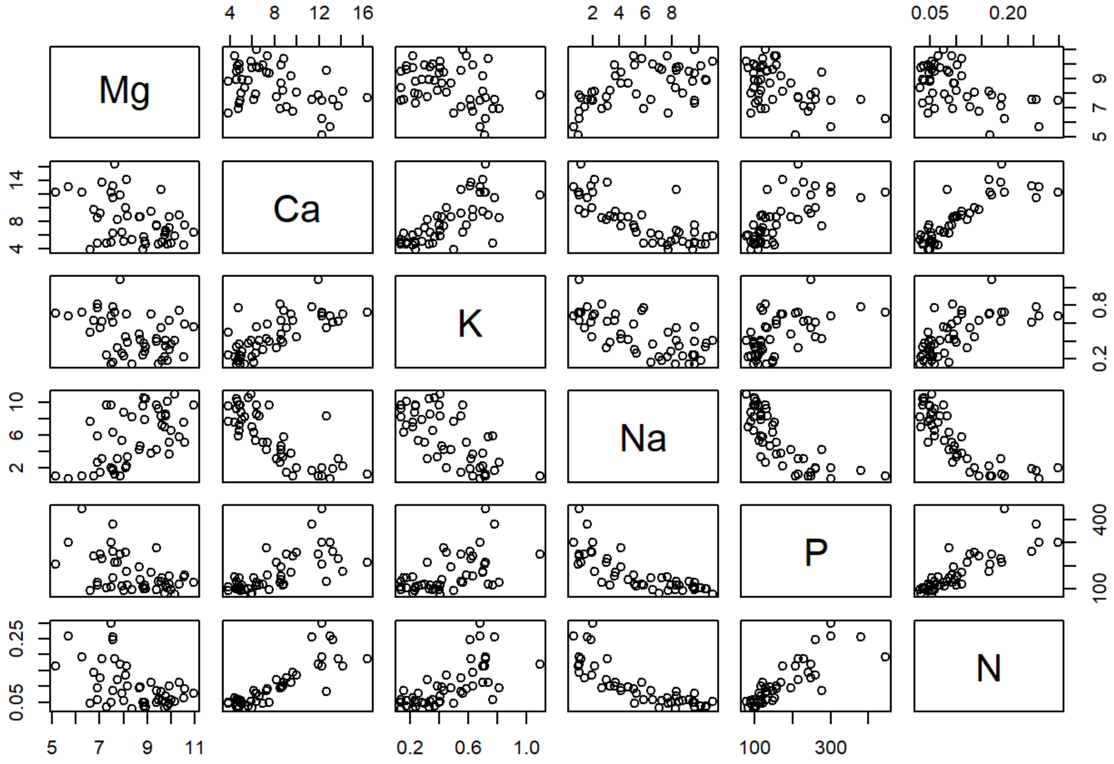
Can’t be visualized anymore. Estimates look quite “empty”.

```
 soils5<- Soils[,c("Mg", "Ca", "K", "Na", "P")]
 plot(soils5)
```



```
h.soil5 <- hd.rot(soils5)
fit.soil5<-kde(soils5, H=diag(h.soil5^2))
```

```
soils6<- Soils[,c("Mg", "Ca", "K", "Na", "P", "N")]
plot(soils6)
```



```
h.soil6 <- hd.rot(soils6)
# fit.soil6<-kde(soils6, H=diag(h.soil6^2))
# this one took too long and in fact crashed my R session. Feel free to try :-)
```

Let's compare the bandwidths used again.

```
h.soil2
```

```
## [1] 0.717701 1.707931
```

```
h.soil3
```

```
## [1] 0.7869971 1.8728366 0.1288146
```

```
h.soil4
```

```
## [1] 0.8433256 2.0068830 0.1380344 2.0272460
```

```
h.soil5
```

```
## [1]  0.8899095  2.1177399  0.1456592  2.1392278 52.7886334
```

These are gently increasing as expected. Let's inspect a bit further.

Which grid sizes (G) have been used for each coordinate axis? (Note that, the number of evaluation points of the KDE is G^d!)

```
length(fit.soil2$eval.points[[1]])
```

```
## [1] 151
```

```
length(fit.soil3$eval.points[[1]])
```

```
## [1] 51
```

```
length(fit.soil4$eval.points[[2]])
```

```
## [1] 21
```

```
length(fit.soil5$eval.points[[3]])
```

```
## [1] 21
```

How many estimated density values does this result in?

```
length(fit.soil2$estimate)
```

```
## [1] 22801
```

```
length(fit.soil3$estimate)
```

```
## [1] 132651
```

```
length(fit.soil4$estimate)
```

```
## [1] 194481
```

```
length(fit.soil5$estimate)
```

```
## [1] 4084101
```

Observe, for instance, that $21^4 = 194481$.

How many (precisely) **zero** estimates are there?

```
sum(fit.soil2$estimate==0)
```

```
## [1] 3269
```

```
sum(fit.soil3$estimate==0)
```

```
## [1] 45473
```

```
sum(fit.soil4$estimate==0)
```

```
## [1] 0
```

```
sum(fit.soil5$estimate==0)
```

```
## [1] 0
```

How many de-facto **zero** estimates are there? (Let’s call a KDE value $\hat{f}_H(x)$ “de facto zero” if $\hat{f}_H(x) < 10^{-6} \times \max_z \hat{f}_H(z)$).

```
sum(fit.soil2$estimate<1e-6*max(fit.soil2$estimate))
```

```
## [1] 3681
```

```
sum(fit.soil3$estimate<1e-6*max(fit.soil3$estimate))
```

```
## [1] 57419
```

```
sum(fit.soil4$estimate<1e-6*max(fit.soil4$estimate))
```

```
## [1] 124389
```

```
sum(fit.soil5$estimate<1e-6*max(fit.soil5$estimate))
```

```
## [1] 3346029
```

Which proportion of estimates is precisely zero?

```
sum(fit.soil2$estimate==0)/length(fit.soil2$estimate)
```

```
## [1] 0.1433709
```

```
sum(fit.soil3$estimate==0)/length(fit.soil3$estimate)
```

```
## [1] 0.3428018
```

```
sum(fit.soil4$estimate==0)/length(fit.soil4$estimate)
```

```
## [1] 0
```

```
sum(fit.soil5$estimate==0)/length(fit.soil5$estimate)
```

```
## [1] 0
```

We see that, from $d = 2$ to $d = 3$, there is the expected behavior: The sparsity is gripping and therefore the proportion of estimates which are zero increases. However, what happens then is more worrying: From $d = 3$ to $d = 4$, the patches of “zero estimates” disappear, which is a “blurring” effect due to the larger bandwidths used.

Which proportion of estimates is de-facto zero?

```
sum(fit.soil2$estimate<1e-6*max(fit.soil2$estimate))/length(fit.soil2$estimate)
```

```
## [1] 0.1614403
```

```
sum(fit.soil3$estimate<1e-6*max(fit.soil3$estimate))/length(fit.soil3$estimate)
```

```
## [1] 0.4328576
```

```
sum(fit.soil4$estimate<1e-6*max(fit.soil4$estimate))/length(fit.soil4$estimate)
```

```
## [1] 0.6395946
```

```
sum(fit.soil5$estimate<1e-6*max(fit.soil5$estimate))/length(fit.soil5$estimate)
```

```
## [1] 0.8192816
```

The proportion of evaluation points attaining any non-negligible density values falls consistently with the dimension; the estimated density dilutes and gets “flattened out” across space as we proceed with higher dimensions.

We conclude that, under any bandwidth setting, KDE is practically useful at most until dimension $d = 3$ or 4. For higher dimensions, it becomes inherently impractical (both interpretationally and computationally). Exceptions could be situations where one is not interested in estimates on a grid, but desires to evaluate the density $\hat{f}_H$ at some specific points (and one does know where to look for points where this is sensible).

4 Nonparametric regression

4.1 Introduction

4.1.1 Problem setup

Let (X, Y) be continuous, univariate random variables with joint density $g(x, y)$ and marginal density of X as $f(x) = \int g(x, y)\, dy$. The densities f and g are unknown.

We are given data $(x_1, y_1), \ldots, (x_n, y_n)$ which form an i.i.d. sampe from (X, Y).

We are interested in a **model** for Y given X which does not require ``parametric” assumptions (such as in linear regression). Let us therefore assume (for now)

$$Y = m(X) + \tilde{\epsilon}$$

where $m : \mathbb{R} \longrightarrow \mathbb{R}$ is an unknown, twice differentiable (“smooth”) function, and $\tilde{\epsilon}$ is some random error with mean zero (which we give some more attention later in this introduction).

Note that the model above implies

$$E(Y|X=x)=m(x),$$

so the regression function $m(x)$ is just the conditional mean of Y given that X takes the value x.

Task: Estimate m.

4.1.2 Motivating kernel regression via KDE

To find an estimate of $m(x)$ at some fixed point x, note that

$$m(x)=E(Y|X=x)=\int y\,g(y|x)\,dy=\int y\frac{g(x,y)}{f(x)}\,dy$$

where have implicitly used the notation $g(y|x)$ for the conditional density of Y given $X=x$. We do know how to estimate the densities $f(\cdot)$ and $g(\cdot,\cdot)$ via kernel density estimation, namely

$$\begin{aligned}\hat{f}\,(x)&=\frac{1}{nh_1}\sum_{i=1}^{n}K\left(\frac{x_i-x}{h_1}\right);\\ \hat{g}(x,y)&=\frac{1}{nh_1h_2}\sum_{i=1}^{n}K\left(\frac{x-x_i}{h_1}\right)K\left(\frac{y-y_i}{h_2}\right).\end{aligned}$$

Then we can estimate

$$\begin{aligned}\hat{m}(x)&=\int y\frac{\hat{g}(x,y)}{\hat{f}\,(x)}\,dy\\ &=\int y\frac{\frac{1}{nh_1h_2}\sum_{i=1}^{n}K\left(\frac{x-x_i}{h_1}\right)K\left(\frac{y-y_i}{h_2}\right)}{\frac{1}{nh_1}\sum_{i=1}^{n}K\left(\frac{x-x_i}{h_1}\right)}\,dy\\ &=\frac{1}{h_2}\frac{1}{\sum_{i=1}^{n}K\left(\frac{x-x_i}{h_1}\right)}\times\sum_{i=1}^{n}K\left(\frac{x-x_i}{h_1}\right)\int yK\left(\frac{y-y_i}{h_2}\right)\,dy\end{aligned}$$

Now use the transformation $z=\frac{y-y_i}{h_2}$ which implies

$$y=h_2z+y_i$$

and hence also $dy=h_2dz$. Focusing on the intergral for the moment,

$$\begin{aligned}\int yK\left(\frac{y-y_i}{h_2}\right)\,dy&=\int(h_2z+y_i)K(z)h_2\,dz\\ &=h_2^2\int zK(z)\,dz+h_2y_i\int K(z)\,dz\\ &=h_2y_i\end{aligned}$$

Hence we have

$$\begin{aligned}\hat{m}(x)&=\frac{1}{h_2}\frac{1}{\sum_{i=1}^{n}K\left(\frac{x-x_i}{h_1}\right)}\times\sum_{i=1}^{n}K\left(\frac{x-x_i}{h_1}\right)h_2y_i\\ &=\frac{\sum_{i=1}^{n}y_iK\left(\frac{x-x_i}{h_1}\right)}{\sum_{i=1}^{n}K\left(\frac{x-x_i}{h_1}\right)}.\end{aligned}$$

We see that the vertical bandwidth h_2 does not matter. Writing then $h \equiv h_1$, this becomes

$$\hat{m}_{NW}(x) = \frac{\sum_{i=1}^n y_i K\left(\frac{x-x_i}{h}\right)}{\sum_{i=1}^n K\left(\frac{x-x_i}{h}\right)}$$

which is known as the **Nadaraya-Watson** estimator of $m(x)$. Note that, for each point $x \in \mathbb{R}$, this is just a weighted mean of all (response) observations, with those responses y_i which are (in terms of their associated predictor values x_i) further away from x receiving smaller weight.

Example 4.1

We will consider two data sets for illustrative purposes:

- the `fossil` data, with ratios of certain strontium isotopes in $n = 106$ fossil shells versus age (in millions of years);
- the `lidar` data, which are from a LIDAR (**Li**ght **D**etection **a**nd **R**anging) experiment with $n = 221$.

Accessing these data requires installation of the R package **SemiPar**.

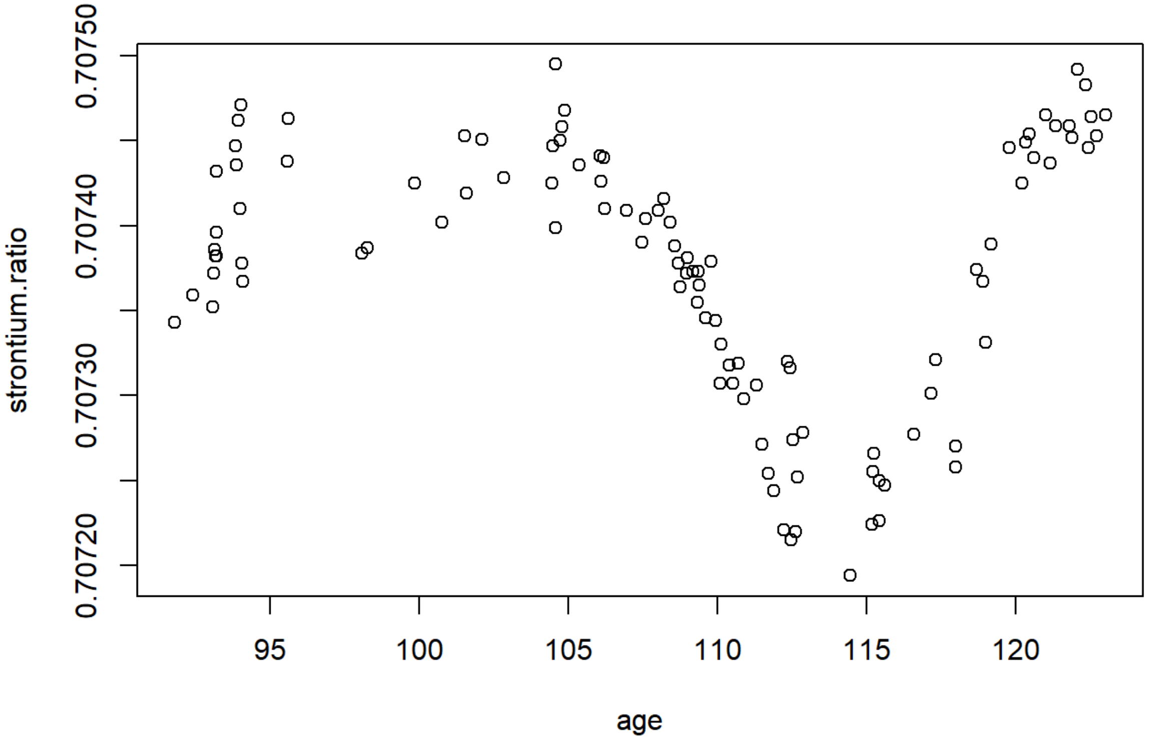
```
# install.packages("SemiPar")
require(SemiPar)
```

```
## Loading required package: SemiPar
```

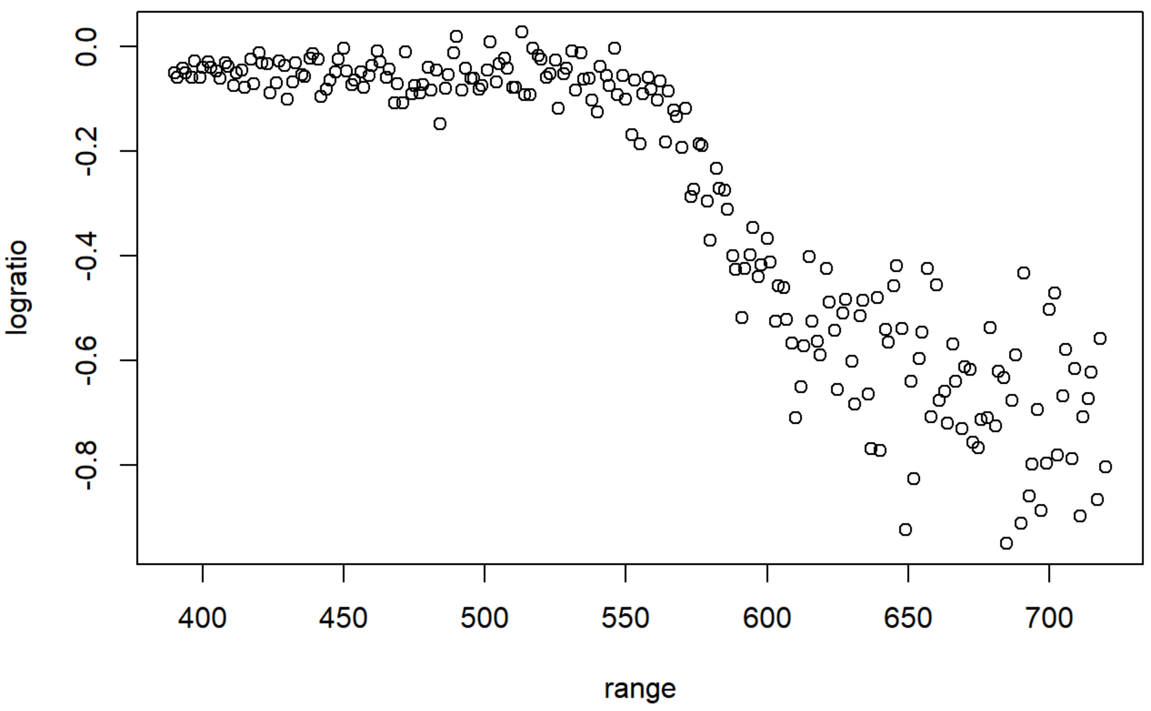
```
##
## Attaching package: 'SemiPar'
```

```
## The following object is masked from 'package:ks':
##
##     matrix.sqrt
```

```
data(fossil)
plot(fossil)
```



```
data(lidar)
plot(lidar)
```

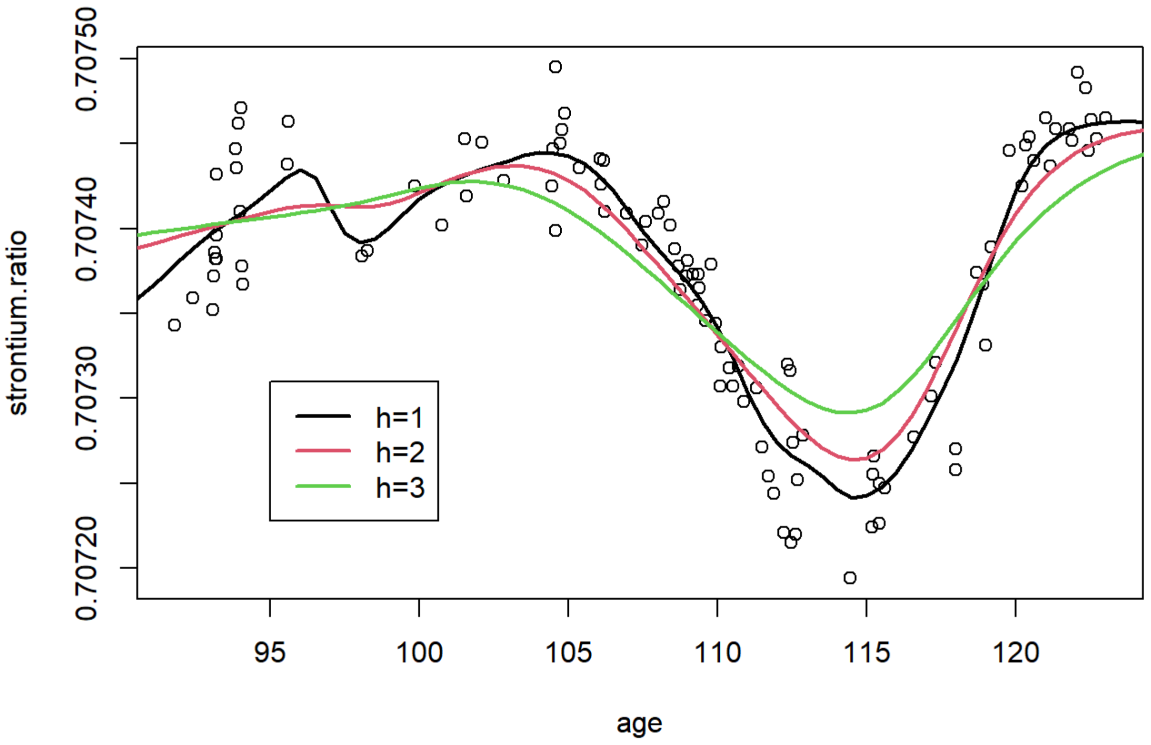


We firstly implement the Nadaraya-Watson estimator, for which we again make use of our `my.kernel` function implemented earlier:

```
my.kernel.smoother<- function(xdat, ydat, xgrid=xdat, h){
  G<- length(xgrid)
  est<-rep(0,G)
  for (j in 1:G){
  est[j]<-
      sum(ydat*my.kernel((xgrid[j]-xdat)/h))/sum(my.kernel((xgrid[j]-xdat)/h))
  }
  return(as.data.frame(cbind(xgrid, est)))
}
```

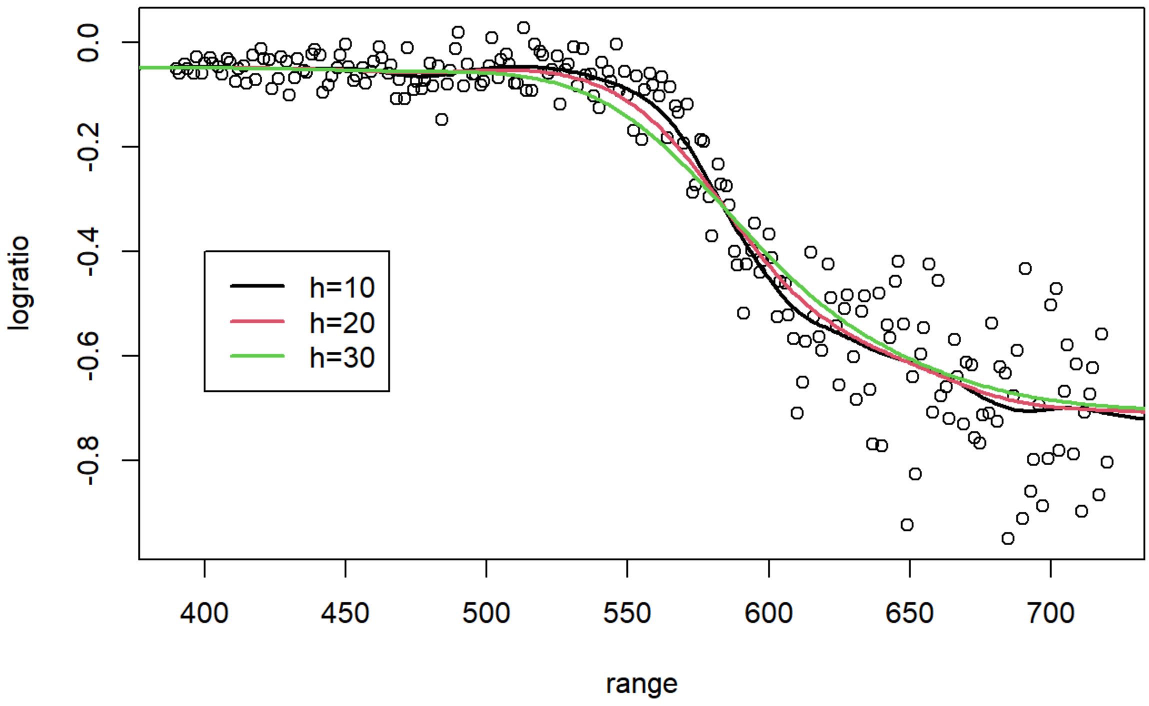
We apply this function for a range of bandwidths on the `fossil` data….

```
age.grid<- seq(90, 130, by=0.5)
fit1<- my.kernel.smoother(fossil$age, fossil$strontium.ratio, age.grid, h=1)
fit2<- my.kernel.smoother(fossil$age, fossil$strontium.ratio, age.grid, h=2)
fit3<- my.kernel.smoother(fossil$age, fossil$strontium.ratio, age.grid, h=3)
plot(fossil)
lines(fit1, col=1, lwd=2)
lines(fit2, col=2, lwd=2)
lines(fit3, col=3, lwd=2)
legend(95,0.70731, c("h=1", "h=2", "h=3"), lwd=c(2,2,2), lty=c(1,1,1), col=c(1,2,3))
```



… and on the `lidar` data:

```
range.grid<-350:750
fitl1<- my.kernel.smoother(lidar$range, lidar$logratio, range.grid, h=10)
fitl2<- my.kernel.smoother(lidar$range, lidar$logratio, range.grid, h=20)
fitl3<- my.kernel.smoother(lidar$range, lidar$logratio, range.grid, h=30)
plot(lidar)
lines(fitl1, col=1, lwd=2)
lines(fitl2, col=2, lwd=2)
lines(fitl3, col=3, lwd=2)
legend(400,-0.4, c("h=10", "h=20", "h=30"), lwd=c(2,2,2), lty=c(1,1,1), col=c(1,2,3))
```



The role of the bandwidth is hence similar as before; such as

- if h is small, we get a “rough” estimate, with low bias, and high variance;
- if h is large, we get a “smooth” estimate, with high bias, and low variance.

The sensitivity of the estimates to the bandwidth seems however not as strong as for the KDEs.

4.1.3 The nonparametric regression model

Looking back at the data examples from Example 4.1, we observe that for the `fossil` data, the variance of the data around the estimated curves appears to be approximately constant along the x axis, whereas for the `lidar` data this is clearly not the case since the data show a “fanning out” effect for high `range` values. Such an unequal variance across the values of x is known as heteroscedasticity, which our model should account for.

Therefore, the nonparametric regression model for i.i.d. data (x_i, y_i), $i = 1, \dots, n$, is generally formulated as

$$Y = m(X) + \sigma(X)\epsilon$$

where $E(\epsilon) = 0$, $\mathrm{Var}(\epsilon) = 1$, ϵ independent of X, and hence

$$E(Y|X=x) = m(x) + \sigma(x) \times 0 = m(x);$$
$$\mathrm{Var}(Y|X=x) = 0 + \sigma^2(x) \times 1 = \sigma^2(x).$$

Models of this type are also called *location-scale models* since both location ($m(x)$) and scale ($\sigma^2(x)$) of Y can be described by a single model.

If $\sigma^2(x) \equiv \sigma^2$ for all x then we call the model *homoscedastic*, otherwise *heteroscadastic*.

The main interest clearly resides in the estimation of $m(x)$. The variance (scale) term is mainly relevant for correct inference on $m(x)$ (such as, variance of its estimate) rather than of intrinsic interest by itself.

4.2 Localized regression

4.2.1 Kernel regression as a weighted least squares problemn

Recall the Nadaraya-Watson (NW) estimator

$$\hat{m}_{NW}(x) = \frac{\sum_{i=1}^n y_i K\left(\frac{x-x_i}{h}\right)}{\sum_{i=1}^n K\left(\frac{x-x_i}{h}\right)}$$

Note this is just the solution of

$$\text{arg min}_{\beta_0} \sum_{i=1}^n K\left(\frac{x-x_i}{h}\right)(y_i - \beta_0)^2$$

Proof: Let us abbreviate $w_i(x) = K\left(\frac{x-x_i}{h}\right)$, and define

$$Q(\beta_0) = \sum_{i=1}^n w_i x(y_i - \beta_0)^2$$

which is the quantity to minimize. We do this by setting its derivative equal to zero,

$$Q'(\beta_0) = -2\sum_{i=1}^n w_i(x)(y_i - \beta_0) \overset{!}{=} 0$$
$$\sum_{i=1}^n w_i(x) y_i = \beta_0 \sum_{i=1}^n w_i(x)$$
$$\beta_0 = \frac{\sum_{i=1}^n y_i w_i(x)}{\sum_{i=1}^n w_i(x)}$$

—

So, de facto $m(x) \approx \beta_0$ acts as a “localized model” for m in a neighborhood of x, say $(x-h, x+h)$. We could improve this localized model by a Taylor expansion of first order

$$m(x) \approx \beta_0 + \beta_1(z - x).$$

This then would require to estimate two parameters, β_0 and β_1, by

$$\text{arg min}_{\beta_0,\beta_1} \sum_{i=1}^n K\left(\frac{x-x_i}{h}\right)(y_i - \beta_0 - \beta_1(x_i - x))^2$$

Note that, by Taylor’s theorem,

$$\beta_0 = m(x)$$
$$\beta_1 = m'(x)$$

(Observe that the latter property would enable us to also do *derivative estimation*).

4.2.2 The local linear regression estimator

Consider

$$Q(\beta_0, \beta_1) = \sum_{i=1}^{n} w_i(x)(y_i - \beta_0 - \beta_1(x_i - x))^2$$

where we are again using the abbreviation $w_i(x) \equiv K\left(\frac{x_i - x}{h}\right)$, and β_0 and β_1 are unknown parameters (Note that these actually depend on x, so we could write $\beta_0(x)$ and $\beta_1(x)$, but this depencene on x is suppressed for notational ease. It is important to be aware of this, though.)

Now we find the values of β_0, β_1 at which $Q(\beta_0, \beta_1)$ attains its minimum by setting the gradient of Q equal to zero,

$$\frac{\partial Q}{\partial \beta_0} = -2\sum_{i=1}^{n} w_i(x)(y_i - \beta_0 - \beta_1(x_i - x))$$
$$\frac{\partial Q}{\partial \beta_1} = -2\sum_{i=1}^{n} w_i(x)(y_i - \beta_0 - \beta_1(x_i - x))(x_i - x)$$

This yields the two equations

$$(1) \quad \sum_{i=1}^{n} w_i(x)y_i = \beta_0 \sum_{i=1}^{n} w_i(x) + \beta_1 \sum_{i=1}^{n} w_i(x)(x_i - x)$$
$$(2) \quad \sum_{i=1}^{n} w_i(x)y_i(x_i - x) = \beta_0 \sum_{i=1}^{n} w_i(x)(x_i - x) + \beta_1 \sum_{i=1}^{n} w_i(x)(x_i - x)^2$$

For better handling define the new notation $S_{n,j} = \sum_{i=1}^{n} w_i(x)(x_i - x)^j$, $j = 0, 1, 2, \cdots$.

This means we can rewrite (1) and (2) as

$$(1) \quad \sum_{i=1}^{n} w_i(x)y_i = \beta_0 S_{n,0} + \beta_1 S_{n,1}$$
$$(2) \quad \sum_{i=1}^{n} w_i(x)y_i(x_i - x) = \beta_0 S_{n,1} + \beta_1 S_{n,2}$$

Now we apply $(1) \times \frac{S_{n,2}}{S_{n,1}} - (2)$ to obtain

$$\frac{S_{n,2}}{S_{n,1}} \sum w_i(x)y_i - \sum w_i(x)y_i(x_i - x) = \beta_0 \frac{S_{n,0}S_{n,2}}{S_{n1}} - \beta_0 S_{n,1}$$
$$S_{n,2} \sum_{i=1}^{n} w_i(x)y_i - S_{n,1} \sum_{i=1}^{n} w_i(x)y_i(x_i - x) = \beta_0 \left(S_{n,0}S_{n,2} - S_{n,1}^2\right)$$

Note that we can write the bracket in the denominator as

$$S_{n,0}S_{n,2} - S_{n,1}^2 = \sum_{i=1}^{n} w_i(x)S_{n,2} - S_{n,1} \sum_{i=1}^{n} w_i(x)(x_i - x) = \sum_{i=1}^{n} w_i(x)(S_{n,2} - (x_i - x)S_{n,1})$$

Hence, the above becomes

$$\hat{\beta}_0 = \frac{\sum_{i=1}^n w_i(x)(S_{n,2} - (x_i - x)S_{n,1})y_i}{\sum_{i=1}^n w_i(x)(S_{n,2} - (x_i - x)S_{n,1})}$$

(hats above β_0 and β_1 omitted in all but last equation for ease of notation). Let us now define the weights

$$v_i(x) = w_i(x)\left(S_{n,2} - (x_i - x)S_{n,1}\right).$$

Therefore, we obtain the *local linear regression estimator*

$$\hat{m}(x) = \hat{\beta}_0 = \frac{\sum_{i=1}^n v_i(x)y_i}{\sum_{i=1}^n v_i(x)}$$

which is, just as the Nadaraya-Watson estimator, a locally weighted sum of the observations (albeit a more complicated one.)

Example 4.2

We would like to apply the local linear regression estimator on the `fossil` and `lidar` data. In order to implement this estimator, we firstly define auxiliary functions for $S_{n,j}$ and $v_i(x)$:

```
Sn<- function(xdat, x, h, j){
snj<-  sum(my.kernel((xdat-x)/h)*(xdat-x)^j)
return(snj)
}
```

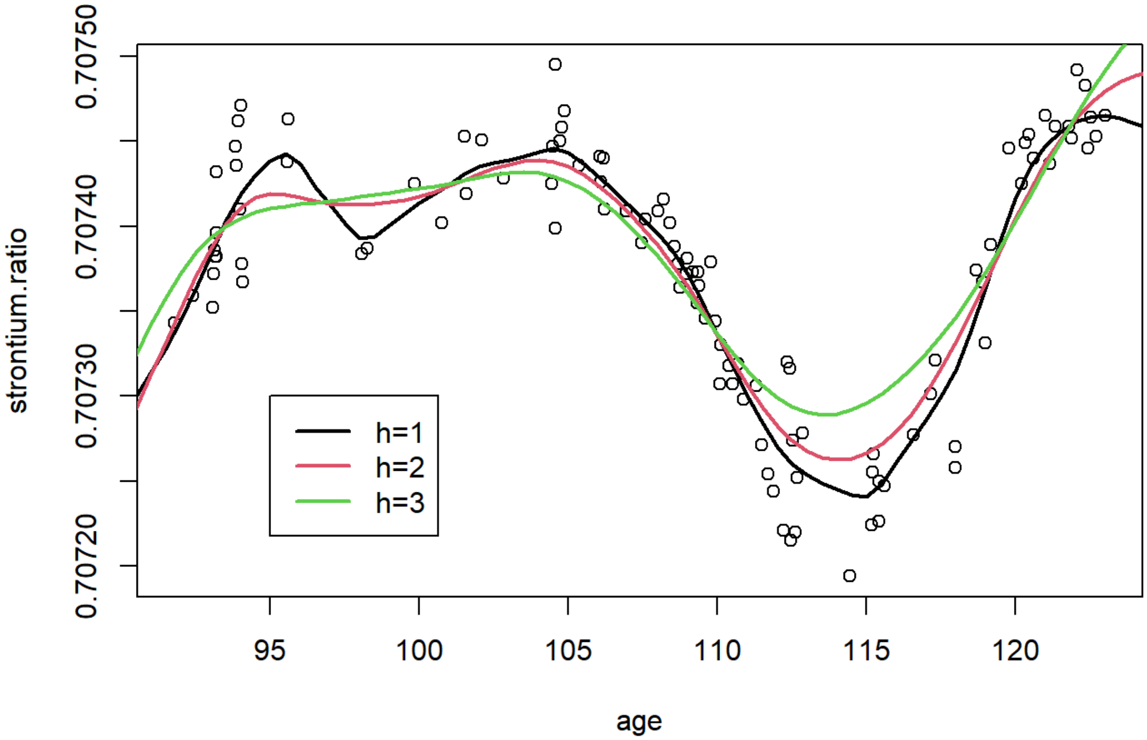
```
vix<-function(xdat, x, h){
  my.kernel((xdat-x)/h)*( Sn(xdat,x,h,2)-(xdat-x)*Sn(xdat,x,h,1) )
}
# produces the entire vector v_1(x),...v_n(x)
```

Now we are in a position to implement the function for the actual local linear smoother:

```
my.ll.smoother<- function(xdat, ydat, xgrid=xdat, h){
  G<- length(xgrid)
  est<-rep(0,G)
  for (j in 1:G){
  est[j]<-
     sum(ydat*vix(xdat,xgrid[j],h))/sum(vix(xdat,xgrid[j],h))
  }
  return(as.data.frame(cbind(xgrid, est)))
}
```

We apply this function now again on the `fossil` data.

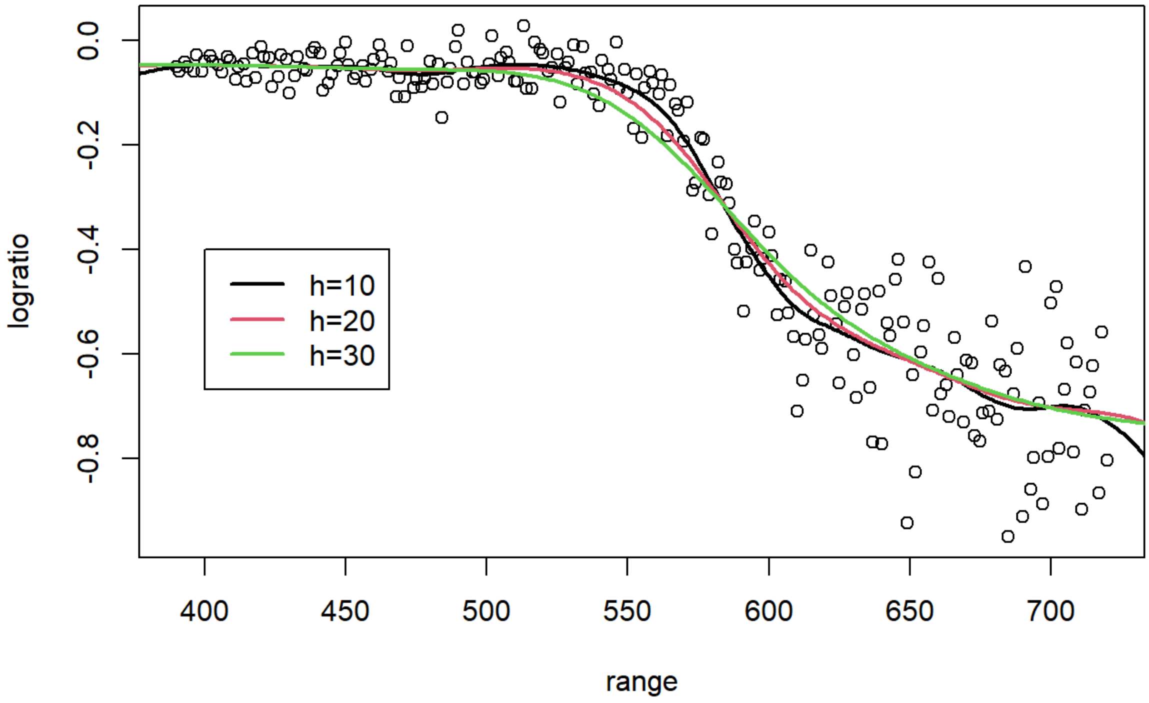
```
age.grid<- seq(90, 130, by=0.5)
fit1<- my.ll.smoother(fossil$age, fossil$strontium.ratio, age.grid, h=1)
fit2<- my.ll.smoother(fossil$age, fossil$strontium.ratio, age.grid, h=2)
fit3<- my.ll.smoother(fossil$age, fossil$strontium.ratio, age.grid, h=3)
plot(fossil)
lines(fit1, col=1, lwd=2)
lines(fit2, col=2, lwd=2)
lines(fit3, col=3, lwd=2)
legend(95,0.70730, c("h=1", "h=2", "h=3"), lwd=c(2,2,2), lty=c(1,1,1), col=c(1,2,3))
```



The most obvious difference to the NW-estimator is the behavior in the left boundary: the smoother follows here the linear trend, rather than incorrectly flattening out at the boundary. The behavior in areas of strong curvature looks also improved. What we see here is a bias-reducing effect of the local linear estimate.

We now apply the local linear estimator on the `lidar` data.

```
range.grid<-350:750
fitl1<- my.ll.smoother(lidar$range, lidar$logratio, range.grid, h=10)
fitl2<- my.ll.smoother(lidar$range, lidar$logratio, range.grid, h=20)
fitl3<- my.ll.smoother(lidar$range, lidar$logratio, range.grid, h=30)
plot(lidar)
lines(fitl1, col=1, lwd=2)
lines(fitl2, col=2, lwd=2)
lines(fitl3, col=3, lwd=2)
legend(400,-0.4, c("h=10", "h=20", "h=30"), lwd=c(2,2,2), lty=c(1,1,1), col=c(1,2,3))
```



Here less difference to the NW estimator is noticeable, except perhaps at the right boundary where the local linear estimator for $h = 10$ seems to overinterpret a potentially emerging linear trend.

4.2.3 Local polynomial regression

Consider Taylor's theorem in ``Young's form": Assume $m^{(p)}$ is bounded at x, then

$$m(z) = \sum_{j=0}^{p} \frac{m^{(j)}(x)}{j!}(z-x)^j + o(|z-x|^p)$$

as $|z - x| \longrightarrow 0$.

This suggests we can locally approximate $m(x)$ by

$$m(z) \approx \sum_{j=0}^{p} \beta_j (z-x)^j,$$

where $\beta_j \equiv \frac{m^{(j)}(x)}{j!}$, and estimate $\beta_0, \ldots, \beta_p$ by minimizing

$$Q(\beta_0, \ldots, \beta_p) = \sum_{i=1}^{n} K\left(\frac{x_i - x}{h}\right)\left(y_i - \sum_{j=0}^{p} \beta_j (x_i - x)^j\right)^2$$

w.r.t. $\beta_0, \ldots, \beta_p$.

In matrix notation, define

$$\boldsymbol{X} = \begin{pmatrix} 1 & x_1 - x & & (x_i - x)^p \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n - x & & (x_n - x)^p \end{pmatrix}, \qquad \boldsymbol{Y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \qquad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_p \end{pmatrix},$$

$$\boldsymbol{W} = \begin{pmatrix} K\left(\frac{x_1 - x}{h}\right) & & \\ & \ddots & \\ & & K\left(\frac{x_n - x}{h}\right) \end{pmatrix} = \begin{pmatrix} w_1(x) & & \\ & \ddots & \\ & & w_n(x) \end{pmatrix}.$$

Then

$$Q(\boldsymbol{\beta}) = (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta})^T \boldsymbol{W} (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta})$$

We now minimize this w.r.t the vector $\boldsymbol{\beta}$:

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\beta}} Q(\boldsymbol{\beta}) = -2\boldsymbol{X}^T \boldsymbol{W} (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}) &\overset{!}{=} 0 \\ \boldsymbol{X}^T \boldsymbol{W} \boldsymbol{Y} &= \boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X} \boldsymbol{\beta} \end{aligned}$$

with solution

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{W} \boldsymbol{Y}$$

(which takes the familar shape of the multiple linear regression estimator, but with the additional weight matrix $\boldsymbol{W}$).

Now define vectors $e_j^T = (0, \dots, 0, 1, 0, \dots, 0) \in \mathbb{R}^{p+1}$ with 1 at j-th position; in particular $e_1^T = (1, 0, \dots, 0)$. Then we see easily

$$\begin{aligned} \hat{m}(x) = \hat{\beta}_0 &= e_1^T \boldsymbol{\beta} \\ &= e_1^T (\boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{W} \boldsymbol{Y} \\ &= \begin{cases} \hat{m}_{NW}(x) & \text{if } p = 0 \\ \hat{m}_{LL}(x) & \text{if } p = 1 \end{cases} \end{aligned}$$

(The last equality is not obvious, but can be shown with a bit of algebra when using the respective forms of $\boldsymbol{X}$ for $p = 0$ and $p = 1$, respectively.)

We also see that a *derivative estimate* can be obtained as

$$\hat{m}'(x) = \hat{\beta}_1 = e_2^T \hat{\boldsymbol{\beta}} = e_2^T (\boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{W} \boldsymbol{Y}$$

4.2.4 Bias and variance

From the matrix representation of $\hat{\boldsymbol{\beta}}$, we find

$$\text{Bias}(\hat{\boldsymbol{\beta}}) = E(\hat{\boldsymbol{\beta}}) - \boldsymbol{\beta} = (\boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{W} E(\boldsymbol{Y}) - \boldsymbol{\beta}.$$

It is worth investing a few thoughts on what $E(\boldsymbol{Y})$ actually is. This *could* have been interpreted as the expectation of the marginal density of $\boldsymbol{Y}$, that is after integration over the predictors. However this is not what is meant and what is usually done. Recall that, for (parametric) multiple regression, all inferences are always conditional on the observed predictor values. Precisely the same is done here. That is, $E(Y)$ is in fact the expectation of the responses conditional on $x_1, \dots, x_n$ (even though usually we will not make this explicit notationally), that is

$$\begin{aligned} E(\boldsymbol{Y}) \equiv E(\boldsymbol{Y}|x_1,\dots,x_n) &= E\left(\begin{array}{c|c} y_1 & \\ \vdots & x_1,\dots,x_n \\ y_n & \end{array}\right) \\ &= \begin{pmatrix} E(y_1|x_1,\dots,x_n) \\ \vdots \\ E(y_n|x_1,\dots,x_n) \end{pmatrix} = \begin{pmatrix} m(x_1) \\ \vdots \\ m(x_n) \end{pmatrix} \equiv \boldsymbol{M}. \end{aligned}$$

So

$$\begin{aligned} \mathrm{Bias}(\hat{\boldsymbol{\beta}}) &= (\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{M} - \boldsymbol{\beta} \\ &= (\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{W}(\boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{M} - \boldsymbol{X}\boldsymbol{\beta}) - \boldsymbol{\beta} \\ &= \boldsymbol{\beta} + (\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{W}(\boldsymbol{M} - \boldsymbol{X}\boldsymbol{\beta}) - \boldsymbol{\beta} \\ &= (\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{W}(\boldsymbol{M} - \boldsymbol{X}\boldsymbol{\beta}) \end{aligned}$$

Note that in this expression, both $\boldsymbol{M}$ and $\boldsymbol{\beta}$ are unknown. Before we go further with this, let's have a look at the variance.

$$\begin{aligned} \mathrm{Var}(\hat{\boldsymbol{\beta}}) &= \mathrm{Var}\left((\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{Y}\right) \\ &= (\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{W}\mathrm{Var}(\boldsymbol{Y})\boldsymbol{W}\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1} \end{aligned}$$

where we again examine $\mathrm{Var}(\boldsymbol{Y})$ as follows

$$\begin{aligned} \mathrm{Var}(\boldsymbol{Y}) &= \mathrm{Var}\left(\begin{array}{c|c} y_1 & \\ \vdots & x_1,\dots,x_n \\ y_n & \end{array}\right) \\ &= [\mathrm{Cov}(y_i, y_j|x_i, x_j)]_{1\le i,j\le n} \\ &= \begin{pmatrix} \mathrm{Var}(y_1|X=x_1) & & \\ & \ddots & \\ & & \mathrm{Var}(y_n|X=x_n) \end{pmatrix} \\ &= \begin{pmatrix} \sigma^2(x_1) & & \\ & \ddots & \\ & & \sigma^2(x_n) \end{pmatrix} \equiv \boldsymbol{V} \end{aligned}$$

so that

$$\mathrm{Var}(\hat{\boldsymbol{\beta}}) = (\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{V}\boldsymbol{W}\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}.$$

Note that in this expression, only $\boldsymbol{V}$ is unknown.

In particular, we get for the bias and variance of the estimate of the regression mean function,

$$\begin{aligned} \mathrm{Bias}(\hat{m}(x)) &= \mathrm{Bias}(e_1^T\hat{\boldsymbol{\beta}}) = e_1^T\mathrm{Bias}(\hat{\boldsymbol{\beta}}) \\ &= e_1^T(\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{W}(\boldsymbol{M} - \boldsymbol{X}\boldsymbol{\beta}); \\ \mathrm{Var}(\hat{m}(x)) &= \mathrm{Var}(e_1^T\hat{\boldsymbol{\beta}}) = e_1^T\mathrm{Var}(\hat{\boldsymbol{\beta}})e_1 \\ &= e_1^T(\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{V}\boldsymbol{W}\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}e_1 \end{aligned}$$

4.2.5 Asymptotic approximations

We focus here on the local linear case, that is $p = 1$, and will only give the proof for the asymptotic bias explicitly. Full results and derivations can be found in Wand & Jones, 1995, and Fan & Gijbels, 1996.

So, in the case $p = 1$, recall that

$$\boldsymbol{\beta} = \begin{pmatrix} m(x) \\ m'(x) \end{pmatrix}, \qquad \boldsymbol{X} = \begin{pmatrix} 1 & x_1 - x \\ \vdots & \vdots \\ 1 & x_n - x \end{pmatrix}, \qquad \boldsymbol{M} = \begin{pmatrix} m(x_1) \\ \vdots \\ m(x_n) \end{pmatrix}$$

So,

$$\begin{aligned} \boldsymbol{M} - \boldsymbol{X}\boldsymbol{\beta} &= \begin{pmatrix} m(x_1) \\ \vdots \\ m(x_n) \end{pmatrix} - \begin{pmatrix} 1 & x_1 - x \\ \vdots & \vdots \\ 1 & x_n - x \end{pmatrix} \begin{pmatrix} m(x_1) \\ \vdots \\ m(x_n) \end{pmatrix} \\ &= \begin{pmatrix} m(x_1) - m(x) - m'(x)(x_1 - x)) \\ \vdots \\ m(x_n) - m(x) - m'(x)(x_n - x) \end{pmatrix} \end{aligned}$$

We know, by Taylor expansion,

$$m(x_i) = m(x) + (x_i - x)m'(x) + \frac{1}{2}(x_i - x)^2 m''(x) + o(|x_i - x|^2)$$

Hence

$$\mathrm{Bias}(\hat{m}(x)) = \frac{1}{2} e_1^T m''(x)(\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{W} \begin{pmatrix} (x_1 - x)^2 \\ \vdots \\ (x_n - x)^2 \end{pmatrix}$$

plus a remainder term which is that $o(\cdot)$ of the given expression. Next, we work out

$$\begin{aligned} \boldsymbol{X}^T\boldsymbol{W} \begin{pmatrix} (x_1 - x)^2 \\ \vdots \\ (x_n - x)^2 \end{pmatrix} &= \begin{pmatrix} 1 & \cdots & 1 \\ x_1 - x & \cdots & x_n - x \end{pmatrix} \begin{pmatrix} w_1(x) & & \\ & \ddots & \\ & & w_n(x) \end{pmatrix} \begin{pmatrix} (x_1 - x)^2 \\ \vdots \\ (x_n - x)^2 \end{pmatrix} \\ &= \begin{pmatrix} 1 & \cdots & 1 \\ x_1 - x & \cdots & x_n - x \end{pmatrix} \begin{pmatrix} w_1(x)(x_1 - x)^2 \\ \vdots \\ w_n(x)(x_n - x)^2 \end{pmatrix} \\ &= \begin{pmatrix} \sum_{i=1}^n w_i(x)(x_i - x)^2 \\ \sum_{i=1}^n w_i(x)(x_i - x)^3 \end{pmatrix} = \begin{pmatrix} S_{n,2} \\ S_{n,3} \end{pmatrix} \end{aligned}$$

Similarly,

$$\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X} = \begin{pmatrix} S_{n,0} & S_{n,1} \\ S_{n,1} & S_{n,2} \end{pmatrix}.$$

It remains to approximate

$$S_{n,j} = \sum_{i=1}^n w_i(x)(x_i - x)^j = \sum_{i=1}^n K\left(\frac{x_i - x}{h}\right)(x_i - x)^j$$

by

$$\begin{aligned}
E(S_{n,j}) &= n\int K\left(\frac{z-x}{h}\right)(z-x)^j f(z)\,dz\\
&= n\int K(u)(hu)^j f(x+hu)h\,du\\
&= nh^{j+1}\int u^j K(u)(f(x)+huf'(x)+O(h^2))\,du\\
&= nh^{j+1}\left(f(x)\int u^jK(u)\,du + hf'(x)\int u^{j+1}K(u)\,du + O(h^2)\right)\\
&= nh^{j+1}\left(f(x)\mu_j + hf'(x)\mu_{j+1} + O(h^2)\right.
\end{aligned}$$

where we have used the transformation $(z-x)/h = u$, hence $dz = h\,du$, and also our earlier notation

$$\mu_j = \int u^j K(u)\,du.$$

[Technical note: Why are we allowed to approximate $S_{n,j}$ by ES_{nj}? We do not want to get hung up on this point as it is technicality which in itself not very interesting, but basically one can show from Chebychev's inequality that for any sequence of real random variables X_n it holds that $X_n = EX_n + O_P\left(\sqrt{\mathrm{Var}(X_n)}\right)$, where O_P means "bounded in probability for large n". So, if the variance term vanishes for large n, which one can show for the $S_{n,j}$ with some effort to be the case, then $ES_{n,j}$ is approximately equal to $S_{n,j}$, with probability approaching 1 for large n. In the literature, this property is also expressed as $S_{n,j} = ES_{n,j} + o_p(1)$, where $o_P(1)$ is a sequence which converges to 0, with a probability converging to 1, for large n. So, technically all appearances of $o(1)$ downstream in this proof are then to be understood as $o_P(1)$ too, even though we will refrain from making this notationally explicit in this exposition. See also Wand & Jones, Appendix A].

It is now important to spot that, for symmetric kernels, the odd moments μ_1, μ_3, are equal to zero. With that, we find

$$\begin{aligned}
\boldsymbol{X}^T\boldsymbol{W}\begin{pmatrix}(x_1-x)^2\\ \vdots\\ (x_n-x)^2\end{pmatrix} &= \begin{pmatrix} nh^3(f(x)\mu_2+o(1))\\ nh^5(f(x)\mu_4+o(1))\end{pmatrix}\\
\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X} &= \begin{pmatrix} nh(f(x)+o(1)) & nh^3(f'(x)\mu_2+o(1))\\ nh^3(f'(x)\mu_2+(o(1)) & nh^3(f(x)\mu_2+o(1))\end{pmatrix}.
\end{aligned}$$

Now, note that the expression for the Bias term above contains the product

$$e_1^T(\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}.$$

where $e_1^T = (1,0)$. This means, we actually only need to work out the first row of $(\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}$. Hence we have

$$(\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1} = \frac{1}{n^2h^4f^2(x)\mu_2 - n^2h^6f'(x)^2\mu_2^2}\begin{pmatrix} nh^3(f(x)\mu_2+o(1)) & -nh^3(f'(x)\mu_2+o(1)\\ \cdots & \cdots\end{pmatrix}$$

so that we can finally compute

$$\begin{aligned}
\mathrm{Bias}(\hat{m}_{LL}(x)) &= \frac{1}{2}m''(x)\frac{h^2}{nh^3f^2(x)\mu_2 - nh^5f'(x)\mu_2}\underbrace{(f(x)\mu_2+o(1), -f'(x)\mu_2+o(1))\begin{pmatrix} nh^3(f(x)\mu_2+o(1))\\ nh^5(f'(x)\mu_4+o(1))\end{pmatrix}}_{nh^3(f^2(x)\mu_2^2+o(1))}\\
&= \frac{1}{2}h^2m''(x)\mu_2 + o(h^2).
\end{aligned}$$

For the variance, a similar calculation can be carried out, which requires, beside the already available approximation of $(\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}$, also an approximation of $\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{V}\boldsymbol{W}\boldsymbol{X}$. This is not more difficult but we omit this here, see also Wand & Jones, page 122, or Fan & Gijbels, page 101. The resulting expression takes the shape

$$\mathrm{Var}(\hat{m}_{LL}(x)) = \frac{1}{nh}\frac{\nu_0}{f(x)} + o\left(\frac{1}{nh}\right),$$

recalling our earlier notation $\nu_j = \int u^j K^2(u)\,du$.

Concluding, we see that, for the local linear estimator, when the bandwidth increases, the bias increases and the variance decreases, as expected. Furthermore, the bias is larger in areas of high curvature of the regression function, and the variance is larger for regions of high error variance or *low* data density. It may also be noted that the local linear estimator is approximately unbiased if the function m is in fact linear.

Corresponding formulae for the local constant (Nadaraya-Watson) can be derived under the setting $p = 0$ (see also Fan & Gijbels (1996), page 17, 62):

$$\begin{aligned}
\mathrm{Bias}(\hat{m}_{NW}(x)) &= \frac{1}{2}h^2\mu_2\left(m''(x) + 2m'(x)\frac{f'(x)}{f(x)}\right) + o(h^2)\\
\mathrm{Var}(\hat{m}_{NW}(x)) &= \frac{1}{nh}\frac{\nu_0}{f(x)} + o\left(\frac{1}{nh}\right)
\end{aligned}$$

The key takeaway from these expressions is that the bias contains an additional term (as compared to the local linear estimator), which depends on the slope of the regression function and the design density, whereas the variance term is exactly the same as for the local linear estimator. This means, in turn, that the local linear estimator *reduces the bias without an increase in variance*. Therefore, in practice, it is the local linear estimator that should be used with preference over the NW-estimator.

This is a general principle which we would find reflected in equations for higher-order local polynomials (not displayed here): “Odd” polynomials are generally to be preferred over “even” ones; and for the estimation of jth derivatives, odd values of $p - j$ are preferred over even values of $p - j$. See also Wand & Jones page 137, Fan & Gijbels page 62.

4.2.6 Bandwidth selection

From the bias and variance approximations of $\hat{m}_{LL}(x)$, i.e. in the case $p = 1$, we have

$$\begin{aligned}
M(h) \equiv MISE(\hat{m}_{LL}) &= \int \mathrm{Bias}^2(\hat{m}_{LL}(x))\,dx + \int \mathrm{Var}(\hat{m}_{LL}(x))\,dx\\
&= \frac{1}{4}h^4\mu_2^2\int m''(x)^2\,dx + \frac{1}{nh}\nu_0\int \sigma^2(x)/f(x)\,dx.
\end{aligned}$$

From this we can find an optimal bandwidth by differentiating as usual,

$$\frac{dM(h)}{dh} = h^3\mu_2^2\int m''(x)^2\,dx - \frac{1}{nh^2}\nu_0\int \sigma^2(x)/f(x)\,dx \overset{!}{=} 0$$

which we solve as follows

$$\begin{aligned}
h^3\mu_2^2\int m^{''}(x)^2\,dx &= \frac{1}{nh^2}\nu_0\int\frac{\sigma^2(x)}{f(x)}\,dx\\
h^5 &= \frac{\nu_0}{\mu_2^2}\frac{\int\sigma^2(x)/f(x)\,dx}{\int m^{''}(x)^2\,dx}n^{-1}\\
h_{opt} &= \left[\frac{\nu_0}{\mu_2^2}\frac{\int\sigma^2(x)/f(x)\,dx}{\int m^{''}(x)^2\,dx}\right]^{1/5}n^{-1/5}
\end{aligned}$$

We firstly note this is again of order $n^{-1/5}$ as for KDE.

An obvious issue with this expression is that it involves the unknown quantities $\sigma^2(x)$, $f(x)$ and $m''(x)$. The idea is use pilot-estimators of m to inform these quantities which are then plugged into h_{opt}. This approach is generally known as “plug-in” techniques.

A particularly simple plug-in technique is the “Rule-of-thumb” bandwidth selector by Fan & Gijbels (1996), for the local linear case ($p = 1$). Under this approach,

- estimate $\hat{m}(x)$ globally by a polynomial of degree 4, i.e.

$$\check{m}(x) = \hat{\alpha}_0 + \check{\alpha}_1 x + \check{\alpha}_2 x^2 + \check{\alpha}_3 x^3 + \check{\alpha}_4 x^4$$

- estimate a constant error standard deviation σ from the residuals of this fitted model,

$$\check{\sigma}^2 = \frac{1}{n-5}\sum_{i=1}^{n}(y_i - \check{m}(x_i))^2$$

- compute

$$\check{m}''(x) = 2\check{\alpha}_2 + 6\check{\alpha}_3 x + 12\check{\alpha}_4 x^2$$

Then the “rule of thumb” bandwidth is

$$h_{ROT} = \left[\frac{\nu_0}{\mu_2^2}\frac{\check{\sigma}^2}{\sum_{i=1}^{n}\check{m}''(x)^2}\right]^{1/5} n^{-1/5}$$

This would be easly implementable with our abilities. However, we will not do this now, as an implementation is available with R function `thumbBW` of R package `locpol`.

More advanced plug-in techniques use full semi-or nonparametric estimates of $m''(x)$, $f(x)$ and $\sigma^2(x)$. An example for an implementation of such a technique is R function `dpill` in R package `KernSmooth`.

Example 4.3

We initially use R function `locpol` to reproduce our earlier, manually implemented, estimates. Note that the default setting of `locpol` is a polynomial degree `deg=1`, that is local linear, and that the default kernel is Epanechnikov, which we change now to Gaussian. Let’s begin with the `fossil` data.

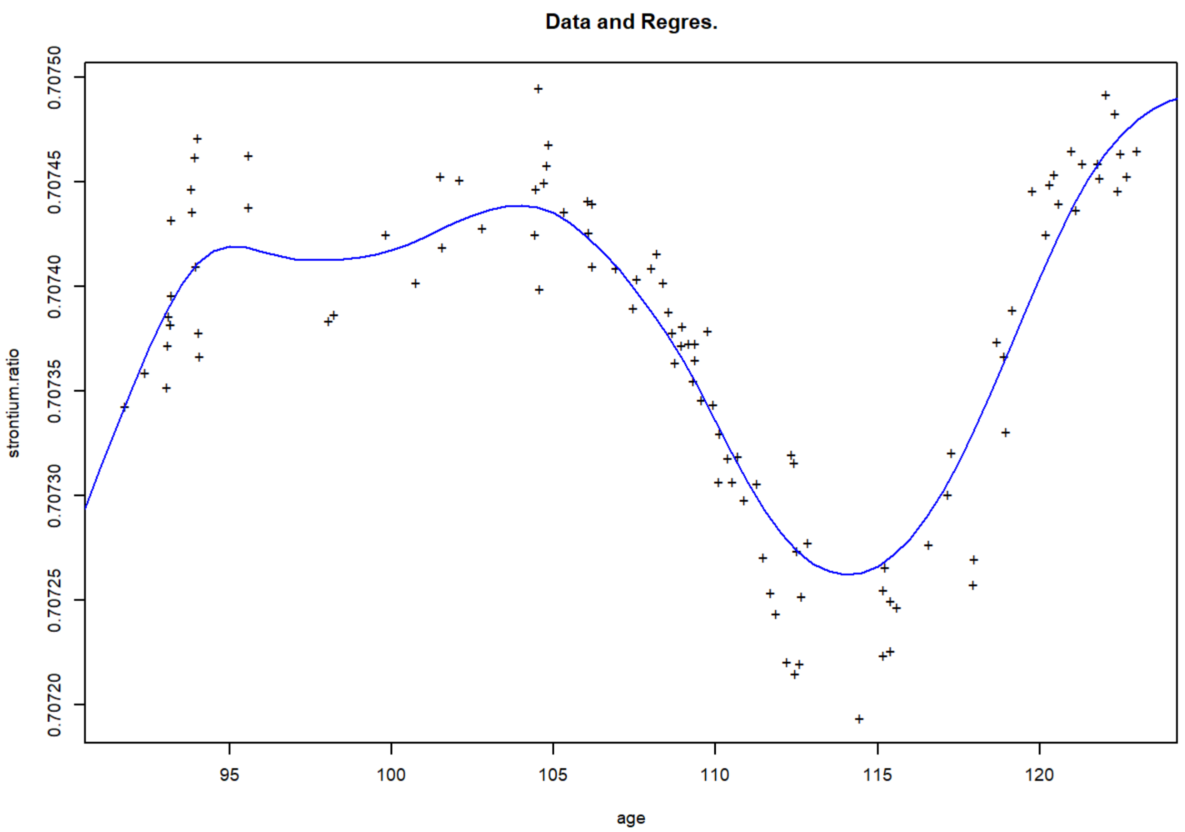
```
require(locpol)
```

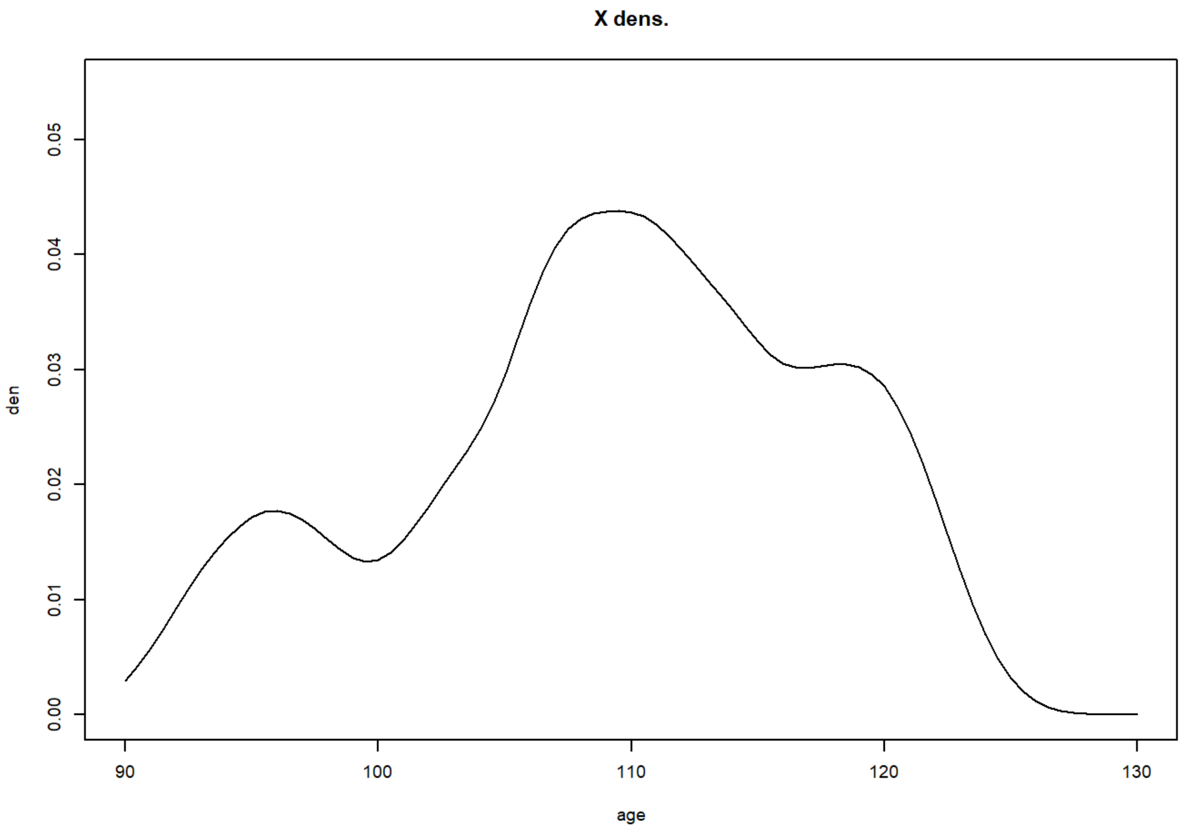
```
## Loading required package: locpol
```

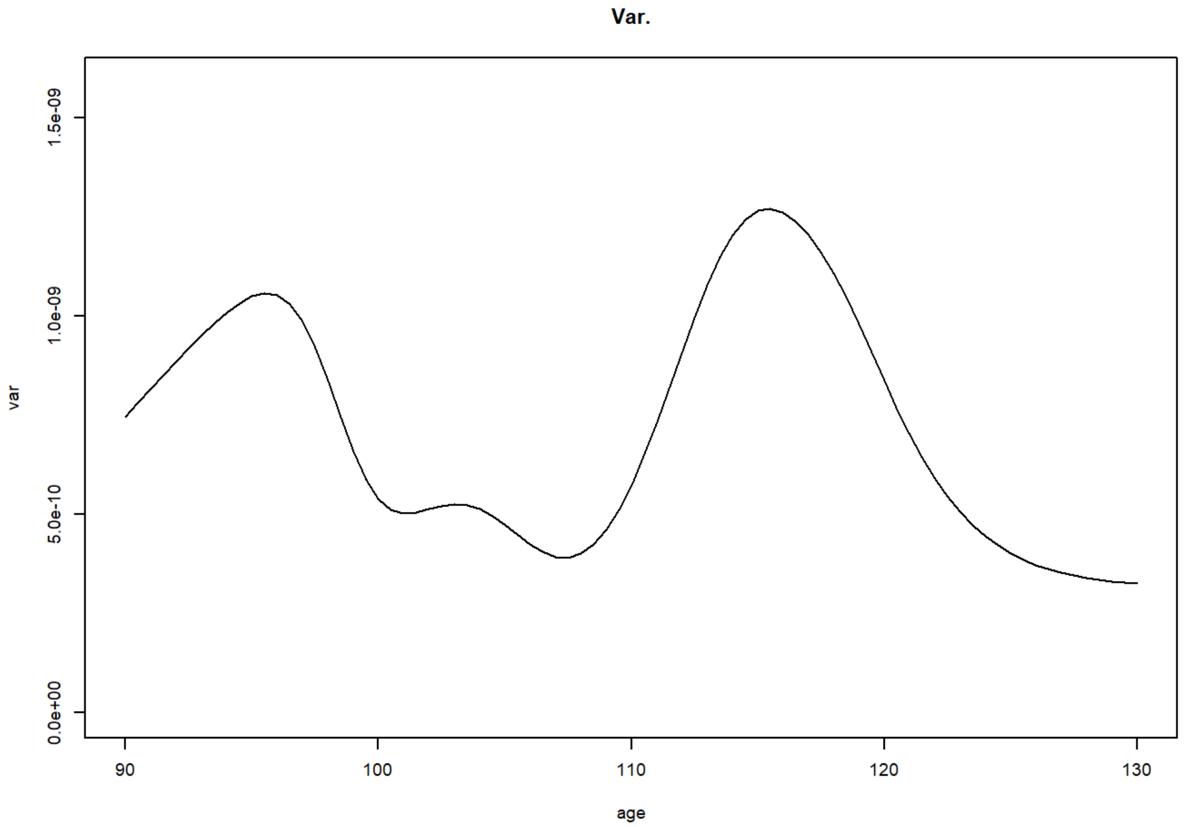
```
fit2a<- locpol(strontium.ratio~age, bw=2, deg=1,  kernel=gaussK, xeval=age.grid,  data=fossil)
```

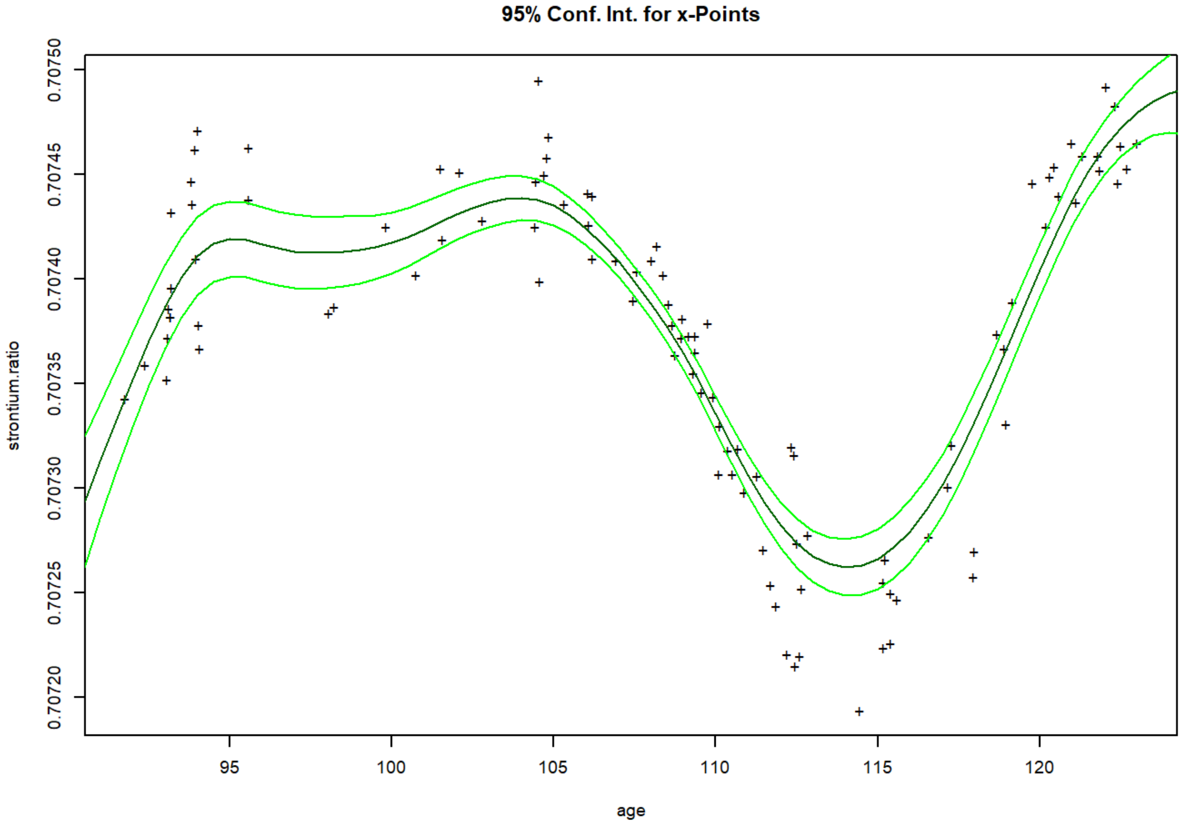
This gives a few standard outputs which we do understand, albeit without having seen all specific methodological details for this estimator:

```
plot(fit2a)
```


Data and Regres.
strontium.ratio
0.70720
0.70725
0.70730
0.70735
0.70740
0.70745
0.70750
95
100
105
110
115
120
age


X dens.
den
0.00
0.01
0.02
0.03
0.04
0.05
90
100
110
120
130
age





Let’s inspect the fitted object more closely

```
names(fit2a)
```

```
##  [1] "bw"         "KName"      "kernel"     "deg"        "xeval"      "mf"
##  [7] "Y"          "X"          "weig"       "lpFit"      "CIwidth"    "residuals"
```

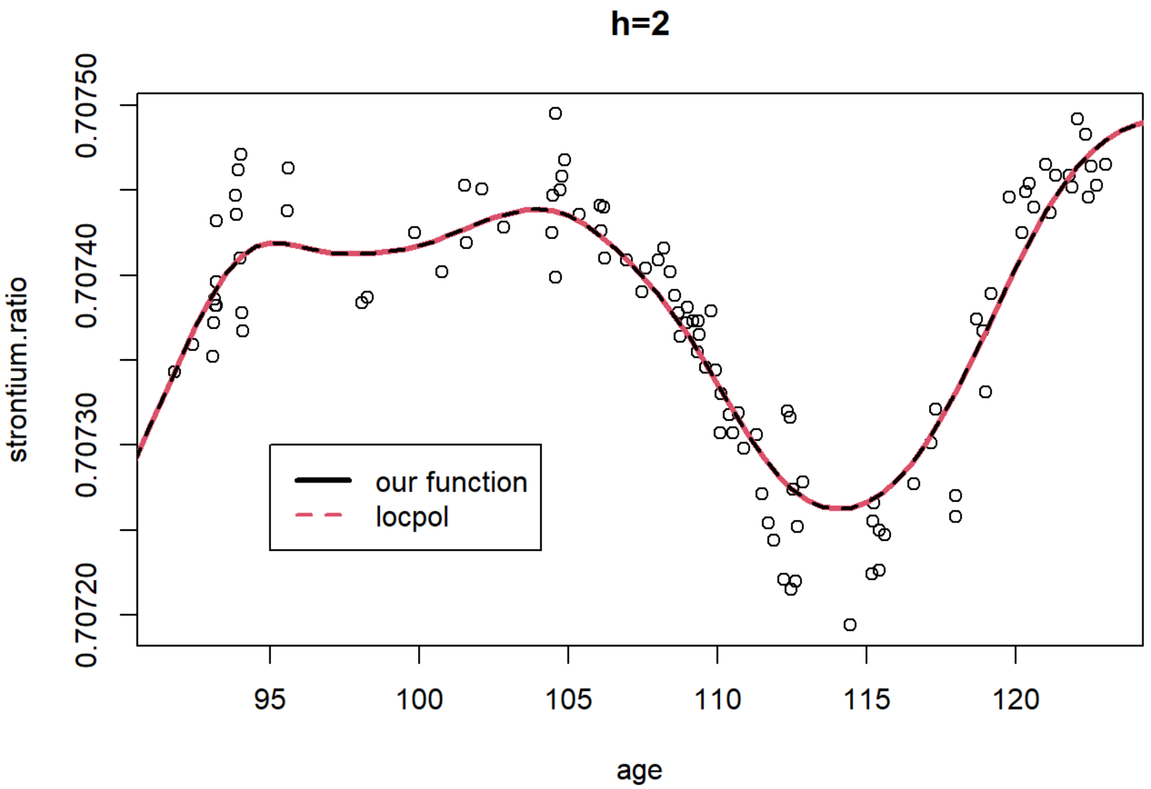
```
fit2a$bw
```

```
## [1] 2
```

Does the result correspond to our manual implementation?

```
age.grid<- seq(90, 130, by=0.5)
fit2<- my.ll.smoother(fossil$age, fossil$strontium.ratio, age.grid, h=2)
plot(fossil, main="h=2")
lines(fit2, col=2, lwd=3)
lines(fit2a$lpFit[1:2], lty=2, lwd=2)

legend(95,0.70730, c("our function", "locpol" ), lwd=c(3,2), lty=c(1,2), col=c(1,2))
```



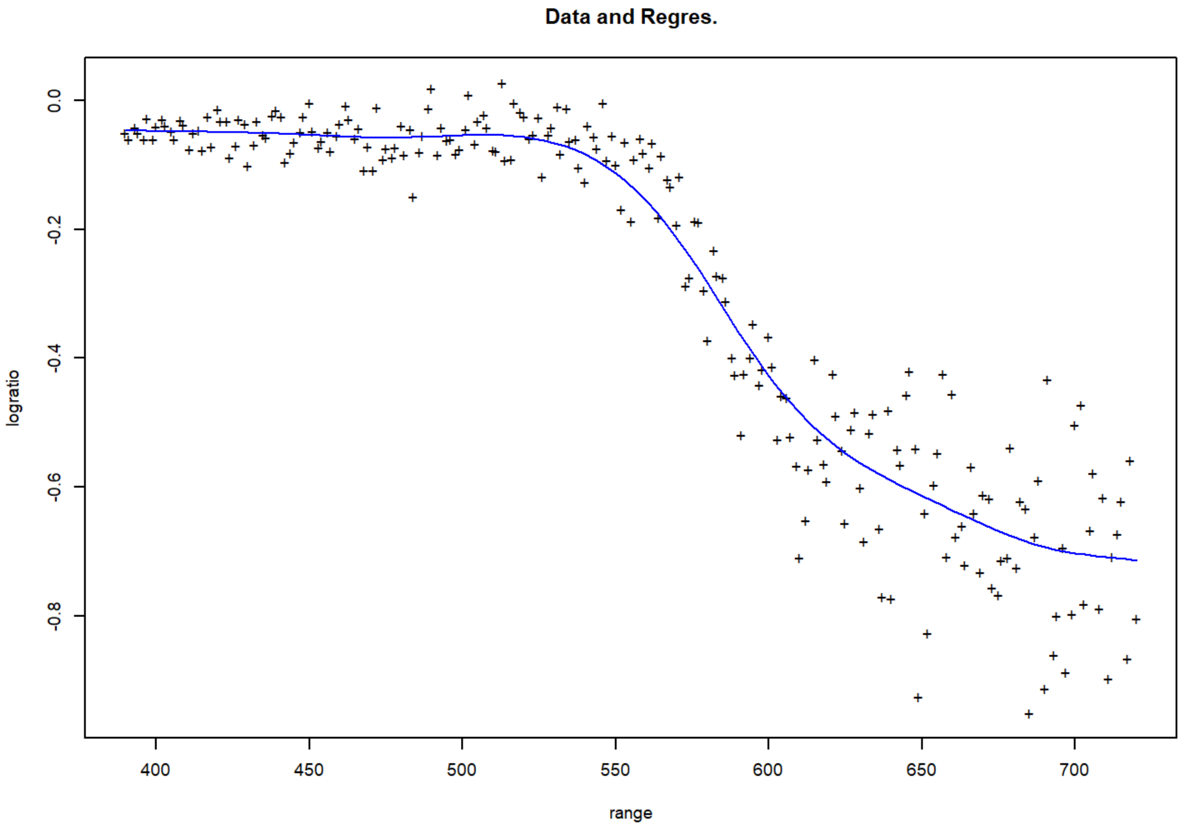
Yes, there is perfect agreement between the two implementations.

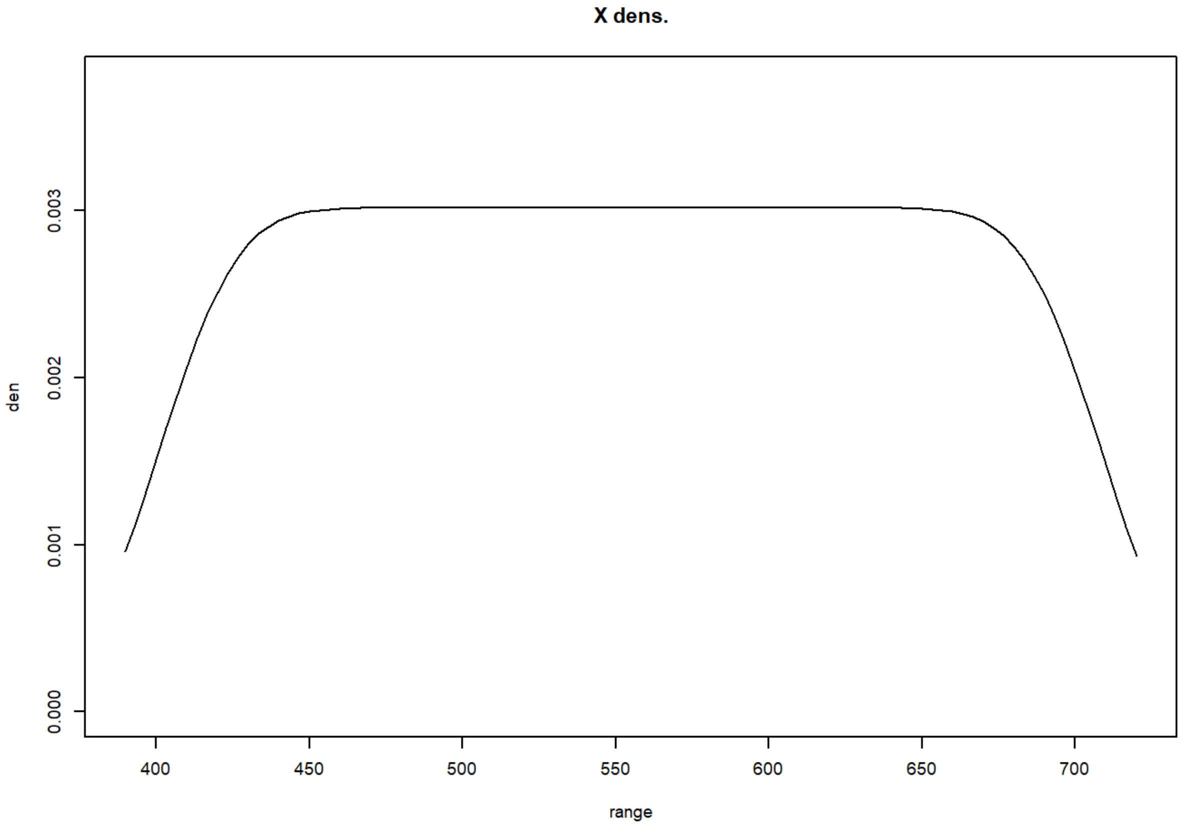
We can also do this for the `lidar` data:

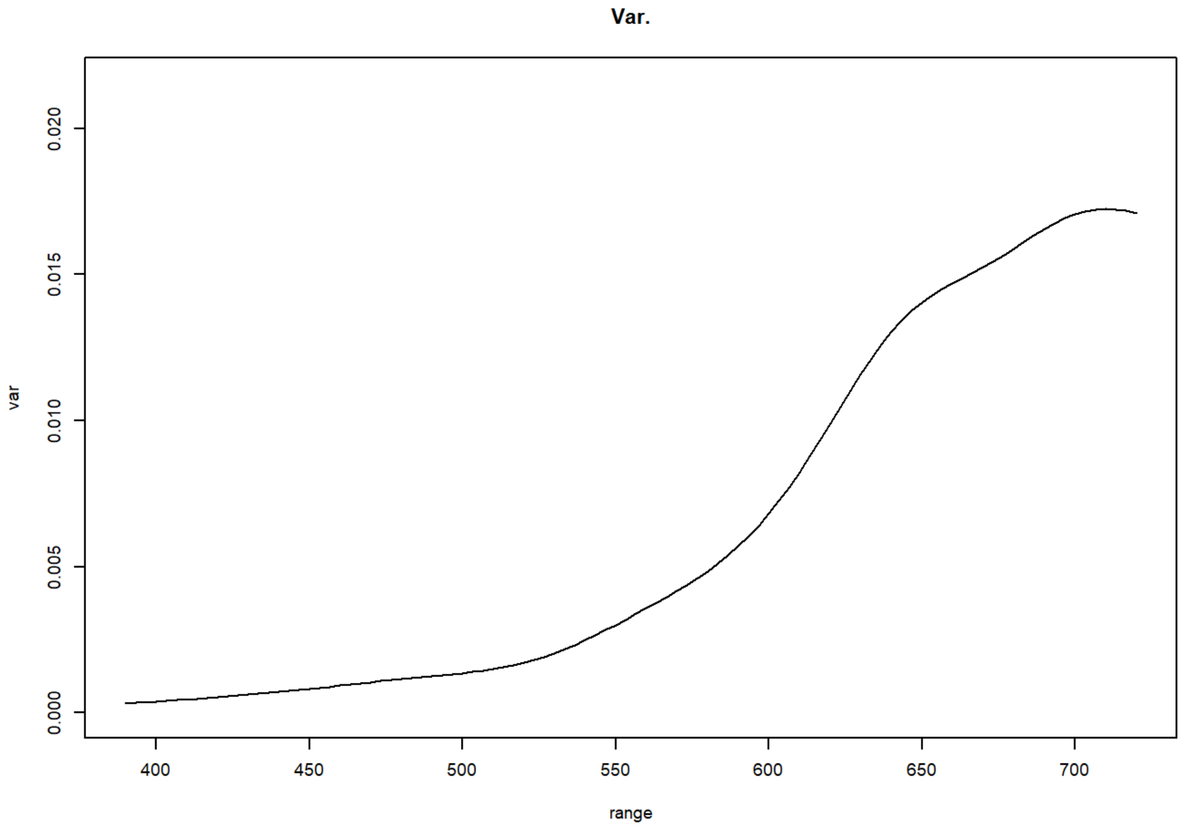
```
fitl2a<- locpol(logratio~range, bw=20, kernel=gaussK, data=lidar)
```

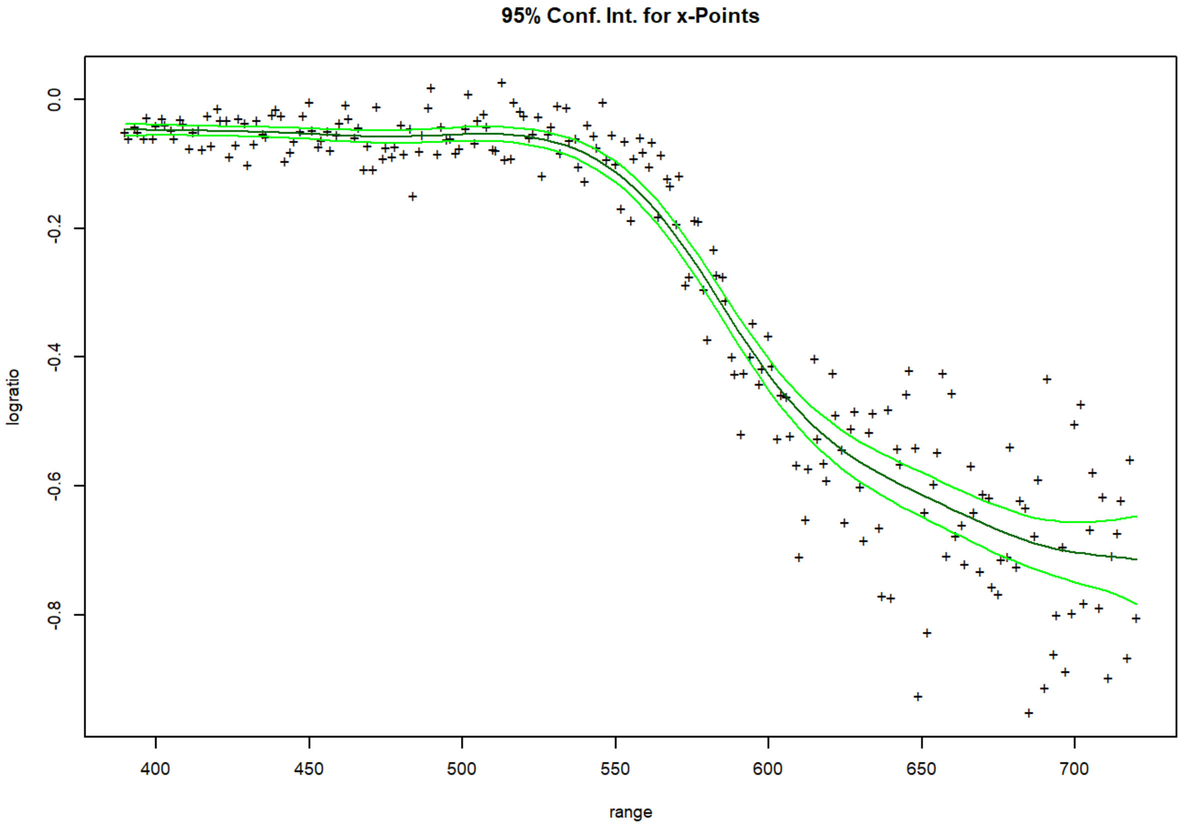
with associated plots

```
plot(fitl2a)
```


Data and Regres.
logratio
0.0
-0.2
-0.4
-0.6
-0.8
400
450
500
550
600
650
700
range


X dens.
den
0.000
0.001
0.002
0.003
400
450
500
550
600
650
700
range





Now let’s look into bandwidth selection. For the “rule-of-thumb”, we get:

```
thumbBw(fossil$age, fossil$strontium.ratio, kernel=gaussK, deg=1)
```

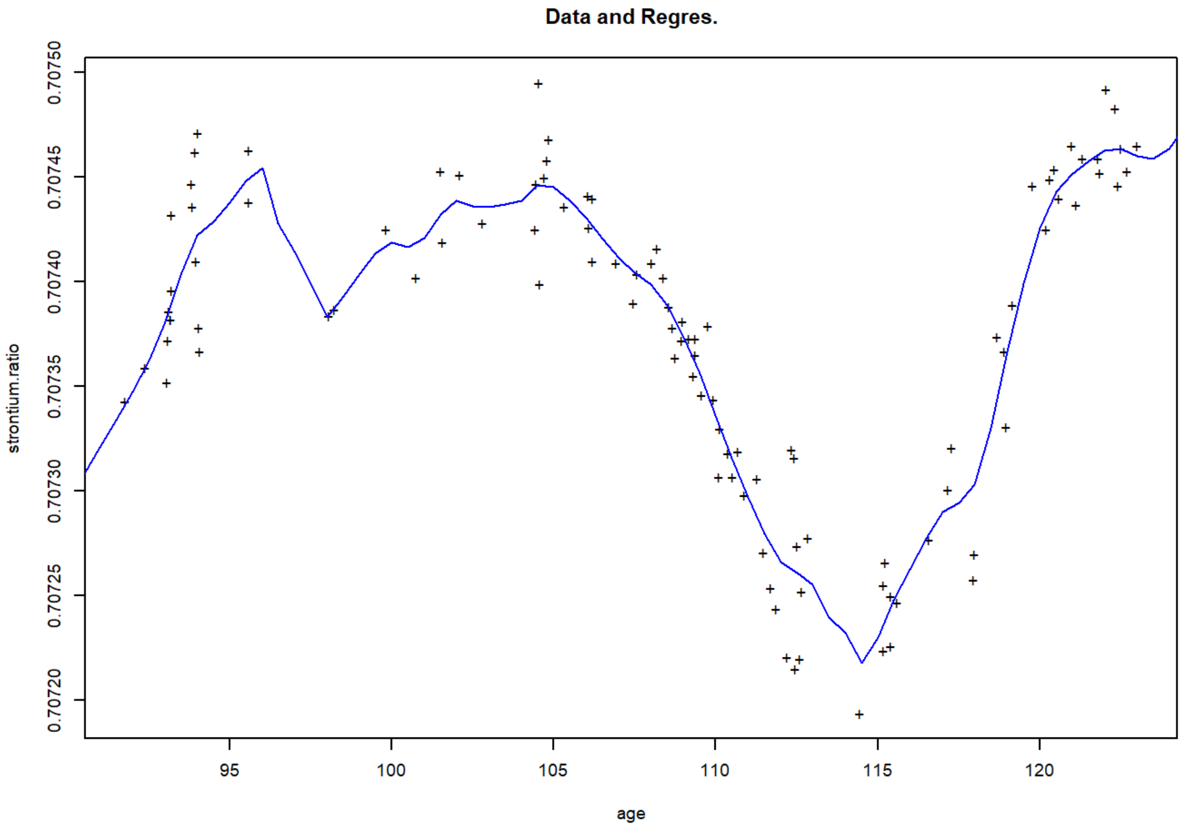
```
## [1] 0.6231473
```

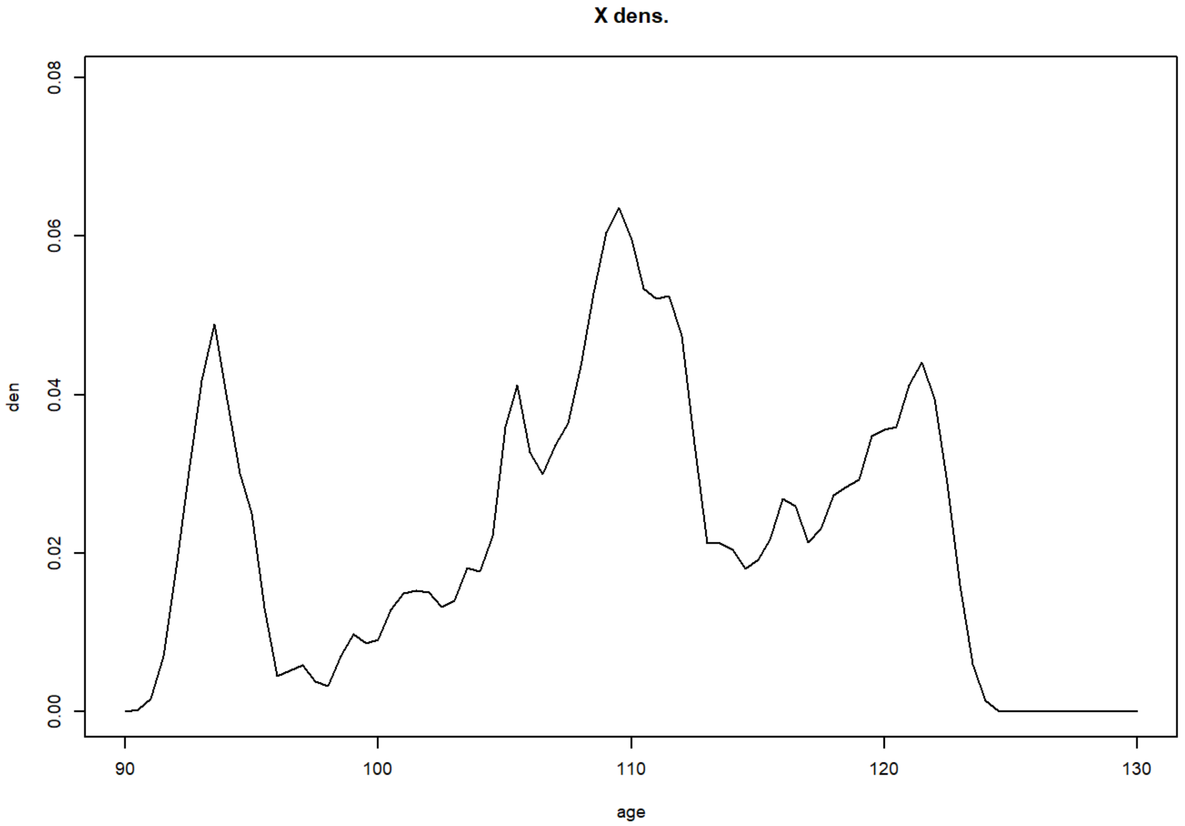
```
thumbBw(lidar$range, lidar$logratio, kernel=gaussK, deg=1)
```

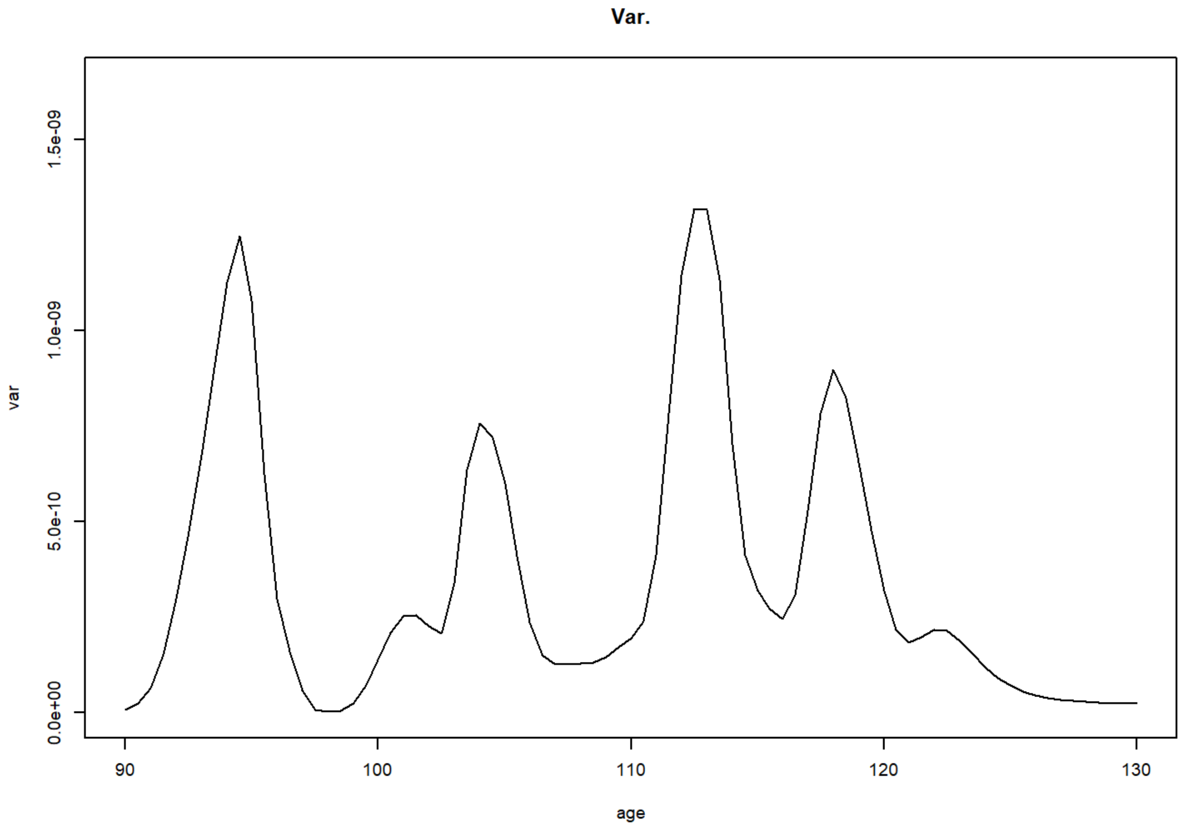
```
## [1] 4.178616
```

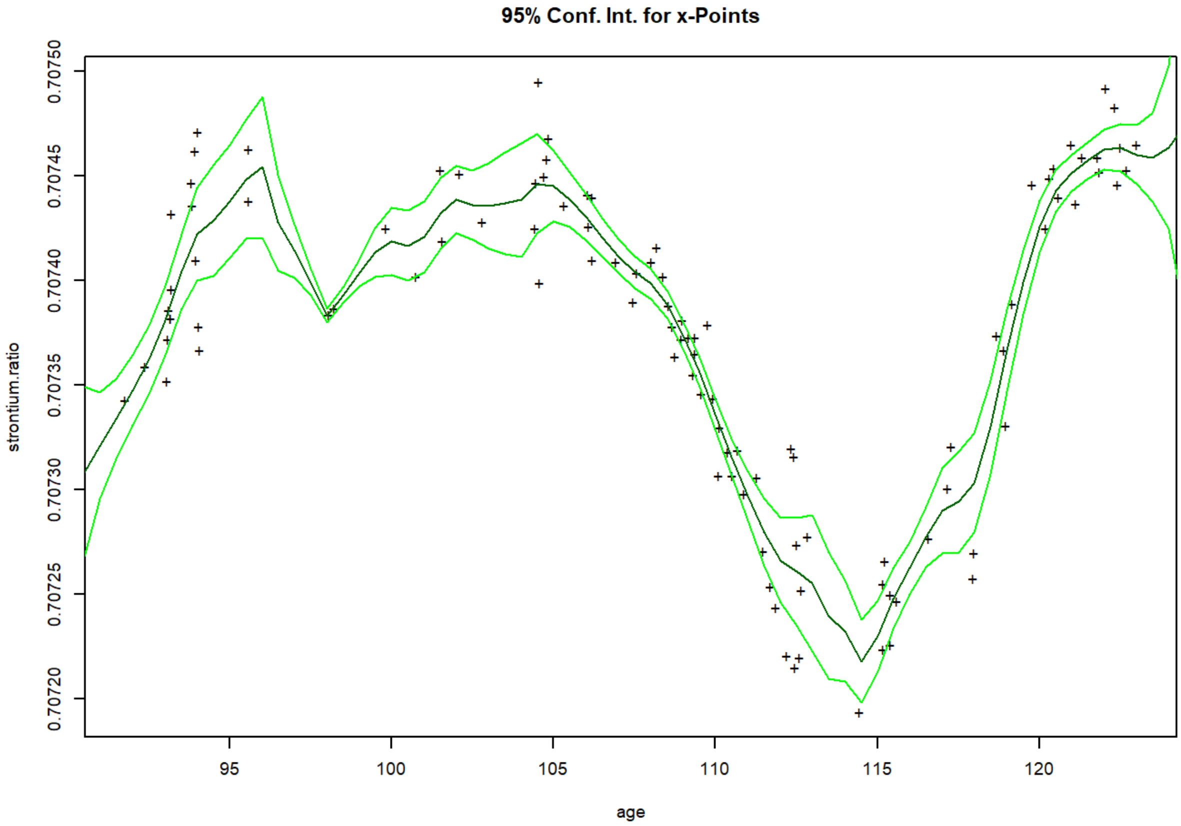
with resulting estimates:

```
fit2b<- locpol(strontium.ratio~age, bw=0.6232473, kernel=gaussK, xeval=age.grid,  data=fossil)
plot(fit2b)
```

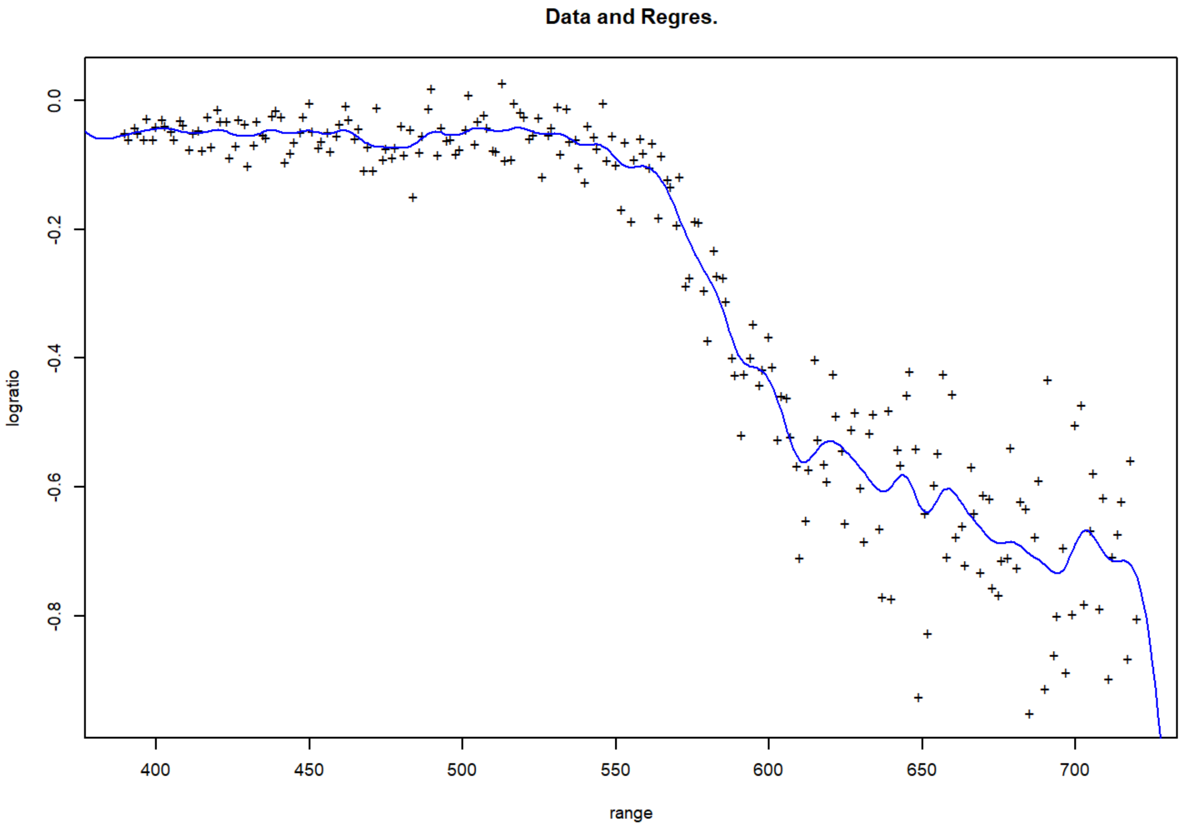

Data and Regres.
strontium.ratio
0.70720
0.70725
0.70730
0.70735
0.70740
0.70745
0.70750
95
100
105
110
115
120
age

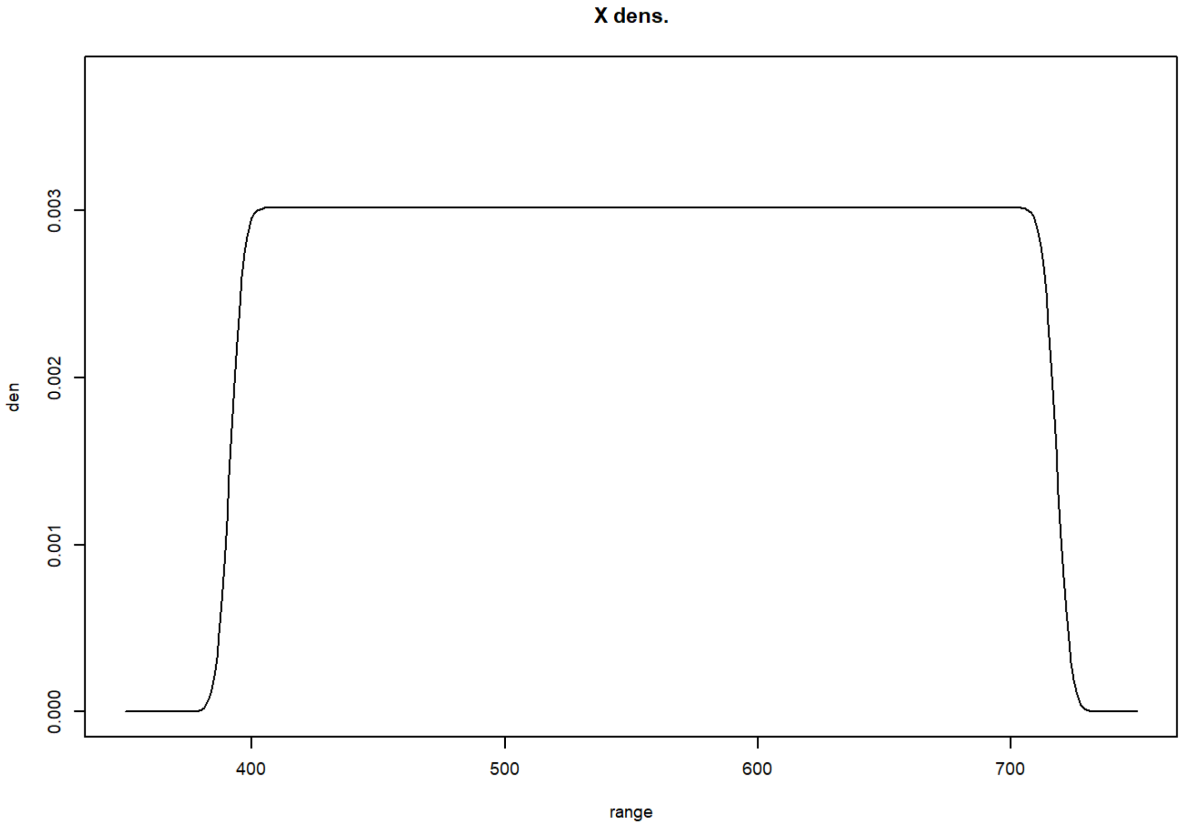

X dens.
den
0.00
0.02
0.04
0.06
0.08
90
100
110
120
130
age

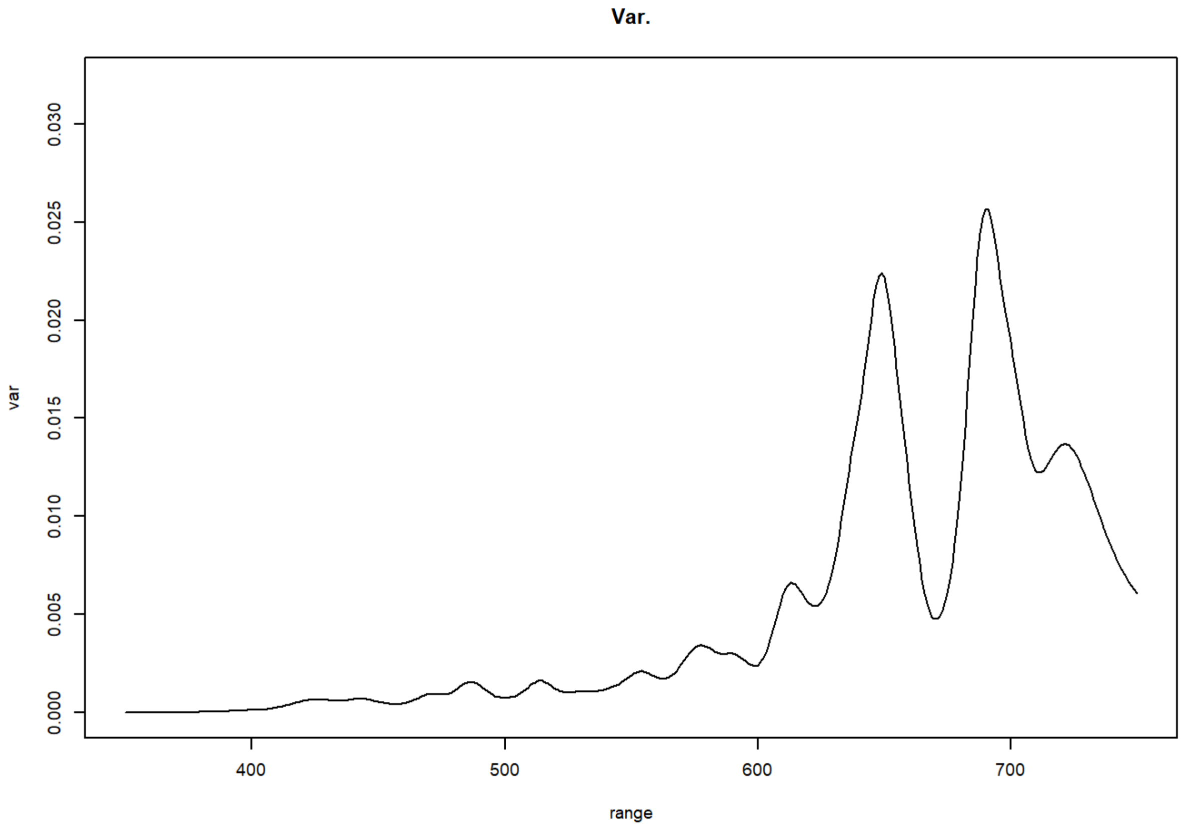


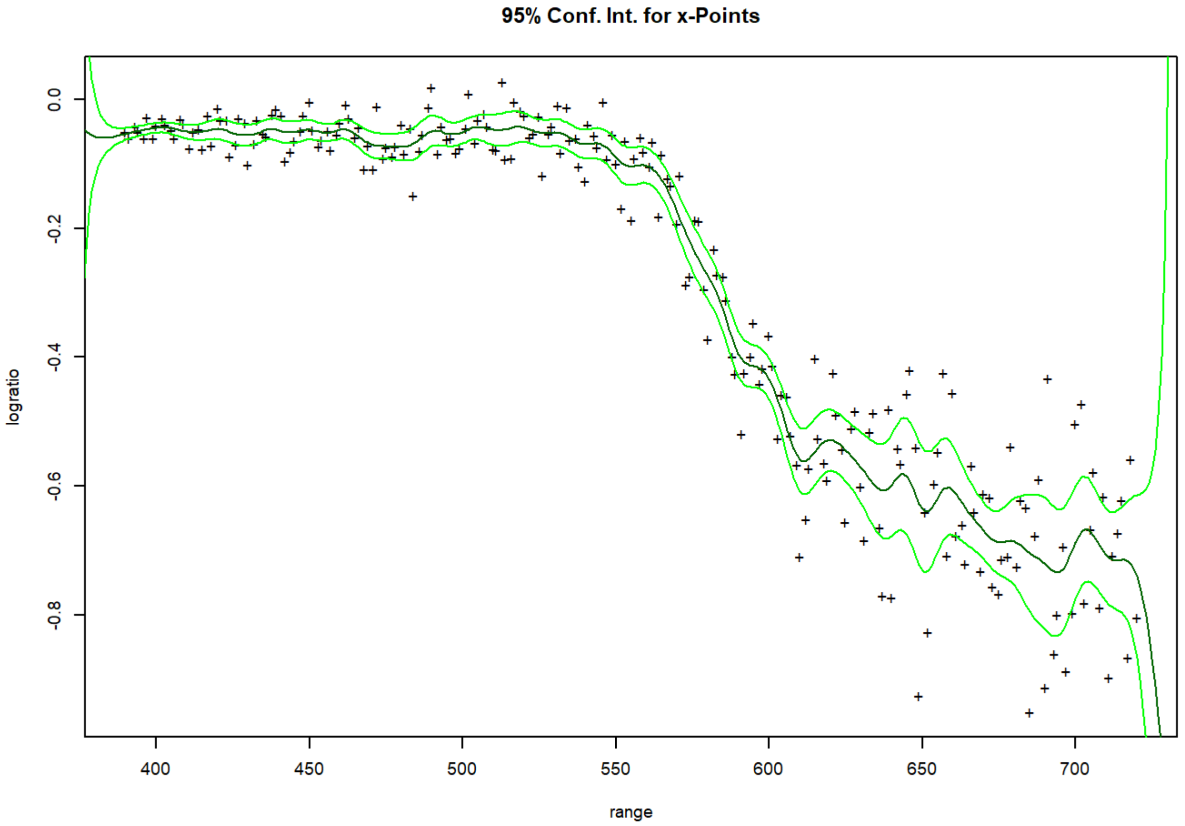


```
fit2lb<- locpol(logratio~range, bw=4.1786, kernel=gaussK, xeval=range.grid,  data=lidar)
plot(fit2lb)
```


Data and Regres.
logratio
0.0
-0.2
-0.4
-0.6
-0.8
range
400
450
500
550
600
650
700


X dens.
den
0.000
0.001
0.002
0.003
range
400
500
600
700





In both cases this seems to have settled on bandwidths which are rather too small. I would suspect that an underestimation of $\hat{\sigma}$ has played a role in both. Perhaps a full plug-in technique performs better? We find function `dpill` in R package `KernSmooth` (“Least squares quartic fits over blocks of data are used to obtain an initial estimate.”):

```
require(KernSmooth)
```

```
## Loading required package: KernSmooth
```

```
## KernSmooth 2.23 loaded
## Copyright M. P. Wand 1997-2009
```

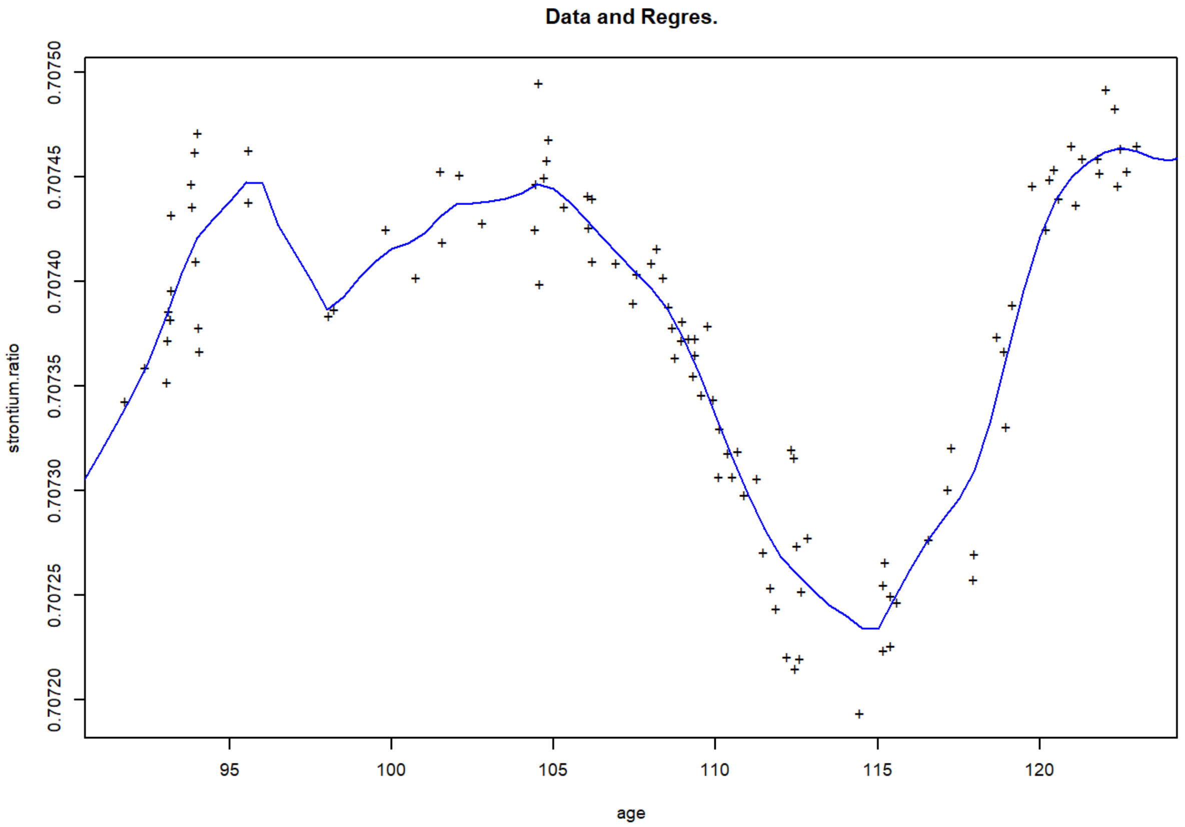
```
dpill(fossil$age, fossil$strontium.ratio)
```

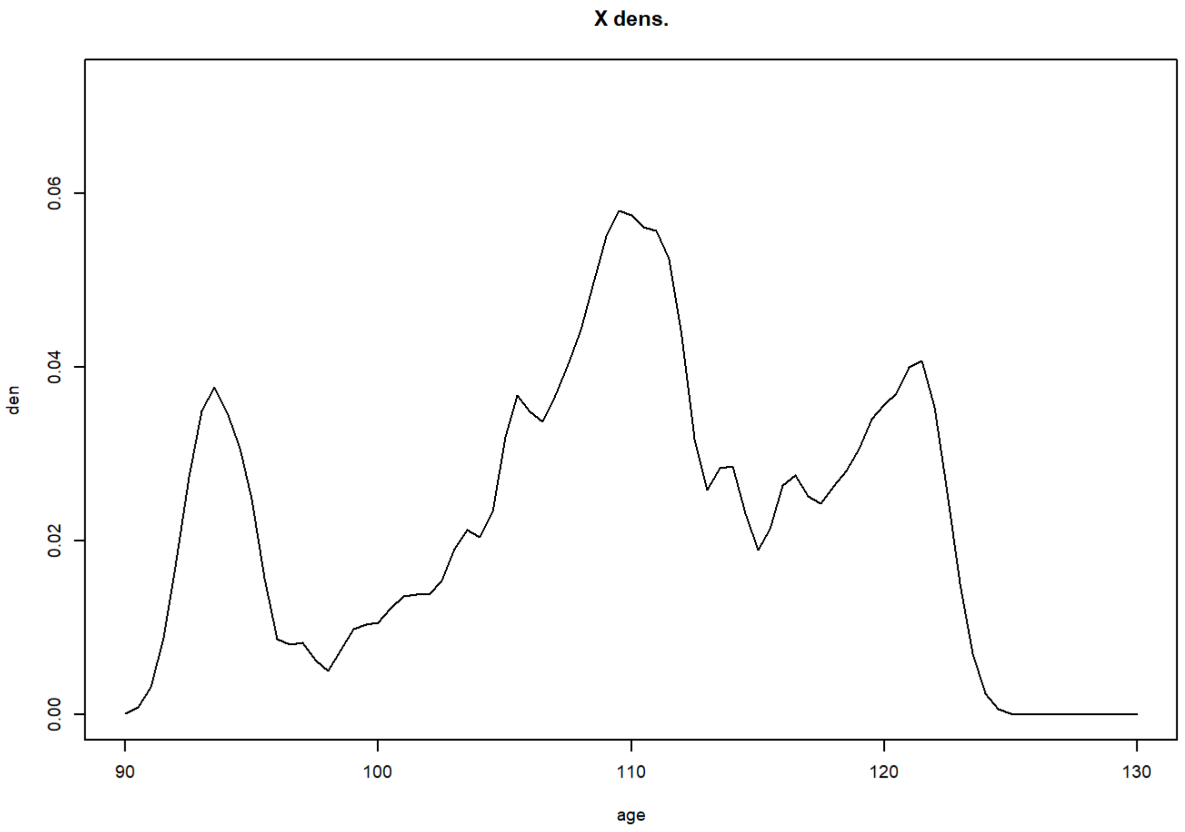
```
## [1] 0.7875148
```

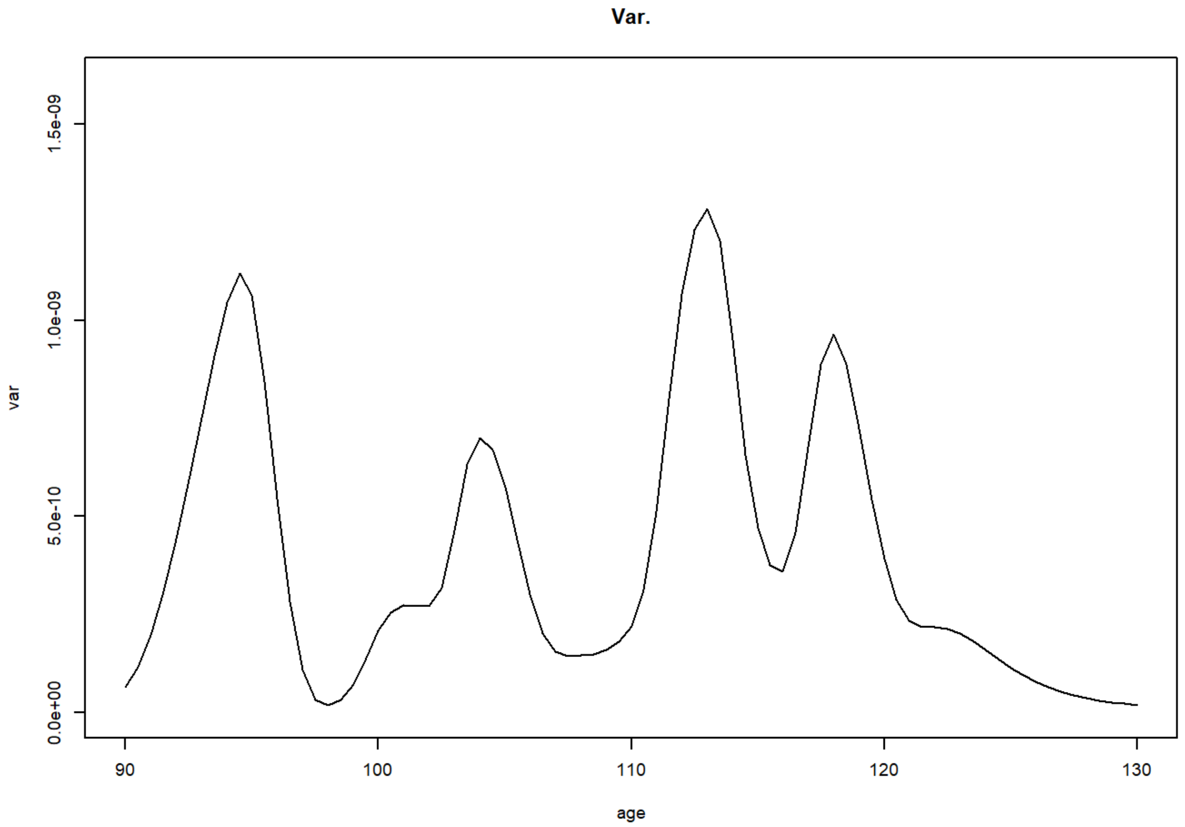
```
dpill(lidar$range, lidar$logratio)
```

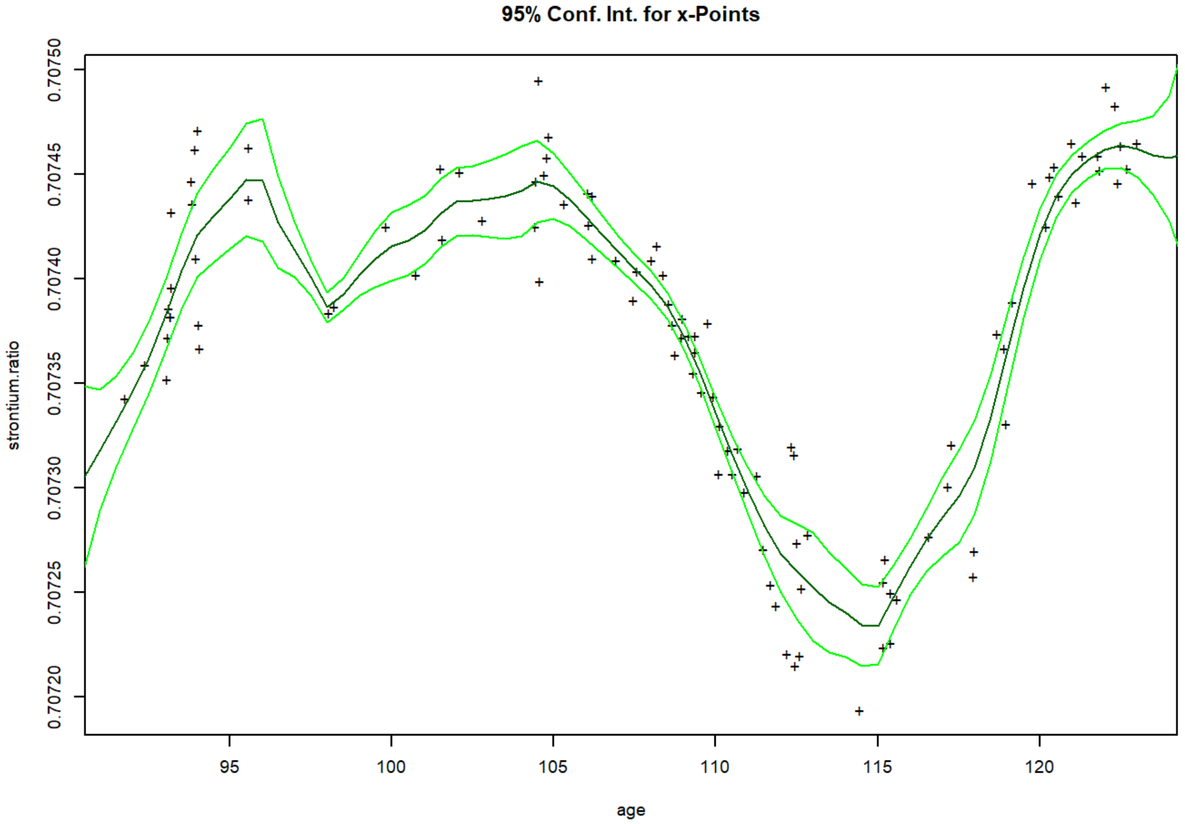
```
## [1] 16.20422
```

```
fit2b<- locpol(strontium.ratio~age, bw=0.7875, kernel=gaussK, xeval=age.grid,  data=fossil)
plot(fit2b)
```

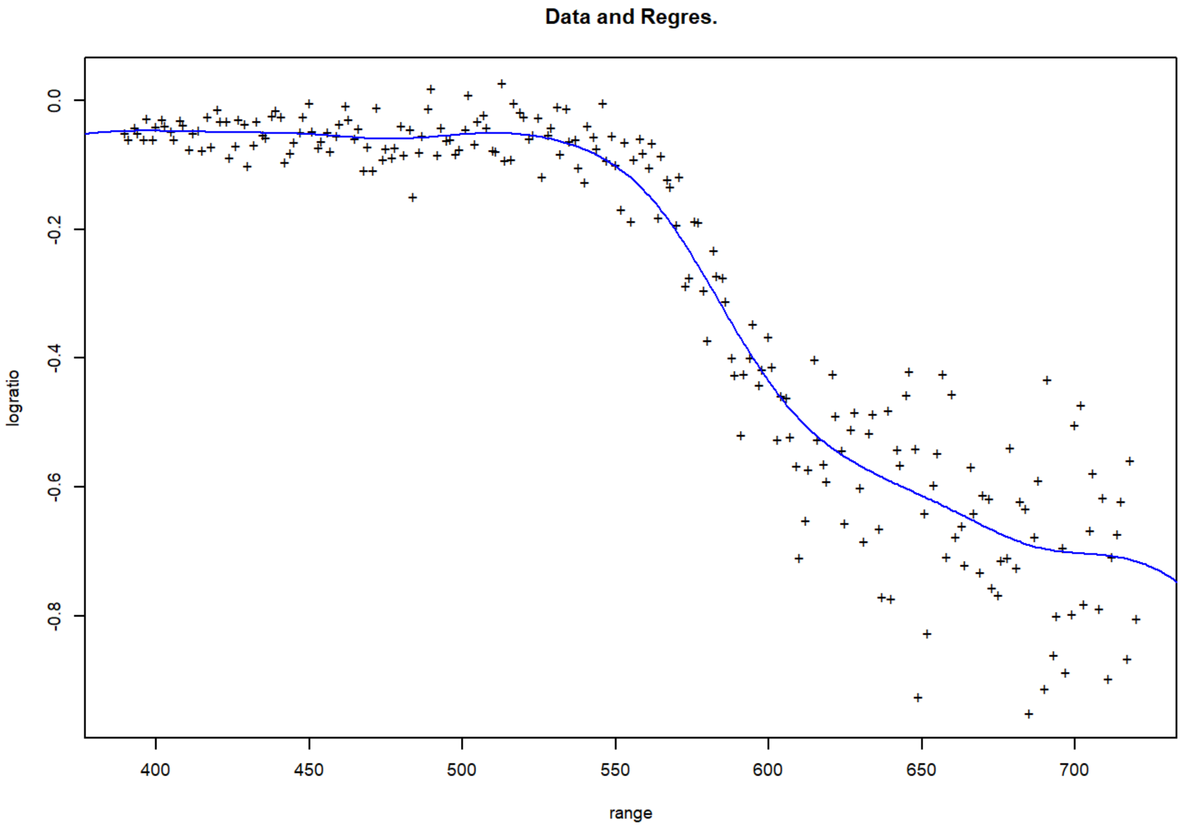

Data and Regres.
strontium.ratio
0.70720
0.70725
0.70730
0.70735
0.70740
0.70745
0.70750
95
100
105
110
115
120
age

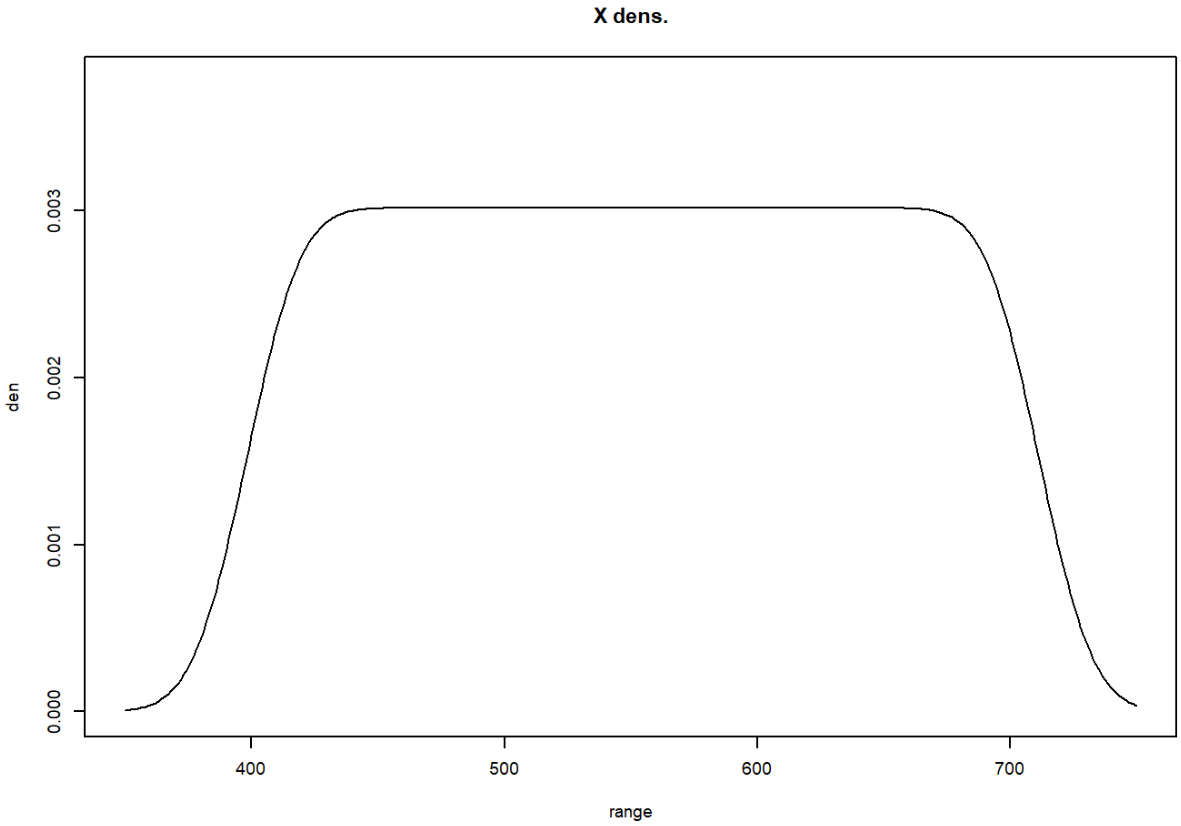

X dens.
den
0.00
0.02
0.04
0.06
90
100
110
120
130
age

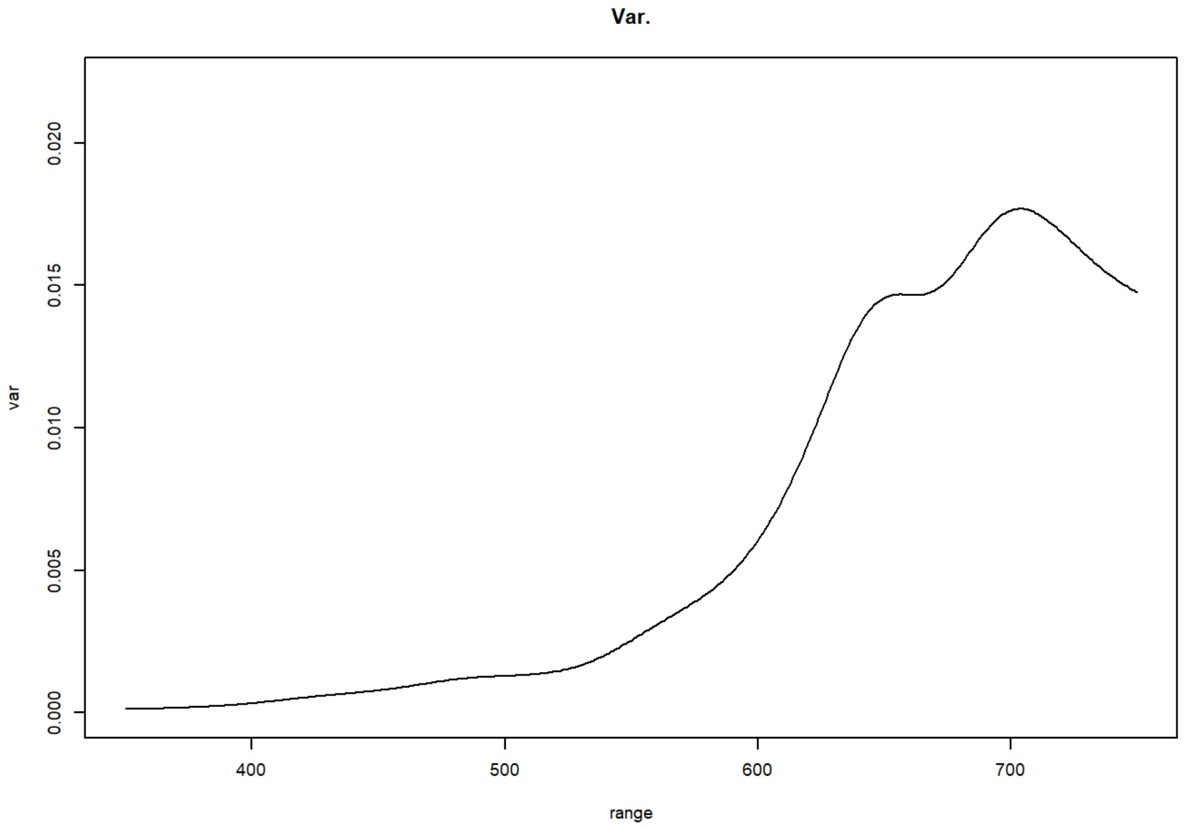


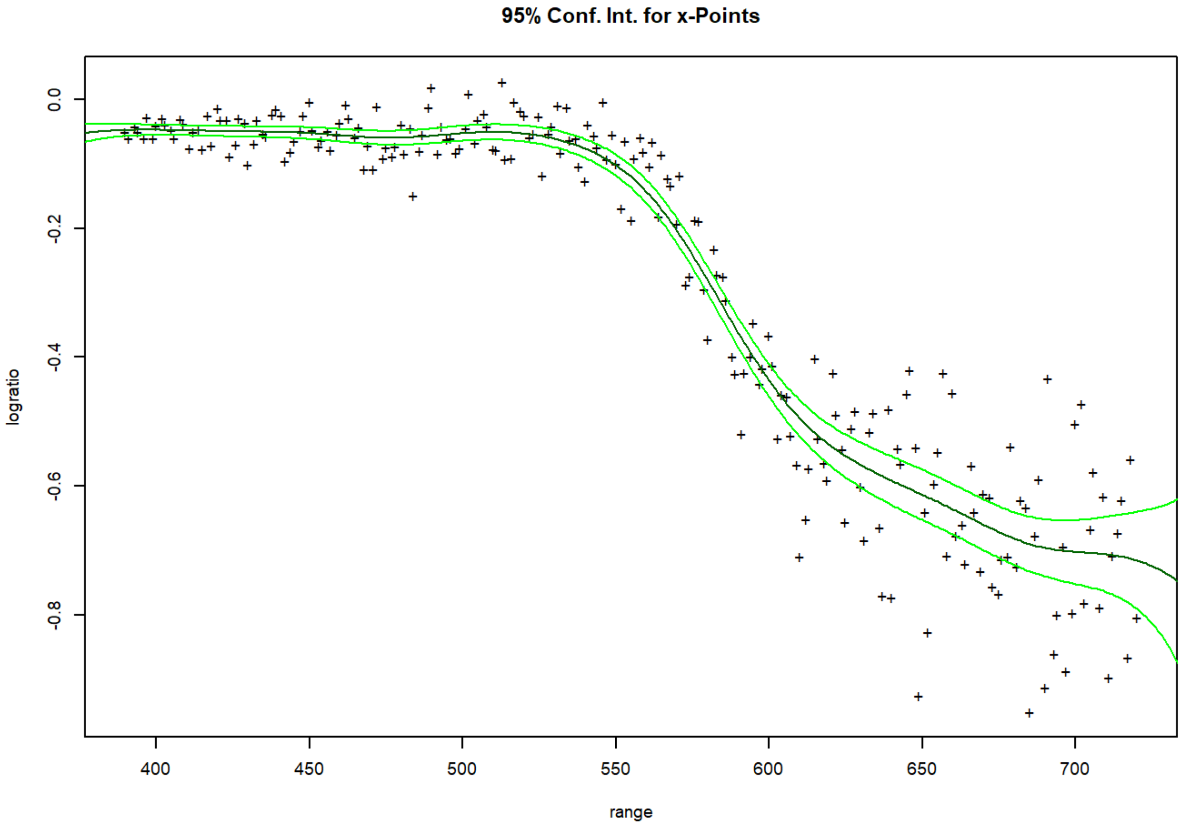


```
fit2lb<- locpol(logratio~range, bw=16.204, kernel=gaussK, xeval=range.grid,  data=lidar)
plot(fit2lb)
```


Data and Regres.
logratio
0.0
-0.2
-0.4
-0.6
-0.8
400
450
500
550
600
650
700
range


X dens.
den
0.000
0.001
0.002
0.003
400
500
600
700
range





Yes this is now a bit better. Note that the difference between the optimal bandwidths of the two approaches is larger for the `lidar` data, which are heteroscedastic, and hence may more likely deliver a poorly performing ROT bandwidth.

4.3 Multivariate local regression

We assume now that there are $d > 1$ predictor variables, yielding data of type

$$(x_{i1}, \ldots, x_{id}, y_i), \quad i = 1, \ldots, n$$

where x_{ij} is the ith observation from the jth predictor variable, for $j = 1, \ldots, d$.

We are again interested in establishing how the response relates to the d predictors. A fully nonparametric location-scale model in this context can be written as

$$y_i = m(x_{i1}, \ldots, x_{id}) + \sigma(x_{i1}, \ldots, x_{id})\epsilon_i$$

where

$m : \mathbb{R}^d \longmapsto \mathbb{R}$ is a sufficiently smooth function (usually twice differentiable in all directions), $\sigma : \mathbb{R}^d \longmapsto \mathbb{R}$ is the scale function (that we will not pay much attention to), and ϵ_i are independent errors with $E(\epsilon_i) = 0$ and $\mathrm{Var}(\epsilon_i) = 1$ (which are also independent of the predictors).

The task, as before, is to estimate the function m.

4.3.1 Multivariate locally constant fitting

A locally constant fit for multivariate data can be done through an obvious extension of the Nadaraya-Watson estimator. For $x \in \mathbb{R}^d$ and

$$x_i = \begin{pmatrix} x_{i1} \\ \vdots \\ x_{id} \end{pmatrix} \in \mathbb{R}^d, \quad i = 1, \ldots, n$$

we have

$$\hat{m}_{NW,H}(x) = \frac{\sum_{i=1} y_i K_H(x_i - x)}{\sum_{i=1} K_H(x_i - x)}$$

where

$$K_H(z) = \frac{1}{|H|^{1/2}} K\left(H^{-1/2} z\right), \quad z \in \mathbb{R}^d,$$

and $K : \mathbb{R}^d \longmapsto \mathbb{R}$ is a d-variate kernel function with properties as defined in Section 3.4.1.

For instance, we could have a multivariate Gaussian kernel

$$K(u) = \frac{1}{(2\pi)^{d/2}} e^{-\frac{1}{2}u^T u}$$

or, for any univariate kernel function, which for the purposes of this subsection we denote by $K_1(\cdot)$, a multivariate kernel is constructed as

$$K(u) = \prod_{j=1}^{d} K_1(u_j)$$

where $u = (u_1, \ldots, u_d)^T$.

So, the estimator $\hat{m}_{LL,H}(x)$ is again just a weighted mean of all observations, with those x_i located closer to the target point x gaining a larger kernel weight than those far away.

Example 4.4

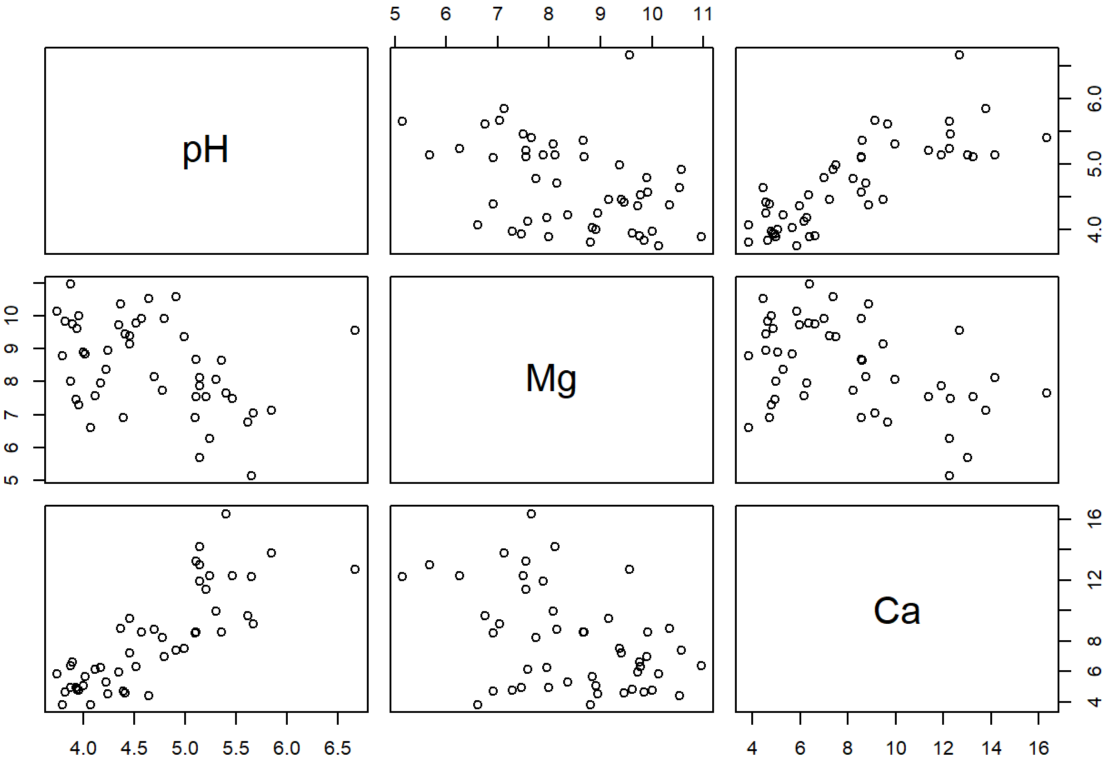
In this example we consider bivariate local regression for the previously considered `soils` data.

Specifically, we consider the varable `pH` from this data set as response, and the variables `Mg` and `Ca` as predictors. For convenience of handling we create a reduced data set `SpH` which only contains these three variables.

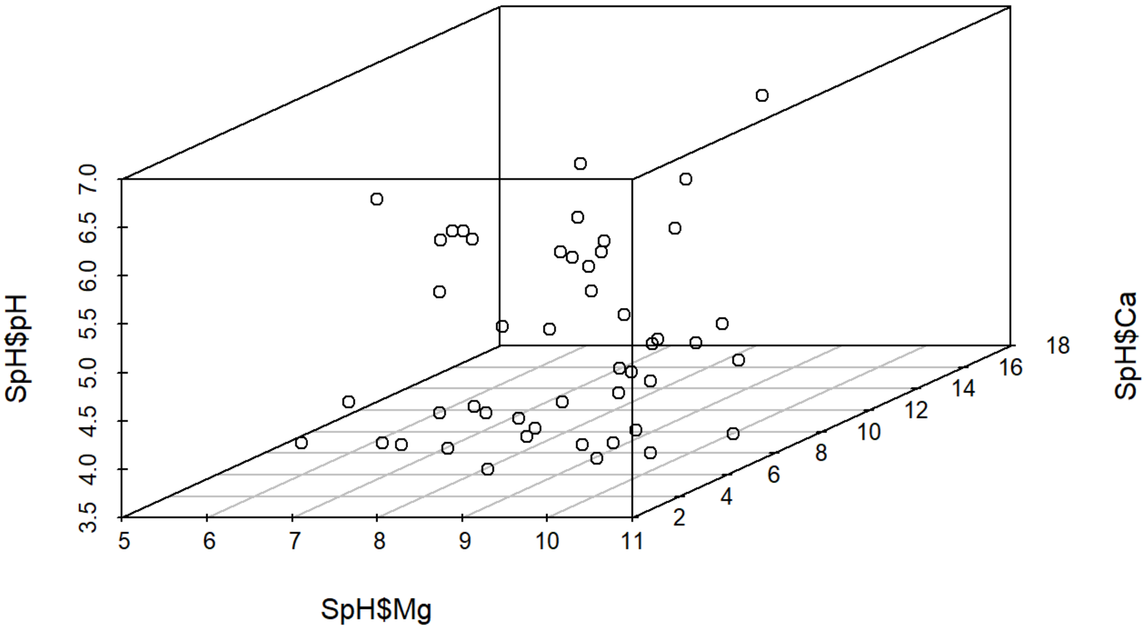
```
require(carData)
require(scatterplot3d)
```

```
## Loading required package: scatterplot3d
```

```
data(Soils)
SpH<- Soils[,c("pH", "Mg", "Ca")]
plot(SpH)
```



```
scatterplot3d(x=SpH$Mg, y=SpH$Ca, z=SpH$pH)
```



We would like to find a nonparametric regression function which predicts `pH` as a function of `Mg` and `Ca`. So, here we have $d = 2$,

$$H = \begin{pmatrix} h_1^2 & \\ & h_2^2 \end{pmatrix}$$

and let us write (with a slight abuse of notation) $x = (x_1, x_2)^T$. Then we can write (see similarly Ex 3.12)

$$K_H(x_i - x) = \frac{1}{|H|^{1/2}} K\left(H^{-\frac{1}{2}}(x_i - x)\right) = \frac{1}{h_1 h_2} K_1\left(\frac{x_{i1} - x_1}{h_1}\right)\left(\frac{x_{i2} - x_1}{h_2}\right)$$

where we also see that the multipplicative constant $\frac{1}{h_1 h_2}$ will cancel out between the numerator and denominator of $\hat{m}_{LL,H}(x)$. With this, we obtain an implementation of $\hat{m}_{LL,H}(x)$ as

```
my.biv.kernel.smoother<- function(x1dat,x2dat, ydat, x1grid, x2grid, h1,h2){
  n<-length(ydat)
  G1<-length(x1grid)
  G2<-length(x2grid)
  est<-matrix(0,G1, G2)
  for (j in 1:G1){
    for (l in 1:G2){
  est[j,l]<-
     sum(ydat*my.kernel((x1dat-x1grid[j])/h1)*my.kernel((x2dat-x2grid[l])/h2))/
    sum(my.kernel((x1dat-x1grid[j])/h1)*my.kernel((x2dat-x2grid[l])/h2))
    }
  }
  return(c("x1grid"=list(x1grid), "x2grid"=list(x2grid), "yhat"=list(est)) )
}
```

where `my.kernel` stems from the early parts of this term and should be set to “Gaussian” for the purposes of this section. Then we set up our estimation grid

```
summary(soils$Mg)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   5.150   7.537   8.515   8.465   9.648  10.960
```

```
summary(soils$Ca)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   3.820   5.040   7.305   8.029   9.735  16.350
```

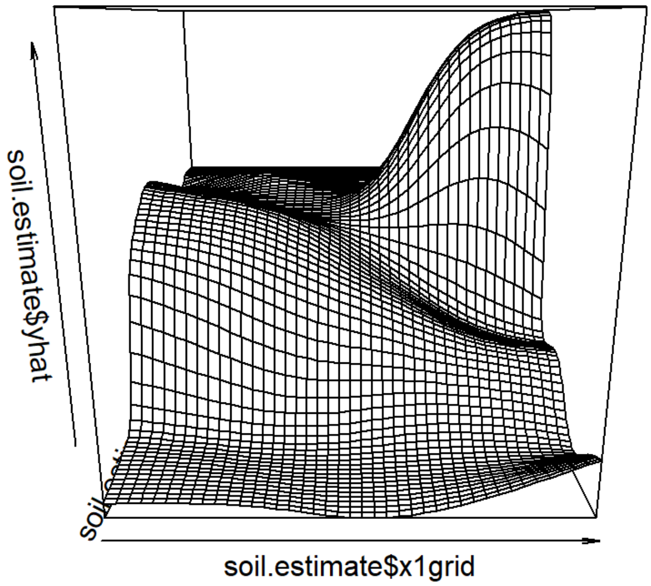
```
Mg.grid<- seq(4,12, by=0.2)
Ca.grid<- seq(3,18, by=0.2)
```

and proceed with the actual estimation (using $h_1 = h_2 = 1$)

```
soil.estimate<- my.biv.kernel.smoother(
  x1dat=SpH$Mg,
  x2dat=SpH$Ca,
  y=SpH$pH,
  x1grid=Mg.grid,
  x2grid=Ca.grid,
  h1=1,
  h2=1
  )
```

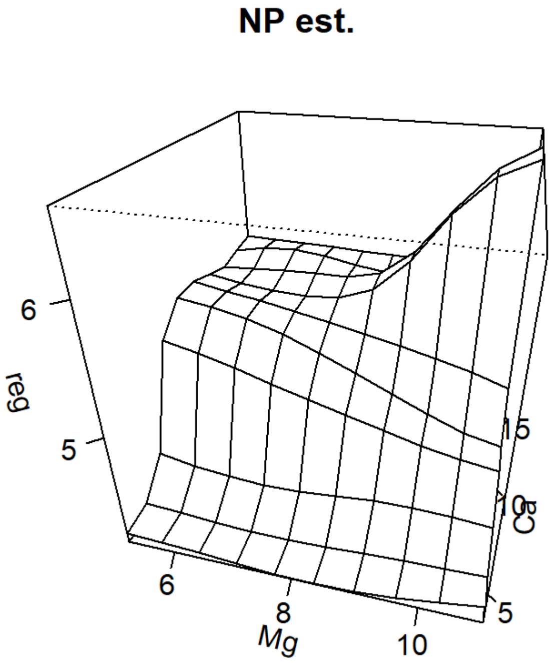
We can visualize the resulting estimator via

```
persp(soil.estimate$x1grid, soil.estimate$x2grid, soil.estimate$yhat )
```



There does exist a function for bivariate locally constant regression in R package `locpol` with which we may want to compare:

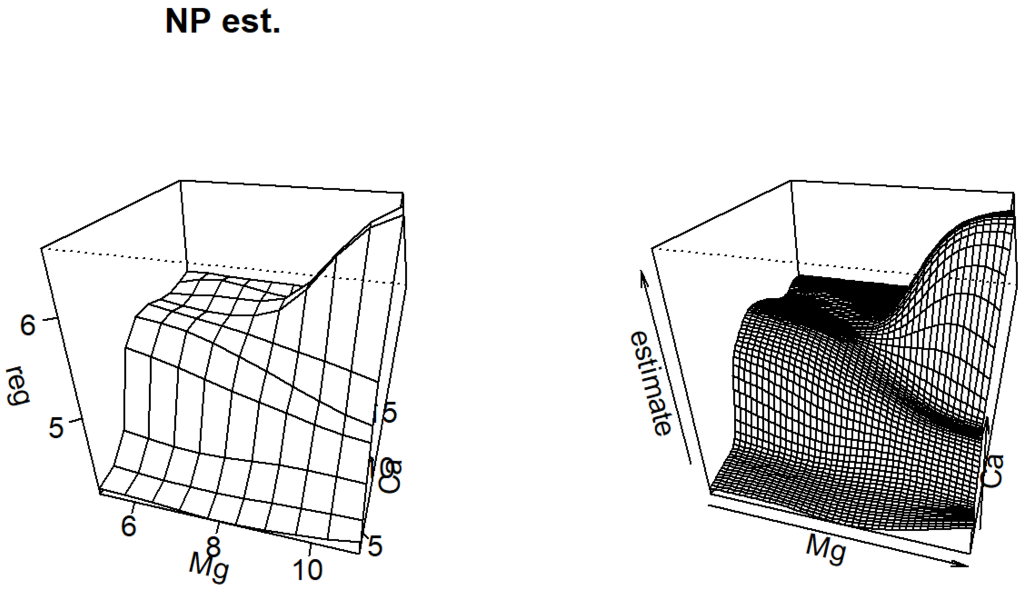
```
require(locpol)
#?bivReg
soil.estimate2<- bivReg(SpH[,c(2,3)],SpH[,1], K=gauK2d, H=diag(c(1^2, 1^2)) )
plot(soil.estimate2)
```


NP est.
6
reg
5
6
8
Mg
10
15
10
Ca
5

```
##           Mg        Ca      den
## 1    5.150000  3.820000 4.122082
## 2    5.795556  3.820000 4.112766
## 3    6.441111  3.820000 4.094727
## 4    7.086667  3.820000 4.069125
## 5    7.732222  3.820000 4.046931
## 6    8.377778  3.820000 4.044557
## 7    9.023333  3.820000 4.063343
## 8    9.668889  3.820000 4.095733
## 9   10.314444  3.820000 4.140457
## 10  10.960000  3.820000 4.197691
## 11   5.150000  5.212222 4.148143
## 12   5.795556  5.212222 4.126309
## 13   6.441111  5.212222 4.102873
## 14   7.086667  5.212222 4.084642
## 15   7.732222  5.212222 4.079640
## 16   8.377778  5.212222 4.090138
## 17   9.023333  5.212222 4.107139
## 18   9.668889  5.212222 4.120965
## 19  10.314444  5.212222 4.130884
## 20  10.960000  5.212222 4.139433
## 21   5.150000  6.604444 4.393768
## 22   5.795556  6.604444 4.339018
## 23   6.441111  6.604444 4.296765
## 24   7.086667  6.604444 4.276067
## 25   7.732222  6.604444 4.283068
## 26   8.377778  6.604444 4.310810
## 27   9.023333  6.604444 4.334763
## 28   9.668889  6.604444 4.338989
## 29  10.314444  6.604444 4.325946
## 30  10.960000  6.604444 4.300048
## 31   5.150000  7.996667 5.240411
## 32   5.795556  7.996667 5.178879
## 33   6.441111  7.996667 5.096935
## 34   7.086667  7.996667 5.004367
## 35   7.732222  7.996667 4.917477
## 36   8.377778  7.996667 4.838199
## 37   9.023333  7.996667 4.754613
## 38   9.668889  7.996667 4.671724
## 39  10.314444  7.996667 4.606007
## 40  10.960000  7.996667 4.560594
## 41   5.150000  9.388889 5.456161
## 42   5.795556  9.388889 5.418723
## 43   6.441111  9.388889 5.358361
## 44   7.086667  9.388889 5.266563
## 45   7.732222  9.388889 5.147610
## 46   8.377778  9.388889 5.016287
## 47   9.023333  9.388889 4.879557
## 48   9.668889  9.388889 4.743009
## 49  10.314444  9.388889 4.628261
## 50  10.960000  9.388889 4.552832
## 51   5.150000 10.781111 5.468445
## 52   5.795556 10.781111 5.416710
## 53   6.441111 10.781111 5.366896
## 54   7.086667 10.781111 5.314059
## 55   7.732222 10.781111 5.256568
## 56   8.377778 10.781111 5.194234
## 57   9.023333 10.781111 5.127815
## 58   9.668889 10.781111 5.060031
## 59  10.314444 10.781111 4.985329
## 60  10.960000 10.781111 4.891742
## 61   5.150000 12.173333 5.402849
## 62   5.795556 12.173333 5.354080
## 63   6.441111 12.173333 5.322063
## 64   7.086667 12.173333 5.310632
## 65   7.732222 12.173333 5.331288
## 66   8.377778 12.173333 5.437431
## 67   9.023333 12.173333 5.712021
## 68   9.668889 12.173333 6.104329
## 69  10.314444 12.173333 6.398217
## 70  10.960000 12.173333 6.532088
## 71   5.150000 13.565556 5.350027
## 72   5.795556 13.565556 5.360873
## 73   6.441111 13.565556 5.382646
## 74   7.086667 13.565556 5.385905
## 75   7.732222 13.565556 5.385959
## 76   8.377778 13.565556 5.451929
## 77   9.023333 13.565556 5.678112
## 78   9.668889 13.565556 6.050617
## 79  10.314444 13.565556 6.374975
## 80  10.960000 13.565556 6.550128
## 81   5.150000 14.957778 5.385161
## 82   5.795556 14.957778 5.448660
## 83   6.441111 14.957778 5.461351
## 84   7.086667 14.957778 5.424899
## 85   7.732222 14.957778 5.374140
## 86   8.377778 14.957778 5.341368
## 87   9.023333 14.957778 5.363720
## 88   9.668889 14.957778 5.496604
## 89  10.314444 14.957778 5.779328
## 90  10.960000 14.957778 6.134147
## 91   5.150000 16.350000 5.420230
## 92   5.795556 16.350000 5.418188
## 93   6.441111 16.350000 5.409614
## 94   7.086667 16.350000 5.399926
## 95   7.732222 16.350000 5.390322
## 96   8.377778 16.350000 5.380893
## 97   9.023333 16.350000 5.372119
## 98   9.668889 16.350000 5.366703
## 99  10.314444 16.350000 5.374227
## 100 10.960000 16.350000 5.422460
```

This does look similar but not necessarily the same. Let us try to adjust the angles from which we look at this:

```
par(mfrow=c(1,2))
plot(soil.estimate2)
persp(soil.estimate$x1grid, soil.estimate$x2grid, soil.estimate$yhat, phi=30, theta=15, xlab="M
g", ylab="Ca", zlab="estimate" )
```



This might be the same. Note that the resolution of the output of `bivReg` is not great, and that there does not seem to be a way of adjusting this.

4.3.2 Multivariate local linear regression

Consider a first-order Taylor expansion of $m : \mathbb{R}^d \longmapsto \mathbb{R}$ around the target point $x \in \mathbb{R}^d$:

$$m(z) = m(x) + \nabla m(x)^T (z - x) + \ldots\ldots$$

where the ∇ operator is defined as

$$\nabla = \left(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_d} \right)^T$$

Now identify $\beta_0 \equiv m(x)$ and

$$\nabla m(x) \equiv \beta_1 = \begin{pmatrix} \beta_{11} \\ \vdots \\ \beta_{1d} \end{pmatrix}$$

This motivates the estimator

$$\hat{m}_{LL,H}(x) = \hat{\beta}_0$$

where $\begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} \in \mathbb{R}^{p+1}$ is found by minimizing

$$Q(\beta_0, \beta_1) = \sum_{i=1}^{n} K_H(x_i - x)\big(y_i - \beta_0 - \beta_1^T(x_i - x)\big)^2$$

which with

$$W_H = \begin{pmatrix} K_H(x_1 - x) & & \\ & \ddots & \\ & & K_H(x_n - x) \end{pmatrix} \in \mathbb{R}^{n \times}, \quad X_d = \begin{pmatrix} 1 & (x_1 - x)^T \\ \vdots & \vdots \\ 1 & (x_n - x)^T \end{pmatrix} \in \mathbb{R}^{n \times d+1}$$

gives

$$\hat{\beta}_0 = e_1^T (X_d W_H X_d)^{-1} X_d^T W_H Y.$$

Some notes on this estimator:

- The asymptotic bias and variance of $\hat{m}_{LL,H}(x)$ have been developed (Wand & Jones, 1995, p. 140, Fan & Gijbels, 1996 p.301), showing similar bias-reducing properties as in the univariate case.
- Optimal bandwidth matrices can in principle be derived based on this concept (Fan & Gijbels, 1996, p. 302, and several journal papers)
- To my knowledge, there does not exist a straightforward, "pure", publicly available implementation of $\hat{\beta}_0$. The closest calls are R functions `loess` and `npreg` which are however somewhat engineered.
- **The curse of dimensionality is hitting here as well!**
- Eventually, the more useful modelling approach is to consider *additive models*,

$$y_i = \beta_0 + m_1(x_{i1}) + \ldots + m_p(x_{ip}) + \text{error}$$

where $m_j : \mathbb{R} \longmapsto \mathbb{R}$ are each estimated by usual, univariate smoothers.

4.4 Additive models

Recall the d-variate nonparametric regression model

$$y_i = m(x_{i1}, \ldots, x_{id}) + \sigma(x_{i1}, \ldots, x_{id})\epsilon_i$$

where the ϵ_i are iid (and also independent from the predictors) with $E(\epsilon_i) = 0$ and $\text{Var}(\epsilon_i) = 1$.

Consider now the y_i as realizations of a random variable Y and the $x_{i1}, \ldots, x_{id}$ as realizations of random variables $X_1, \ldots, X_d$, respectively. Then we can rewrite the model as

$$E(Y|X_1, \ldots, X_d) = m(X_1, \ldots, X_d)$$

Differently to previous sections, we make now the conditional expectation notationally explicit. (That is, in this section 4.4, when no condition is given in the expectation, the expectation is *not conditional*!)

To avoid the curse of dimensionality for the d-variate nonparametric regression estimator, we consider an approximated version of the model displayed above, namely

$$E(Y|X_1, \ldots, X_d) = \beta_0 + m_1(X_1) + \ldots + m_d(X_d)$$

with smooth functions $m_j : \mathbb{R} \longrightarrow \mathbb{R}$, $z \longmapsto m_j(z)$, $j = 1, \ldots, d$.

Is this model well-defined? To answer this, consider an alternative model where we have

$$\begin{aligned}\tilde{m}_1(z) &= m_1(z) - c\\ \tilde{m}_2(z) &= m_2(z) + c\\ \tilde{m}_j(z) &= m_j(z) \quad \text{for } j = 3, \ldots, d\end{aligned}$$

where $d \geq 3$ and $c > 0$. Then clearly

$$\sum_{j=1}^{d} m_j(z) = \sum_{j=1}^{d} \tilde{m}_j(z)$$

so the model is not identifiable. Therefore, in the additive model, all the d functions are assumed to be mean-centered, i.e.

$$E_{X_j}(m_j(X_j)) = \int m_j(z) f_{X_j}(z)\, dz = 0$$

where $f_{X_j}(z)$ is the density of X_j. (This assumption can be made without loss of generality, and in order to do this we do not need to know the densities f_{X_j} or the ``un-centered” means of the functions m_j. The model will implicitly be estimated under this assumption, and we are only interested in the shapes of the trends anyway.)

Note that by the theorem of iterated expectation, for two random variables X and Y, one has $E(Y) = E_X(E(Y|X))$. This implies then, for the unconditional(!) response Y of our nonparametric, additive model,

$$\begin{aligned}E(Y) &= E_{X_1,\ldots,X_d}\left(Y|X_1, \ldots, X_d\right)\\ &= E_{X_1,\ldots,X_d}\left(\beta_0 + m_1(X_1) + \ldots + m_d(X_d)\right)\\ &= \beta_0 + E_{X_1,\ldots,X_d}(m_1(X_1)) + \ldots + E_{X_1,\ldots,X_d}(m_d(X_d))\\ &= \beta_0 + E_{X_1}(m_1(X_1)) + \ldots + E_{X_d}(m_d(X_d))\\ &= \beta_0\end{aligned}$$

So we have immediately the estimator

$$\hat{\beta}_0 = \frac{1}{n} \sum_{i=1}^{n} y_i$$

To estimate the functions m_j, note that we can write

$$\begin{aligned}E(Y|X_1, \ldots, X_d) - \beta_0 - \sum_{\ell \neq j}^{d} m_\ell(X_\ell) &= m_j(X_j)\\ E\left(Y - \beta_0 - \sum_{\ell \neq j}^{d} m_\ell(X_\ell)|X_1, \ldots, X_d\right) &= m_j(X_j)\end{aligned}$$

Therefore, considering the content of the large bracket as a “working response”, for known β_0 and functions m_ℓ, $\ell \neq j$, the function m_j can be estimated. By cycling over j this idea leads to the **Backfitting Algorithm**:

1. Initialize $\hat{\beta}_0 = \frac{1}{n} \sum_{i=1}^{n} y_i$, $\hat{m}_j^{[0]} \equiv 0$, $j = 1, \ldots, d$, where the notation $[s]$ stands for `iteration cycle’.
2. For $j = 1, \ldots, d$, and iteration $s \geq 1$, cycle

$$r_i = y_i - \hat{\beta}_0 - \sum_{\ell=1}^{j-1} \hat{m}_\ell^{[s]}(x_{i\ell}) - \sum_{\ell=j+1}^{d} \hat{m}_\ell^{[s-1]}(x_{i\ell})$$

where $\hat{m}_\ell^{[s]}(x_{i\ell})$ corresponds to terms which have already been updated in the s-th cycle, and $\hat{m}_\ell^{[s-1]}(x_{i\ell})$ corresponds to terms which haven’t. Then $m_j^{[s]}$ is estimated by fitting the model

$$r_i = m_j(x_{ij}) + \tilde{\epsilon}_{ij}$$

to pairs $(r_i, x_{ij}), i = 1, \dots, n$, where $(\tilde{\epsilon}_{ij})$ is some iid error term with mean zero, which is a usual, one-dimensional nonparametric regression problem. Any nonparametric estimator can be used for this, such as the Nadaraya-Watson or local linear estimators. We do this until

3. convergence is reached, which is usually established by verifying that the $\hat{m}_j^{[s]}$ stop changing over successive cycle indexes s.

Example 4.5

We use the R function `gam` in R package **gam**. This function implements the backfitting exactly as derived above, and does support local linear smoothing via the wrapper function `lo` . The implementation of this local linear estimator is based on the `loess` framework which is a slight variant of “our” local linear estimator which uses a `span` parameter (representing the proportion of data points in the neighboorhood of x which are used for the estimation) rather than a fixed bandwidth, and a tricube kernel.

```r
require(gam)
```

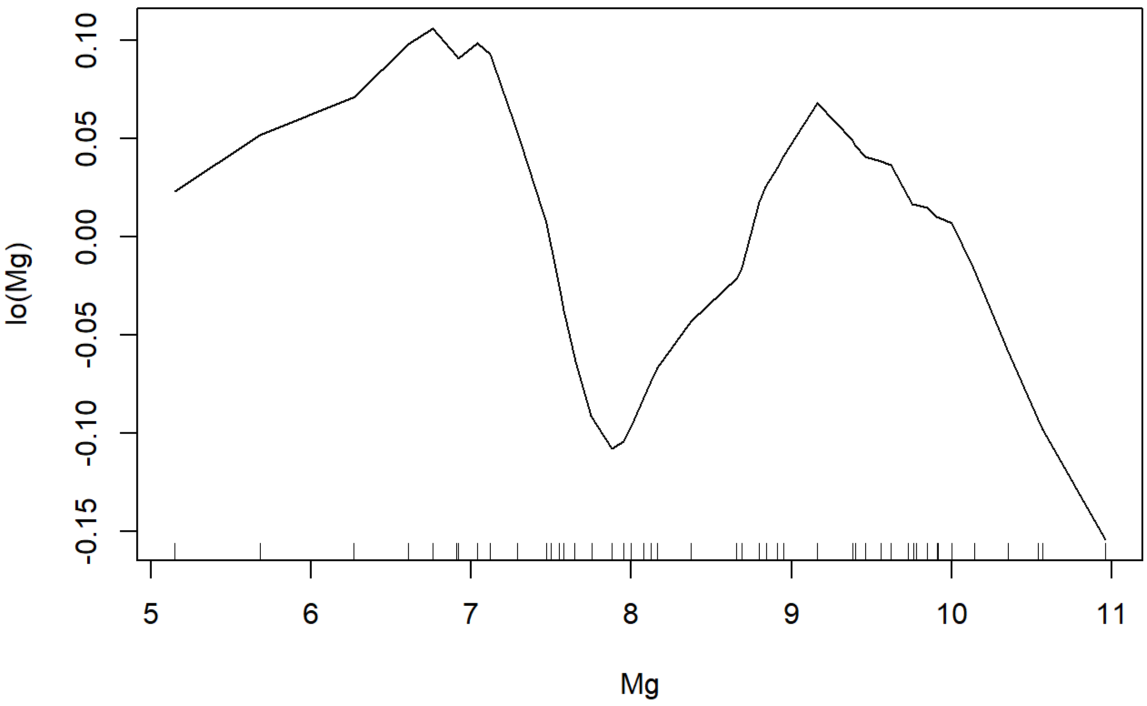
```
## Loading required package: gam
```

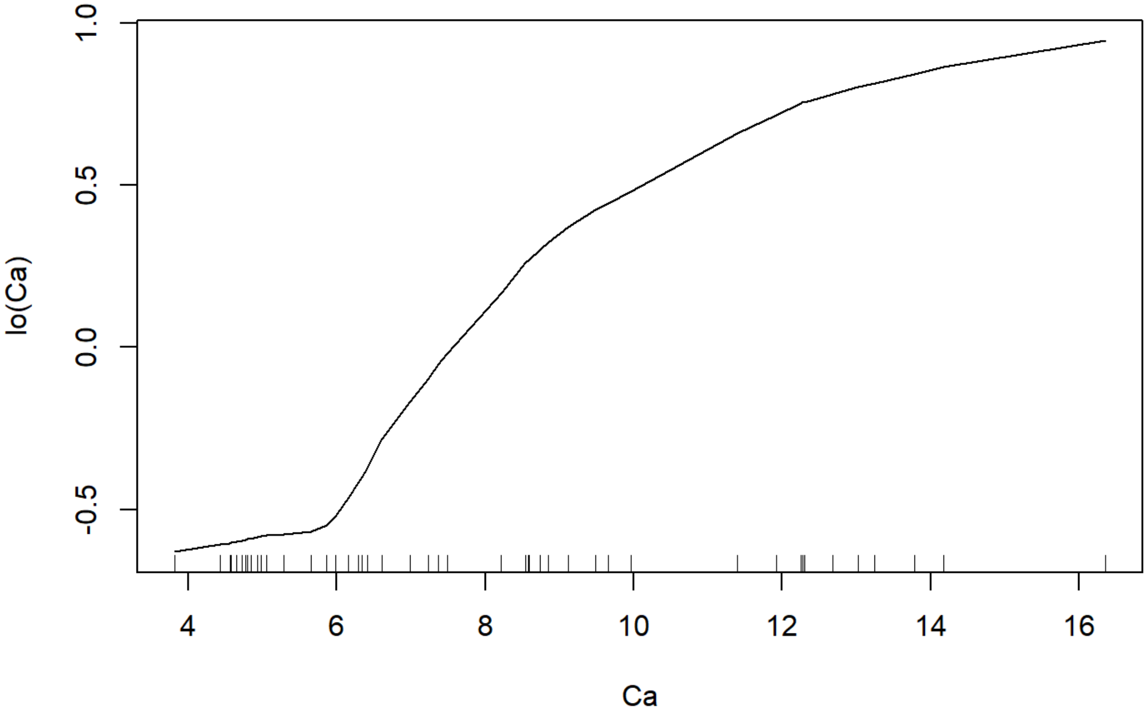
```
## Loading required package: splines
```

```
## Loading required package: foreach
```

```
## Loaded gam 1.22-3
```

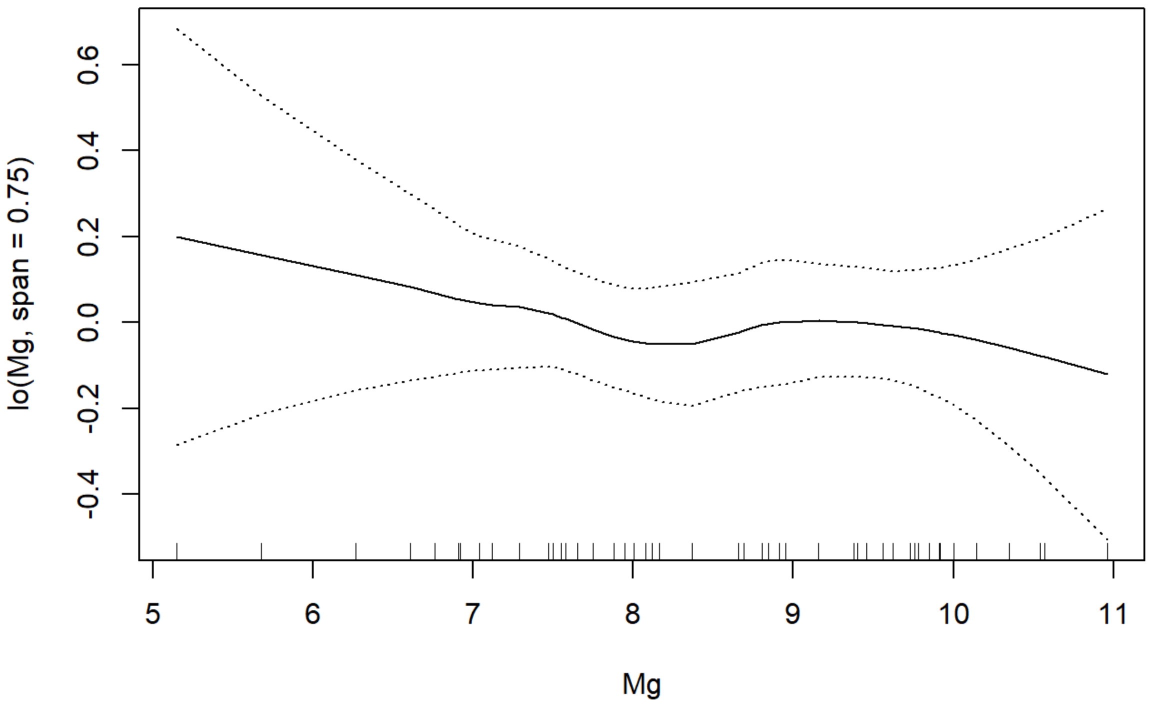
```r
require(carData)
data(Soils)
soil.gam2<-gam(pH~lo(Mg)+lo(Ca), data=Soils)
plot(soil.gam2)
```

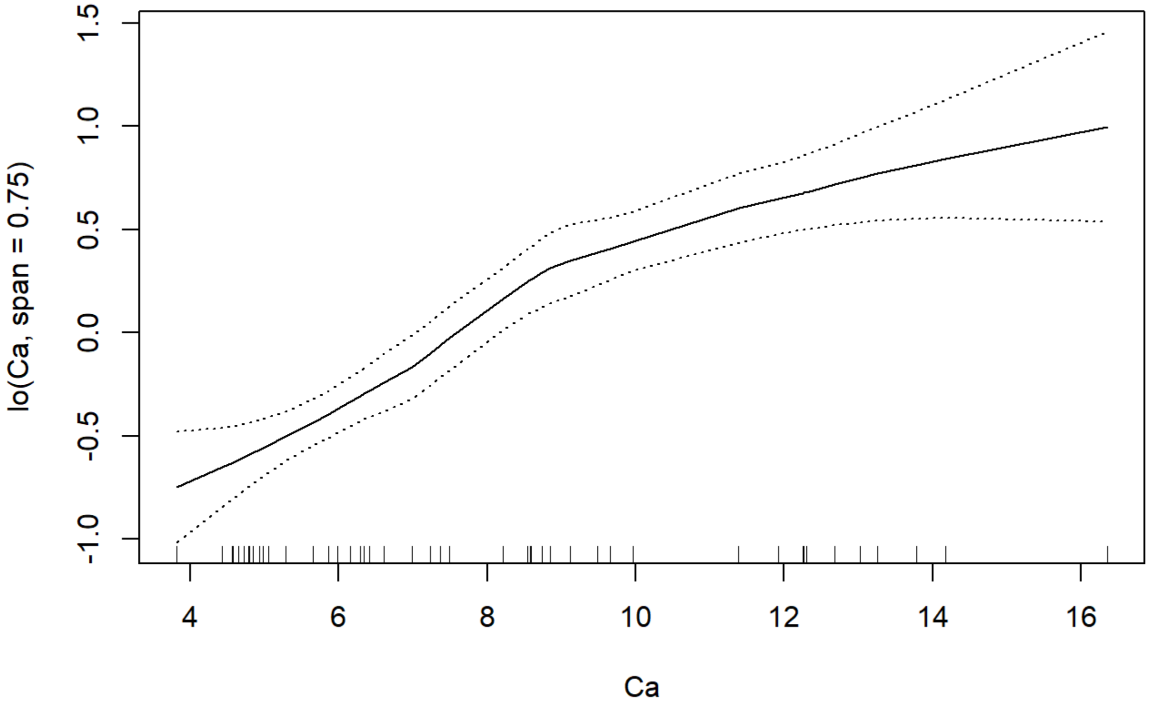




We can change the span (default 0.5) to other values, and also plot (pointwise) confidence intervals:

```
soil.gam2a<-gam(pH~lo(Mg, span=0.75)+lo(Ca, span=0.75), data=Soils)
plot(soil.gam2a, se=TRUE)
```



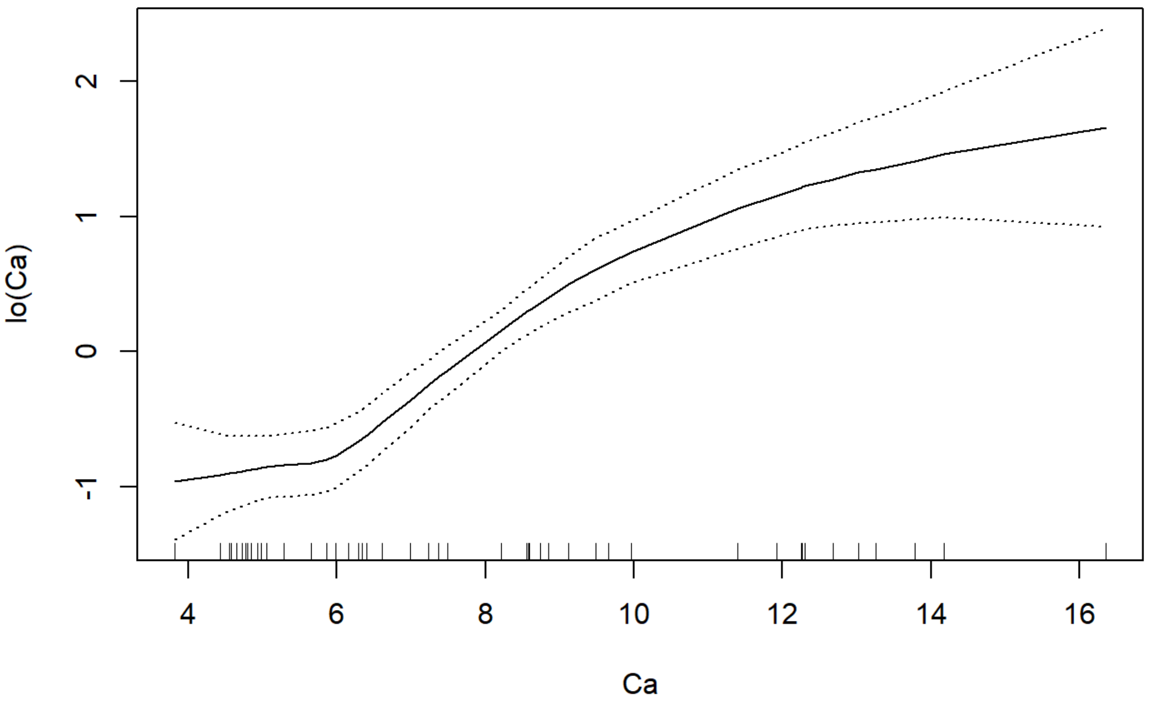


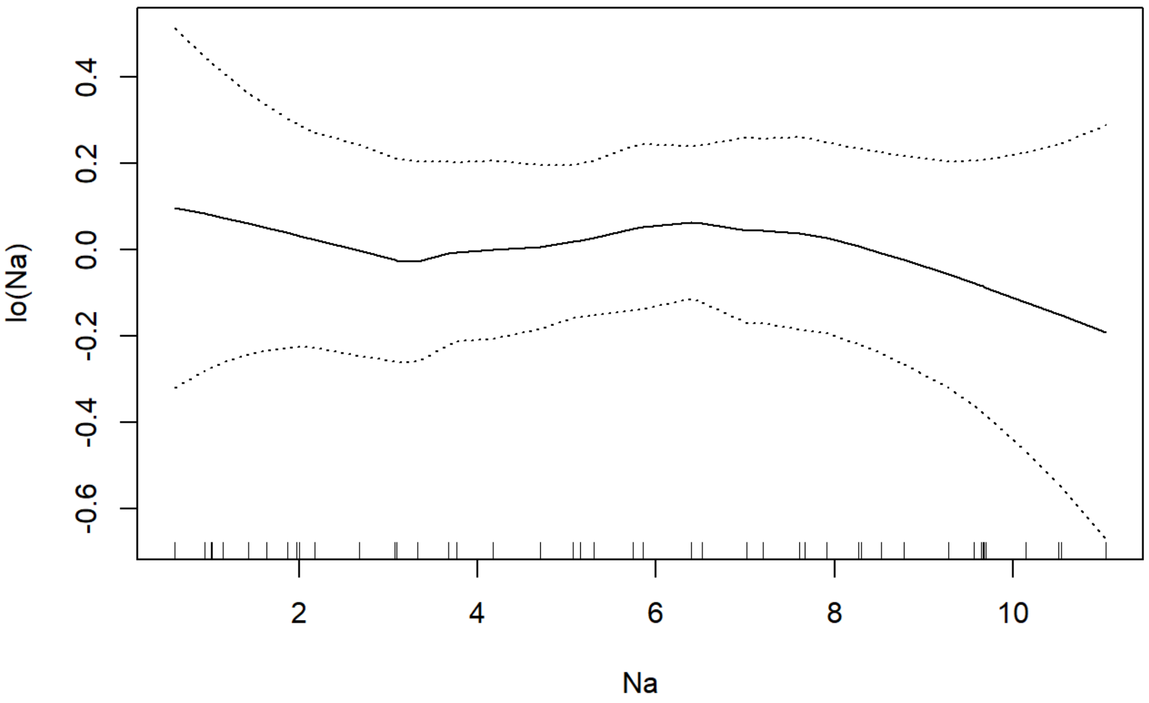
This also works for larger numbers of predictors:

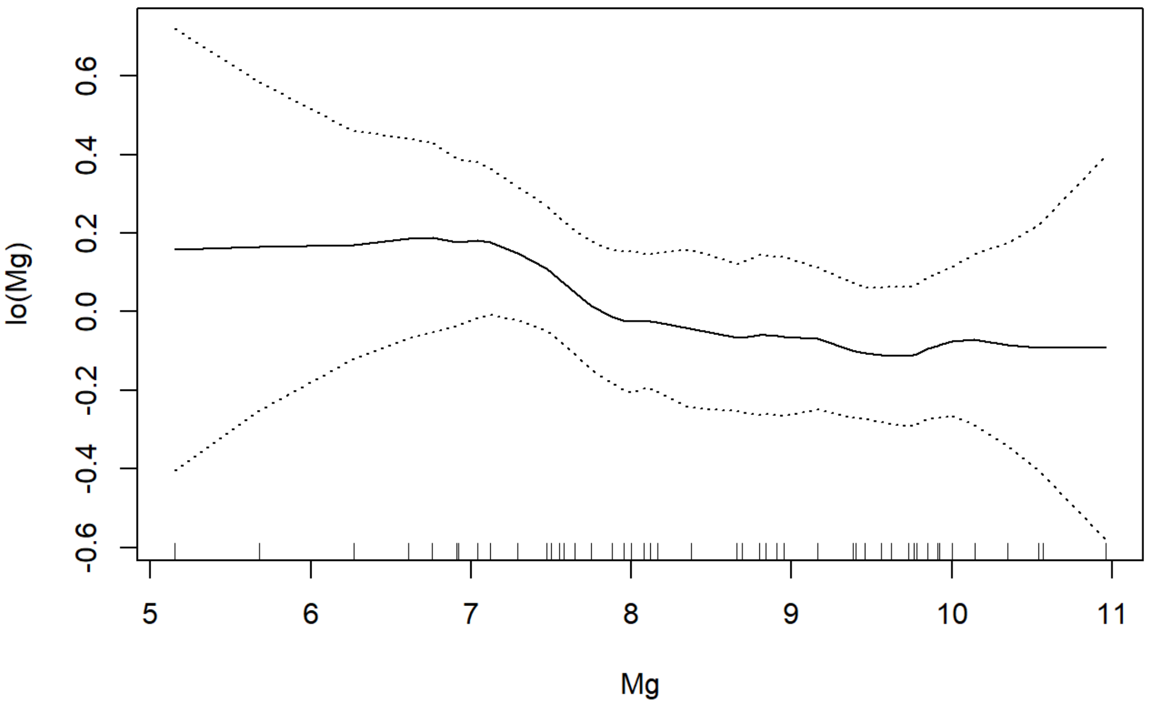
```
soil.gam6<- gam(pH~lo(Ca)+lo(Na)+lo(Mg)+lo(P)+lo(N)+lo(K), data=Soils)
```

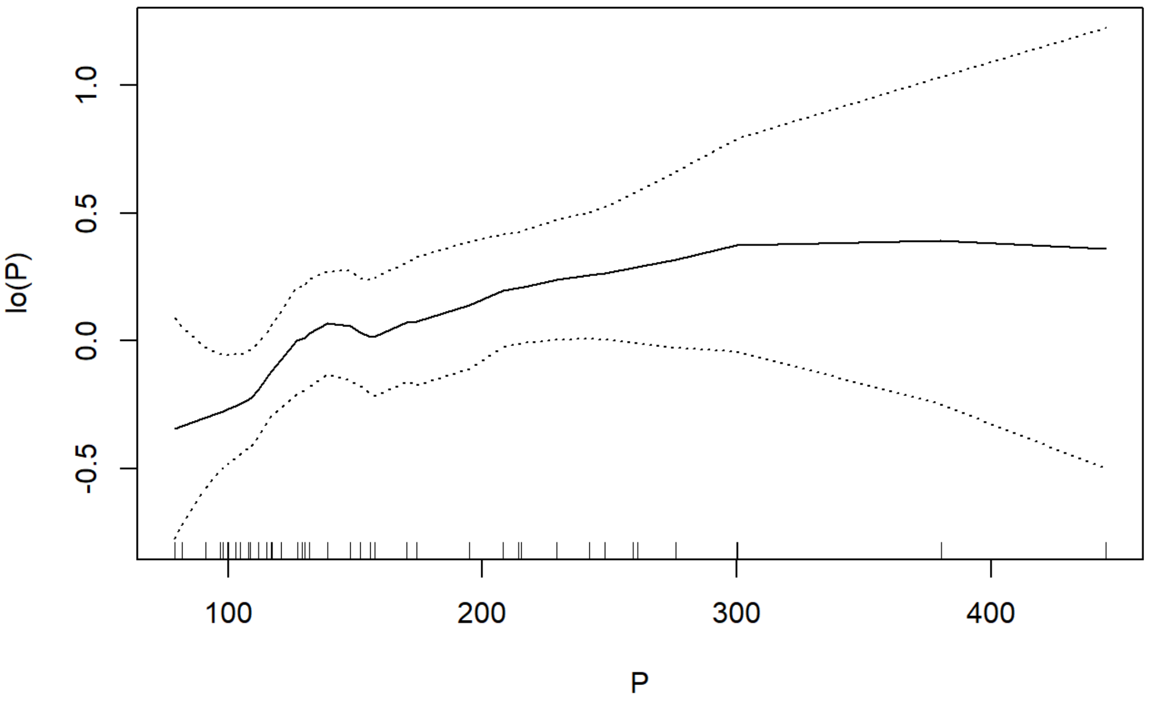
```
## Warning in lo.wam(x, z, wz, fit$smooth, which, fit$smooth.frame, bf.maxit, :
## lo.wam convergence not obtained in 30 iterations
```

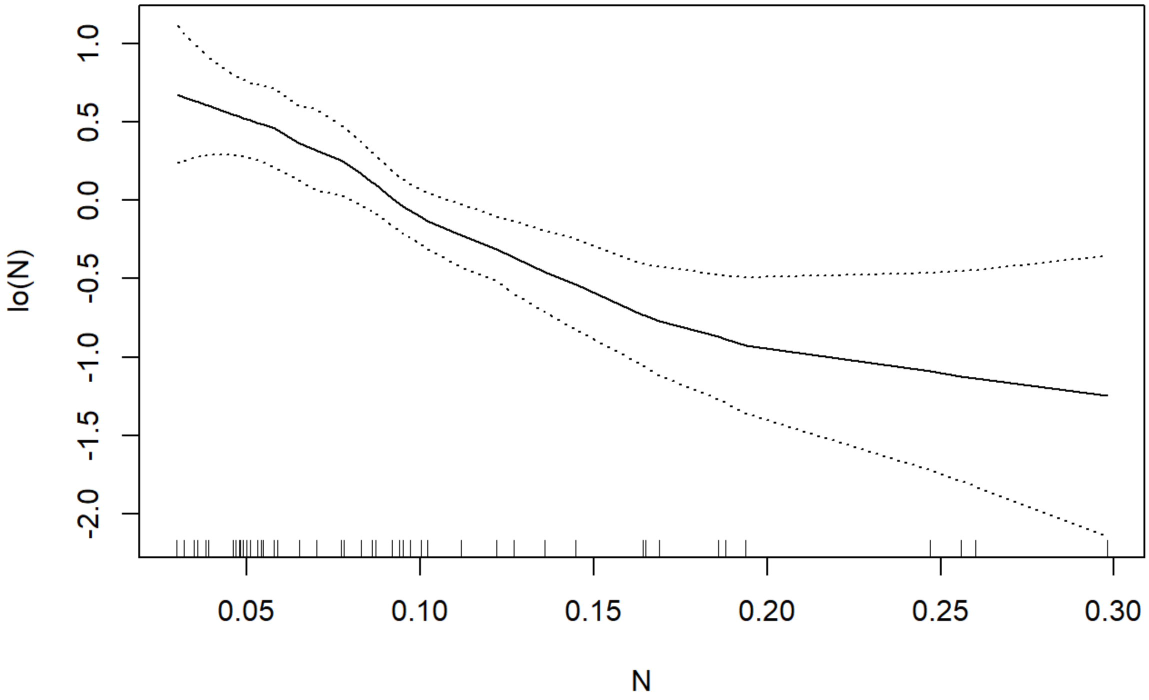
```
plot(soil.gam6, se=TRUE)
```

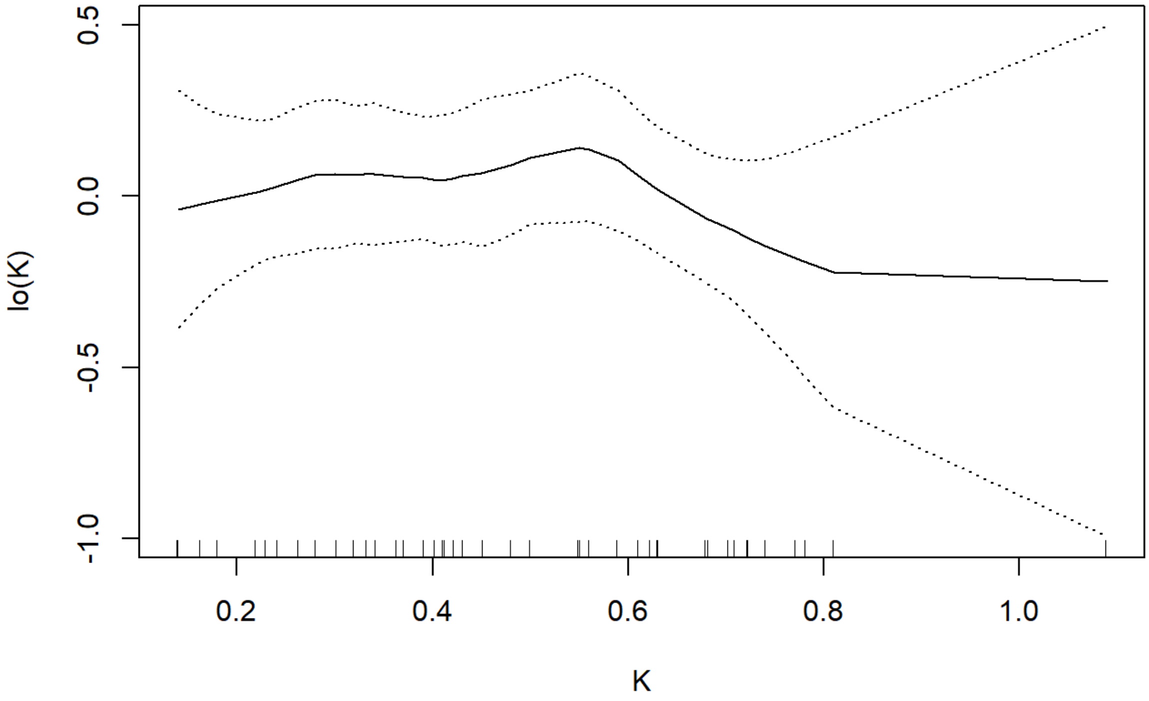

lo(Ca)
2
1
0
-1
4
6
8
10
12
14
16
Ca


lo(Na)
0.4
0.2
0.0
-0.2
-0.4
-0.6
2
4
6
8
10
Na


lo(Mg)
0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6
5
6
7
8
9
10
11
Mg


lo(P)
1.0
0.5
0.0
-0.5
100
200
300
400
P





```
summary(soil.gam6)
```

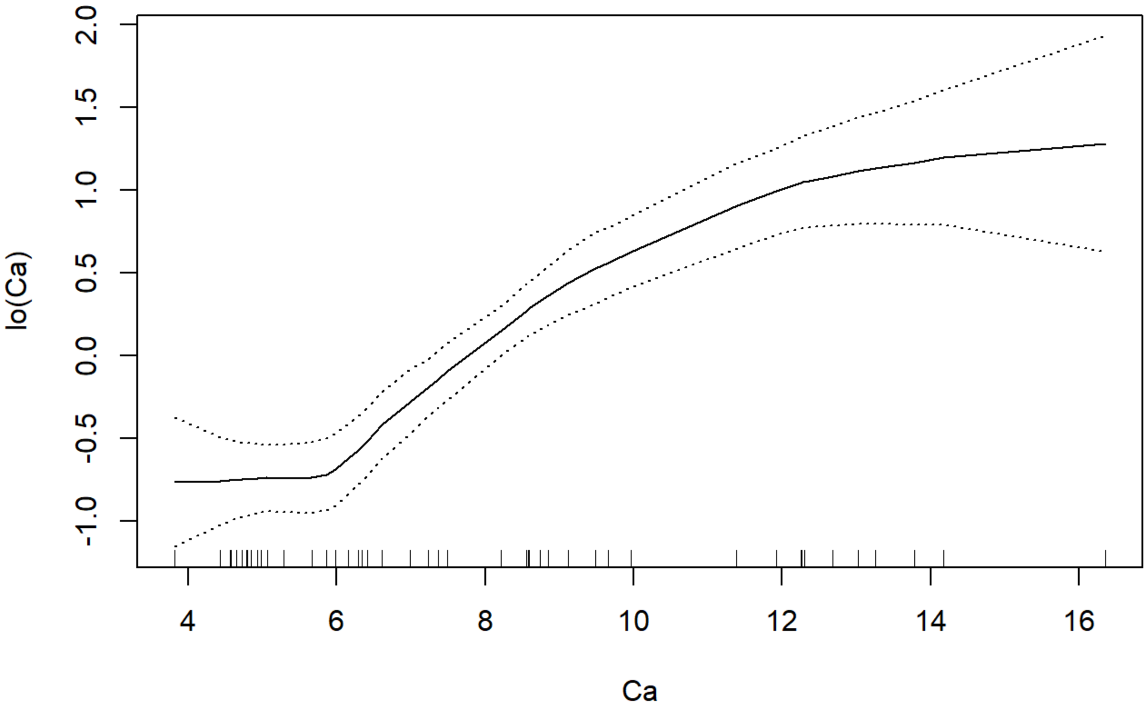
```
##
## Call: gam(formula = pH ~ lo(Ca) + lo(Na) + lo(Mg) + lo(P) + lo(N) +
##     lo(K), data = Soils)
## Deviance Residuals:
##       Min        1Q    Median        3Q       Max
## -0.499466 -0.216506 -0.009361  0.213990  0.522473
##
## (Dispersion Parameter for gaussian family taken to be 0.135)
##
##     Null Deviance: 21.2153 on 47 degrees of freedom
## Residual Deviance: 3.1175 on 23.0939 degrees of freedom
## AIC: 56.7898
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##               Df  Sum Sq Mean Sq  F value    Pr(>F)
## lo(Ca)     1.000 14.8328 14.8328 109.8799 2.982e-10 ***
## lo(Na)     1.000  0.0000  0.0000   0.0000 0.9991789
## lo(Mg)     1.000  0.0897  0.0897   0.6648 0.4232095
## lo(P)      1.000  0.0332  0.0332   0.2460 0.6245605
## lo(N)      1.000  2.3331  2.3331  17.2833 0.0003778 ***
## lo(K)      1.000  0.0499  0.0499   0.3699 0.5489880
## Residuals 23.094  3.1175  0.1350
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##             Npar Df Npar F   Pr(F)
## (Intercept)
## lo(Ca)          2.7 4.2311 0.01841 *
## lo(Na)          2.3 0.3946 0.70327
## lo(Mg)          3.1 0.3870 0.76775
## lo(P)           3.4 1.0840 0.38096
## lo(N)           3.1 1.2676 0.30914
## lo(K)           3.4 0.8246 0.50530
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

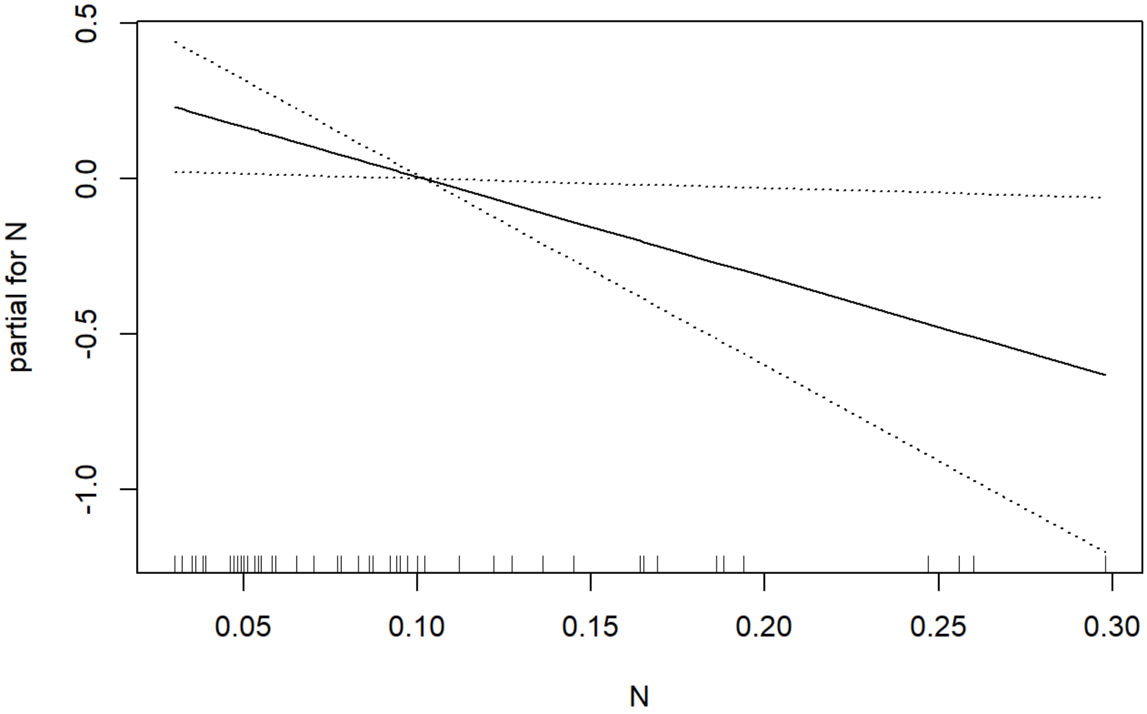
The `summary` output describes two types of significances: The “Anova for Parametric Effects” establishes whether the term as a whole is needed (whether parametric or nonparametric), and the “Anova for Nonparametric Effects” establishes whether there is evidence that the nonparametric term should be preferred over the parametric one. Note that this is ANOVA-based so depends on the order of inclusion. Another useful technique to assess the relevance of the individual terms is to look at the curve plots with the confidence intervals.

- if you can lay a horizontal line through the confidence intervals, the entire term may not be needed
- if you can lay *any* line through the confidence intervals, the nonparametric term may not be needed (but still the linear one)

If you mix and match parametric and nonparametric terms, you get a **semiparametric model**:

```
soil.gam.reduced<- gam(pH~lo(Ca)+N, data=Soils)
plot(soil.gam.reduced, se=TRUE)
```



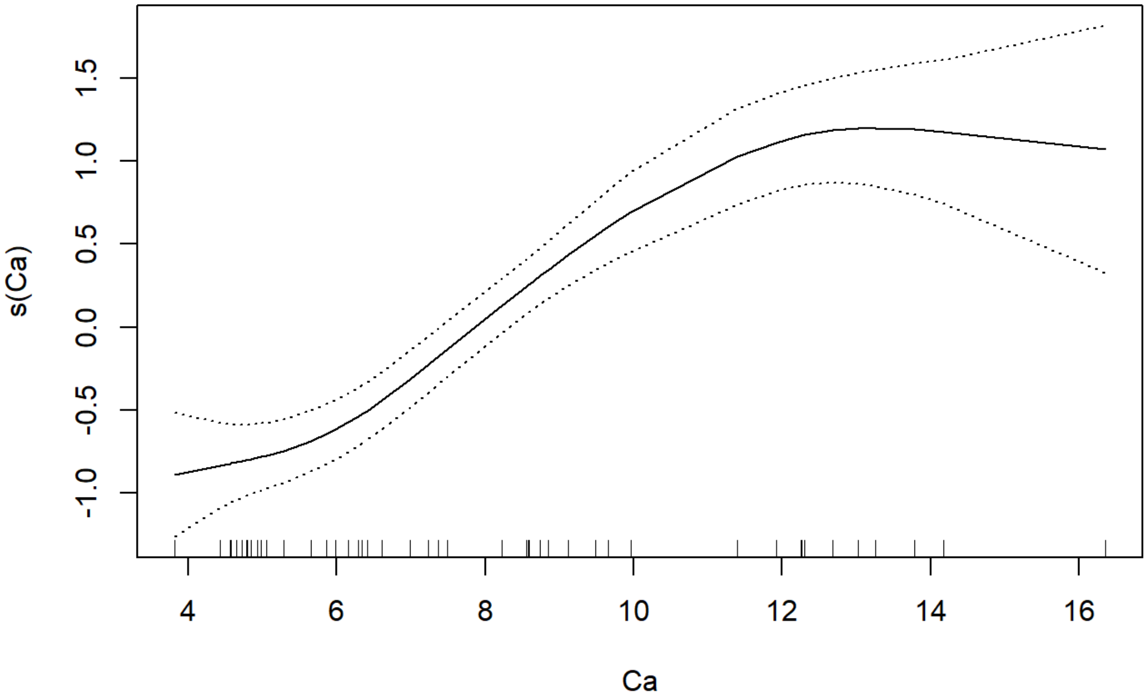


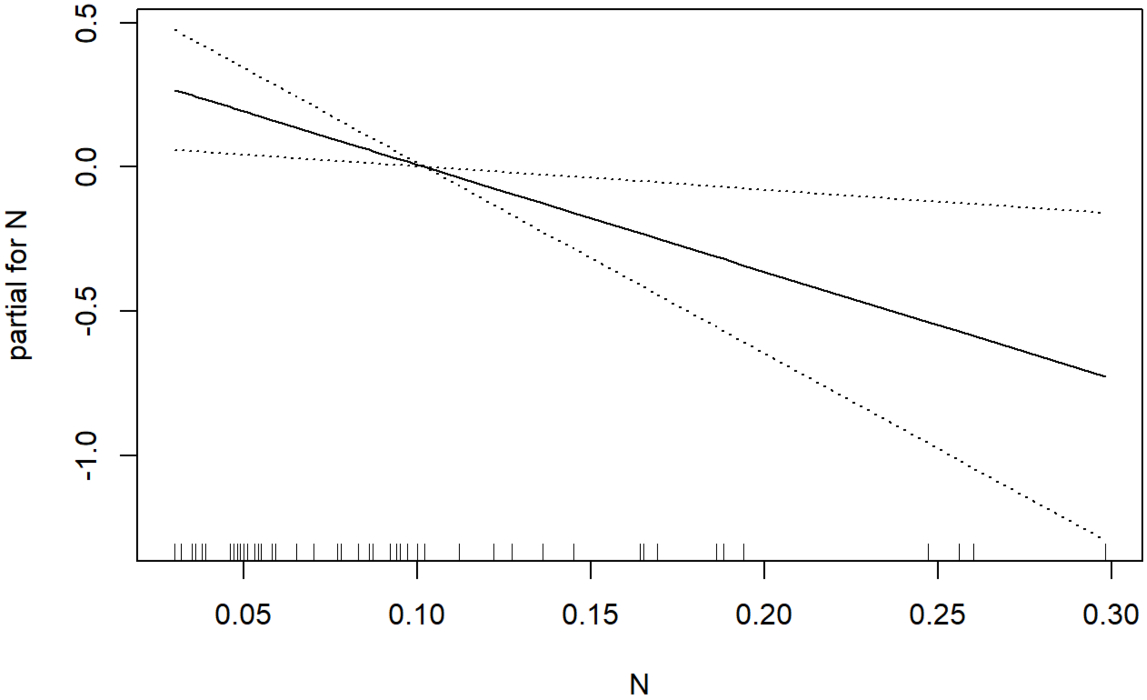
```
summary(soil.gam.reduced)
```

```
##
## Call: gam(formula = pH ~ lo(Ca) + N, data = Soils)
## Deviance Residuals:
##       Min        1Q    Median        3Q       Max
## -0.702791 -0.233828 -0.008408  0.229444  0.857314
##
## (Dispersion Parameter for gaussian family taken to be 0.1245)
##
##     Null Deviance: 21.2153 on 47 degrees of freedom
## Residual Deviance: 5.2626 on 42.2822 degrees of freedom
## AIC: 43.5459
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##               Df  Sum Sq Mean Sq  F value    Pr(>F)
## lo(Ca)     1.000 13.8723 13.8723 111.4568 1.996e-13 ***
## N          1.000  0.6074  0.6074   4.8802   0.03264 *
## Residuals 42.282  5.2626  0.1245
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##             Npar Df Npar F    Pr(F)
## (Intercept)
## lo(Ca)          2.7 5.5649 0.003416 **
## N
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note there exist other smoothers which you can use….

```
soil.gam.reduced<- gam(pH~s(Ca)+N, data=Soils)
plot(soil.gam.reduced, se=TRUE)
```





```
summary(soil.gam.reduced)
```

```
##
## Call: gam(formula = pH ~ s(Ca) + N, data = Soils)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -0.72518 -0.20404 -0.02767  0.19568  0.74505
##
## (Dispersion Parameter for gaussian family taken to be 0.1233)
##
##     Null Deviance: 21.2153 on 47 degrees of freedom
## Residual Deviance: 5.1791 on 42 degrees of freedom
## AIC: 43.3426
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##           Df  Sum Sq Mean Sq  F value    Pr(>F)
## s(Ca)      1 13.8723 13.8723 112.4977 1.879e-13 ***
## N          1  0.8079  0.8079   6.5515   0.01417 *
## Residuals 42  5.1791  0.1233
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##             Npar Df Npar F   Pr(F)
## (Intercept)
## s(Ca)             3 5.3143 0.00339 **
## N
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

4.5 Smoothing Splines

We return now to univariate smoothing, that is we postulate a model

$$y_i = m(x_i) + \sigma(x_i)\epsilon_i$$

under the usual assumptions. A good smoother $\hat{m}$ will

- have $\hat{m}(x_i)$ "close" to the y_i (in some sense)
- be reasonably "smooth".

Idea: Create an explicit **global** criterion which encapsulates these two goals. One approach is to find a function m (assumed twice continuously differentiable) which minimizes

$$\sum_{i=1}^{n}(y_i - m(x_i))^2 + \lambda \int_{-\infty}^{\infty} m''(u)^2\, du$$

where $\lambda \geq 0$ is a smoothing parameter (analogous to the bandwidth, h) such that

- $\lambda = 0$: no smoothing (interpolation),
- $\lambda \longrightarrow \infty$: maximal smoothing (linear global fit).

One can show (Green & Silverman, 1984) that solutions to the minimization problem above are

- cubic polynomial pieces ('cubic splines') which connect at the locations of the x_i (but not necessarily at exactly (x_i, y_i)!) such that $\hat{m}'$ and $\hat{m}''$ are continuous;

- equivalent to solutions of the penalized least squares (PLS) problem

$$\text{PLS}(M) = (Y-M)^T(Y-M) + \lambda M^T K M$$

where (as before)

$$Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \qquad M = \begin{pmatrix} m(x_1) \\ \vdots \\ m(x_n) \end{pmatrix}$$

and $K \in R^{n\times n}$ is a matrix with a rather complex structure which however only depends on the differences $x_{i+1} - x_i$, $i = 1, \ldots, n-1$ (Green and Silverman, 1994, page 12, Fahrmeir and Tutz, 2001, page 181).

So one can find an estimate of M by differentiating

$$\frac{d\,\text{PLS}(M)}{dM} = -2(Y-M) + 2\lambda KM \overset{!}{=} 0.$$

Hence

$$\begin{aligned} Y - M &= \lambda KM \\ Y &= M + \lambda KM \\ (I_n + \lambda K)M &= Y \\ \hat{M} &= (I+\lambda K)^{-1}Y \end{aligned}$$

Note that this is easily implemented (when looking up K), and only requires a *single* global estimation, rather than an estimation at each target point (as for local regression). As disadvantage, one could state that now the $n \times n$ matrix $I + \lambda K$ needs to be inverted, which may be computationally demanding for large data sets.

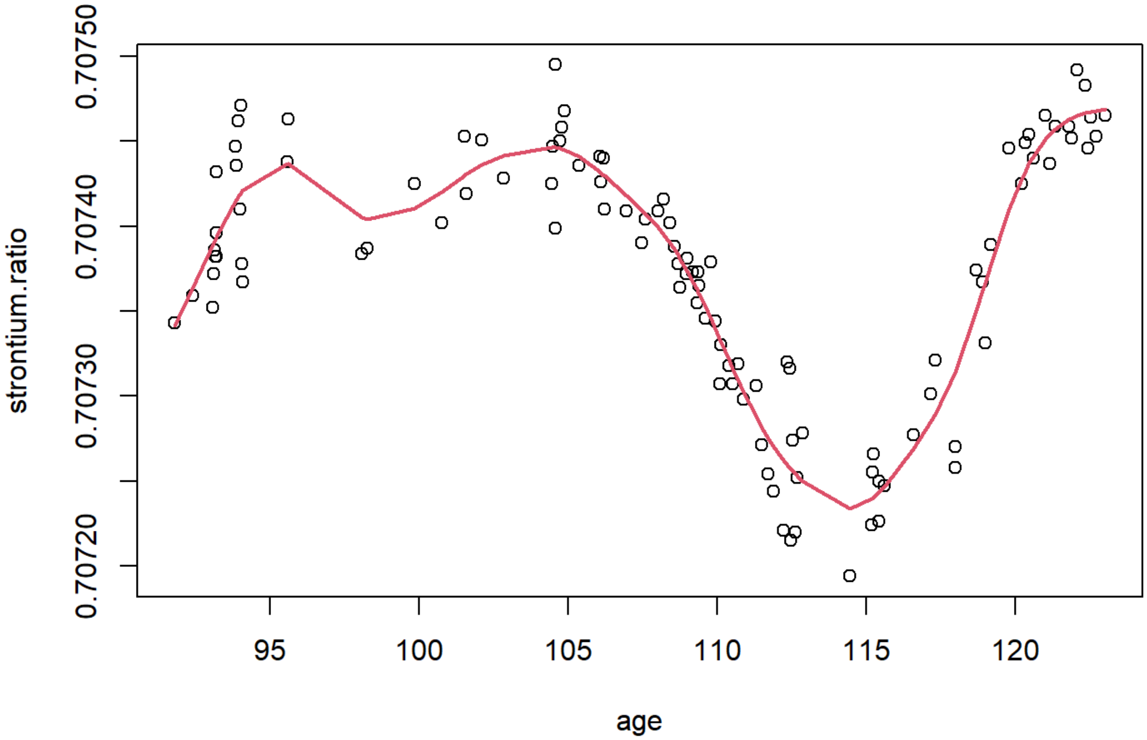
Example 4.6

We look again at the fossil data, and produce a smoothing spline fit using function `sm.spline` in R package `pspline`.

```
require(pspline)
```

```
## Loading required package: pspline
```

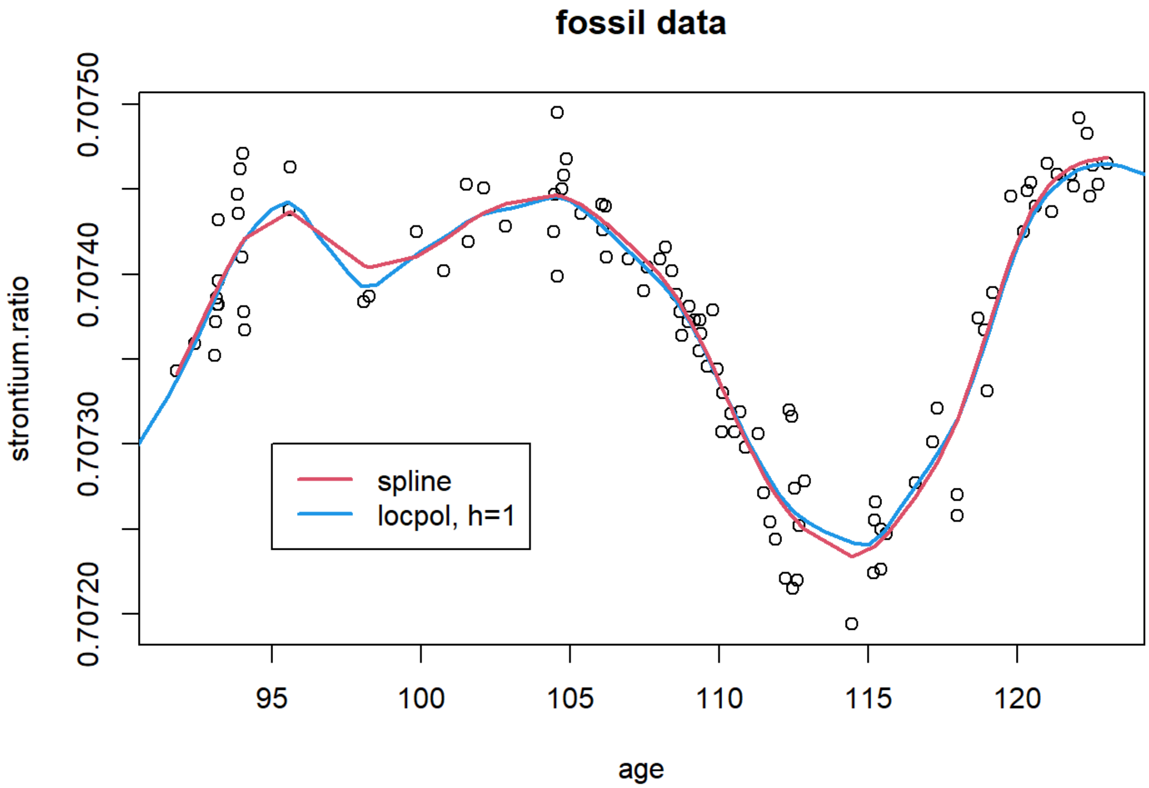
```
require(SemiPar)
data(fossil)
fossil.spline <- sm.spline(fossil$age, fossil$strontium.ratio)
plot(fossil)
lines(fossil.spline, col=2, lwd=2)
```



With some investigation by looking at the help file (`?sm.spline`), we find that there is an argument `norder` which defaults to the value 2 and that this corresponds indeed to a cubic smoothing spline. We also find from this help file that this function chooses the value λ through a cross-validation criterion.

Let us compare this estimate to a local linear estimator. We choose the bandwidth $h = 1$ for this purpose which is close to the optimal bandwidth found through `dpill` earlier.

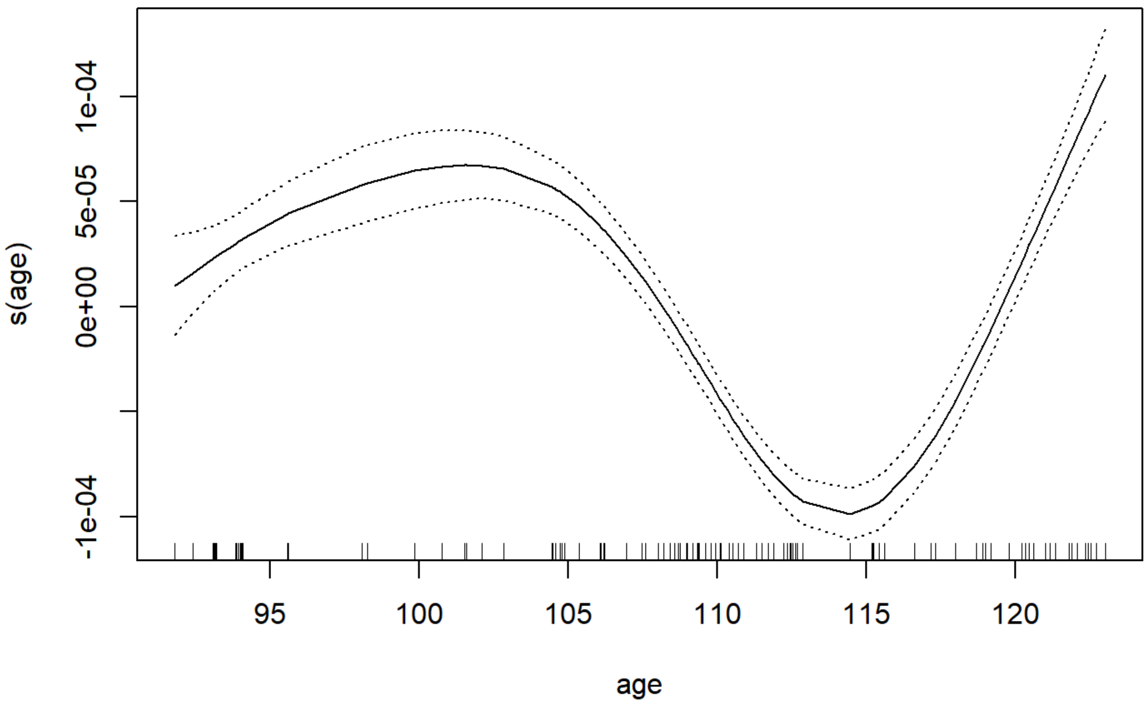
```
age.grid<- seq(90, 130, by=0.5)
require(locpol)
fit2b<- locpol(strontium.ratio~age, bw=1, deg=1,  kernel=gaussK, xeval=age.grid,  data=fossil)
plot(fossil, main="fossil data")
lines(fit2b$lpFit[1:2], lty=1, lwd=2, col=4)
lines(fossil.spline, col=2, lwd=2)
legend(95,0.70730, c("spline", "locpol, h=1"), lwd=c(2,2), lty=c(1,1), col=c(2,4))
```



We see that the smoothing splines performs smoother at the first dip but manages to get deeper into the second. This could indicate superior behavior of the spline estimate. On the other hand, the estimate appears “edgy” at some points which could be related to the fact that the smoothing spline function is only evaluated at the data points.

A “trick” which is sometimes useful is to apply the `gam` function even for univariate smoothing. This allows access to confidence intervals and other inferential devices. For instance, we get

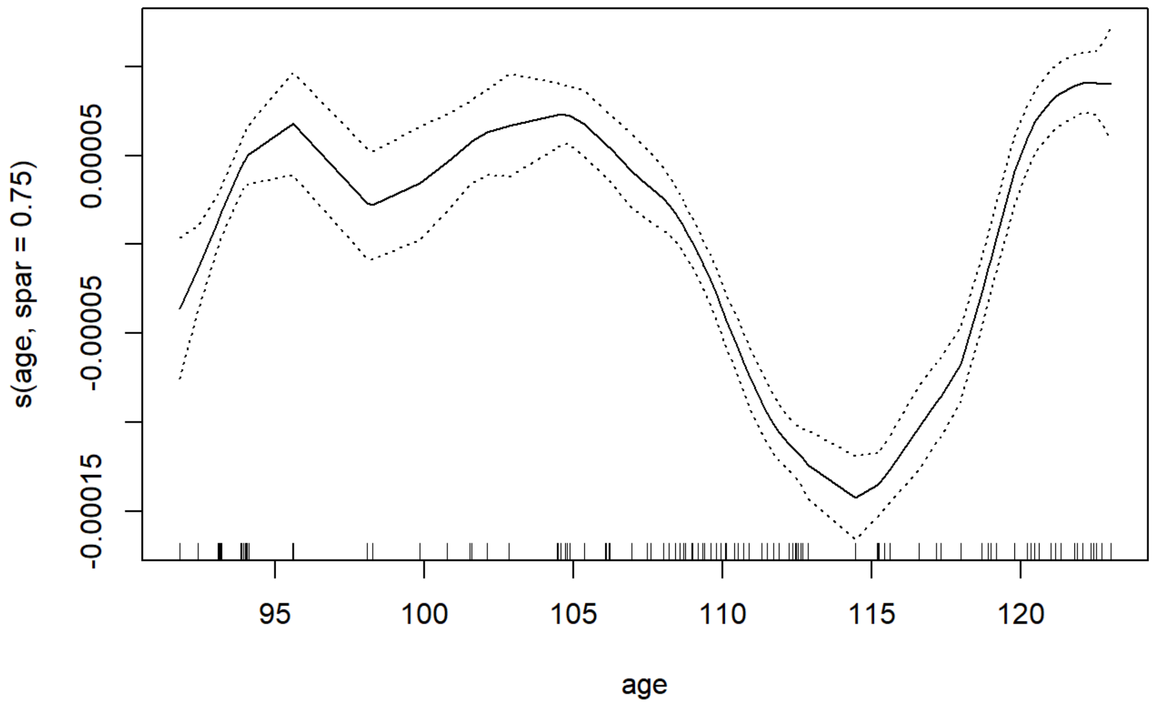
```
require(gam)
fossil.spline2 <- gam(strontium.ratio~s(age), data=fossil)
plot(fossil.spline2, se=TRUE)
points(fossil)
```



Here again, a look at the help file (`?s`) tells us that this smoother uses 4 degrees of freedom. Since a cubic polynomial has 4 parameters, this indicates correspondence to a cubic smoothing spline. Looking deeper, one finds that the function `s()` in fact makes use of the R function `smooth.spline` in the base R package `stats` , which however implements a B-spline fit, which is different. (Unfortunately, the R function names in this field are very confusing…)

The smoothness of this spline fit is steered by an option `spar` which defaults to the value 1. From the image above and from what we have seen previously for this data set, this looks too large. Let’s try something less smooth…

```
# ?s
fossil.spline3 <- gam(strontium.ratio~s(age, spar=0.75), data=fossil)
plot(fossil.spline3, se=TRUE)
```



This looks now to be of the familiar magnitude. Understandig how smoothing parameters from different smoothers relate to each other is an advanced topic… we stop however here.