MATH5714 Linear Regression, Robustness and Smoothing

Jochen Voss

University of Leeds, Semester 1, 2025/26

Contents

Preface

The MATH5714M module includes all the material from MATH3714 (Linear Regression and Robustness), plus additional material on smoothing.

The level 3 component covers linear regression and its extensions. In simple linear regression, we fit a line through data points $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^2$ using the model

$$y_i = \alpha + \beta x_i + \varepsilon_i$$

for all $i \in \{1, 2, \ldots, n\}$, where the aim is to find values for the intercept α and the slope β such that the fitted line is as close as possible to the data points.

However, linear models are not always appropriate. Figure 1 illustrates an example where a straight line model would be inadequate. For such data, we need more flexible methods that can capture non-linear patterns.

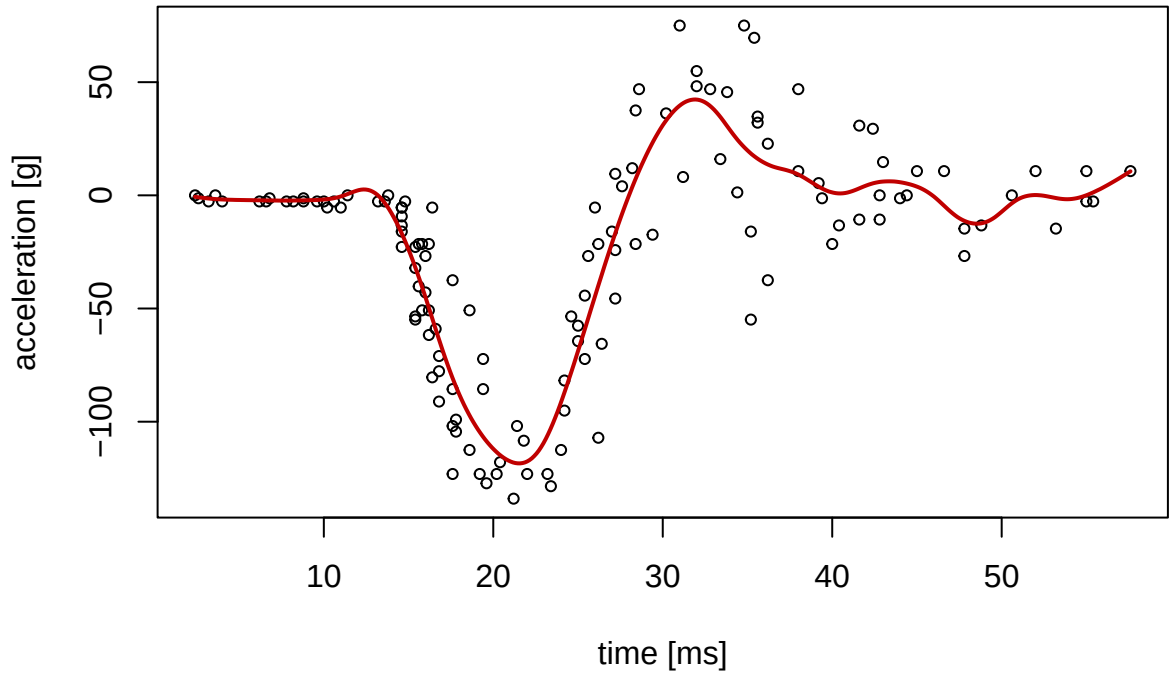


Figure 1: An illustration of a dataset where a linear (straight line) model is not appropriate. The data represents a series of measurements of head acceleration in a simulated motorcycle accident, used to test crash helmets (the `mcycle` dataset from the `MASS` R package).

The level 5 component introduces techniques for fitting smooth curves through data—methods collectively known as **smoothing**. Unlike linear regression, which imposes a straight line on all the data, smoothing methods work locally: they estimate the relationship at different points using nearby observations, then blend these local estimates together to create a smooth curve. The red curve in Figure 1 shows such a smoothed fit, which captures the complex pattern in the data without assuming any particular functional form.

We will approach these smoothing methods gradually. We begin with the simpler problem of "kernel density estimation": given a sample from an unknown distribution, how can we estimate the probability density? This introduces fundamental concepts (kernels, bandwidth selection, and the bias-variance trade-off) that apply throughout all smoothing methods. We then extend these ideas to regression smoothing, where we estimate the relationship between variables rather than a probability distribution.

1 Kernel Density Estimation

In this section, we suppose we are given data $x_1, \dots, x_n \in \mathbb{R}$ from an unknown probability density f, and our goal is to estimate f from the data. As explained in the preface, this seemingly simpler problem introduces key concepts (kernels, bandwidth selection) that we will later apply to regression smoothing.

1.1 Probability Densities

Before we consider how to estimate a density, let us recall what a density is. A random variable $X \in \mathbb{R}$ has **density** $f \colon \mathbb{R} \to [0, \infty)$ if

$$P(X \in [a, b]) = \int_a^b f(x)\, dx$$

for all $a, b \in \mathbb{R}$ with $a < b$. Densities are also called "probability densities" or "probability density functions".

Graphically, the integral $\int_a^b f(x)\, dx$ can be interpreted as the area under the graph of f. This is illustrated in figure 2.

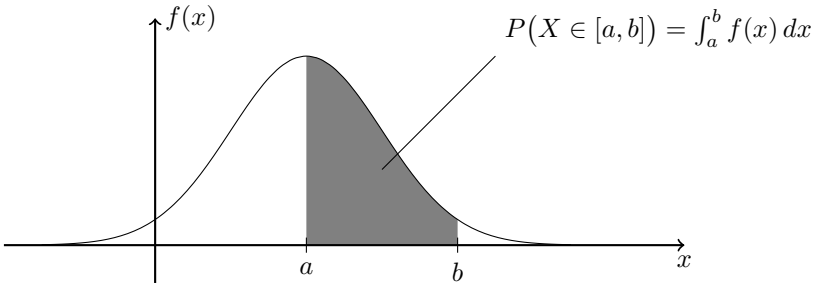


Figure 2: An illustration of how the area under the graph of a density can be interpreted as a probability.

A function f is the density of a random variable X if and only if $f \geq 0$ and

$$\int_{-\infty}^{\infty} f(x)\, dx = 1.$$

In the following, we will see how to estimate f from data.

1.2 Histograms

A commonly used technique to approximate the density of a sample is to plot a histogram. To illustrate this, we use the `faithful` dataset built into R, which contains waiting times between eruptions and the duration of the eruption for the Old Faithful geyser in the Yellowstone National Park. (You can type `help(faithful)` in R to learn more about this dataset.) Here we focus on the waiting times only. A simple histogram for this dataset is shown in figure 3.

```
x <- faithful$waiting
hist(x, probability = TRUE,
     main = NULL, xlab = "time between eruptions [mins]")
```

The histogram splits the data range into "buckets", with bar heights proportional to the number of samples in each bucket.

As we have seen, the area under the graph of the density f over the interval $[a, b]$ corresponds to the probability $P(X \in [a, b])$. For the histogram to approximate the density, we need to scale the height $h_{a,b}$ of the bucket $[a, b]$ so that the area in the histogram is also close to this probability. Since we don't know the probability $P(X \in [a, b])$ exactly, we have to approximate it as

$$P(X \in [a, b]) \approx \frac{n_{a,b}}{n},$$

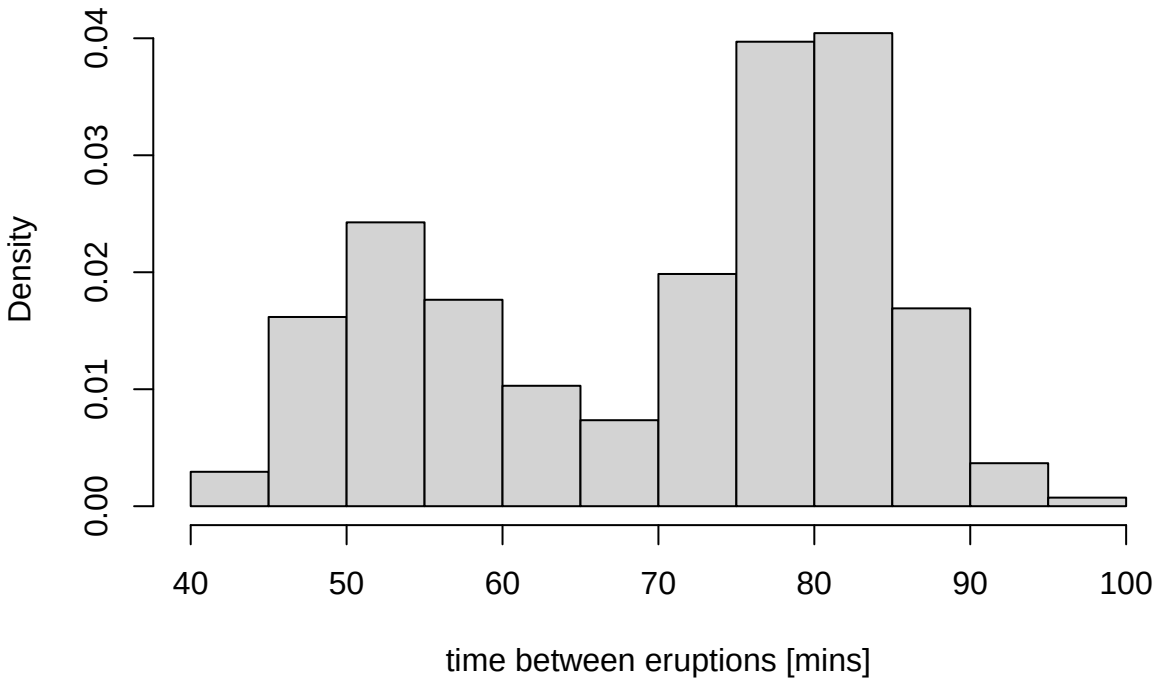


Figure 3: This figure shows how a histogram can be used to approximate a probability density. From the plot one can see that the density of the waiting times distribution seems to be bi-modal with peaks around 53 and 80 minutes.

where $n_{a,b}$ is the number of samples in the bucket $[a, b]$, and n is the total number of samples. So we need

$$\begin{aligned}(b-a)\cdot h_{a,b} &= \text{area of the histogram bar} \\ &\approx \text{area of the density} \\ &= P(X \in [a,b]) \\ &\approx \frac{n_{a,b}}{n}.\end{aligned}$$

and thus we choose

$$h_{a,b} = \frac{1}{(b-a)n} n_{a,b}.$$

We can write the count $n_{a,b}$ as a sum over the data:

$$n_{a,b} = \sum_{i=1}^{n} I_{[a,b]}(x_i),$$

where $I_{[a,b]}(x) = 1$ if $x \in [a, b]$ and $I_{[a,b]}(x) = 0$ otherwise. Substituting this into the formula for the histogram, we get

$$h_{a,b} = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{b-a} I_{[a,b]}(x_i). \tag{1}$$

This shows that the histogram can be written as an average of indicator functions.

Histograms have several limitations:

- Histograms are only meaningful if the buckets are chosen well. If the buckets are too large, not much of the structure of f can be resolved. If the buckets are too small, the estimate $P(X \in [a, b]) \approx n_{a,b}/n$ will be poor and the histogram will no longer resemble the shape of f.
- Even if reasonable bucket sizes are chosen, the result can depend quite strongly on the exact locations of these buckets.
- Histograms have sharp edges at bucket boundaries, creating discontinuities in the density estimate even when the true density f is smooth.

Kernel density estimation addresses these issues by using smooth weight functions, called "kernels", that gradually decrease as we move away from x. Each sample x_i contributes to the estimate at x with a weight that depends on the distance $|x - x_i|$. Nearby samples get higher weight, distant samples get lower weight, and the transition is smooth.

1.3 Kernels

To make this idea precise, we first define what we mean by a kernel.

Definition 1.1. A **kernel** is a function $K \colon \mathbb{R} \to \mathbb{R}$ such that

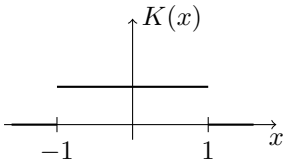
- $\int_{-\infty}^{\infty} K(x)\,dx = 1$,
- $K(x) = K(-x)$ (*i.e.* K is symmetric) and
- $K(x) \geq 0$ (*i.e.* K is positive) for all $x \in \mathbb{R}$.

The first condition ensures that K must decay as $|x|$ becomes large. The second condition, symmetry, ensures that K is centred around 0, and the third condition, positivity, makes K a probability density. (While most authors list all three properties shown above, sometimes the third condition is omitted.)

There are many different kernels in use. Some examples are listed below; a more exhaustive list is available on Wikipedia.

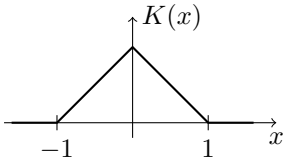
- Uniform Kernel:

$$K(x) = \begin{cases} 1/2 & \text{if } -1 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$



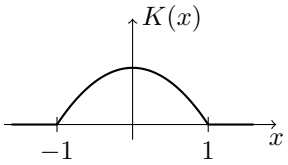
- Triangular Kernel:

$$K(x) = \begin{cases} 1 - |x| & \text{if } -1 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$



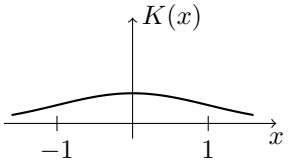
- Epanechnikov Kernel:

$$K(x) = \begin{cases} \frac{3}{4}(1 - x^2) & \text{if } -1 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$



- Gaussian Kernel:

$$K(x) = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)$$



The "width" of the kernel determines how smooth the resulting estimate is. A narrow kernel gives more weight to nearby samples and produces a more detailed (but potentially noisier) estimate, while a wider kernel averages over more distant samples and produces a smoother estimate.

To control the width, we use a **bandwidth parameter** $h > 0$ and define the **rescaled kernel** K_h by

$$K_h(y) = \frac{1}{h} K(y/h)$$

for all $y \in \mathbb{R}$. The bandwidth parameter h controls the width: small h gives a narrow kernel, large h gives a wide kernel.

Lemma 1.1. *If K is a kernel, then K_h is also a kernel.*

Proof. We need to verify the three properties from definition 1.1. Using the substitution $z = y/h$ we find

$$\begin{aligned}
\int_{-\infty}^{\infty} K_h(y)\, dy &= \int_{-\infty}^{\infty} \frac{1}{h} K(y/h)\, dy \\
&= \int_{-\infty}^{\infty} \frac{1}{h} K(z)\, h\, dz \\
&= \int_{-\infty}^{\infty} K(z)\, dz \\
&= 1.
\end{aligned}$$

For the second property, we have

$$K_h(-y) = \frac{1}{h} K(-y/h) = \frac{1}{h} K(y/h) = K_h(y),$$

where we used the symmetry of K. Finally, since $K(y/h) \geq 0$ for all y, we also have $K_h(y) = \frac{1}{h} K(y/h) \geq 0$ for all y. □

Remark. Note that the choice of kernel K and bandwidth h is not unique. For a given kernel shape, different authors use different normalisations. For example, we could replace our triangular kernel K from equation (2) with $\tilde{K}(x) = K(x/\sqrt{6})/\sqrt{6}$, which has variance 1. If we simultaneously replace h with $\tilde{h} = h/\sqrt{6}$, then the rescaled kernel remains unchanged: $\tilde{K}_{\tilde{h}} = K_h$.

1.4 Kernel Density Estimation

We can now define the kernel density estimate.

Definition 1.2. For a kernel K, bandwidth $h > 0$ and $x \in \mathbb{R}$, the **kernel density estimate** for $f(x)$ is given by

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

where K_h is the rescaled kernel defined above.

Note the similarity to the histogram formula (1): both are averages of weight functions applied to the data. The key difference is that the kernel $K_h(x - x_i)$ is centred around each data point x_i, whereas the indicator function $I_{[a,b]}(x_i)$ is fixed at the bucket location $[a, b]$.

Similar to the bucket width in histograms, the bandwidth parameter h controls how smooth the density estimate $\hat{f}_h$ is: small h gives narrow kernels that put weight only on very nearby samples, while large h gives wide kernels that average over more distant samples.

Later sections will discuss how to choose the kernel K for good properties.

Remark. If the kernel K is continuous, then the rescaled kernel K_h and the kernel density estimate $\hat{f}_h$ are also continuous. This is a useful property, as it means that kernel density estimates avoid the discontinuities that histograms exhibit at bucket boundaries.

1.5 Kernel Density Estimation in R

Kernel density estimates can be computed in R using the built-in `density()` function. If `x` is a vector containing the data, then `density(x)` computes a basic kernel density estimate, using the Gaussian kernel. The function has a number of optional arguments, which can be used to control details of the estimate:

- `bw = ...` can be used to control the bandwidth h. If no numeric value is given, a heuristic is used. R standardizes all kernels to have variance 1, so the `bw` parameter differs from our bandwidth h by a kernel-dependent constant factor (see the remark after Lemma 1.1). Specifically, `bw=1` corresponds to the case where the rescaled kernel K_h has variance 1.
- `kernel = ...` can be used to choose the kernel. Choices include `"rectangular"` (the uniform kernel), `"triangular"`, `"epanechnikov"` and `"gaussian"`. The default value is `"gaussian"`.

See `help(density)` for details.

To illustrate the use of `density()`, we use a dataset about the annual amount of snow falling in Buffalo, New York for different years:

```
# downloaded from https://teaching.seehuhn.de/data/buffalo/
x <- read.csv("data/buffalo.csv")
snowfall <- x$snowfall
```

The return value of `density` is an R object which contains information about the kernel density estimate.

```
m <- density(snowfall)
str(m)
```

```
## List of 8
##  $ x         : num [1:512] -4.17 -3.72 -3.26 -2.81 -2.35 ...
##  $ y         : num [1:512] 4.28e-06 4.94e-06 5.68e-06 6.50e-06 7.42e-06 ...
##  $ bw        : num 9.72
##  $ n         : int 109
##  $ old.coords: logi FALSE
##  $ call      : language density.default(x = snowfall)
##  $ data.name : chr "snowfall"
##  $ has.na    : logi FALSE
##  - attr(*, "class")= chr "density"
```

The fields `$x` and `$y` contain the x and y coordinates, respectively, of points on the $x \mapsto \hat{f}_h(x)$ curve, which approximates f. The field `$bw` shows the numeric value for the bandwidth chosen by the heuristic. The returned object can also directly be used as an argument to `plot()` and `lines()`, to add the graph of $\hat{f}_h$ to a plot.

The following code illustrates the use of `density()` with different bandwidth parameters. The result is shown in figure 4.

```
par(mfrow = c(2,2))

for (bw in c(1, 2, 4, 8)) {
  plot(density(snowfall, bw = bw, kernel = "triangular", n = 1000),
    xlim = c(25,200),
    ylim = c(0, 0.025),
    xlab = paste("bandwidth =", bw),
    main = NA)
}
```

Summary

- Histograms approximate densities using indicator functions on fixed buckets.
- Kernel density estimates replace fixed buckets with smooth kernels centred on each data point.
- The bandwidth parameter controls the smoothness of the kernel density estimate.
- In R, the `density()` function computes kernel density estimates.

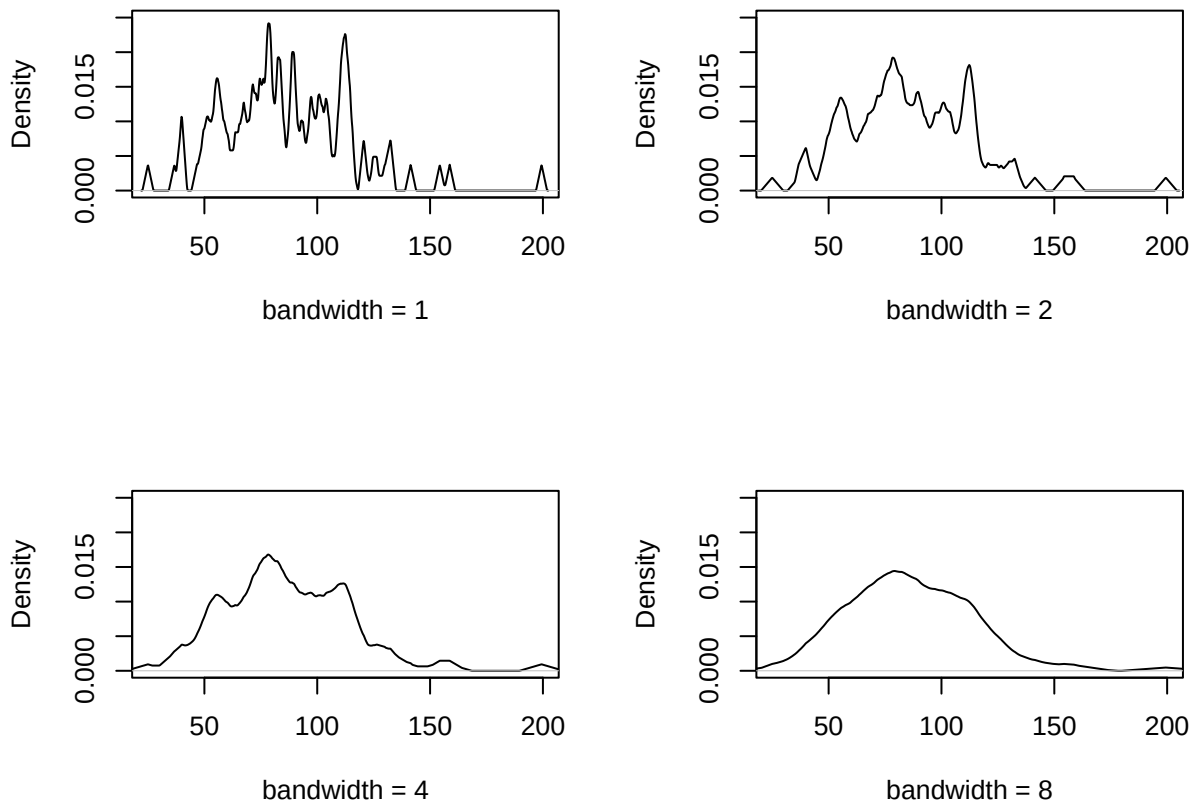


Figure 4: This figure illustrates the effect of the bandwidth parameter on the kernel density estimate. Small bandwidths give a more detailed estimate, but also show more noise. Large bandwidths give a smoother estimate, but may miss local structure of the density.

Interlude: Vectorisation in R

In this section we explore techniques for writing efficient R code, using kernel density estimation as an example. We start with a straightforward implementation using for-loops, then show how to speed it up by replacing loops with operations on entire vectors and matrices at once. These techniques apply to many computational problems in statistics.

For our experiments we will use the same "snowfall" dataset as in the previous section.

```
# downloaded from https://teaching.seehuhn.de/data/buffalo/
x <- read.csv("data/buffalo.csv")
snowfall <- x$snowfall
```

Direct Implementation

The kernel density estimate for given data $x_1, \dots, x_n \in \mathbb{R}$ is

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

Our aim is to implement this formula in R.

We use the triangular kernel from equation (2):

```
K <- function(x) pmax(1 - abs(x), 0)
```

The use of `pmax()` allows us to evaluate `K()` for several x-values in parallel:

```
K(c(-1, -0.5, 0, 0.5, 1))
```

```
## [1] 0.0 0.5 1.0 0.5 0.0
```

With this in place, we can now easily compute the kernel density estimate. As mentioned in the previous section, the choice of kernel normalization and bandwidth is not unique: rescaling the kernel and adjusting the bandwidth can give the same K_h. The R function `density()` uses a specific convention: the `bw` parameter is defined to be the standard deviation of the smoothing kernel. For the triangular kernel, this means that `density()` uses a rescaled version with variance 1, which differs from our kernel in equation (2) by a factor of $1/\sqrt{6}$. Thus, to get output comparable to `bw=4` (bottom left panel in figure 4), we use $h = 4\sqrt{6}$:

```
h <- 4 * sqrt(6) # bandwidth
x <- 100 # the point at which to estimate f
mean(1/h * K((x - snowfall) / h))
```

```
## [1] 0.0108237
```

To plot the estimated density, we typically evaluate $\hat{f}_h$ on a grid of x-values. Since $x_1, \dots, x_n$ denotes the input data, we use $\tilde{x}_1, \dots, \tilde{x}_m$ for the evaluation points:

```
x.tilde <- seq(25, 200, by = 1)
f.hat <- numeric(length(x.tilde)) # empty vector to store the results
for (j in 1:length(x.tilde)) {
  f.hat[j] <- mean(1/h * K((x.tilde[j] - snowfall) / h))
}
```

Figure 5 shows the result is similar to figure 4.

```
plot(x.tilde, f.hat, type = "l",
     xlab = "snowfall", ylab = "density",
     ylim = c(0, 0.025))
```

This code is slower than `density()` (as we will see below), but gives us full control over the computation.

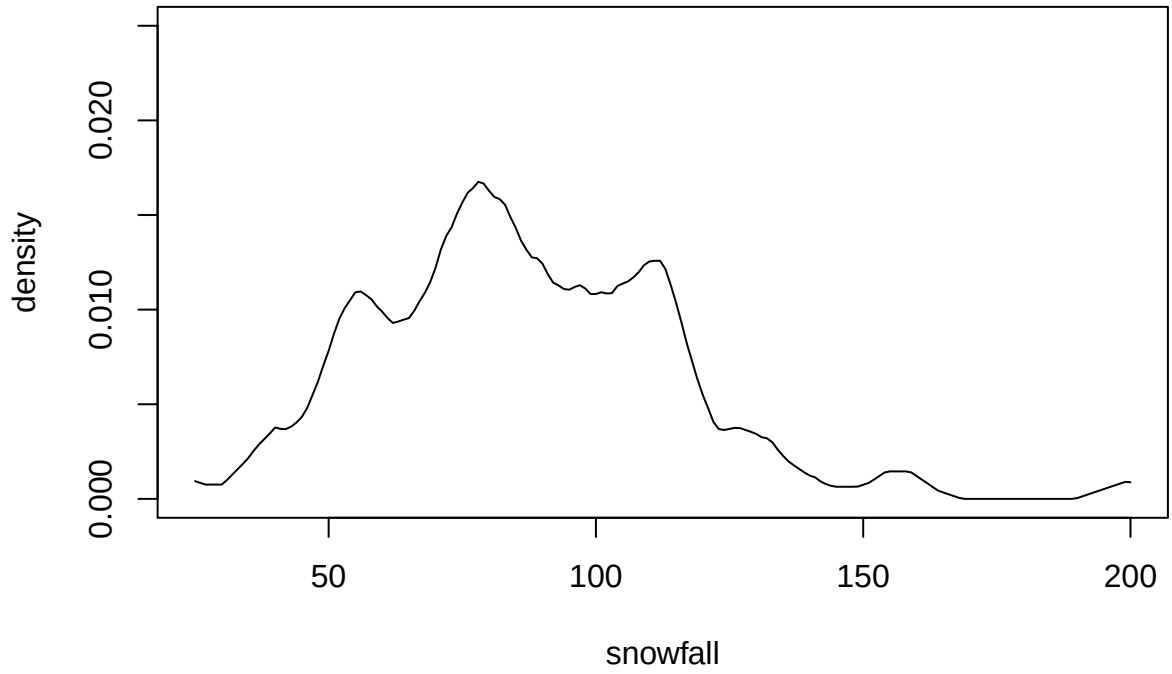


Figure 5: Manually computed kernel density estimate for the snowfall data, corresponding to `bw=4` in `density()`.

Vectorization

We now explore how to speed up our implementation. First, we encapsulate the code above into a function:

```
KDE1 <- function(x.tilde, x, K, h) {
  f.hat <- numeric(length(x.tilde))
  for (j in 1:length(x.tilde)) {
    f.hat[j] <- mean(1/h * K((x.tilde[j] - x) / h))
  }
  f.hat
}
```

A common source of inefficiency in R is the use of loops. The `for`-loop in `KDE1()` computes differences $\tilde{x}_j - x_i$ for all j. We have already avoided a second loop by treating `x.tilde[j] - x` as a vector operation, using `pmax()` in `K`, and using `mean()` for the sum. This approach of replacing explicit loops with operations on entire vectors is called **vectorisation**.

To eliminate the loop over `j`, we use `outer()`, which applies a function to all pairs of elements from two vectors. The result is a matrix with rows corresponding to the first vector and columns to the second. For example:

```
x <- c(1, 2, 3)
y <- c(1, 2)
outer(x, y, "-")
```

```
##      [,1] [,2]
## [1,]    0   -1
## [2,]    1    0
## [3,]    2    1
```

We use `outer(x.tilde, x, "-")` to compute all $\tilde{x}_j - x_i$ at once, giving an $m \times n$ matrix to which we apply `K()`. This fully vectorised implementation avoids all explicit loops:

```
KDE2 <- function(x.tilde, x, K, h) {
  dx <- outer(x.tilde, x, "-")
```

```
  rowMeans(1/h * K(dx / h))
}
```

We can verify that both functions return the same result:

```
KDE1(c(50, 100, 150), snowfall, K, h)
```

```
## [1] 0.0078239091 0.0108236983 0.0007448277
```

```
KDE2(c(50, 100, 150), snowfall, K, h)
```

```
## [1] 0.0078239091 0.0108236983 0.0007448277
```

There is another, minor optimisation we can make. Instead of dividing the $m \times n$ matrix elements of `dx` by `h`, we can achieve the same effect by dividing the vectors `x` and `x.tilde`, *i.e.* only $m + n$ numbers, by h. This reduces the number of operations required. Similarly, instead of multiplying the $m \times n$ kernel values by $1/h$, we can divide the m row means by h:

```
KDE3 <- function(x.tilde, x, K, h) {
  dx <- outer(x.tilde / h, x / h, "-")
  rowMeans(K(dx)) / h
}
```

Again, `KDE3()` returns the same result:

```
KDE3(c(50, 100, 150), snowfall, K, h)
```

```
## [1] 0.0078239091 0.0108236983 0.0007448277
```

We measure execution times using `microbenchmark`:

```
library("microbenchmark")

microbenchmark(
  KDE1 = KDE1(x.tilde, snowfall, K, h),
  KDE2 = KDE2(x.tilde, snowfall, K, h),
  KDE3 = KDE3(x.tilde, snowfall, K, h),
  times = 1000)
```

```
## Unit: microseconds
##  expr      min       lq      mean   median       uq      max neval
##  KDE1 1429.998 1445.291 1480.8799 1451.400 1459.928 4175.440  1000
##  KDE2  192.946  194.463  254.6251  195.078  196.103 5123.360  1000
##  KDE3  173.184  174.660  210.5374  175.275  176.177 5141.564  1000
```

The output shows execution times for 1000 runs. The introduction of `outer()` in `KDE2()` gives a substantial speedup, and `KDE3()` improves further.

Finally, we compare to `density()`, using arguments `from`, `to`, and `n` to specify the same grid $\tilde{x}$:

```
KDE.builtin <- function(x.tilde, x, kernel_name, h) {
    density(x,
      kernel = kernel_name, bw = h / sqrt(6),
      from = min(x.tilde), to = max(x.tilde), n = length(x.tilde))$y
}
microbenchmark(density = KDE.builtin(x.tilde, snowfall, "triangular", h), times = 1000)
```

```
## Unit: microseconds
##     expr     min      lq     mean  median       uq      max neval
##  density 155.185 159.449 185.5912 167.731 180.2565 8014.762  1000
```

The result shows that `density()` is faster than `KDE3()`, but by a factor of less than 2. The reason is that `density()` uses a more efficient algorithm based on Fast Fourier Transform to compute an approximation to the kernel density estimate. By comparing the outputs of `KDE3()` and `KDE.builtin()` we can see that the approximation is very accurate:

```r
f.hat1 <- KDE3(x.tilde, snowfall, K, h)
f.hat2 <- KDE.builtin(x.tilde, snowfall, "triangular", h)
max(abs(f.hat1 - f.hat2))
```

```
## [1] 1.935582e-05
```

Summary

- Vectorisation is the technique of replacing explicit loops with operations on entire vectors.
- Vectorised code dramatically improves performance in R.
- The `outer()` function enables fully vectorised implementations for pairwise operations.
- Further speedups can be achieved by reducing the number of operations on large matrices.

2 The Error of Kernel Density Estimates

In the previous section we introduced the kernel density estimate

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) \tag{3}$$

for estimating the density f, and we argued that $\hat{f}_h(x) \approx f(x)$. The aim of the current section is to quantify the error of this approximation and to understand how this error depends on the true density f and on the bandwidth $h > 0$.

2.1 A Statistical Model

As usual, we will make a statistical model for the data $x_1, \dots, x_n$, and then use this model to analyse how well the estimator performs. The statistical model we will consider here is extremely simple: we model the x_i using random variables

$$X_1, \dots, X_n \sim f, \tag{4}$$

which we assume to be independent and identically distributed (i.i.d.). Here, the notation $X \sim f$, where f is a probability density, simply denotes that the random variable X has density f.

It is important to not confuse x (the point where we are evaluating the densities during our analysis) with the data x_i. A statistical model describes the data, so here we get random variables $X_1, \dots, X_n$ to describe the behaviour of $x_1, \dots, x_n$, but it does not describe x. The number x is not part of the data, so will never be modelled by a random variable.

While the model is very simple, for example it is much simpler than the model we use in the level 3 part of the module for linear regression, the associated parameter estimation problem is more challenging. The only "parameter" in this model is the function $f\colon \mathbb{R} \to \mathbb{R}$ instead of just a vector of numbers. The space of all possible density functions f is infinite dimensional, so this is a more challenging estimation problem than the one we consider, for example, for linear regression. Since f is not a "parameter" in the usual sense, sometimes this problem is called a "non-parametric" estimation problem.

Our estimate for the density f is the function $\hat{f}_h\colon \mathbb{R} \to \mathbb{R}$, where $\hat{f}_h(x)$ is given by (3) for every $x \in \mathbb{R}$.

2.2 Moments of Kernels

For our analysis we will need to consider properties of K and K_h in more detail.

Definition 2.1. The kth **moment** of a kernel K, for $k \in \mathbb{N}_0 = \{0, 1, 2, ...\}$, is given by

$$\mu_k(K) = \int_{-\infty}^{\infty} x^k K(x)\, dx.$$

The second moment μ_2 is sometimes also called the **variance** of the kernel K.

Using the properties of K, we find the following results:

- Since $x^0 = 1$ for all $x \in \mathbb{R}$, the 0th moment is $\mu_0(K) = \int_{-\infty}^{\infty} K(x)\, dx = 1$ for every kernel K.
- Since K is symmetric, the function $x \mapsto xK(x)$ is antisymmetric and we have

  $$\mu_1(K) = \int_{-\infty}^{\infty} xK(x)\, dx = 0$$

  for every kernel K.

The moments of the rescaled kernel K_h can be computed from the moments of K.

Lemma 2.1. *Let K be a kernel, $k \in \mathbb{N}_0$ and $h > 0$. Then*

$$\mu_k(K_h) = h^k \mu_k(K).$$

Proof. We have

$$\begin{aligned}\mu_k(K_h) &= \int_{-\infty}^{\infty} x^k K_h(x)\,dx \\ &= \int_{-\infty}^{\infty} x^k \frac{1}{h} K\Big(\frac{x}{h}\Big)\,dx.\end{aligned}$$

Using the substitution $y = x/h$ we find

$$\begin{aligned}\mu_k(K_h) &= \int_{-\infty}^{\infty} (hy)^k \frac{1}{h} K(y)\,h\,dy \\ &= h^k \int_{-\infty}^{\infty} y^k K(y)\,dy \\ &= h^k \mu_k(K).\end{aligned}$$

This completes the proof. □

By lemma 1.1, if K is a kernel, then K_h is also a kernel which implies that K_h is a probability density. If Y is a random variable with density K_h, written as $Y \sim K_h$ in short, then we find

$$\mathbb{E}(Y) = \int y K_h(y)\,dy = \mu_1(K_h) = 0$$

and

$$\mathrm{Var}(Y) = \mathbb{E}(Y^2) = \int y^2 K_h(y)\,dy = \mu_2(K_h) = h^2\,\mu_2(K). \tag{5}$$

Thus, Y is centred and the variance of Y is proportional to h^2.

2.3 The Roughness of Kernels

In addition to the moments of a kernel, we will need one more kernel property for our analysis.

Definition 2.2. The **roughness** of a kernel K is given by

$$R(K) := \int_{-\infty}^{\infty} K(x)^2\,dx.$$

The roughness measures how concentrated the kernel is: a kernel with higher roughness has more mass concentrated in a smaller region, while a kernel with lower roughness is more spread out.

Similar to the moments, the roughness of the rescaled kernel K_h can be expressed in terms of the roughness of K.

Lemma 2.2. *Let K be a kernel and $h > 0$. Then*

$$R(K_h) = \frac{1}{h} R(K).$$

Proof. We have

$$\begin{aligned}R(K_h) &= \int_{-\infty}^{\infty} K_h(x)^2\,dx \\ &= \int_{-\infty}^{\infty} \Big(\frac{1}{h} K\Big(\frac{x}{h}\Big)\Big)^2\,dx \\ &= \int_{-\infty}^{\infty} \frac{1}{h^2} K\Big(\frac{x}{h}\Big)^2\,dx.\end{aligned}$$

Using the substitution $y = x/h$ we find

$$\begin{aligned} R(K_h) &= \int_{-\infty}^{\infty} \frac{1}{h^2} K(y)^2 \, h \, dy \\ &= \frac{1}{h} \int_{-\infty}^{\infty} K(y)^2 \, dy \\ &= \frac{1}{h} R(K). \end{aligned}$$

This completes the proof. □

2.4 Mean Squared Error

In this section we will quantify the error of the kernel density estimate $\hat{f}_h(x)$ for estimating $f(x)$ by computing its mean squared error. The mean squared error combines two sources of error: bias (systematic error) and variance (random error due to sampling). We will see how the bandwidth h affects both of these components, and how they must be balanced to minimise the overall error.

2.4.1 Result

The following theorem characterises the mean squared error of the kernel density estimate for small bandwidth h.

Theorem 2.1. *Let $X_1, \dots, X_n$ be i.i.d. random variables with density f, and let $\hat{f}_h(x)$ be the kernel density estimate defined in (3). Then, as $h \downarrow 0$, the mean squared error of $\hat{f}_h(x)$ is given by*

$$\mathrm{MSE}(\hat{f}_h(x)) = \frac{1}{nh} f(x) R(K) + \frac{1}{4} \mu_2(K)^2 f''(x)^2 h^4 + o(1/nh) + o(h^4), \tag{6}$$

where $R(K) = \int K(x)^2 \, dx$ is the roughness of the kernel and $\mu_2(K) = \int x^2 K(x) \, dx$ is the second moment of the kernel.

Here we use little-o notation: the term $o(1/nh)$ represents error terms of the form $e_1(h)/n$ where $e_1(h) = o(1/h)$ as $h \downarrow 0$, meaning that $e_1(h)/(1/h) \to 0$, and the constants in the definition of $o(1/h)$ do not depend on n. Similarly, $o(h^4)$ represents error terms $e_2(h)$ which do not depend on n and satisfy $e_2(h)/h^4 \to 0$ as $h \downarrow 0$. Thus $o(1/nh)$ really means "$o(1/h)$, where the constants are proportional to $1/n$' ', and the constants in the definition of $o(h^4)$ do not depend on n.

The theorem shows that the MSE has two main terms with opposing behaviour: the first term $\frac{1}{nh} f(x) R(K)$ comes from the variance of the estimate and decreases as h increases, while the second term $\frac{1}{4} \mu_2(K)^2 f''(x)^2 h^4$ comes from the squared bias and increases as h increases. These competing effects lead to an optimal choice of bandwidth that balances bias and variance.

We will not give a complete formal proof of this theorem, but we will derive the main components in the following subsections. First we compute the bias of the estimate and show that it is approximately $\frac{\mu_2(K) f''(x)}{2} h^2$ for small h. Then we compute the variance and show that it is approximately $\frac{f(x) R(K)}{nh}$ for small h. Combining these results using the standard identity $\mathrm{MSE} = \mathrm{Var} + \mathrm{bias}^2$ yields the theorem.

2.4.2 Bias

2.4.2.1 The Bias of the Estimate As usual, the **bias** of our estimate is the difference between what the estimator gives on average and the truth. For our estimation problem we get

$$\mathrm{bias}(\hat{f}_h(x)) = \mathbb{E}(\hat{f}_h(x)) - f(x).$$

The expectation on the right-hand side averages over the randomness in the data, by using $X_1, \dots, X_n$ from the model in place of the data.

Substituting in the definition of $\hat{f}_h(x)$ from equation (3) we find

$$\begin{aligned}\mathbb{E}(\hat{f}_h(x)) &= \mathbb{E}\Big(\frac{1}{n}\sum_{i=1}^{n} K_h(x - X_i)\Big) \\ &= \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}(K_h(x - X_i))\end{aligned}$$

and since the X_i are identically distributed, we can replace all X_i with X_1 (or any other of them) to get

$$\begin{aligned}\mathbb{E}(\hat{f}_h(x)) &= \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}(K_h(x - X_1)) \\ &= \frac{1}{n} n\, \mathbb{E}(K_h(x - X_1)) \\ &= \mathbb{E}(K_h(x - X_1)).\end{aligned}$$

Since the model assumes X_1 (and all the other X_i) to have density f, we can write this expectation as an integral to get

$$\begin{aligned}\mathbb{E}(\hat{f}_h(x)) &= \int_{-\infty}^{\infty} K_h(x - y)\, f(y)\, dy \\ &= \int_{-\infty}^{\infty} f(y)\, K_h(y - x)\, dy \\ &= \int_{-\infty}^{\infty} f(z + x)\, K_h(z)\, dz\end{aligned}$$

where we used the symmetry of K_h and the substitution $z = y - x$.

2.4.2.2 The Bias for Small Bandwidth Considering again the formula

$$\mathbb{E}(\hat{f}_h(x)) = \int_{-\infty}^{\infty} f(x + y)\, K_h(y)\, dy,$$

we see that we can interpret this integral as an expectation with respect to a random variable $Y \sim K_h$:

$$\mathbb{E}(\hat{f}_h(x)) = \mathbb{E}(f(x + Y)). \tag{7}$$

Equation (5) shows that for $h \downarrow 0$ the random variable concentrates more and more around 0 and thus $x + Y$ concentrates more and more around x. For this reason we expect $\mathbb{E}(\hat{f}_h(x)) \approx f(x)$ for small h.

To get a more quantitative version of this argument, we consider the Taylor approximation of f around the point x:

$$f(x + y) \approx f(x) + y f'(x) + \frac{y^2}{2} f''(x).$$

Substituting this into equation (7) we find

$$\begin{aligned}\mathbb{E}(\hat{f}_h(x)) &\approx \mathbb{E}\Big(f(x) + Y f'(x) + \frac{Y^2}{2} f''(x)\Big) \\ &= f(x) + \mathbb{E}(Y) f'(x) + \frac{1}{2}\mathbb{E}(Y^2) f''(x) \\ &= f(x) + \frac{1}{2} h^2 \mu_2(K) f''(x)\end{aligned}$$

for small h. Considering the bias again, this gives

$$\operatorname{bias}(\hat{f}_h(x)) = \mathbb{E}(\hat{f}_h(x)) - f(x) \approx \frac{\mu_2(K) f''(x)}{2} h^2 \tag{8}$$

which shows that the bias of the estimator decreases quadratically as h gets smaller.

2.4.3 Variance

We now turn to computing the variance of the estimator. Recall from the bias analysis above that

$$\mathbb{E}(\hat{f}_h(x)) = \mathbb{E}(K_h(x - X_i)) \tag{9}$$

for all $i \in \{1, \dots, n\}$. We will use similar arguments to derive the variance.

We use the formula

$$\mathrm{Var}(\hat{f}_h(x)) = \mathbb{E}(\hat{f}_h(x)^2) - \mathbb{E}(\hat{f}_h(x))^2$$

and consider the two terms separately.

2.4.3.1 Second Moment For the second moment term in the definition of the variance we get

$$\begin{aligned}
\mathbb{E}(\hat{f}_h(x)^2) &= \mathbb{E}\Big(\frac{1}{n}\sum_{i=1}^n K_h(x - X_i)\frac{1}{n}\sum_{j=1}^n K_h(x - X_j)\Big) \\
&= \frac{1}{n^2}\mathbb{E}\Big(\sum_{i,j=1}^n K_h(x - X_i)K_h(x - X_j)\Big) \\
&= \frac{1}{n^2}\sum_{i,j=1}^n \mathbb{E}\Big(K_h(x - X_i)K_h(x - X_j)\Big)
\end{aligned}$$

Since the X_i are independent, the values of i and j in this sum do not matter. For the n terms where $i = j$ we can assume that both indices equal 1, and for the $n(n-1)$ terms where $i \neq j$ we can assume $i = 1$ and $j = 2$, without changing the value of the expectation. So we get

$$\begin{aligned}
\mathbb{E}(\hat{f}_h(x)^2) &= \frac{1}{n^2}\Big(n\mathbb{E}(K_h(x - X_1)^2\Big) + n(n-1)\mathbb{E}(K_h(x - X_1)K_h(x - X_2))\Big) \\
&= \frac{1}{n^2}\Big(n\mathbb{E}(K_h(x - X_1)^2\Big) + n(n-1)\mathbb{E}(K_h(x - X_1))\mathbb{E}(K_h(x - X_2))\Big) \\
&= \frac{1}{n^2}\Big(n\mathbb{E}(K_h(x - X_1)^2\Big) + n(n-1)\mathbb{E}(K_h(x - X_1))^2\Big) \\
&= \frac{1}{n}\mathbb{E}\Big(K_h(x - X_1)^2\Big) + \frac{n-1}{n}\mathbb{E}(\hat{f}_h(x))^2,
\end{aligned}$$

where we used equation (9) for the last term. Using this equation, we can write the variance as

$$\begin{aligned}
\mathrm{Var}(\hat{f}_h(x)) &= \mathbb{E}(\hat{f}_h(x)^2) - \mathbb{E}(\hat{f}_h(x))^2 \\
&= \frac{1}{n}\mathbb{E}(K_h(x - X_1)^2) + \Big(\frac{n-1}{n} - 1\Big)\mathbb{E}(\hat{f}_h(x))^2.
\end{aligned}$$

Since $(n-1)/n - 1 = -1/n$ we get

$$\mathrm{Var}(\hat{f}_h(x)) = \frac{1}{n}\Big(\mathbb{E}(K_h(x - X_1)^2) - \mathbb{E}(\hat{f}_h(x))^2\Big). \tag{10}$$

2.4.3.2 Approximating $\mathbb{E}(K_h(x - X_1)^2)$ Similar to the bias analysis, we will use Taylor expansion of f around the point x to understand the behaviour of the variance for small h. Some more care is needed here, because this time the result also depends on the sample size n and we will consider the joint limit of $n \to \infty$ and $h \to 0$. For the first term in equation (10) we find

$$\begin{aligned}
\mathbb{E}(K_h(x - X_1)^2) &= \int K_h(x - y)^2 f(y)\, dy \\
&= \int K_h(y - x)^2 f(y)\, dy \\
&= \int \frac{1}{h^2} K\Big(\frac{y - x}{h}\Big)^2 f(y)\, dy.
\end{aligned}$$

Now we use the substitution $z = (y - x)/h$. This gives

$$\mathbb{E}(K_h(x - X_1)^2) = \int \frac{1}{h^2} K(z)^2 f(x + hz)\, h\, dz$$

Finally, we use Taylor approximation to get

$$\begin{aligned}\mathbb{E}(K_h(x-X_1)^2) &\approx \int \frac{1}{h}K(z)^2\Big(f(x)+hz\,f'(x)+\frac{1}{2}h^2z^2\,f''(x)\Big)\,dz \\ &= \frac{1}{h}f(x)\int K(z)^2\,dz + f'(x)\int zK(z)^2\,dz + \frac{1}{2}hf''(x)\int z^2K(z)^2\,dz \\ &= \frac{1}{h}f(x)\int K(z)^2\,dz + \frac{1}{2}hf''(x)\int z^2K(z)^2\,dz\end{aligned}$$

where the first-order term disappears since $z \mapsto zK(z)^2$ is an antisymmetric function. Using the definition of the roughness of a kernel, $R(K) = \int K(x)^2\,dx$, this gives the result

$$\mathbb{E}(K_h(x-X_1)^2) \approx \frac{1}{h}f(x)R(K) + \frac{1}{2}hf''(x)\int z^2K(z)^2\,dz \tag{11}$$

for small h.

2.4.3.3 The Variance for Small Bandwidth Here we consider the term $\mathbb{E}(\hat{f}_h(x))^2$ in the formula for the variance. From the bias analysis above we know

$$\mathbb{E}(\hat{f}_h(x)) \approx f(x) + \frac{1}{2}h^2\mu_2(K)f''(x)$$

and thus we get

$$\mathbb{E}(\hat{f}_h(x))^2 \approx f(x)^2 + h^2\mu_2(K)f(x)f''(x) + \frac{1}{4}h^4\mu_2(K)^2f''(x)^2 \tag{12}$$

for small h.

Substituting equations (11) and (12) into equation (10) we finally find

$$\mathrm{Var}(\hat{f}_h(x)) = \frac{1}{n}\Big(\frac{1}{h}f(x)R(K) - f(x)^2 + \cdots\Big),$$

where all the omitted terms go to zero as $h \downarrow 0$. Omitting one more term and keeping only the leading term we find

$$\mathrm{Var}(\hat{f}_h(x)) \approx \frac{1}{nh}f(x)R(K) \tag{13}$$

as $h \downarrow 0$.

2.4.4 Result

Recall that the mean squared error can be decomposed as

$$\mathrm{MSE}(\hat{f}_h(x)) = \mathbb{E}\Big((\hat{f}_h(x)-f(x))^2\Big) = \mathrm{Var}(\hat{f}_h(x)) + \mathrm{bias}(\hat{f}_h(x))^2.$$

Substituting equations (8) and (13) into the formula for the MSE, we obtain the main two terms in theorem 2.1:

$$\mathrm{MSE}(\hat{f}_h(x)) \approx \frac{1}{nh}f(x)R(K) + \frac{1}{4}\mu_2(K)^2f''(x)^2h^4.$$

The additional error terms $o(1/nh)$ and $o(h^4)$ in the theorem statement account for the higher-order terms neglected in the Taylor approximations used to derive the bias and variance.

These two main terms have opposing behaviour: the variance term $\frac{1}{nh}f(x)R(K)$ *increases* as $h \downarrow 0$, whilst the squared bias term $\frac{1}{4}\mu_2(K)^2f''(x)^2h^4$ *decreases* as $h \downarrow 0$. We will need to balance these two effects to find the optimal value of h.

2.5 Optimal Bandwidth for Pointwise MSE

The two terms on the right-hand side of (6) are balanced in that the first term decreases for large h while the second term decreases for small h. By taking derivatives with respect to h, we can find the optimal value of h. Ignoring the higher order terms, we get

$$\frac{\partial}{\partial h}\,\mathrm{MSE}(\hat{f}_h(x)) = -\frac{1}{nh^2}f(x)R(K) + \mu_2(K)^2f''(x)^2h^3$$

and thus the derivative equals zero, if

$$\frac{1}{nh^2} f(x)R(K) = \mu_2(K)^2 f''(x)^2 h^3$$

or, equivalently,

$$h = h_{\text{opt}} := \Big(\frac{f(x)R(K)}{n\mu_2(K)^2 f''(x)^2}\Big)^{1/5}.$$

It is easy to check that this h corresponds to the minimum of the MSE. This shows how the optimal bandwidth depends both on the kernel and on the target density f. In practice, this formula is hard to use, since f'' is unknown. (We don't even know f!)

Substituting the optimal value of h back into equation (6), we get

$$\begin{aligned}\text{MSE}_{\text{opt}} &= \frac{1}{n} f(x)R(K)\Big(\frac{n\mu_2(K)^2 f''(x)^2}{f(x)R(K)}\Big)^{1/5} + \frac{1}{4}\mu_2(K)^2 f''(x)^2\Big(\frac{f(x)R(K)}{n\mu_2(K)^2 f''(x)^2}\Big)^{4/5} \\ &= \frac{5}{4}\,\frac{1}{n^{4/5}}\,\Big(R(K)^2\mu_2(K)\Big)^{2/5}\,\Big(f(x)^2|f''(x)|\Big)^{2/5}.\end{aligned}$$

This expression clearly shows the contribution of n, K and f:

- If the bandwidth is chosen optimally, as n increases the bandwidth h decreases proportionally to $1/n^{1/5}$ and the MSE decreases proportionally to $1/n^{4/5}$. For comparison, in a Monte Carlo estimate for an expectation, the MSE decreases proportionally to $1/n$. The error in kernel density estimation decreases slightly slower than for Monte Carlo estimates.
- The error is proportional to $\big(R(K)^2\mu_2(K)\big)^{2/5}$. Thus we should use kernels where the value $R(K)^2\mu_2(K)$ is small.
- The error is proportional to $\big(f(x)^2|f''(x)|\big)^{2/5}$. We cannot influence this term, but we can see that x where f is large or has high curvature have higher estimation error.

2.6 Integrated Error

Equation (6) gives the mean squared error when trying to estimate the density $f(x)$ at a fixed point x. Usually we are interested in estimating the function f rather than individual points $f(x)$. In this case, we consider the **integrated mean squared error (IMSE)**:

$$\text{IMSE}(\hat{f}_h) := \int_{-\infty}^{\infty} \text{MSE}(\hat{f}_h(x))\,dx.$$

Using our result from above we find

$$\begin{aligned}\text{IMSE}(\hat{f}_h) &\approx \int \Big(\frac{1}{nh} f(x)R(K) + \frac{1}{4}\mu_2(K)^2 f''(x)^2 h^4\Big)\,dx \\ &= \frac{1}{nh} R(K)\int f(x)\,dx + \frac{h^4}{4}\mu_2(K)^2\int f''(x)^2\,dx \\ &= \frac{1}{nh} R(K) + \frac{1}{4}\mu_2(K)^2 R(f'')h^4,\end{aligned}$$

where we (mis-)use the definition of roughness as an abbreviation to express the integral over f''.

As in the pointwise case above, we can use differentiation to find the optimal value of h. Here we get

$$h_{\text{opt}} = \Big(\frac{R(K)}{n\mu_2(K)^2 R(f'')}\Big)^{1/5}.$$

and the corresponding error is

$$\text{IMSE}_{\text{opt}} = \frac{5}{4}\,\frac{1}{n^{4/5}}\,\Big(R(K)^2\mu_2(K)\Big)^{2/5}\,R(f'')^{1/5}. \tag{14}$$

Thus, in order to minimise the error we still need to choose $h \propto n^{-1/5}$ and we should choose a kernel K which minimises the value $R(K)^2\mu_2(K)$.

Summary

- We introduced a statistical model for density estimation where data are i.i.d. samples from the unknown density f.
- Kernel moments $\mu_k(K) = \int x^k K(x)dx$ characterise kernel properties and scale with bandwidth: $\mu_k(K_h) = h^k \mu_k(K)$.
- For small bandwidth h, the bias is approximately $\frac{\mu_2(K)f''(x)}{2}h^2$, decreasing quadratically as $h \to 0$.
- The variance for small h is approximately $\frac{f(x)R(K)}{nh}$, where $R(K) = \int K(x)^2 dx$ is the kernel roughness, and increases as $h \to 0$.
- The mean squared error (MSE) combines both bias and variance. The integrated mean squared error (IMSE) integrates the MSE over all points and provides a global measure of estimation quality.

3 Kernel Density Estimation in Practice

In this section we conclude our discussion of kernel density estimation by considering practical aspects of implementing the method. We discuss how to choose a kernel, how to select an appropriate bandwidth in practice, and how the method extends to higher dimensions.

3.1 Choice of Kernel

The integrated error in equation (14) is proportional to $\left(R(K)^2\mu_2(K)\right)^{2/5}$, and none of the remaining terms in the equation depends on the choice of the kernel. Thus, we can minimise the error by choosing a kernel which has minimal $R(K)^2\mu_2(K)$. For a given kernel, it is easy to work out the value of $R(K)^2\mu_2(K)$.

Example 3.1. For the uniform kernel we have

$$K(x) = \begin{cases} 1/2 & \text{if } -1 \le x \le 1 \\ 0 & \text{otherwise.} \end{cases}$$

From this we find

$$R(K) = \int_{-\infty}^{\infty} K(x)^2\,dx = \int_{-1}^{1} \frac{1}{4}\,dx = \frac{1}{2}$$

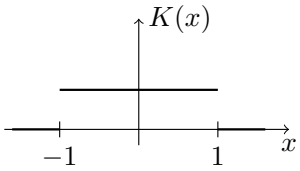
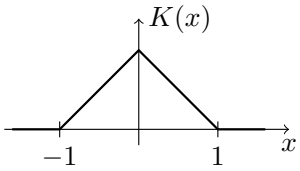
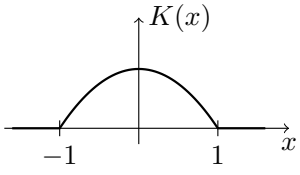
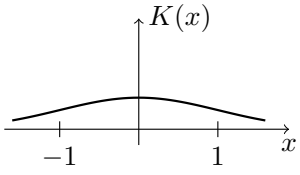
and

$$\mu_2(K) = \int_{-\infty}^{\infty} x^2K(x)\,dx = \int_{-1}^{1} \frac{1}{2}x^2\,dx = \frac{1}{6}x^3\Big|_{x=-1}^{1} = \frac{1}{6}(1-(-1)) = \frac{1}{3}.$$

Thus, for the uniform kernel we have

$$R(K)^2\mu_2(K) = \left(\frac{1}{2}\right)^2\frac{1}{3} = \frac{1}{12} \approx 0.083333.$$

Calculations similar to the ones in the example give the following values:

kernel		$\mu_2(K)$	$R(K)$	$R(K)^2\mu_2(K)$
Uniform		$\frac{1}{3}$	$\frac{1}{2}$	0.08333
Triangular		$\frac{1}{6}$	$\frac{2}{3}$	0.07407
Epanechnikov		$\frac{1}{5}$	$\frac{3}{5}$	0.07200
Gaussian		1	$\frac{1}{2\sqrt{\pi}}$	0.07958

The best value in the table is obtained for the Epanechnikov kernel, with $R(K)^2\mu_2(K) = 9/125 = 0.072$. One can show that this value is indeed optimal amongst all kernels. Since the difference in error for the kernels listed above is only a few percent, any of these kernels would be a reasonable choice.

3.2 Bandwidth Selection

Our formulas for the optimal bandwidth contain the terms $f(x)^2|f''(x)|$ for fixed x and $R(f'')$ for the integrated error. Since f is unknown, neither of these quantities are available and instead different rules of thumb are used in the literature. Here we present one possible choice of bandwidth estimator.

Suppose that f is a normal density, with mean μ and variance σ^2. Then we have

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-(x-\mu)^2/2\sigma^2).$$

Taking derivatives we get

$$f'(x) = -\frac{1}{\sqrt{2\pi\sigma^2}} \frac{x-\mu}{\sigma^2} \exp(-(x-\mu)^2/2\sigma^2)$$

and

$$f''(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \Big(\frac{(x-\mu)^2}{\sigma^4} - \frac{1}{\sigma^2} \Big) \exp(-(x-\mu)^2/2\sigma^2)$$

Patiently integrating the square of this function gives

$$R(f'') = \int_{-\infty}^{\infty} f''(x)^2 \, dx = \cdots = \frac{3}{8\sigma^5\sqrt{\pi}}.$$

This can be used as a simple "plug-in rule" with σ estimated by the sample standard deviation.

We now demonstrate how this rule of thumb could be used in R to obtain a kernel density estimate for the snowfall data. We will use the Epanechnikov kernel. For compatibility with the kernels built into R, we rescale this kernel, so that $\mu_2(K) = 1$, *i.e.* we consider $K_{\sqrt{5}}$ in place of K. An easy calculation shows that the roughness is then $R(K) = 3/(5\sqrt{5})$.

```
# downloaded from https://teaching.seehuhn.de/data/buffalo/
x <- read.csv("data/buffalo.csv")
snowfall <- x$snowfall
n <- length(snowfall)

# Roughness of the Epanechnikov kernel, after rescaling with h = sqrt(5)
# so that the second moment becomes mu_2 = 1:
R.K <- 3 / (5 * sqrt(5))

# Rule of thumb:
R.fpp <- 3 / (8 * sd(snowfall)^5 * sqrt(pi))

# formula for the optimal h
my.bw <- (R.K / (n * 1^2 * R.fpp))^0.2
my.bw
```

```
## [1] 11.58548
```

R has a variety of different builtin methods to estimate bandwidths. See `help("bw.nrd0")` for R's built-in bandwidth selectors. For comparison to our result, we list here the bandwidths suggested by some of R's algorithms:

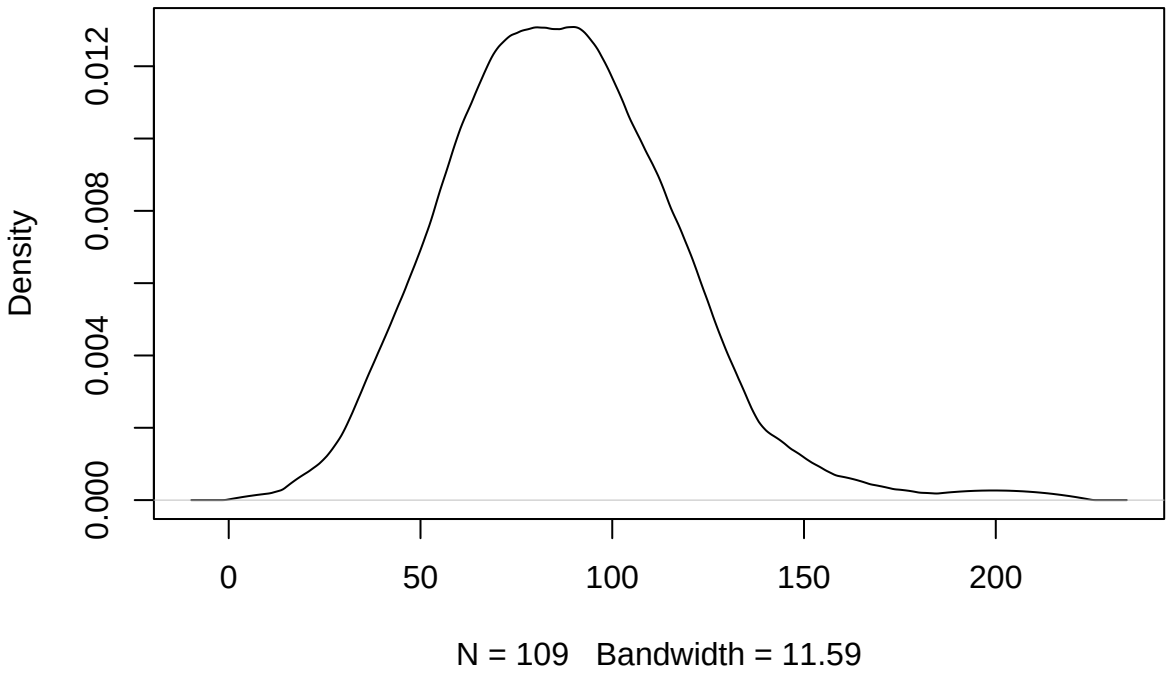
```
data.frame(
  name = c("nrd0", "nrd", "SJ"),
  bw = c(bw.nrd0(snowfall), bw.nrd(snowfall), bw.SJ(snowfall)))
```

```
##   name        bw
## 1 nrd0  9.724206
```

```
## 2  nrd 11.452953
## 3   SJ 11.903840
```

All of these values seem close the value we obtained manually. Using our bandwidth estimate, we get the following estimated density.

```
plot(density(snowfall, bw = my.bw, kernel = "epanechnikov"),
     main = NA)
```



In practice one would just use one of the built-in methods, for example using `bw="SJ"` instead of estimating the bandwidth manually.

3.3 Higher Dimensions

So far we have only considered the one-dimensional case, where the samples x_i are real numbers. In this subsection we will sketch how these methods will need to be adjusted for the multivariate case of $x_i = (x_{i,1}, \dots, x_{i,p}) \in \mathbb{R}^p$.

In this setup, a **kernel** is a function $K \colon \mathbb{R}^p \to \mathbb{R}$ (generalizing Definition 1.1) such that

- $\int \cdots \int K(x)\, dx_p \cdots dx_1 = 1$,
- $K(x) = K(-x)$ and
- $K(x) \geq 0$ for all $x \in \mathbb{R}^p$,

where the integral in the first condition is now over all p coordinates.

Example 3.2. If $K_1, \dots, K_p$ are one-dimensional kernels, then the product

$$K(x_1, \dots, x_p) := K_1(x_1) \cdots K_p(x_p)$$

is a kernel in p dimensions. If we use the product of p Gaussian kernels, we get

$$\begin{aligned} K(x) &= \prod_{i=1}^{p} \frac{1}{\sqrt{2\pi}} \exp(-x_i^2/2) \\ &= \frac{1}{(2\pi)^{p/2}} \exp\Big(-\frac{1}{2}(x_1^2 + \cdots + x_p^2)\Big). \end{aligned}$$

There are different possibilities for rescaling these kernels:

- If all coordinates live on "comparable scales" (*e.g.*, if they are measured in the same units), the formula
$$K_h(x) = \frac{1}{h^p} K(x/h)$$
for all $x \in \mathbb{R}^p$ can be used, where $h > 0$ is a bandwidth parameter as before. The scaling by $1/h^p$ is required to ensure that the integral of K_h equals 1, so that K_h is a kernel again.
- If different scaling is desirable for different components, the formula
$$K_h(x) = \frac{1}{h_1 \cdots h_p} K(x_1/h_1, \ldots, x_p/h_p)$$
for all $x \in \mathbb{R}^p$ can be used, where $h = (h_1, \ldots, h_p)$ is a vector of bandwidth parameters.
- A more general version would be to use a symmetric, positive definite bandwidth matrix $H \in \mathbb{R}^{p \times p}$. In this case the required scaling is
$$K_H(x) = \frac{1}{\det(H)} K(H^{-1}x)$$
for all $x \in \mathbb{R}^p$.

For all of these choices, the kernel density estimator is given by
$$\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^{n} K_h(x - x_i)$$
(using K_H for the third option) for all $x \in \mathbb{R}^p$. Bandwidth selection in the multivariate case is a difficult problem and we will not discuss this here.

In the previous sections we have used kernel methods to estimate probability densities from observed data. In the following sections we will see a different application of kernel methods: using them for regression, where we estimate a function m from noisy observations.

Summary

- The Epanechnikov kernel minimises $R(K)^2\mu_2(K)$ and gives the smallest integrated error, though differences between common kernels are small.
- Bandwidth selection uses plug-in rules that estimate $R(f'')$ from the data, often by assuming a normal reference distribution.
- R provides built-in bandwidth selectors such as `bw.nrd0()`, `bw.nrd()`, and `bw.SJ()` for practical use.
- Kernel density estimation extends to higher dimensions $\mathbb{R}^p$ using product kernels, with bandwidth parameters that can differ across coordinates.

Problem Sheet 1

This problem sheet is for self-study only. It is not assessed.

1. Consider the following function:

$$K(x) = \begin{cases} \frac{2}{3}(1-|x|^3) & \text{if } |x| \leq 1, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

a. Show that this function integrates to 1 over its domain.

Solution: We have

$$\begin{aligned} \int_{-\infty}^{\infty} K(x), dx &= \frac{2}{3}\int_{-1}^{1}(1-|x|^3), dx \\ &= \frac{4}{3}\int_{0}^{1}(1-x^3), dx \\ &= \frac{4}{3}\left[x - \frac{x^4}{4}\right]_0^1 \\ &= \frac{4}{3}\left(1-\frac{1}{4}\right) \\ &= 1. \end{aligned}$$

b. Show that K satisfies the conditions of a kernel.

Solution: We have already seen that K integrates to 1 over its domain. Since $|x| = |-x|$ for all $x \in \mathbb{R}$, the function K is symmetric. Finally, since $|x|^3 \leq 1$ for all $x \in \mathbb{R}$ with $|x| \leq 1$, we have $K(x) \geq 0$ for all $x \in \mathbb{R}$.

c. Compute the moments $\mu_0(K)$, $\mu_1(K)$ and $\mu_2(K)$ of K.

Solution: The kth moment of K is given by

$$\mu_k(K) = \int_{-\infty}^{\infty} x^k K(x)\, dx = \frac{2}{3}\int_{-1}^{1} x^k(1-|x|^3)\, dx.$$

For $k = 0$, we know $\mu_0(K) = 1$, from part a. For $k = 1$, we have $\mu_1(K) = 0$, since K is symmetric. For $k = 2$, we find

$$\begin{aligned} \mu_2(K) &= \frac{2}{3}\int_{-1}^{1} x^2(1-|x|^3)\, dx \\ &= \frac{4}{3}\int_{0}^{1} x^2(1-x^3)\, dx \\ &= \frac{4}{3}\left[\frac{x^3}{3} - \frac{x^6}{6}\right]_0^1 \\ &= \frac{4}{3}\left(\frac{1}{3}-\frac{1}{6}\right) \\ &= \frac{2}{9}. \end{aligned}$$

2. Consider a normal density with mean and variance ², given by:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

a. Calculate f'(x) and f''(x) for this density.

Solution: Using the chain rule:

$$\begin{aligned} f'(x) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \cdot \left(-\frac{x-\mu}{\sigma^2}\right) \\ &= -f(x) \cdot \frac{x-\mu}{\sigma^2} \end{aligned}$$

Taking derivatives again, using the product rule:

$$\begin{aligned} f''(x) &= -f'(x) \cdot \frac{x-\mu}{\sigma^2} - f(x) \cdot \frac{1}{\sigma^2} \\ &= f(x) \cdot \frac{(x-\mu)^2}{\sigma^4} - f(x) \cdot \frac{1}{\sigma^2} \\ &= f(x) \cdot \left(\frac{(x-\mu)^2}{\sigma^4} - \frac{1}{\sigma^2}\right) \end{aligned}$$

b. Using the formula

$$\text{bias}(\hat{f}_h(x)) \approx \frac{\mu_2(K)}{2} f''(x) h^2,$$

show that for fixed K and h, the bias satisfies the following proportionality:

$$\text{bias}(\hat{f}_h(x)) \propto \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\left(\frac{(x-\mu)^2}{\sigma^4} - \frac{1}{\sigma^2}\right).$$

Solution: Substituting our expression for $f''(x)$ into the bias formula we get

$$\begin{aligned} \text{bias}(\hat{f}_h(x)) &\approx \frac{\mu_2(K)}{2} f''(x) h^2 \\ &= \frac{\mu_2(K)}{2} \cdot f(x) \cdot \left(\frac{(x-\mu)^2}{\sigma^4} - \frac{1}{\sigma^2}\right) h^2 \\ &= \frac{\mu_2(K) h^2}{2\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\left(\frac{(x-\mu)^2}{\sigma^4} - \frac{1}{\sigma^2}\right). \end{aligned}$$

c. For which values of x is the bias positive, negative, or zero? Why does this make sense intuitively?

Solution: The exponential term is always positive, so the sign depends on the sign of the term $(x-\mu)^2/\sigma^4 - 1/\sigma^2$. This equals zero when $(x-\mu)^2 = \sigma^2$, or when $x = \mu \pm \sigma$. The bias is negative when $|x-\mu| < \sigma$, is positive when $|x-\mu| > \sigma$.

This makes sense because the positive bandwidth h smoothes the density, so the bias is negative when the density is concave (near the maxiumum) and positive when it is convex (towards the tails).

3. Consider the variance of a kernel density estimate $\hat{f}_h(x)$ based on a sample $X_1, \ldots, X_n$ with common density f:

$$\text{Var}(\hat{f}_h(x)) = \frac{1}{n^2} \sum_{i,j=1}^{n} \mathbb{E}\Big(K_h(x - X_i) K_h(x - X_j)\Big)$$

a. Let $n = 3$. Write out all terms in the sum explicitly and identify which terms involve the only one random variable X_i and which involve more than one random variable.

Solution: For $n = 3$, expanding the double sum gives 9 terms:

Terms with same random variable ($i = j$):

$$\begin{aligned} &\mathbb{E}(K_h(x - X_1) K_h(x - X_1)) \\ &\mathbb{E}(K_h(x - X_2) K_h(x - X_2)) \\ &\mathbb{E}(K_h(x - X_3) K_h(x - X_3)) \end{aligned}$$

Terms with different random variables ($i \neq j$):

$$\begin{aligned}
&\mathbb{E}(K_h(x-X_1)K_h(x-X_2))\\
&\mathbb{E}(K_h(x-X_1)K_h(x-X_3))\\
&\mathbb{E}(K_h(x-X_2)K_h(x-X_1))\\
&\mathbb{E}(K_h(x-X_2)K_h(x-X_3))\\
&\mathbb{E}(K_h(x-X_3)K_h(x-X_1))\\
&\mathbb{E}(K_h(x-X_3)K_h(x-X_2))
\end{aligned}$$

b. For general n, how many terms in the sum have $i = j$ and how many have $i \neq j$?

Solution: When $i = j$, we are choosing the same index from $\{1, \dots, n\}$ once, giving n terms. When $i \neq j$, we are choosing two different indices from $\{1, \dots, n\}$, giving $n(n-1)$ terms. The total number of terms is thus $n + n(n-1) = n^2$, as expected.

c. Using these counts and the independence of the X_i, show that:

$$\mathrm{Var}(\hat{f}_h(x)) = \frac{1}{n}\mathbb{E}\Big(K_h(x-X_1)^2\Big) + \frac{n-1}{n}\mathbb{E}\Big(\hat{f}_h(x)\Big)^2.$$

Solution: Using part b, we can split the sum into two parts:

$$\begin{aligned}
\mathrm{Var}(\hat{f}_h(x)) &= \frac{1}{n^2}\Big(n\mathbb{E}(K_h(x-X_1)^2) + n(n-1)\mathbb{E}(K_h(x-X_1)K_h(x-X_2))\Big)\\
&= \frac{1}{n}\mathbb{E}(K_h(x-X_1)^2) + \frac{n-1}{n}\mathbb{E}(K_h(x-X_1))\,\mathbb{E}(K_h(x-X_2)),
\end{aligned}$$

where we used the independence of the X_i in the last step. Since $\mathbb{E}(K_h(x-X_1)) = \mathbb{E}(K_h(x-X_2)) = \mathbb{E}(\hat{f}_h(x))$, this gives the required result.

4. Consider the following data:

```
x <- c(89.6, 82.5, 70.9, 83.8, 92.4, 86.5, 77.3, 89.2,
       93.1, 84.7, 78.5, 88.3, 85.6, 90.4, 76.8)
```

a. Determine the sample standard deviation of these data.

Solution:

```
sigma <- sd(x)
sigma
```

```
## [1] 6.410126
```

The R ouputs shows that the standard deviation is 6.410126.

b. Using the "plug-in rule" from section 3.2, what bandwidth would you choose for a kernel density estimate of these data using the triangular kernel?

Solution:

The plug-in rule assumes that the density is normal, to get

$$R(f'') = \frac{3}{8\sigma^5\sqrt{\pi}}.$$

Substituting the sample standard deviation into this formula gives

```
R.fpp <- 3 / (8 * sigma^5 * sqrt(pi))
R.fpp
```

```
## [1] 1.954895e-05
```

We also need the sample size, and the roughness and second moment of the kernel:

```
n <- length(x)
R.K <- 2/3
mu2.K <- 1/6
```

Using these quantities and the formula from section 2.6 gives the result:

```
h <- (R.K / (n * mu2.K^2 * R.fpp))^(1/5)
h
```

```
## [1] 9.607254
```

4 Kernel Smoothing

We now consider the statistical model

$$Y_i = m(x_i) + \varepsilon_i,$$

where $m\colon \mathbb{R} \to \mathbb{R}$ is a smooth function and ε_i are independent random variables with $\mathbb{E}(\varepsilon_i) = 0$. We are given data (x_i, y_i) for $i \in \{1, \dots, n\}$ and our aim is to estimate the function m. The task of estimating the function m from data is called **smoothing**.

4.1 The Nadaraya-Watson Estimator

Since we have

$$\mathbb{E}(Y_i) = \mathbb{E}(m(x_i) + \varepsilon_i) = m(x_i) + \mathbb{E}(\varepsilon_i) = m(x_i),$$

we could attempt to use a Monte-Carlo approach where we estimate the expectation $\mathbb{E}(Y_i)$ using an average of many Y values. This approach is not feasible in practice, since typically we will only have *a single* observation y_i corresponding to a given x_i. The idea of the Nadaraya-Watson Estimator is to average the y_i corresponding to nearby x_i instead. A weighted average is used, which gives less weight to further away values. This leads to the following definition.

Definition 4.1. The **Nadaraya-Watson Estimator** for $m(x)$ is given by

$$\hat{m}_h(x) = \frac{\sum_{i=1}^n K_h(x - x_i) y_i}{\sum_{j=1}^n K_h(x - x_j)},$$

where K_h is a kernel scaled to bandwidth h as in definition 1.2.

The problem of finding m using kernel functions is called **kernel smoothing** or **kernel regression**. In this context, the bandwidth h is also called the **smoothing parameter**. The Nadaraya-Watson Estimator is not the only method for kernel smoothing. We will learn about different methods in the next sections.

Using the shorthand

$$w_i(x) := \frac{K_h(x - x_i)}{\sum_{j=1}^n K_h(x - x_j)}$$

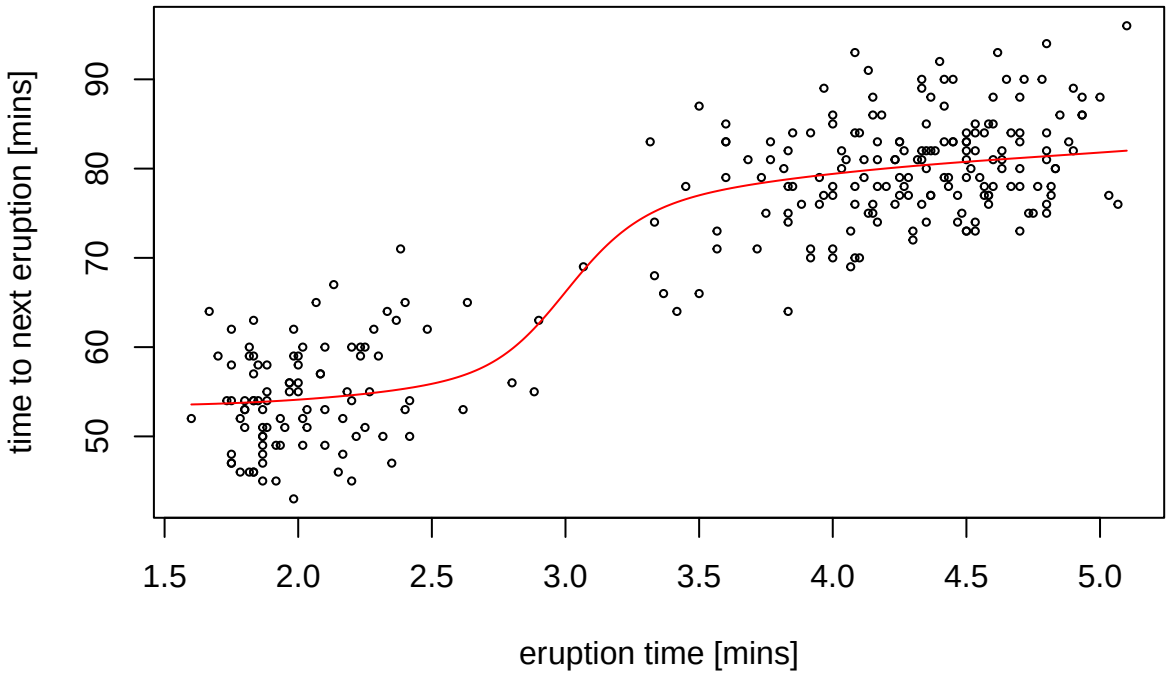
we can write the Nadaraya-Watson Estimator as $\hat{m}_h(x) = \sum_{i=1}^n w_i(x) y_i$ and since

$$\begin{aligned}\sum_{i=1}^n w_i(x) &= \sum_{i=1}^n \frac{K_h(x - x_i)}{\sum_{j=1}^n K_h(x - x_j)} \\ &= \frac{\sum_{i=1}^n K_h(x - x_i)}{\sum_{j=1}^n K_h(x - x_j)} \\ &= 1,\end{aligned}$$

this is indeed a weighted average.

Example 4.1. The `faithful` dataset built into R contains 272 observations of waiting time between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park. We can use the `ksmooth()` function to compute Nadaraya-Watson estimate for the waiting time after an eruption of a given length. Here we use a Gaussian kernel with bandwidth 1.

```
x <- faithful$eruptions
y <- faithful$waiting
plot(x, y, cex = .5,
     xlab = "eruption time [mins]", ylab = "time to next eruption [mins]")
lines(ksmooth(x, y, kernel = "normal", bandwidth = 1, n.points = 1000),
      col = "red")
```

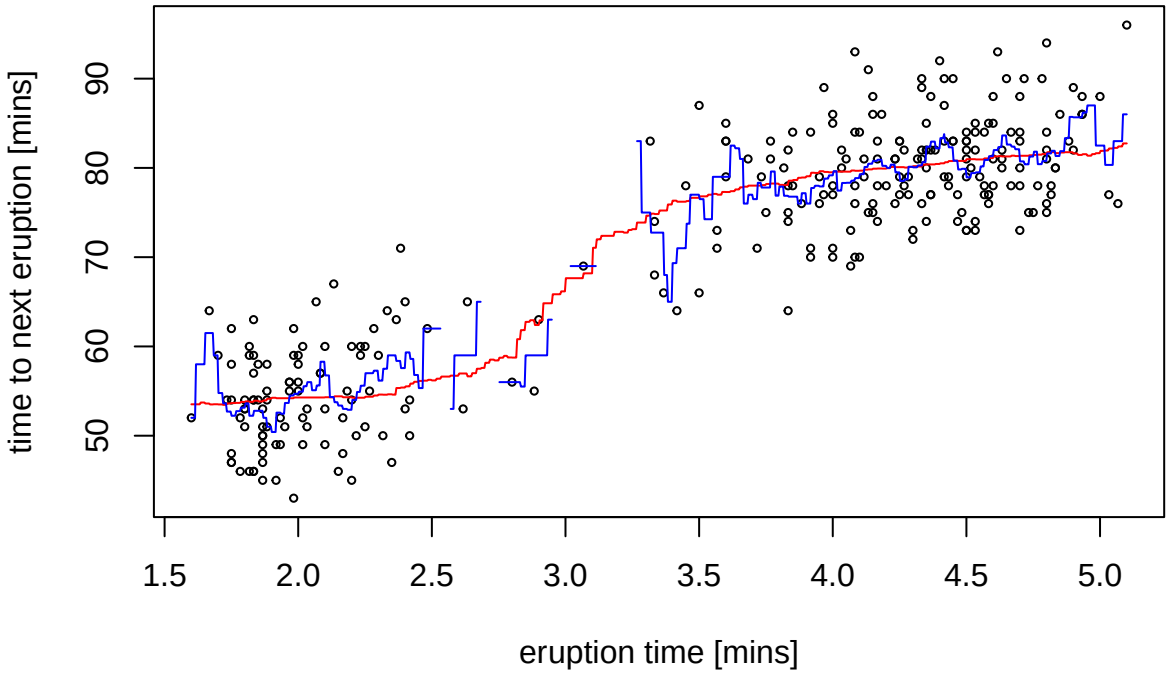


The estimate $\hat{m}_h$ (red line) smoothly connects the two clusters visible in the scatter plot.

For kernels with bounded support, *e.g.* for the Epanechnikov kernel, the denominator $\sum_{j=1}^{n} K_h(x - x_j)$ will equal zero for x which are too far away from all of the x_i. For these x, the weights w_i and thus also the estimate $\hat{m}_h(x)$ are undefined. This problem can easily be seen in practice, when the bandwidth is chosen too small.

Example 4.2. To illustrate the problem of the estimate becoming undefined far away from the data points, we redo the previous estimate using a uniform kernel:

```
plot(x, y, cex = .5,
     xlab = "eruption time [mins]", ylab = "time to next eruption [mins]")
lines(ksmooth(x, y, kernel = "box", bandwidth = 1, n.points = 1000),
      col = "red")
lines(ksmooth(x, y, kernel = "box", bandwidth = 0.1, n.points = 1000),
      col = "blue")
```



For $h = 1$ (red line) we get a line $\hat{m}_h$ which is less smooth than the estimate using the Gaussian kernel, but is otherwise looks similar to the previous estimate. In contrast, if we reduce the bandwidth to $h = 0.1$ (blue line), gaps start to appear in the plot of $\hat{m}_h$ where the spacing of the data points is too large.

4.2 Estimation Error

Here we will discuss how fast the estimation error decreases in the limit of $n \to \infty$, *i.e.* for the case when we have a large dataset to use for the estimate. As before, we will find that we need to decrease the bandwidth h as n increases.

To allow for n to change, we will introduce a statistical model also for the inputs x_i. (This is different from what we did in the level 3 part of the module for linear regression.) Here we will consider the following model:

- $X_1, \dots, X_n$ are independent and identically distributed with density f.
- $\eta_1, \dots \eta_n$ are independent, with $\mathbb{E}(\eta_i) = 0$ and $\mathrm{Var}(\eta_i) = 1$.
- $\varepsilon_i = s(X_i)\eta_i$ for all $i \in \{1, \dots, n\}$, where $s \colon \mathbb{R} \to (0, \infty)$ is a smooth function.
- $Y_i = m(X_i) + \varepsilon_i$ where $m \colon \mathbb{R} \to \mathbb{R}$ is a smooth function.

While this extended model allows us to increase n, it also creates a practical problem: the estimator

$$\hat{m}_h(x) = \frac{\sum_{i=1}^n K_h(x - X_i)Y_i}{\sum_{j=1}^n K_h(x - X_j)},$$

now has random terms both in the numerator and in the denominator. This will make it more challenging to determine the behaviour of $\mathbb{E}(\hat{m}_h(x))$ and $\mathrm{Var}(\hat{m}_h(x))$ as $n \to \infty$ and $h \downarrow 0$. We can write $\hat{m}_h(x)$ as

$$\hat{m}_h(x) = \frac{\frac{1}{n}\sum_{i=1}^n K_h(x - X_i)Y_i}{\frac{1}{n}\sum_{j=1}^n K_h(x - X_j)} = \frac{\frac{1}{n}\sum_{i=1}^n K_h(x - X_i)Y_i}{\hat{f}_h(x)} = \frac{\hat{r}_h(x)}{\hat{f}_h(x)} \tag{15}$$

where $\hat{f}_h(x)$ is the kernel density estimator from Section 1 and

$$\hat{r}_h(x) := \frac{1}{n}\sum_{i=1}^n K_h(x - X_i)Y_i.$$

We will consider the numerator and denominator of equation (15) separately.

Denominator

From equations (8) and (13) we know that

$$\mathbb{E}(\hat{f}_h(x)) \approx f(x) + \frac{\mu_2(K)f''(x)}{2}h^2$$

and

$$\mathrm{Var}(\hat{f}_h(x)) \approx \frac{1}{nh}f(x)R(K)$$

as $h \downarrow 0$.

Numerator

We start by considering the numerator $\hat{r}_h(x)$. The arguments used here will be very similar to the arguments used in Section 2.4.3 on the variance of kernel density estimates.

The expectation of $\hat{r}_h(x)$ is

$$\begin{aligned}
\mathbb{E}(\hat{r}_h(x)) &= \mathbb{E}\Big(\frac{1}{n}\sum_{i=1}^n K_h(x - X_i)Y_i\Big) \\
&= \frac{1}{n}\sum_{i=1}^n \mathbb{E}(K_h(x - X_i)Y_i) \\
&= \mathbb{E}(K_h(x - X)Y) \\
&= \mathbb{E}\Big(K_h(x - X)(m(X) + s(X)\eta)\Big).
\end{aligned}$$

We use integrals to average over the randomness in X and η, denoting the density of η by φ:

$$\begin{aligned}\mathbb{E}(\hat{r}_h(x)) &= \int\int K_h(x-\xi)(m(\xi)+s(\xi)e)\,\varphi(e)\,de\,f(\xi)\,d\xi \\ &= \int K_h(x-\xi)\Big(m(\xi)+s(\xi)\int e\,\varphi(e)\,de\Big)\,f(\xi)\,d\xi \\ &= \int K_h(x-\xi)m(\xi)f(\xi)\,d\xi,\end{aligned}$$

since

$$\int e\,\varphi(e)\,de = \mathbb{E}(\eta) = 0.$$

Writing

$$r(x) := m(x)f(x)$$

as an abbreviation, we finally get

$$\mathbb{E}(\hat{r}_h(x)) = \int K_h(x-\xi)r(\xi)\,d\xi.$$

We now formalise an argument which we already used earlier.

Lemma 4.1. *Let $g\colon \mathbb{R}\to\mathbb{R}$ be two times continuously differentiable and let K be a kernel function. Then we have*

1. $\int K_h(x-\xi)g(\xi)\,d\xi = g(x) + \frac{1}{2}\mu_2(K)g''(x)h^2 + o(h^2)$ *as* $h\downarrow 0$, *and*
2. $\int K_h(x-\xi)^2 g(\xi)\,d\xi = \frac{1}{h}R(K)g(x) + o(1/h)$ *as* $h\downarrow 0$.

Proof. The first statement is proved using substitution and Taylor expansion of r around x as shown in the derivation of equation (8). The second statement is proved similarly, as shown in the derivation of equation (11). □

Using the first part of lemma 4.1 for $g = r$ we get

$$\mathbb{E}(\hat{r}_h(x)) = r(x) + \frac{1}{2}\mu_2(K)r''(x)h^2 + o(h^2).$$

For the variance of $\hat{r}_h(x)$ we get

$$\begin{aligned}\operatorname{Var}(\hat{r}_h(x)) &= \operatorname{Var}\Big(\frac{1}{n}\sum_{i=1}^n K_h(x-X_i)Y_i\Big) \\ &= \frac{1}{n^2}\sum_{i=1}^n \operatorname{Var}(K_h(x-X_i)Y_i) \\ &= \frac{1}{n}\operatorname{Var}(K_h(x-X)Y) \\ &= \frac{1}{n}\Big(\mathbb{E}(K_h(x-X)^2Y^2) - \mathbb{E}(K_h(x-X)Y)^2\Big).\end{aligned}$$

We have already seen that

$$\mathbb{E}(K_h(x-X)Y) = \mathbb{E}(\hat{r}_h(x)) = r(x) + \frac{1}{2}\mu_2(K)r''(x)h^2 + o(h^2)$$

and thus

$$\mathbb{E}(K_h(x-X)Y)^2 = r(x)^2 + O(h^2).$$

Using the second part of lemma 4.1 one can show that

$$\begin{aligned}\mathbb{E}(K_h(x-X)^2Y^2) &= \int\int K_h(x-\xi)^2\big(m(\xi)+s(\xi)e\big)^2\,\varphi(e)\,de\,f(\xi)\,d\xi \\ &= \int K_h(x-\xi)^2(m(\xi)^2+s(\xi)^2)f(\xi)\,d\xi \\ &= \frac{1}{h}R(K)(m(x)^2+s(x)^2)f(x)+o(1/h).\end{aligned}$$

Combining these equations we find

$$\operatorname{Var}(\hat{r}_h(x)) \approx \frac{1}{nh}R(K)(m(x)^2+s(x)^2)f(x)+\frac{1}{n}r(x)^2$$

as $n\to\infty$, $h\downarrow 0$ and $nh\to\infty$.

Mean Squared Error

To turn our results about $\hat{r}_h$ and our previous results about $\hat{f}$ into an error estimate for

$$\hat{m}_h(x)=\frac{\hat{r}_h(x)}{\hat{f}_h(x)},$$

we consider Taylor expansion of the function $g(y)=1/y$:

$$\begin{aligned}g(y+h) &= g(y)+g'(y)h+o(h) \\ &= \frac{1}{y}-\frac{1}{y^2}h+o(h).\end{aligned}$$

Using this approximation we get

$$\begin{aligned}\hat{m}_h(x) &= \hat{r}_h(x)g(\hat{f}_h(x)) \\ &= \hat{r}_h(x)g(f(x)+\hat{f}_h(x)-f(x)) \\ &\approx \hat{r}_h(x)\Big(\frac{1}{f(x)}-\frac{1}{f(x)^2}(\hat{f}_h(x)-f(x))\Big) \\ &= \frac{\hat{r}_h(x)}{f(x)}-\frac{\hat{r}_h(x)(\hat{f}_h(x)-f(x))}{f(x)^2}.\end{aligned}$$

With the help of this trick, we have achieved that now all random terms are in the denominator and thus we can take expectations easily:

$$\begin{aligned}\mathbb{E}(\hat{m}_h(x)) &= \frac{\mathbb{E}(\hat{r}_h(x))}{f(x)}-\frac{\mathbb{E}\Big(\hat{r}_h(x)(\hat{f}_h(x)-f(x))\Big)}{f(x)^2} \\ &\approx \frac{\mathbb{E}(\hat{r}_h(x))}{f(x)}-\frac{\mathbb{E}(\hat{r}_h(x))\mathbb{E}(\hat{f}_h(x)-f(x))}{f(x)^2}.\end{aligned}$$

Substituting in our previous results we get

$$\begin{aligned}\mathbb{E}(\hat{m}_h(x)) &\approx \frac{r(x)+\frac{1}{2}\mu_2(K)r''(x)h^2+o(h^2)}{f(x)}-\frac{(r(x)+\frac{1}{2}\mu_2(K)r''(x)h^2+o(h^2))\frac{1}{2}\mu_2(K)f''(x)h^2}{f(x)^2} \\ &= \frac{r(x)}{f(x)}+\frac{1}{2}\frac{\mu_2(K)r''(x)}{f(x)}h^2-\frac{1}{2}\frac{\mu_2(K)r(x)f''(x)}{f(x)^2}h^2+o(h^2)\end{aligned}$$

Using $r(x)=f(x)m(x)$ we find the derivative $r'(x)=f'(x)m(x)+f(x)m'(x)$ as well as the second derivative $r''(x)=f''(x)m(x)+2f'(x)m'(x)+f(x)m''(x)$. This gives

$$\begin{aligned}\mathbb{E}(\hat{m}_h(x)) &= m(x)+\frac{1}{2}\frac{\mu_2(K)r''(x)}{f(x)}h^2-\frac{1}{2}\frac{\mu_2(K)m(x)f''(x)}{f(x)}h^2+o(h^2) \\ &= m(x)+\frac{1}{2}\frac{\mu_2(K)(2f'(x)m'(x)+f(x)m''(x))}{f(x)}h^2+o(h^2) \\ &= m(x)+\mu_2(K)\Big(\frac{f'(x)}{f(x)}m'(x)+\frac{1}{2}m''(x)\Big)h^2+o(h^2)\end{aligned}$$

and

$$\text{bias}(\hat{m}_h(x)) = \mu_2(K)\Big(\frac{f'(x)}{f(x)}m'(x) + \frac{1}{2}m''(x)\Big)h^2 + o(h^2).$$

A similar calculation gives the approximate variance as

$$\text{Var}(\hat{m}_h(x)) = \frac{1}{nh}\frac{\sigma^2(x)R(K)}{f(x)} + o\Big(\frac{1}{nh}\Big).$$

So finally we have

$$\begin{aligned}\text{MSE}(\hat{m}_h(x)) = \frac{h^4\mu_2(K)^2}{4}\Big(m''(x) + \frac{2m'(x)f'(x)}{f(x)}\Big)^2 \\ + \frac{1}{nh}\frac{\sigma^2(x)R(K)}{f(x)} + o\Big(\frac{1}{nh}\Big) + o(h^4).\end{aligned}$$

Notes:

- A more careful calculation will need to take into account that $\hat{m}_h(x)$ may be undefined. All expectations and variances are conditional on $\hat{f}(x) \neq 0$. One can show that $P(\hat{f}(x) \neq 0) \to 1$ as $n \to \infty$ for all $x \in \mathbb{R}$ with $f(x) > 0$, so this is not a problem.
- The MSE is of order $O(n^{-4/5})$ when we choose $h \sim n^{-1/5}$, as before.
- The formula for the variance shows that the regression curve is more stable in those areas where there are plenty of observations.
- The bias-squared is either dominated by the second derivative $m''(x)$ - when we are close to a local extremum of $m(x)$ (turning point), or by the first derivative $m'(x)$ when we have few observations.
- This calculation is helpful in creating confidence intervals for the estimate $\hat{m}_h(x)$ in which $\sigma^2(x)$ can be estimated by

  $$\hat{\sigma}^2(x) = \sum w_i(x)(y_i - \hat{m}_h^{(i)}(x_i))^2,$$

  where $\hat{m}_h^{(i)}(x_i)$ is the estimate of m at the point x_i using all of the data except for the observation at (x_i, y_i).

Summary

- The Nadaraya-Watson estimator provides a kernel-based method for nonparametric regression, estimating $m(x)$ as a weighted average of nearby observations.
- For kernels with bounded support, the estimator may be undefined when data points are too sparse (bandwidth too small).
- Under a random design model, the estimator has bias of order $O(h^2)$ and variance of order $O(1/(nh))$.
- The optimal bandwidth is $h \sim n^{-1/5}$, giving mean squared error of order $O(n^{-4/5})$.
- The bias depends on both $m''(x)$ and the ratio $f'(x)/f(x)$, reflecting curvature in the regression function and the density of observations.
- The variance is smaller in regions where observations are plentiful (large $f(x)$) and increases with the local noise level $\sigma^2(x)$.

5 Local Polynomial Regression

Local polynomial regression is a generalisation of the Nadaraya-Watson estimator that addresses some of its limitations, particularly boundary bias. The method combines the two ideas of linear regression with weights and polynomial regression. The aim is still to estimate the model mean $m\colon \mathbb{R} \to \mathbb{R}$ from given data $(x_1, y_1), \dots, (x_n, y_n)$.

5.1 Linear Regression with Weights

In the level 3 part of the module, we introduced the least squares method for linear regression. This method estimates the regression coefficients by minimising the residual sum of squares:

$$r(\beta) = \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} \big(y_i - (X\beta)_i\big)^2.$$

Here we will extend this method to include weights for the observations. Given weights $w_1, \dots, w_n > 0$, the weighted least squares method minimises

$$r_w(\beta) = \sum_{i=1}^{n} w_i \varepsilon_i^2 = \sum_{i=1}^{n} w_i \big(y_i - (X\beta)_i\big)^2.$$

In matrix notation, this function can be written as

$$r_w(\beta) = (y - X\beta)^\top W (y - X\beta),$$

where W is a diagonal matrix with the weights on the diagonal:

$$W = \begin{pmatrix} w_1 & 0 & 0 & \cdots & 0 \\ 0 & w_2 & 0 & \cdots & 0 \\ 0 & 0 & w_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & w_n \end{pmatrix}. \tag{16}$$

Similar to lemma 2.1 in the level 3 notes, we can take derivatives to find the minimum of r_w. The result is

$$\hat{\beta} = (X^\top W X)^{-1} X^\top W y.$$

Since W appears once in the inverse and once before the y, we can multiply W by any number without changing the result. Thus we don't need to "normalise" the weights w_i to sum to one.

As before, the fitted value for inputs $(\tilde{x}_1, \dots, \tilde{x}_p) \in \mathbb{R}^p$ is given by

$$\hat{\beta}_0 + \hat{\beta}_1 \tilde{x}_1 + \cdots + \hat{\beta}_p \tilde{x}_p = \tilde{x}^\top \hat{\beta},$$

where $\tilde{x} = (1, \tilde{x}_1, \dots, \tilde{x}_p)$.

5.2 Polynomial Regression

In these notes we only consider regression with a single input variable, $x \in \mathbb{R}$. In this case we can easily fit a polynomial of degree p to the data by using $x, x^2, \dots, x^p$ as the input variables. The corresponding model is

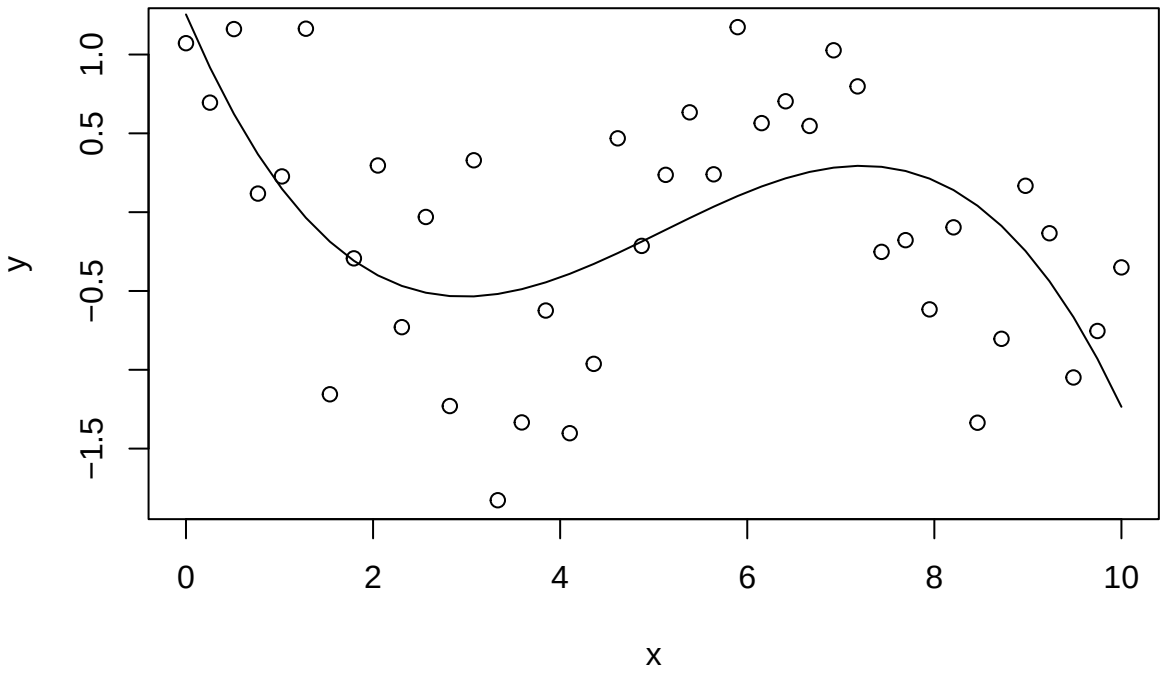
$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_p x^p + \varepsilon.$$

This leads to the design matrix

$$X = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^p \\ 1 & x_2 & x_2^2 & \cdots & x_2^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^p \end{pmatrix}.$$

Example 5.1. To illustrate polynomial regression, we fit a third-order polynomial to a simple, simulated dataset. Since the operator `^` has a special meaning inside `lm()`, we have to use the function `I()` (which disables the special meaning of `+` and `^` for its arguments) when computing the inputs x^2 and x^3.

```r
set.seed(20211102)

n <- 40
x <- seq(0, 10, length.out = n)
y <- cos(x) + rnorm(n, sd = 0.5)

m <- lm(y ~ x + I(x^2) + I(x^3))

plot(x, y)
lines(x, fitted(m))
```



The resulting regression line seems like a reasonable fit for the data. We note that a third-order polynomial grows very quickly to $\pm\infty$ as $|x|$ increases. Thus, the fitted model cannot be used for extrapolating beyond the range of the data.

When the regression is set up in this way, the design matrix often suffers from collinearity. To check for this, we can consider the condition number κ. For the example above we get the following value:

```r
kappa(m, exact = TRUE)
```

```
## [1] 1849.947
```

This is a large value, indicating collinearity. To improve the setup of the problem we can use the model

$$y = \beta_0 + \beta_1(x - \tilde{x}) + \beta_2(x - \tilde{x})^2 + \cdots + \beta_p(x - \tilde{x})^p + \varepsilon,$$

where $\tilde{x}$ is inside the interval of x values. This leads to the design matrix

$$X = \begin{pmatrix} 1 & (x_1 - \tilde{x}) & (x_1 - \tilde{x})^2 & \cdots & (x_1 - \tilde{x})^p \\ 1 & (x_2 - \tilde{x}) & (x_2 - \tilde{x})^2 & \cdots & (x_2 - \tilde{x})^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (x_n - \tilde{x}) & (x_n - \tilde{x})^2 & \cdots & (x_n - \tilde{x})^p \end{pmatrix}. \tag{17}$$

Example 5.2. Continuing from the previous example, we can see that writing the model as in (17) greatly improves the condition number:

```r
x.tilde <- 5
m2 <- lm(y ~ I(x-x.tilde) + I((x-x.tilde)^2) + I((x-x.tilde)^3))
kappa(m2, exact = TRUE)
```

```
## [1] 76.59629
```

While there is still collinearity, the condition number is now much smaller.

Polynomials of higher degree take very large values as $|x|$ increases and often make poor global models. Instead, these polynomials are best used for local interpolation of data.

5.3 Polynomial Regression with Weights

The idea of local polynomial regression is to combine polynomial regression with weights which give more weight to close by observations: to get an estimate at a point $\tilde{x} \in \mathbb{R}$, we define

$$w_i := K_h(\tilde{x} - x_i),$$

where K_h is a scaled kernel function as before. Using the diagonal matrix W from (16) for the weights and the matrix X from (17) as the design matrix, we can fit a polynomial of degree p to the data which fits the data near $\tilde{x}$ well. The regression coefficients are again estimated as

$$\hat{\beta} = (X^\top W X)^{-1} X^\top W y$$

and the model mean near $\tilde{x}$ is given by

$$\hat{m}_h(x; \tilde{x}) = \hat{\beta}_0 + \hat{\beta}_1(x - \tilde{x}) + \hat{\beta}_2(x - \tilde{x})^2 + \cdots + \hat{\beta}_p(x - \tilde{x})^p,$$

where the coefficients $\hat{\beta}$ depend on $\tilde{x}$. The model mean at $x = \tilde{x}$ simplifies to

$$\begin{aligned}
\hat{m}_h(\tilde{x}) &= \hat{m}_h(\tilde{x}; \tilde{x}) \\
&= \hat{\beta}_0 + \hat{\beta}_1(\tilde{x} - \tilde{x}) + \hat{\beta}_2(\tilde{x} - \tilde{x})^2 + \cdots + \hat{\beta}_p(\tilde{x} - \tilde{x})^p \\
&= \hat{\beta}_0.
\end{aligned}$$

Using matrix notation, we can write this as

$$\begin{aligned}
\hat{m}_h(\tilde{x}) &= \hat{\beta}_0 \\
&= e_0^\top \hat{\beta} \\
&= e_0^\top (X^\top W X)^{-1} X^\top W y,
\end{aligned}$$

where $e_0 = (1, 0, \ldots, 0) \in \mathbb{R}^{p+1}$.

Since both X and W depend on $\tilde{x}$, we need to evaluate $(X^\top W X)^{-1} X^\top W y$ separately for each $\tilde{x}$ where an estimate of $\hat{m}_h$ is needed. To get a regression line, this needs to be done over a grid of $\tilde{x}$ values. Thus, this method can be computationally expensive.

5.4 Special Cases

Here we discuss the special cases of $p = 0$, $p = 1$, and $p = 2$.

$p = 0$

For $p = 0$, the polynomial consists only of the constant term β_0. In this case, the design matrix X simplifies to $X = (1, \ldots, 1) \in \mathbb{R}^{n \times 1}$. Thus we have

$$\begin{aligned}
X^\top W X &= (1, \ldots, 1)^\top W (1, \ldots, 1) \\
&= \sum_{i=1}^{n} w_i \\
&= \sum_{i=1}^{n} K_h(\tilde{x} - x_i).
\end{aligned}$$

Similarly, we have

$$\begin{aligned} X^\top W y &= (1, \dots, 1)^\top W y \\ &= \sum_{i=1}^n w_i y_i \\ &= \sum_{i=1}^n K_h(\tilde{x} - x_i) y_i. \end{aligned}$$

Thus we find

$$\begin{aligned} \hat{m}_h(\tilde{x}) &= (X^\top W X)^{-1} X^\top W y \\ &= \frac{\sum_{i=1}^n K_h(\tilde{x} - x_i) y_i}{\sum_{i=1}^n K_h(\tilde{x} - x_i)}. \end{aligned}$$

This is the same formula as in definition 4.1: for $p = 0$ the local polynomial regression estimator is the same as the Nadaraya-Watson estimator.

$p = 1$

For $p = 1$ the polynomial is a straight line, allowing to model the value as well as the slope of the mean line. The resulting estimator is called the **local linear estimator**. This sometimes gives a better fit than the Nadaraya-Watson estimator, for example at the boundaries of the domain.

Example 5.3. To better understand how the local linear estimator works, we can implement it directly from the formulas derived above. Recall from earlier in this section that $\hat{m}_h(\tilde{x}) = \hat{\beta}_0$, where $\hat{\beta}$ is computed via weighted least squares. For the local linear case ($p = 1$), the matrix formula $\hat{\beta} = (X^\top W X)^{-1} X^\top W y$ leads to the expression

$$\hat{m}_h(\tilde{x}) = \frac{T_1 T_2 - T_3 T_4}{B_1 B_2 - B_3^2},$$

where the seven terms are sums involving the kernel weights.

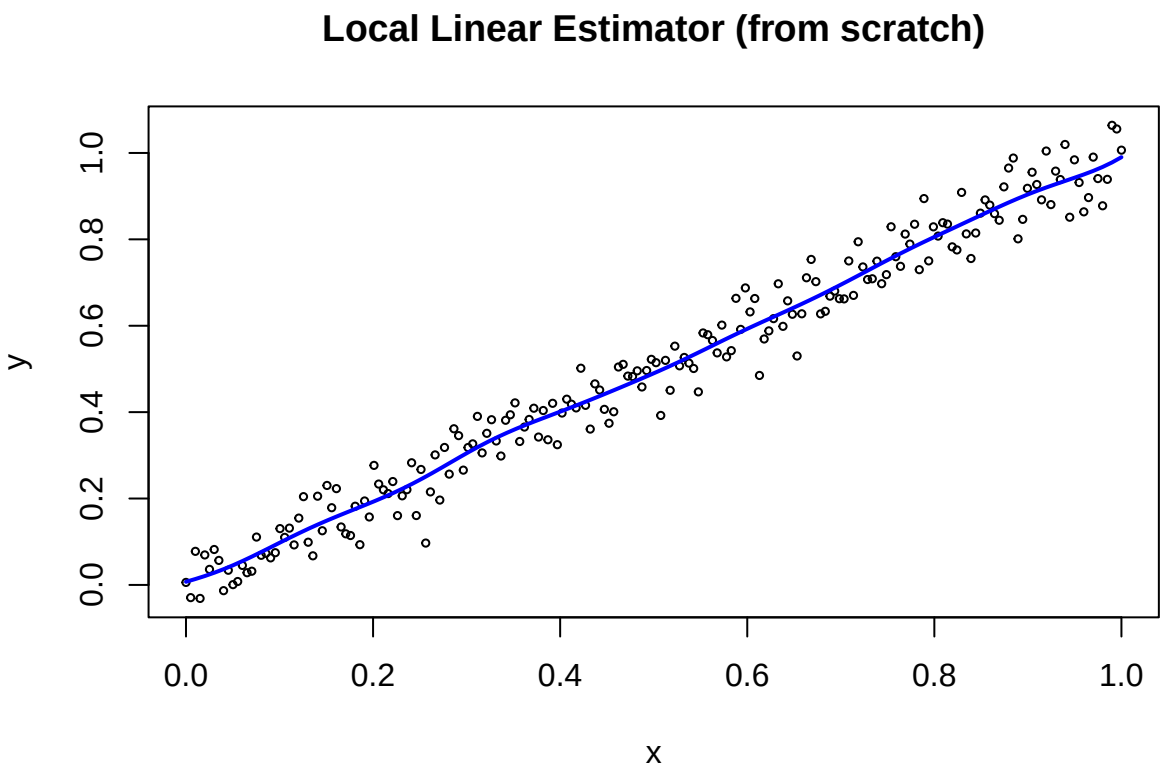
Here is a direct implementation in R:

```
# Local linear regression estimator
local_linear <- function(x_train, y_train, x_eval, h) {
  # Matrix of differences: dx[i,j] = x_eval[j] - x_train[i]
  # Each column corresponds to one evaluation point x_eval[j]
  dx <- -outer(x_train, x_eval, "-")

  # Kernel weights K_h(x_eval[j] - x_train[i])
  K <- dnorm(dx, sd = h)

  # The seven terms from the weighted regression formula:
  T1 <- colSums(y_train * K)
  T2 <- colSums(x_train * (-dx) * K)
  T3 <- colSums(x_train * y_train * K)
  T4 <- colSums((-dx) * K)
  B1 <- colSums(K)
  B2 <- colSums(x_train^2 * K)
  B3 <- colSums(x_train * K)

  # The local linear estimate at each evaluation point
  (T1*T2 - T3*T4) / (B1*B2 - B3^2)
}
```

We can test this implementation on a simple dataset:

```
set.seed(20251114)
```

```r
n <- 200
x <- seq(0, 1, length.out = n)
y <- x + rnorm(n, sd = 0.05)

# Compute local linear estimate on a grid
x_grid <- seq(0, 1, length.out = 100)
m_ll <- local_linear(x, y, x_grid, h = 0.05)

plot(x, y, cex = .5, main = "Local Linear Estimator (from scratch)")
lines(x_grid, m_ll, col = "blue", lwd = 2)
```



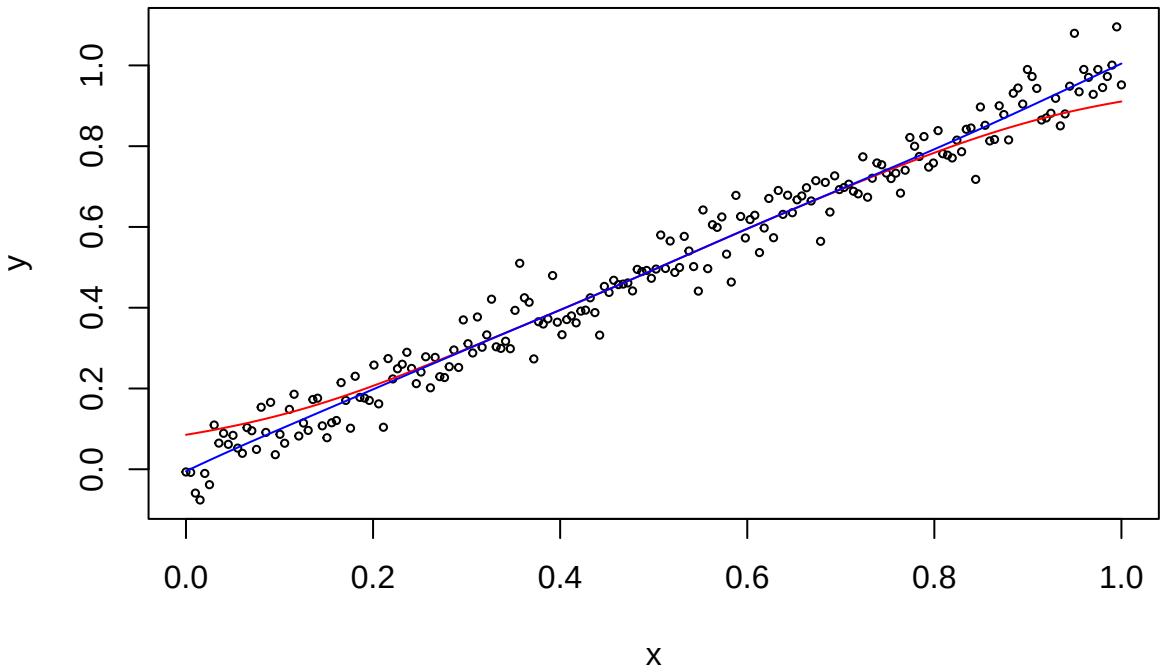
The implementation shows clearly how the kernel weights K_h combine with the data to produce the estimate at each point.

Example 5.4. We can use the R function `locpoly` from the `KernSmooth` package to compute locally polynomial regression estimates. Here we plot the estimate for $p = 1$ (blue line) together with the Nadaraya-Watson estimator (red line), for a simple, simulated dataset. Unfortunately, the function `locpoly()` has an interpretation of the bandwidth which is different from what `ksmooth()` uses. Experimentally I found that `bandwidth = 0.3` for `ksmooth()` corresponds to `bandwidth = 0.11` for `locpoly()`: the output of `ksmooth(..., bandwidth = 0.3)` and of `locpoly(.., degree = 0, bandwidth = 0.11)` is nearly identical.

```r
set.seed(20211103)

n <- 200
x <- seq(0, 1, length.out = n)
y <- x + rnorm(n, sd = 0.05)
plot(x, y, cex = .5)

m1 <- ksmooth(x, y, kernel = "normal", bandwidth = 0.3)
lines(m1, col = "red")

library(KernSmooth)
```

```
## KernSmooth 2.23 loaded
```

```
## Copyright M. P. Wand 1997-2009
m2 <- locpoly(x, y, degree = 1, bandwidth = 0.11)
lines(m2, col = "blue")
```



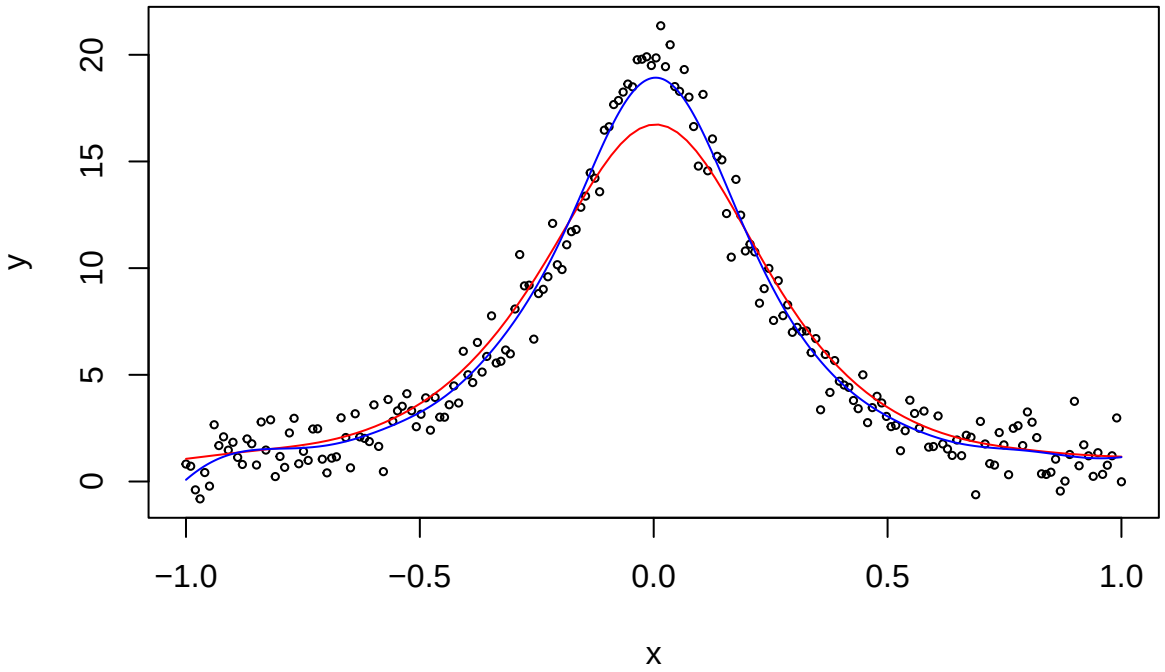
Near the boundaries the Nadaraya-Watson estimator is biased, because on the left-hand boundary all nearby samples correspond to larger values of the mean line, and similarly on the right-hand boundary all nearby samples correspond to smaller values of the mean line. In contrast, the local polynomial estimate retains its slope right up to the boundary.

$p = 2$

For $p = 2$ the local polynomials are parabolas. This sometimes allows us to reduce bias near peaks.

Example 5.5. We compare the Nadaraya-Watson estimator to locally polynomial regression with $p = 2$, using a simulated dataset which has a peak in the middle of the domain. We choose the same bandwidths as in the previous example.

```
set.seed(20211103)

n <- 200
x <- seq(-1, 1, length.out = n)
y <- 1/(x^2 + 0.05) + rnorm(n, sd = 1)
plot(x, y, cex = .5)

m1 <- ksmooth(x, y, kernel = "normal", bandwidth = 0.3, n.points = 100)
lines(m1, col = "red")

library(KernSmooth)
m2 <- locpoly(x, y, degree = 2, bandwidth = 0.11)
lines(m2, col = "blue")
```



We can see that the Nadaraya-Watson estimator (red line) is biased near the peak, because all nearby samples correspond to smaller values of the mean line. In contrast, the local polynomial estimate (blue line) has much lower bias.

Summary

- Local polynomial regression combines weighted least squares with polynomial models, using kernel weights $w_i = K_h(\tilde{x} - x_i)$ to emphasise nearby observations.
- The estimate at $\tilde{x}$ is $\hat{m}_h(\tilde{x}) = \hat{\beta}_0$ where $\hat{\beta} = (X^\top W X)^{-1} X^\top W y$ must be computed separately for each evaluation point.
- For $p = 0$ (constant fit), the method reduces to the Nadaraya-Watson estimator; for $p = 1$ (local linear), it reduces boundary bias; for $p = 2$ (local quadratic), it reduces bias near peaks.
- Local linear regression ($p = 1$) performs better than Nadaraya-Watson at boundaries because the linear fit adapts to local slope.
- In R, the `locpoly()` function from `KernSmooth` provides efficient implementations, though the method can be implemented directly from the weighted regression formula.

Problem Sheet 2

This problem sheet is for self-study only. It is not assessed.

5. Consider the Nadaraya-Watson estimator

$$\hat{m}_h(x) = \frac{\sum_{i=1}^n K_h(x - x_i) y_i}{\sum_{j=1}^n K_h(x - x_j)}.$$

Show that if K is continuous with $K(0) > 0$, then $\hat{m}_h(x_i) \to y_i$ as $h \downarrow 0$ for each data point x_i.

Solution: Evaluating the estimator at the data point x_i, we have

$$\hat{m}_h(x_i) = \frac{\sum_{j=1}^n K_h(x_i - x_j) y_j}{\sum_{j=1}^n K_h(x_i - x_j)} = \frac{K_h(0) y_i + \sum_{j \neq i} K_h(x_i - x_j) y_j}{K_h(0) + \sum_{j \neq i} K_h(x_i - x_j)}.$$

Dividing numerator and denominator by $K_h(0) = \frac{1}{h} K(0)$ gives

$$\hat{m}_h(x_i) = \frac{y_i + \sum_{j \neq i} \frac{K_h(x_i - x_j)}{K_h(0)} y_j}{1 + \sum_{j \neq i} \frac{K_h(x_i - x_j)}{K_h(0)}}.$$

For $j \neq i$, we have $x_i - x_j \neq 0$, so $(x_i - x_j)/h \to \pm\infty$ as $h \downarrow 0$. Since K is continuous and integrates to 1, we must have $K(u) \to 0$ as $|u| \to \infty$. Therefore

$$\frac{K_h(x_i - x_j)}{K_h(0)} = \frac{K((x_i - x_j)/h)}{K(0)} \to \frac{0}{K(0)} = 0$$

as $h \downarrow 0$, using $K(0) > 0$. Since there are only finitely many terms in the sums (namely $n - 1$ terms), we conclude that

$$\hat{m}_h(x_i) \to \frac{y_i + 0}{1 + 0} = y_i$$

as $h \downarrow 0$.

6. The bias of the Nadaraya-Watson estimator $\hat{m}_h(x)$ satisfies

$$\operatorname{bias}(\hat{m}_h(x)) \approx \mu_2(K) \left(\frac{f'(x)}{f(x)} m'(x) + \frac{1}{2} m''(x) \right) h^2.$$

a. Explain why the term $f'(x)/f(x)$ appears in the bias formula for the Nadaraya-Watson estimator but not in the bias formula for kernel density estimation.

Solution: In kernel density estimation, we estimate $f(x)$ directly from the data. The bias arises purely from the Taylor expansion of the density f around x, giving bias $\approx \frac{\mu_2(K)}{2} f''(x) h^2$.

In contrast, the Nadaraya-Watson estimator estimates the conditional mean $m(x) = \mathbb{E}(Y \mid X = x)$, but the estimator involves both the regression function m and the density f of the inputs X_i. As shown in section 4, the estimator can be written as a ratio $\hat{m}_h(x) = \hat{r}_h(x)/\hat{f}_h(x)$, where $\hat{r}_h(x)$ estimates $r(x) = m(x) f(x)$.

When analysing the bias of this ratio, we must expand both r and f. Since $r = mf$, the derivative $r'' = m''f + 2m'f' + mf''$ involves cross-terms between m and f. After simplification, the term $2m'f'$ in the numerator combines with the denominator f to produce the $\frac{f'(x)}{f(x)} m'(x)$ term in the bias.

Intuitively, the density gradient $f'(x)$ affects which observations contribute to the weighted average. In regions where f is changing rapidly, nearby x-values are distributed asymmetrically around x, which biases the estimate of $m(x)$ when m has non-zero slope.

b. In a region where observations are sparse (small $f(x)$, large $|f'(x)|$), is the bias dominated by $m'(x)$ or $m''(x)$? Interpret this geometrically.

Solution: The bias formula has two terms:

$$\text{bias}(\hat{m}_h(x)) \approx \mu_2(K) \left(\frac{f'(x)}{f(x)} m'(x) + \frac{1}{2} m''(x) \right) h^2.$$

When $f(x)$ is small and $|f'(x)|$ is large, the ratio $|f'(x)/f(x)|$ becomes large. Consequently, the term $\frac{f'(x)}{f(x)} m'(x)$ dominates over $\frac{1}{2} m''(x)$, and the bias is dominated by $m'(x)$.

Geometrically, in a sparse region at the edge of the data distribution, most nearby observations come from one side—the direction where f is larger. The kernel averages y-values from points that are systematically shifted in one direction along the x-axis. If m has any slope ($m'(x) \neq 0$), this one-sided averaging pulls the estimate towards y-values that are systematically too high or too low, causing bias proportional to the slope $m'(x)$.

c. Suggest a scenario where the Nadaraya-Watson estimator would have high bias even if m is nearly linear.

Solution: If m is nearly linear, then $m''(x) \approx 0$ and the bias simplifies to

$$\text{bias}(\hat{m}_h(x)) \approx \mu_2(K) \frac{f'(x)}{f(x)} m'(x) h^2.$$

This can still be large if both $m'(x)$ (the slope) and $f'(x)/f(x)$ (the relative density gradient) are large.

A concrete scenario: Suppose we want to estimate a steep linear regression function $m(x) = a + bx$ with large slope b, and the input data X_i are exponentially distributed or otherwise concentrated near one boundary. Near the sparse tail of the distribution, $f(x)$ is small while $|f'(x)|$ remains substantial, making $|f'(x)/f(x)|$ large. Combined with the large slope $m'(x) = b$, this produces substantial bias even though $m''(x) = 0$.

This is precisely the "boundary bias" phenomenon discussed in section 5: at the edges of the data range, the Nadaraya-Watson estimator is biased because observations are available only on one side. Local linear regression ($p = 1$) addresses this by fitting a line locally, which can extrapolate correctly even when observations are asymmetrically distributed.

6 *k*-Nearest Neighbour Regression

In the previous sections we used a fixed bandwidth h to determine the scale on which "closeness" of existing samples to a new input x was measured. While this approach generally works well, problems can appear in regions where samples are sparse (*e.g.* in example 4.2). This problem can be addressed by choosing h adaptively, using larger bandwidths where samples are sparse and smaller bandwidths in regions where there are many samples. The k-nearest neighbour method is one of several methods which implements this idea.

6.1 Definition of the Estimator

Definition 6.1. For $k \in \{1, \dots, n\}$, the ***k*-nearest neighbour**, or k-NN estimate for the model mean $m(x)$ is given by

$$\hat{m}_k(x) := \frac{1}{k} \sum_{i \in J_k(x)} y_i, \tag{18}$$

where

$$J_k(x) := \big\{ i \;\big|\; x_i \text{ is one of the } k \text{ nearest observations to } x \big\}.$$

The k-NN estimate $\hat{m}_k(x)$ is the average of the k responses where the inputs are closest to x. We can interpret equation (18) as a weighted average

$$\hat{m}_k(x) = \sum_{i=1}^{n} w_i(x) y_i,$$

where the weights are given by

$$w_i(x) = \begin{cases} \frac{1}{k}, & \text{if } i \in J_k(x), \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

If several x_i have the same distance to x, some tie-breaking rule must be used to decide which indices to include in the set $J_k(x)$. This case is so unlikely that the choice of rule is not important. One could, for example, pick one of the tied neighbours at random.

The method can be used both for the one-dimensional case $x \in \mathbb{R}$, and for vector-valued inputs $x \in \mathbb{R}^p$. For the one-dimensional case, it is advantageous to sort the data in order of increasing x_i. In this case, the position of x in the list of the x_i can be found using a binary search, and the nearest neighbours can be identified by search to the left and right of this position. For $p > 1$ the method becomes computationally very expensive, since the data needs to be sorted afresh for every new input x. Advanced data structures like "cover trees" can be used to speed up the process of finding the nearest neighbours.

6.2 Properties

The parameter k controls the "smoothness" of the estimate. In the extreme case $k = n$, we have $J_n(x) = \{1, \dots, n\}$ and

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

for all x, *i.e.* for this case $\hat{m}_k$ is constant. The other extreme is the case of $k = 1$, where $\hat{m}_k(x)$ always equals the value of the closest x_i and has jumps at the mid-points between the data points.

Later, in section 8 on cross-validation, we will learn how k can be chosen from data.

Independent of the value of k, the function $\hat{m}_k$ is always a step function, with jumps at points x where two points have equal distance from x.

6.3 Numerical Experiment

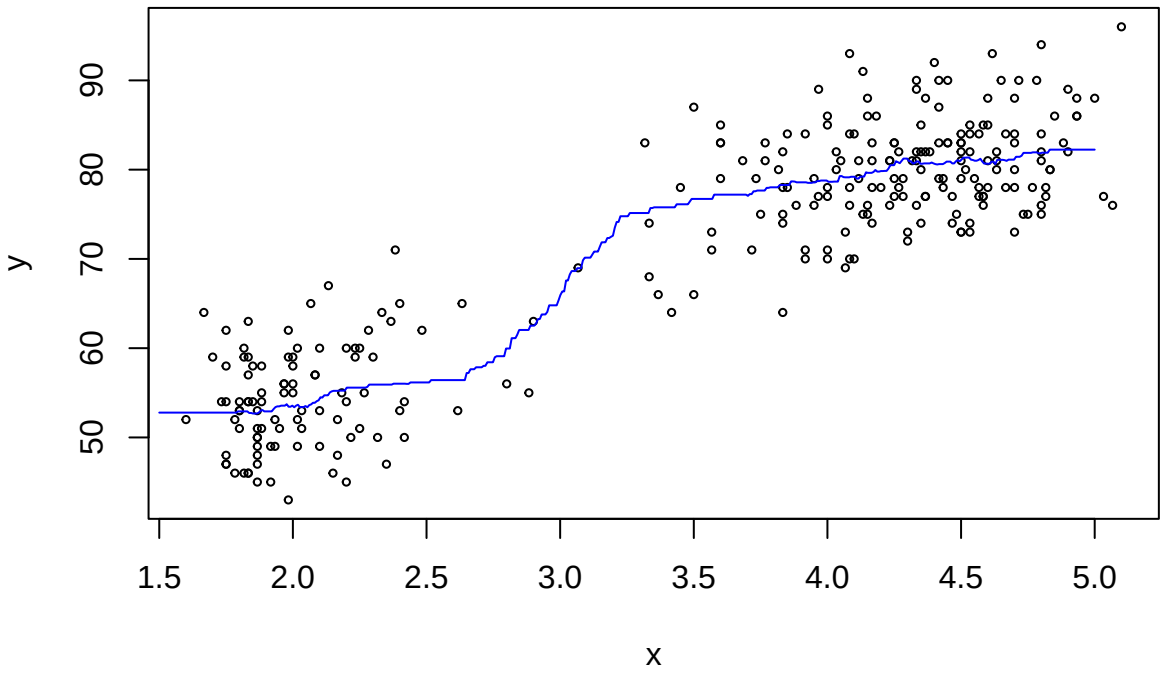
In R, an implementation of the k-NN method can be found in the `FNN` package. This package implements not only k-NN regression, but also k-NN classification, and it implements sophisticated algorithms to speed up the search for the nearest neighbours in higher-dimensional spaces. The function to perform k-NN regression is `knn.reg()`.

Example 6.1. Here we compute a k-NN estimate for the mean of the `faithful` dataset, which we have already encountered in examples 4.1 and 4.2. We start by storing the data in the variables `x` and `y`:

```
x <- faithful$eruptions
y <- faithful$waiting
```

Now we use `knn.reg()` to compute the k-NN estimate on a grid of values. The help page for `knn.reg()` explains that we need to convert the input to either a matrix or a data frame; here we use data frames. The grid of input values where we want to estimate the k-NN estimate is passed in via the optional argument `test = ...`. Here we use the arbitrarily chosen value $k = 50$.

```
library(FNN)

x.test <- seq(1.5, 5, length.out = 500)

m <- knn.reg(data.frame(x=x),
             y = y,
             test = data.frame(x=x.test),
             k = 50)

plot(x, y, cex=.5)
lines(x.test, m$pred, col = "blue")
```



The estimated mean curve looks reasonable.

6.4 Variants of the Method

- For one-dimensional inputs and even k, the **symmetric k-NN estimate** averages the responses corresponding to the $k/2$ nearest neighbours smaller than x and the $k/2$ nearest neighbours larger than x.
- To obtain a continuous estimate, we can define a "local bandwidth" $h(x)$ as

  $$h(x) = c \max\{|x - x_i| \mid i \in J_k(x)\}$$

  for some constant c, and then use the Nadaraya-Watson estimator with this bandwidth:

  $$\tilde{m}(x) = \frac{\sum_{i=1}^{n} K_{h(x)}(x - x_i) y_i}{\sum_{j=1}^{n} K_{h(x)}(x - x_j)},$$

where K is a kernel function as before. If we use $c = 1$ together with the uniform kernel

$$K(x) = \begin{cases} 1/2 & \text{if } -1 \leq x \leq 1 \\ 0 & \text{otherwise,} \end{cases}$$

this method coincides with the k-NN estimator.

Summary

- The k-NN estimator averages the k nearest responses, providing an adaptive bandwidth that adjusts to local data density.
- The parameter k controls smoothness: larger k gives smoother estimates, while $k = 1$ produces a step function matching the nearest observation.
- The method extends naturally to higher dimensions $\mathbb{R}^p$, though finding nearest neighbours becomes computationally expensive without specialised data structures.
- In R, the `FNN` package provides efficient implementations via `knn.reg()`.
- Variants include symmetric k-NN and continuous versions using local bandwidth $h(x)$ based on the distance to the kth nearest neighbour.

7 Spline Smoothing

The previous sections showed how kernel methods and k-nearest neighbour methods estimate the regression function $m\colon \mathbb{R} \to \mathbb{R}$ from data $(x_1, y_1), \ldots, (x_n, y_n)$. In this section we introduce an alternative approach based on **spline smoothing**. Instead of local averaging, we fit a smooth function globally to the entire dataset, using a penalty term to control the smoothness.

7.1 Smoothing Splines

Spline smoothing finds a function m that balances two competing goals:

1. Fit the data well: minimise the residual sum of squares $\sum_{i=1}^{n} \big(y_i - m(x_i)\big)^2$.
2. Remain smooth: avoid excessive curvature.

We measure the curvature of a function m using the integral of the squared second derivative:

$$\int_{-\infty}^{\infty} \big(m''(x)\big)^2 \, dx.$$

A function with high curvature produces a large integral; a nearly linear function produces a small integral. A linear function $m(x) = a + bx$ has $m''(x) = 0$, giving an integral of zero.

Definition 7.1. The **smoothing spline** estimate for the regression function m is the function $\hat{m}_\lambda$ which minimises

$$I(m) = \sum_{i=1}^{n} \big(y_i - m(x_i)\big)^2 + \lambda \int_{-\infty}^{\infty} \big(m''(x)\big)^2 \, dx \tag{19}$$

over all twice differentiable functions $m\colon \mathbb{R} \to \mathbb{R}$. The parameter $\lambda \geq 0$ is called the **smoothing parameter**.

The smoothing parameter λ controls the trade-off between fitting the data and smoothness:

- For $\lambda = 0$, only the residual sum of squares matters, and the solution is the interpolating function which passes through all data points.
- For $\lambda \to \infty$, only the smoothness matters, and the solution is the linear regression line (which has zero curvature).
- For intermediate values of λ, the solution balances fit and smoothness.

This is similar to ridge regression (see section 16 of the level 3 notes), where a penalty term $\lambda\|\beta\|^2$ is added to the residual sum of squares to control the size of the coefficients.

7.2 Cubic Splines

Before we can state the solution to the optimization problem in definition 7.1, we need to introduce the concept of a **cubic spline**.

Definition 7.2. A **cubic spline** with knots $\kappa_1 < \kappa_2 < \cdots < \kappa_k$ is a function $s\colon \mathbb{R} \to \mathbb{R}$ such that

- s is a cubic polynomial on each interval $(-\infty, \kappa_1)$, (κ_1, κ_2), ..., (κ_{k-1}, κ_k), and (κ_k, ∞), and
- s is twice continuously differentiable, *i.e.* s, s' and s'' are all continuous.

A **natural cubic spline** is a cubic spline which is linear on the intervals $(-\infty, \kappa_1)$ and (κ_k, ∞).

The points $\kappa_1, \ldots, \kappa_k$ are called **knots**. At each knot, the function s transitions from one cubic polynomial to another, but the transition is smooth because s, s' and s'' are all continuous.

Example 7.1. Consider the knots at $\kappa_1 = 0$, $\kappa_2 = 1$, and $\kappa_3 = 2$. We will show that the function defined by

- On $(-\infty, 0)$: $s(x) = \frac{3}{2}x$
- On $(0, 1)$: $s(x) = \frac{3}{2}x - \frac{1}{2}x^3$
- On $(1, 2)$: $s(x) = \frac{3}{2}(2 - x) - \frac{1}{2}(2 - x)^3$
- On $(2, \infty)$: $s(x) = \frac{3}{2}(2 - x)$ (linear)

is a natural cubic spline, by checking values and continuity at the knots. For $\kappa_1 = 0$ we have:

- $s(0^-) = 0$ and $s(0^+) = 0$
- $s'(0^-) = \frac{3}{2}$ and $s'(0^+) = \frac{3}{2}$
- $s''(0^-) = 0$ and $s''(0^+) = 0$

For the knot $\kappa_2 = 1$ we get:

- $s(1^-) = \frac{3}{2} - \frac{1}{2} = 1$ and $s(1^+) = \frac{3}{2} - \frac{1}{2} = 1$
- $s'(1^-) = \frac{3}{2} - \frac{3}{2} = 0$ and $s'(1^+) = -\frac{3}{2} + \frac{3}{2} = 0$
- $s''(1^-) = -3$ and $s''(1^+) = -3$

Finally, for the knot $\kappa_3 = 2$ we find:

- $s(2^-) = 0$ and $s(2^+) = 0$
- $s'(2^-) = -\frac{3}{2}$ and $s'(2^+) = -\frac{3}{2}$
- $s''(2^-) = 0$ and $s''(2^+) = 0$

Thus, s is indeed a natural cubic spline.

The following theorem shows that the solution to the smoothing spline problem is always a natural cubic spline.

Theorem 7.1. *The solution $\hat{m}_\lambda$ to the optimization problem in definition 7.1 is a natural cubic spline with knots at the data points $x_1, \dots, x_n$.*

Proof. We sketch the main idea of the proof, showing that the solution must be a cubic spline.

The "calculus of variations" shows that among all twice-differentiable functions $f\colon [x_i, x_{i+1}] \to \mathbb{R}$ with given values and derivatives at the endpoints, the cubic polynomial minimises $\int_{x_i}^{x_{i+1}} [f''(x)]^2 dx$. Equality holds only if $\hat{m}_\lambda$ is already cubic on $[x_i, x_{i+1}]$.

Now suppose $\hat{m}_\lambda$ is the solution but $\hat{m}_\lambda$ is not a cubic polynomial on some interval $[x_i, x_{i+1}]$. We will show that this leads to a contradiction.

Let p be the unique cubic polynomial on $[x_i, x_{i+1}]$ which matches $\hat{m}_\lambda$ in value and first derivative at both endpoints: $p(x_i) = \hat{m}_\lambda(x_i)$, $p'(x_i) = \hat{m}'_\lambda(x_i)$, $p(x_{i+1}) = \hat{m}_\lambda(x_{i+1})$, and $p'(x_{i+1}) = \hat{m}'_\lambda(x_{i+1})$. Now consider the function $\tilde{m}$ which equals $\hat{m}_\lambda$ outside $[x_i, x_{i+1}]$ and equals p on $[x_i, x_{i+1}]$.

Since $\tilde{m}$ agrees with $\hat{m}_\lambda$ at all data points, both functions produce the same residual sum of squares. However, $\tilde{m}$ has a smaller penalty term:

$$\int_{-\infty}^{\infty} \left(\tilde{m}''(x)\right)^2 dx < \int_{-\infty}^{\infty} \left(\hat{m}''_\lambda(x)\right)^2 dx,$$

by the property of cubic polynomials mentioned above. This contradicts the assumption that $\hat{m}_\lambda$ minimises equation (19). Therefore, $\hat{m}_\lambda$ must be a cubic polynomial on each interval $[x_i, x_{i+1}]$.

A similar argument shows that $\hat{m}_\lambda$ must be twice continuously differentiable at the knots, completing the proof that $\hat{m}_\lambda$ is a cubic spline. □

The theorem shows that the smoothing spline is a natural cubic spline with n knots at the data points $x_1, \dots, x_n$. Such a spline consists of:

- A linear function on $(-\infty, x_1)$: 2 parameters
- Cubic polynomials on each of the $n-1$ intervals between consecutive knots: $4(n-1)$ parameters
- A linear function on (x_n, ∞): 2 parameters

This gives $2 + 4(n-1) + 2 = 4n$ parameters initially. However, at each of the n knots, we require continuity of the function, its first derivative, and its second derivative. This imposes $3n$ constraints, leaving $4n - 3n = n$ free parameters. With n free parameters and n data points $(x_1, y_1), \dots, (x_n, y_n)$, we could make the spline pass through all data points exactly—a perfect interpolating fit. This corresponds to $\lambda = 0$. For $\lambda > 0$, the penalty term in equation (19) forces the spline to be smoother, in exchange for imperfect fit.

7.3 Degrees of Freedom

As with linear regression, we can write the smoothing spline estimate in matrix form. Let $y = (y_1, \dots, y_n)^\top$ be the vector of responses, and let $\hat{y} = (\hat{m}_\lambda(x_1), \dots, \hat{m}_\lambda(x_n))^\top$ be the vector of fitted values. Then we can write

$$\hat{y} = S_\lambda y, \tag{20}$$

where $S_\lambda \in \mathbb{R}^{n \times n}$ is the **smoother matrix**. This is analogous to the hat matrix $H = X(X^\top X)^{-1}X^\top$ in linear regression (see section 2 of the level 3 notes).

The smoother matrix S_λ depends on the smoothing parameter λ and on the data points $x_1, \dots, x_n$, but not on the responses $y_1, \dots, y_n$. We omit the explicit formula for S_λ, but note that efficient algorithms exist for computing it.

Definition 7.3. The **effective degrees of freedom** of the smoothing spline estimate is

$$\mathrm{df}(\lambda) = \mathrm{tr}(S_\lambda),$$

where tr denotes the trace of a matrix (the sum of the diagonal elements).

The effective degrees of freedom measure the complexity of the fitted model:

- For $\lambda = 0$, the smoother matrix is $S_0 = I$ (the identity matrix), and we have $\mathrm{df}(0) = n$. This corresponds to interpolation, where we use all n data points.
- For $\lambda \to \infty$, the smoother matrix converges to the hat matrix of linear regression, and we have $\mathrm{df}(\infty) = 2$. This corresponds to fitting a straight line.
- For intermediate values of λ, we have $2 < \mathrm{df}(\lambda) < n$.

The effective degrees of freedom provide an alternative way to specify the amount of smoothing: choose a target value for $\mathrm{df}(\lambda)$ and find the corresponding λ.

7.4 Smoothing Splines in R

R computes smoothing splines using the built-in function `smooth.spline()`, which takes the following arguments:

- `x` and `y`: the data points.
- `spar`: the smoothing parameter. This is related to λ but uses a different scale: specifically, $\lambda = r \cdot 256^{3 \cdot \mathrm{spar} - 1}$ where r is a data-dependent scaling factor. If not specified, the function chooses a default value.
- `df`: the target degrees of freedom. If specified, the function finds the value of `spar` which gives the desired degrees of freedom.

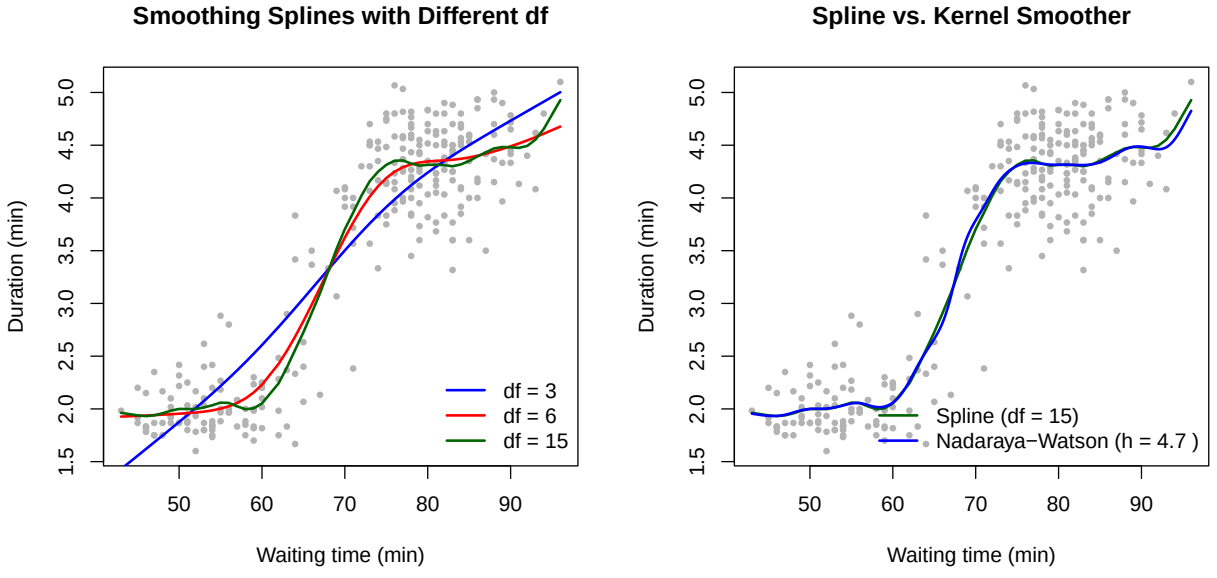
The return value is an object which contains the fitted spline. The most important components are:

- `$x` and `$y`: the fitted spline evaluated at the data points (or at a grid of points if you set the optional argument `all.knots = FALSE`).
- `$df`: the effective degrees of freedom of the fitted spline.
- `$lambda`: the smoothing parameter λ.

Example 7.2. We illustrate smoothing splines using the `faithful` dataset, which contains waiting times between eruptions and eruption durations for the Old Faithful geyser. We fit smoothing splines with different degrees of freedom and compare them to kernel methods.

```
# Load the data
data(faithful)
x <- faithful$waiting
y <- faithful$eruptions

# Sort the data for plotting
ord <- order(x)
x <- x[ord]
y <- y[ord]

# Fit smoothing splines with different df
```

```r
par(mfrow = c(1, 2))

# Left panel: varying degrees of freedom
plot(x, y, main = "Smoothing Splines with Different df",
     xlab = "Waiting time (min)", ylab = "Duration (min)",
     pch = 16, col = "grey70", cex = 0.7)

# Fit splines with df = 3, 6, and 15
fit3 <- smooth.spline(x, y, df = 3)
fit6 <- smooth.spline(x, y, df = 6)
fit15 <- smooth.spline(x, y, df = 15)

lines(fit3, col = "blue", lwd = 2)
lines(fit6, col = "red", lwd = 2)
lines(fit15, col = "darkgreen", lwd = 2)

legend("bottomright",
       legend = c("df = 3", "df = 6", "df = 15"),
       col = c("blue", "red", "darkgreen"),
       lwd = 2, bty = "n")

# Right panel: comparison with kernel smoother
plot(x, y, main = "Spline vs. Kernel Smoother",
     xlab = "Waiting time (min)", ylab = "Duration (min)",
     pch = 16, col = "grey70", cex = 0.7)

# Smoothing spline with df = 15
lines(fit15, col = "darkgreen", lwd = 2)

# Nadaraya-Watson kernel smoother with automatic bandwidth
h <- bw.nrd(x)
fit_nw <- ksmooth(x, y, kernel = "normal", bandwidth = h)
lines(fit_nw, col = "blue", lwd = 2)

legend("bottomright",
       legend = c("Spline (df = 15)",
                  paste("Nadaraya-Watson (h =", round(h, 1), ")")),
       col = c("darkgreen", "blue"),
       lwd = 2, bty = "n")
```



The left panel shows how degrees of freedom control smoothness: df = 3 gives an almost linear fit, while df = 15 follows the data closely. The right panel compares the smoothing spline (df = 15) with the Nadaraya-Watson estimator (using the normal reference rule for bandwidth selection). Both methods produce similar estimates, though the spline has better boundary behaviour.

7.5 Comparison with Kernel Methods

Both smoothing splines and kernel methods estimate the regression function m from data, but they differ in strengths and weaknesses.

Advantages of smoothing splines:

- **Global method:** The spline fits the entire dataset at once, which gives better results at the boundaries of the data range.
- **Automatic smoothness:** The solution is always twice continuously differentiable.
- **Efficient computation:** Solving a system of linear equations computes the spline, avoiding local fits at many points.

Advantages of kernel methods:

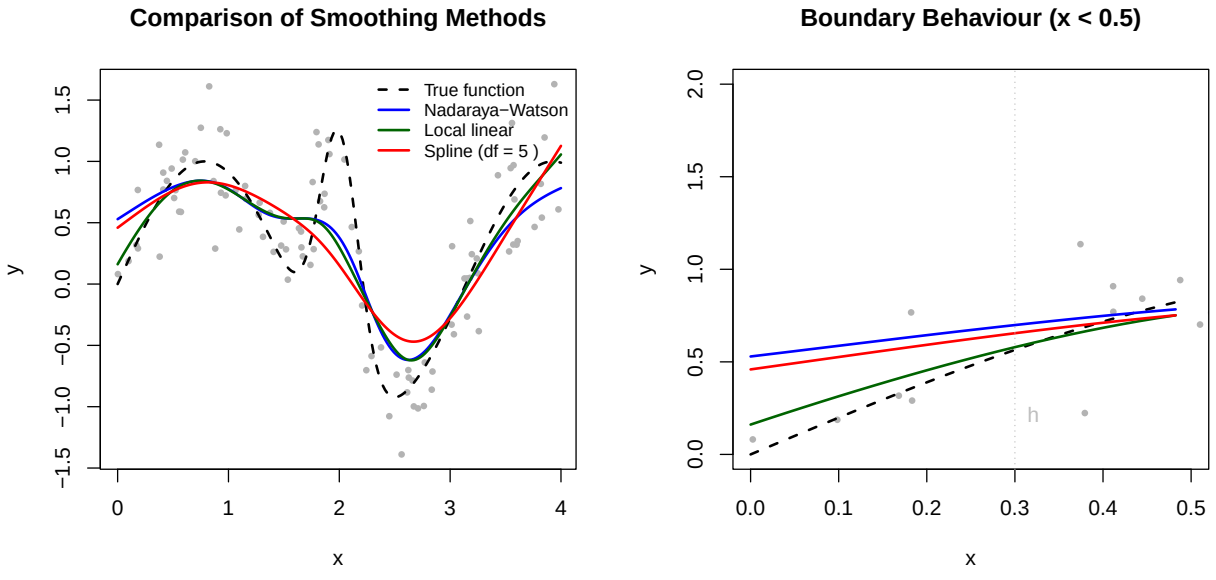
- **Local adaptation:** Kernel methods can adapt to local features of the data, for example by using different bandwidths in different regions (as in k-NN methods).
- **Geometric interpretation:** Kernel methods use weighted averages, which are easier to understand than the penalized least squares formulation of splines.
- **Robustness:** Robust kernels or trimmed extreme values make kernel methods resistant to outliers.

In practice, the choice between smoothing splines and kernel methods often depends on the specific application and the properties of the data.

Example 7.3. We compare smoothing splines with Nadaraya-Watson and local linear regression using simulated data where the true regression function is known. This allows us to assess the performance of each method, particularly near the boundaries.

```
# Generate simulated data
set.seed(123)
n <- 100
x <- sort(runif(n, 0, 4))
m_true <- function(t) sin(2*t) + 2*exp(-16*(t-2)^2)
y <- m_true(x) + rnorm(n, 0, 0.3)

# Grid for evaluation
x_grid <- seq(0, 4, length = 200)
y_true <- m_true(x_grid)

# Nadaraya-Watson estimator
nw_estimate <- function(x_new, x, y, h) {
  K <- dnorm((x_new - x) / h)
  sum(K * y) / sum(K)
}

# Local linear estimator
local_linear <- function(x_new, x, y, h) {
  w <- dnorm((x - x_new) / h)
  X <- cbind(1, x - x_new)
  beta <- solve(t(X) %*% diag(w) %*% X) %*% t(X) %*% diag(w) %*% y
  beta[1]
}

# Choose bandwidth using rule of thumb
h <- 0.3

# Compute estimates
```

```r
y_nw <- sapply(x_grid, nw_estimate, x = x, y = y, h = h)
y_ll <- sapply(x_grid, local_linear, x = x, y = y, h = h)

# Smoothing spline with fixed degrees of freedom
fit_spline <- smooth.spline(x, y, df = 5)
y_spline <- predict(fit_spline, x_grid)$y

# Plotting
par(mfrow = c(1, 2))

# Left panel: all three methods
plot(x, y, main = "Comparison of Smoothing Methods",
     xlab = "x", ylab = "y", pch = 16, col = "grey70", cex = 0.7)
lines(x_grid, y_true, col = "black", lwd = 2, lty = 2)
lines(x_grid, y_nw, col = "blue", lwd = 2)
lines(x_grid, y_ll, col = "darkgreen", lwd = 2)
lines(x_grid, y_spline, col = "red", lwd = 2)

legend("topright",
       legend = c("True function", "Nadaraya-Watson",
                  "Local linear", paste("Spline (df =", round(fit_spline$df, 1), ")")),
       col = c("black", "blue", "darkgreen", "red"),
       lty = c(2, 1, 1, 1), lwd = 2, bty = "n", cex = 0.8)

# Right panel: zoom on boundary region
plot(x, y, main = "Boundary Behaviour (x < 0.5)",
     xlim = c(0, 0.5), ylim = c(0, 2),
     xlab = "x", ylab = "y", pch = 16, col = "grey70", cex = 0.7)

# Only plot in boundary region
idx <- x_grid <= 0.5
lines(x_grid[idx], y_true[idx], col = "black", lwd = 2, lty = 2)
lines(x_grid[idx], y_nw[idx], col = "blue", lwd = 2)
lines(x_grid[idx], y_ll[idx], col = "darkgreen", lwd = 2)
lines(x_grid[idx], y_spline[idx], col = "red", lwd = 2)

# Add vertical line at x = h to show bandwidth
abline(v = h, col = "grey", lty = 3)
text(h, 0.2, "h", pos = 4, col = "grey")
```



The comparison reveals several key differences:

1. **Boundary effects**: The Nadaraya-Watson estimator (blue) shows clear bias near $x = 0$, underestimating the function. The local linear estimator (green) performs better at the boundary but still shows bias. The smoothing spline (red) handles the boundary well through its global nature.
2. **Peak capture**: All methods struggle with the sharp peak at $x = 2$, but the smoothing spline adapts well despite using a global smoothing parameter.
3. **Smoothness**: The spline produces the smoothest estimate, constrained to be twice differentiable everywhere. The kernel methods show more local variation.

The choice of method depends on the specific requirements—boundary behaviour, local adaptation, or global smoothness.

Methods for selecting the smoothing parameter λ or the effective degrees of freedom using cross-validation are discussed in section 8.

Summary

- Smoothing splines balance fit and smoothness using a penalty term on curvature.
- The solution is a natural cubic spline with knots at the data points.
- Smoothness is controlled by λ or equivalently by effective degrees of freedom.
- Compared to kernel methods, splines are global and handle boundaries better.

8 Cross-validation

In this section we will discuss methods for choosing the bandwidth h for kernel-based methods, the size k of the neighbourhood used in k-nearest neighbour regression, and the smoothing parameter λ (or equivalently the degrees of freedom) for spline smoothing. The methods we will use here are based on cross-validation.

The idea of cross-validation is to measure goodness of fit by comparing each sample y_i to a fitted value $\tilde{y}_i$, where the fitted values $\tilde{y}_i$ are computed from a subset of the data which excludes x_i. This way, a method cannot achieve a misleadingly good fit by overfitting the data. There are different ways to implement this idea:

- In ***k*-fold cross-validation**, the data is partitioned into k subsamples of approximately equal size. Only k models are fitted, each one leaving out the data from one subsample. Then for every i there is exactly one model which does not use (x_i, y_i), and to compute the fitted value for (x_i, y_i), we use this model. Since fewer data are used to fit each model, this method gives less accurate results than leave-one-out cross-validation. For this method to work, it is important that the subsamples are independent of each other.
- In **leave-one-out cross-validation**, a separate model is fitted for each $i \in \{1, \dots, n\}$, leaving out just the sample (x_i, y_i) for this model. This is the special case of k-fold cross-validation where $k = n$. Since n models need to be fitted to the data, for large n the method can be computationally expensive.

8.1 Regression

In linear regression, we find the regression line by minimising the residual sum of squares. One could be tempted to try the same approach here and to find the "best" h by minimising

$$r(h) := \sum_{i=1}^{n} (y_i - \hat{m}_h(x_i))^2.$$

Unfortunately, this approach does not work: for $h \downarrow 0$ we have $\hat{m}_h(x_i) \to y_i$ and thus $r(h) \to 0$. For this reason, minimising $r(h)$ always finds $h = 0$ as the optimal value of h. To solve the problem we use leave-one-out cross-validation and minimise

$$r_{\text{LOO}}(h) := \sum_{i=1}^{n} (y_i - \hat{m}_h^{(i)}(x_i))^2$$

instead, where $\hat{m}^{(i)}$ is the kernel regression estimate computed without using sample i:

$$\hat{m}_h^{(i)}(x) = \frac{\sum_{j \mid j \neq i} K_h(x - x_j) y_j}{\sum_{j \mid j \neq i} K_h(x - x_j)}$$

for the Nadaraya-Watson estimator and similarly for local polynomial regression.

A similar approach can be used to find the optimal k for a k-nearest neighbour estimate.

For spline smoothing (see section 7), we need to choose the smoothing parameter λ or, equivalently, the effective degrees of freedom. The same leave-one-out cross-validation approach can be used: we minimise

$$r_{\text{LOO}}(\lambda) := \sum_{i=1}^{n} (y_i - \hat{s}_\lambda^{(i)}(x_i))^2,$$

where $\hat{s}_\lambda^{(i)}$ is the smoothing spline fitted to the data with sample i omitted.

In R, the `smooth.spline()` function provides automatic parameter selection via cross-validation when the argument `cv = TRUE` is used:

```
# Fit smoothing spline with CV-selected smoothing parameter
fit <- smooth.spline(x, y, cv = TRUE)
```

By default, `smooth.spline()` uses ordinary (leave-one-out) cross-validation to select the effective degrees of freedom.

8.2 Kernel Density Estimation

When using kernel density estimation in practice, we need to choose the bandwidth h. One idea for finding a good h is by using maximum likelihood estimation: We could try to choose h to maximize the likelihood

$$L(h; x_1, \dots, x_n) = \prod_{i=1}^{n} \hat{f}_h(x_i),$$

but this gives the solution $h = 0$. So, instead we maximize the leave-one-out likelihood, which is given by

$$L_{\mathrm{LOO}}(h; x_1, \dots, x_n) = \prod_{i=1}^{n} \hat{f}_h^{(i)}(x_i).$$

This technique is known as **maximum likelihood cross-validation**. When this method is used in practice, it is advantageous to maximise the logarithm

$$\begin{aligned}\mathcal{L}_{\mathrm{LOO}}(h; x_1, \dots, x_n) &:= \log\Big(L_{\mathrm{LOO}}(h; x_1, \dots, x_n)\Big) \\ &= \log\Big(\prod_{i=1}^{n} \hat{f}_h^{(i)}(x_i)\Big) \\ &= \sum_{i=1}^{n} \log(\hat{f}_h^{(i)}(x_i))\end{aligned}$$

instead of L_{LOO}, since the product in the definition of the likelihood can be strongly affected by numerical errors.

An alternative method to find a good h considers the integrated mean squared error (IMSE) as a measure for the error. This is the same quantity we also used to derive our theoretical results:

$$\mathrm{IMSE}(\hat{f}_h) = \int_{-\infty}^{\infty} \mathrm{MSE}(\hat{f}_h(x))\, dx.$$

Unfortunately, the h which minimises this expression depends on properties of f and in section 3.2 we were only able to find a heuristic "plug-in" estimate to approximate the best h. The following lemma shows how a variant of leave-one-out cross-validation can be used to estimate the optimal h from data.

Lemma 8.1. *Let $\hat{f}_h$ be the kernel density estimate with bandwidth h and let*

$$e(h) := \int \mathbb{E}(\hat{f}_h(x)^2) dx - 2 \int \mathbb{E}(\hat{f}_h(x)) f(x) dx.$$

Then the following statements hold.

(a) We have $\mathrm{IMSE}(\hat{f}_h) = e(h) + \mathrm{const}$*, where* const *stands for a term which does not depend on h.*

(b) Let $\hat{f}_h^{(i)}$ be the kernel density estimate computed from the data with sample i omitted. Then

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

is an (approximately) unbiased estimator for $e(h)$.

Proof. The first statement can be shown by expanding the square in the definition of the (I)MSE:

$$\begin{aligned}
\mathrm{IMSE}(\hat{f}_h) &= \int_{-\infty}^{\infty} \mathrm{MSE}(\hat{f}_h(x))\,dx \\
&= \int_{-\infty}^{\infty} \mathbb{E}\Big(\big(\hat{f}_h(x) - f(x)\big)^2 \Big)\,dx \\
&= \int_{-\infty}^{\infty} \mathbb{E}\Big(\hat{f}_h(x)^2 - 2\hat{f}_h(x)f(x) + f(x)^2 \Big)\,dx \\
&= \int_{-\infty}^{\infty} \mathbb{E}\Big(\hat{f}_h(x)^2 \Big)\,dx - 2\int_{-\infty}^{\infty} \mathbb{E}\Big(\hat{f}_h(x)f(x) \Big)\,dx \\
&\qquad\qquad + \int_{-\infty}^{\infty} \mathbb{E}\Big(f(x)^2 \Big)\,dx \\
&= \int_{-\infty}^{\infty} \mathbb{E}\Big(\hat{f}_h(x)^2 \Big)\,dx - 2\int_{-\infty}^{\infty} \mathbb{E}(\hat{f}_h(x))f(x)\,dx \\
&\qquad\qquad + \int_{-\infty}^{\infty} f(x)^2\,dx,
\end{aligned}$$

where we used the fact that $f(x)$ is not random. Since the last term does not depend on h, the first claim is proved.

For the second statement we need to consider the kernel density estimates computed from random data $X_1, \dots, X_n \sim f$, i.i.d. We have to show that

$$\mathbb{E}(\mathrm{CV}(h)) = e(h) = \int \mathbb{E}(\hat{f}_h(x)^2)dx - 2\int \mathbb{E}(\hat{f}_h(x))f(x)dx.$$

For the first term in the definition of CV we get

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

where we used Fubini's theorem to exchange the expectation with the integral. For the second term we have

$$\begin{aligned}
\mathbb{E}\Big(\frac{1}{n}\sum_{i=1}^{n} \hat{f}_h^{(i)}(X_i) \Big) &= \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\Big(\hat{f}_h^{(i)}(X_i) \Big) \\
&= \mathbb{E}\Big(\hat{f}_h^{(1)}(X_1) \Big),
\end{aligned}$$

since the X_i are independent and identically distributed. Since the estimator $\hat{f}_h^{(1)}$ is computed from $X_2, \dots, X_n$, which are independent of X_1, we can evaluate the expectation on the right-hand side by first computing $\mathbb{E}\Big(\hat{f}_h^{(1)}(x) \Big)$ as a function of x, then using X_1 in place of x, and computing the expectation over X_1 afterwards. This gives

$$\begin{aligned}
\mathbb{E}\Big(\frac{1}{n}\sum_{i=1}^{n} \hat{f}_h^{(i)}(X_i) \Big) &= \mathbb{E}\Big(\mathbb{E}(\hat{f}_h^{(1)}(X_1) \mid X_1) \Big) \\
&= \int \mathbb{E}(\hat{f}_h^{(1)}(x))\, f(x)\, dx,
\end{aligned}$$

since $X_1 \sim f$. Finally, since $\hat{f}_h^{(1)}$ and $\hat{f}_h$ only differ in the sample size, by using $n-1$ and n samples respectively, we have

$$\mathbb{E}(\hat{f}_h^{(1)}(x)) \approx \mathbb{E}(\hat{f}_h(x)).$$

Combining these equations completes the proof. □

Using the result from the lemma we see that we can choose h to minimise $\mathrm{CV}(h)$ in order to get a candidate for the bandwidth h. This procedure is known as **integrated squared error cross-validation**.

These two approaches differ slightly in the optimal h, but the first one is easier to implement as there is no integration involved.

Summary

- Minimising residual sum of squares directly leads to $h = 0$ (overfitting); cross-validation avoids this by evaluating fitted values on held-out data.
- In leave-one-out cross-validation, each observation is predicted using a model fitted without that observation; k-fold CV is a faster approximation.
- For regression, we minimise $r_{\text{LOO}}(h) = \sum_i (y_i - \hat{m}_h^{(i)}(x_i))^2$.
- For KDE, two approaches: maximum likelihood CV (maximise $\sum_i \log \hat{f}_h^{(i)}(x_i)$) or integrated squared error CV (minimise CV(h), which estimates IMSE).

9 Examples

To conclude these notes, we give three examples where we use cross-validation to choose the tuning parameter in kernel density estimation, kernel regression, and k-nearest neighbour regression.

9.1 Kernel Density Estimation

Here we show how to find a good bandwidth for kernel density estimation, by using cross-validation. From lemma 8.1 we know that we can choose the h which minimises

$$\mathrm{CV}(h) = \int \hat{f}_h(x)^2 dx - \frac{2}{n}\sum_{i=1}^{n} \hat{f}_h^{(i)}(x_i) =: A - B. \tag{21}$$

We will consider the snow fall dataset again:

```
# data from https://teaching.seehuhn.de/data/buffalo/
buffalo <- read.csv("data/buffalo.csv")
x <- buffalo$snowfall
n <- length(x)
```

In order to speed up the computation of the $\hat{f}_h^{(i)}$, we implement the kernel density estimate "by hand". Thus, instead of using the built-in function `density`, we use the formula

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

from definition 1.2.

We use a number of "tricks" in the R code:

- For numerically computing the integral of $\hat{f}_h^2$ in term A we evaluate $\hat{f}_h$ on a grid of x-values, say $\tilde{x}_1, \dots, \tilde{x}_m$.
- When computing $\hat{f}_h(\tilde{x}_j)$ we need to compute all pair differences $\tilde{x}_j - x_i$. In R, this can efficiently be done using the command `outer(x, x.tilde, "-")`, which returns the pair differences as an $n \times m$ matrix.
- Here we use a Gaussian kernel, so that K_h can be evaluated using `dnorm(..., sd = h)` in R. This function can be applied to the matrix of pair differences; the result is a matrix `K` where row i, column j stores the value $K_h(\tilde{x}_j - x_i)$.
- The kernel density estimate $\hat{f}_h$ now corresponds to the column means of the matrix `K`. In R, these can be efficiently computed using the command `colMeans()`.
- Term A in equation (21) can now be approximated by the sum of the $\hat{f}_h(\tilde{x}_j)$, multiplied by the distance between the grid points:

$$A = \int \hat{f}_h(x)^2\, dx \approx \sum_{j=1}^{m} \hat{f}_h(\tilde{x}_j)^2\, \Delta\tilde{x}.$$

- To compute term B in equation (21), we can use the formula

$$\begin{aligned}\sum_{j=1}^{n} \hat{f}_h^{(j)}(x_j) &= \sum_{j=1}^{n} \frac{1}{n-1} \sum_{i \neq j} K_h(x_j - x_i) \\ &= \frac{1}{n-1} \sum_{\substack{i,j=1 \\ i \neq j}}^{n} K_h(x_j - x_i).\end{aligned}$$

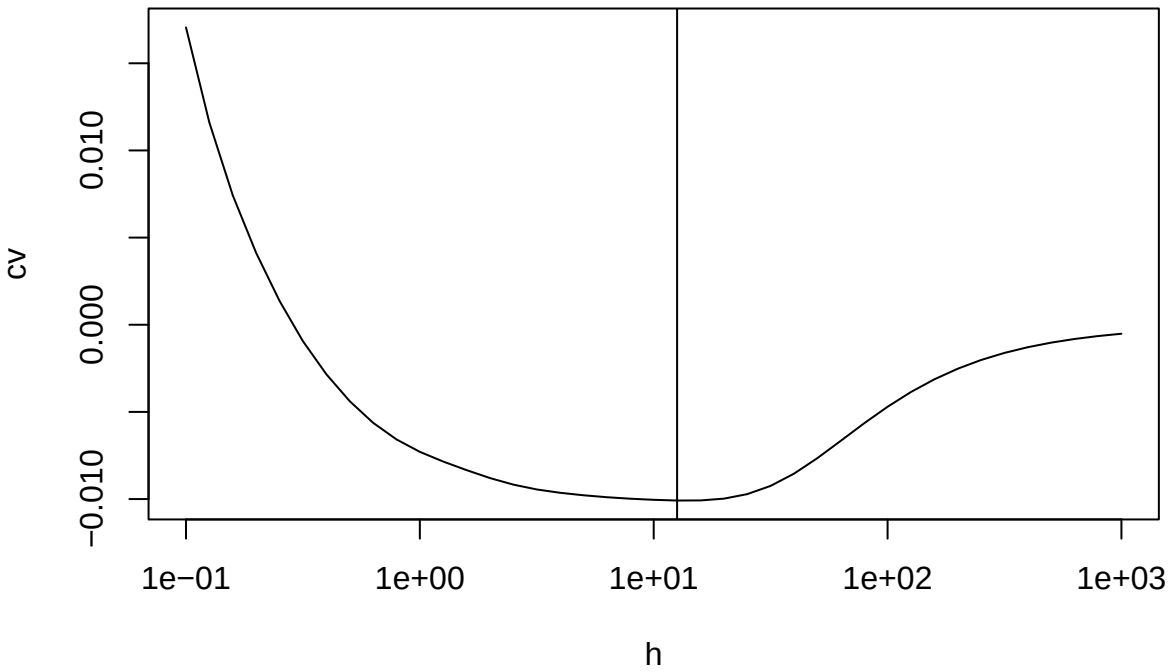
  Here we can use `outer()` again, and then implement the condition $i \neq j$ by setting the matrix elements corresponding to $i = j$ equal to 0 before taking the sum.

Using these ideas, we can implement the function $\mathrm{cv}(h)$ in R as follows:

```r
cv.h <- function(h) {
    x.min <- min(x) - 3*h
    x.max <- max(x) + 3*h
    m <- 1000
    dx <- (x.max - x.min) / (m - 1)
    x.tilde <- seq(x.min, x.max, length.out = m)

    K <- dnorm(outer(x, x.tilde, "-"), sd = h)
    f.hat <- colMeans(K)
    A <- sum(f.hat^2 * dx)

    K <- dnorm(outer(x, x, "-"), sd = h)
    diag(K) <- 0
    B <- 2 * sum(K) / (n-1) / n

    return(A - B)
}
```

Finally, we evaluate the function `cv.h()` on a grid of h-values to find a good value of h:

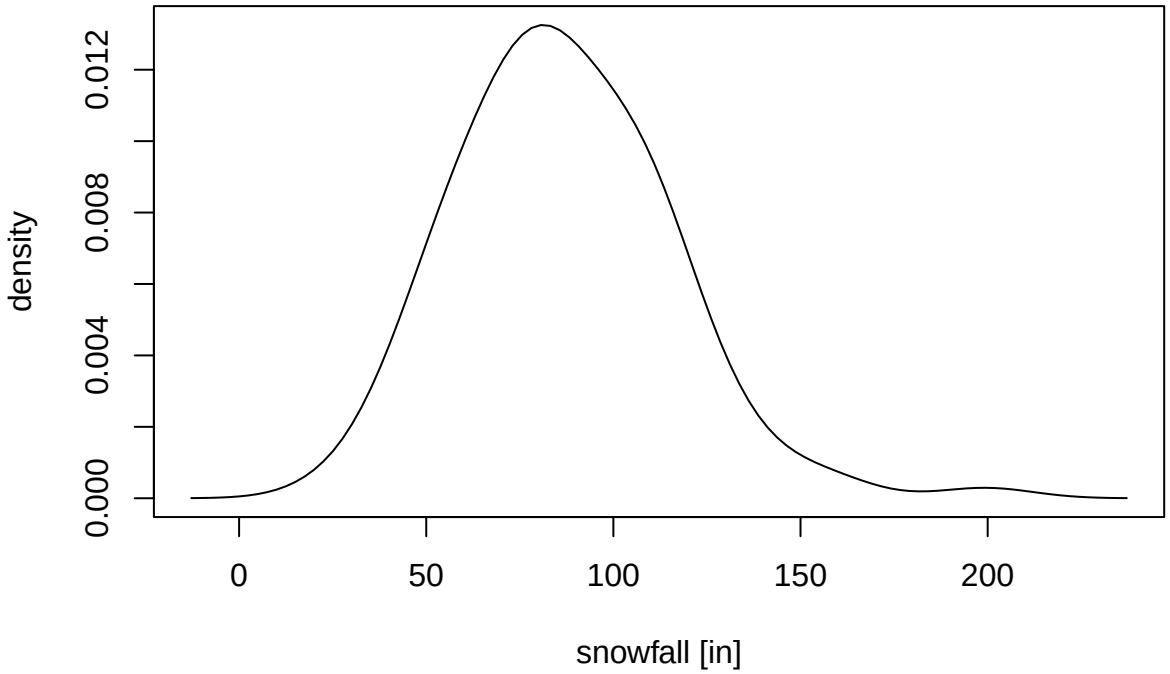
```r
h <- 10^seq(-1, 3, length.out = 41)
cv <- numeric(length(h))
for (i in seq_along(h)) {
    cv[i] <- cv.h(h[i])
}
plot(h, cv, log="x", type = "l")

best.h <- h[which.min(cv)]
abline(v = best.h)
```



The optimal bandwidth is $h = 12.59$. The kernel density estimate using this h is shown in the following figure.

```r
x.min <- min(x) - 3*best.h
x.max <- max(x) + 3*best.h
m <- 100
x.tilde <- seq(x.min, x.max, length.out = m)
```

```r
K <- dnorm(outer(x, x.tilde, "-"), sd = best.h)
f.hat <- colMeans(K)

plot(x.tilde, f.hat, type = "l",
     xlab = "snowfall [in]", ylab = "density")
```



9.2 Kernel Regression

To illustrate cross-validation for the different smoothing methods, we use the `faithful` dataset again.

```r
x <- faithful$eruptions
y <- faithful$waiting
```

We compare the methods using the leave-one-out mean squared error

$$r(h) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{m}^{(i)}(x_i))^2.$$

We start by considering the Nadaraya-Watson estimator. Here we have to compute

$$\hat{m}_h^{(i)}(x_i) = \frac{\sum_{j=1,j\neq i}^{n} K_h(x_i - x_j)y_j}{\sum_{j=1,j\neq i}^{n} K_h(x_i - x_j)}$$

for all $i \in \{1, \ldots, n\}$. To evaluate this expression in R, we use the same ideas as before:

- We use `outer(x, x, "-")` to compute all pair differences $x_i - x_j$.
- We use `dnorm(..., sd = h)` to compute K_h.
- We can obtain the leave-one-out estimate by setting the diagonal of K to zero.

One new idea is needed to compute the products $K_h(x_i - x_j)y_j$ in an efficient way:

- If we "multiply" a matrix `K` to a vector `y` using `*` (instead of using `%*%` for the usual matrix vector multiplication), the product is performed element-wise. If `y` has as many elements as `K` has rows, then the result is the matrix $(k_{ij}y_i)_{i,j}$, *i.e.* each row of `K` is multiplied with the corresponding element of `y`.

Combining these ideas, we get the following function to compute the leave-one-out estimate for the mean squared error of the Nadaraya-Watson estimator:

```
r.NW <- function(h) {
    K <- dnorm(outer(x, x, "-"), sd = h)

    # compute a leave-one-out estimate
    diag(K) <- 0

    m.hat <- colSums(K*y) / colSums(K)
    mean((m.hat - y)^2)
}
```

We will also consider local linear smoothing, *i.e.* local polynomial smoothing where the degree p of the polynomials is $p = 1$. As we have seen in the section about Polynomial Regression with Weights, the local linear estimator can be computed as

$$\hat{m}_h(x) = e_0^\top (X^\top W X)^{-1} X^\top W y,$$

where X and W are defined as in equations (17) and (16). Here we use the "linear" case, $p = 1$. For this case it is easy to check that we have

$$X^\top W X = \begin{pmatrix} \sum_j K_h(x - x_j) & \sum_j K_h(x - x_j) x_j \\ \sum_j K_h(x - x_j) x_j & \sum_j K_h(x - x_j) x_j^2 \end{pmatrix}$$

and

$$X^\top W y = \begin{pmatrix} \sum_j K_h(x - x_j) y_j \\ \sum_j K_h(x - x_j) x_j y_j \end{pmatrix}.$$

Using the formula

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

for the inverse of a general 2×2-matrix, we find

$$\hat{m}_h(x) = \frac{T_1 T_2 - T_3 T_4}{B_1 B_2 - B_3^2},$$

where

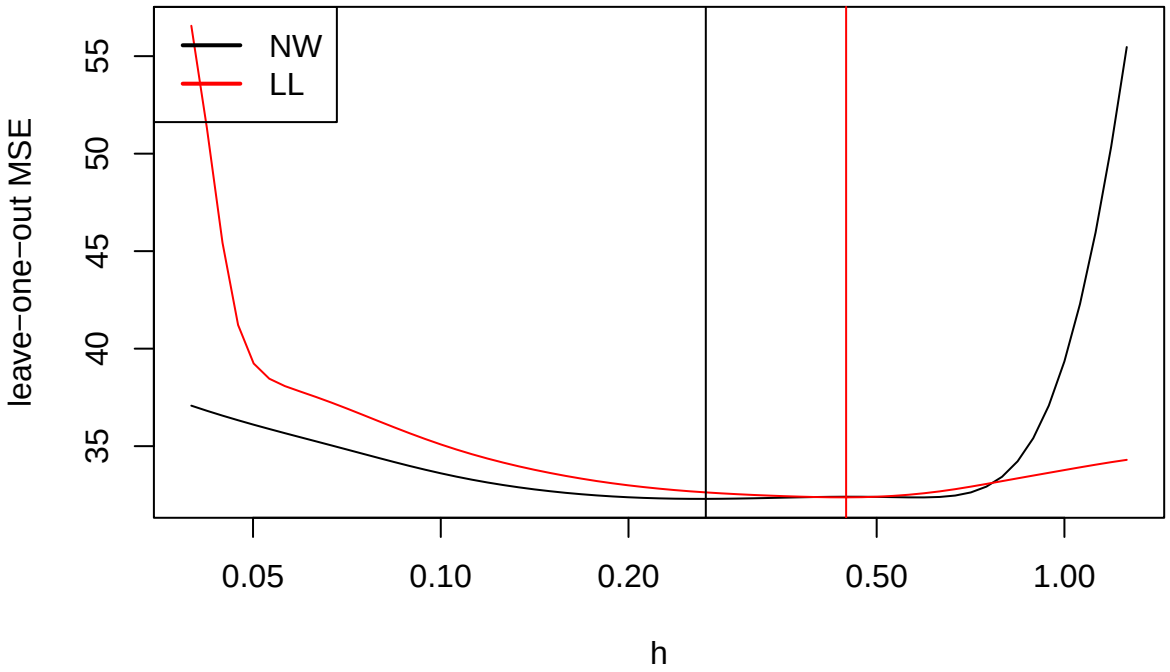
$$\begin{aligned} T_1 &= \sum_{j=1}^n K_h(x - x_j) y_j, \\ T_2 &= \sum_{j=1}^n K_h(x - x_j) x_j (x_j - x), \\ T_3 &= \sum_{j=1}^n K_h(x - x_j) x_j y_j, \\ T_4 &= \sum_{j=1}^n K_h(x - x_j) (x_j - x), \\ B_1 &= \sum_{j=1}^n K_h(x - x_j), \\ B_2 &= \sum_{j=1}^n K_h(x - x_j) x_j^2, \\ B_3 &= \sum_{j=1}^n K_h(x - x_j) x_j. \end{aligned}$$

As before, for a leave-one-out estimate we need to compute these sums over all $j \neq i$. Since each of the seven terms listed above contains the term $K_h(x - x_j)$ inside the sum, we can achieve this by setting the corresponding elements of the matrix `K` to zero.

```
r.LL <- function(h) {
    dx <- outer(x, x, "-")
    K <- dnorm(dx, sd = h)

    # compute a leave-one-out estimate
    diag(K) <- 0

    T1 <- colSums(y*K)
    T2 <- colSums(x*dx*K)
    T3 <- colSums(x*y*K)
    T4 <- colSums(dx*K)
    B1 <- colSums(K)
    B2 <- colSums(x^2*K)
    B3 <- colSums(x*K)
    m.hat <- (T1*T2 - T3*T4) / (B1*B2 - B3^2)

    mean((m.hat - y)^2)
}
```

Now we evaluate the function `r.NW()` and `r.LL()` on a grid of h-values to find the optimal h for each method.

```
h <- 10^seq(-1.4, 0.1, length.out = 61)
mse.nw <- numeric(length(h))
mse.ll <- numeric(length(h))
for (i in seq_along(h)) {
    mse.nw[i] <- r.NW(h[i])
    mse.ll[i] <- r.LL(h[i])
}
plot(h, mse.nw, log="x", type = "l", ylim = range(mse.nw, mse.ll),
     ylab = "leave-one-out MSE")
lines(h, mse.ll, col="red")

best.h.NW <- h[which.min(mse.nw)]
abline(v = best.h.NW)
best.h.LL <- h[which.min(mse.ll)]
abline(v = best.h.LL, col="red")

legend("topleft", legend = c("NW", "LL"), col = c("black", "red"),
       lwd = 2)
```

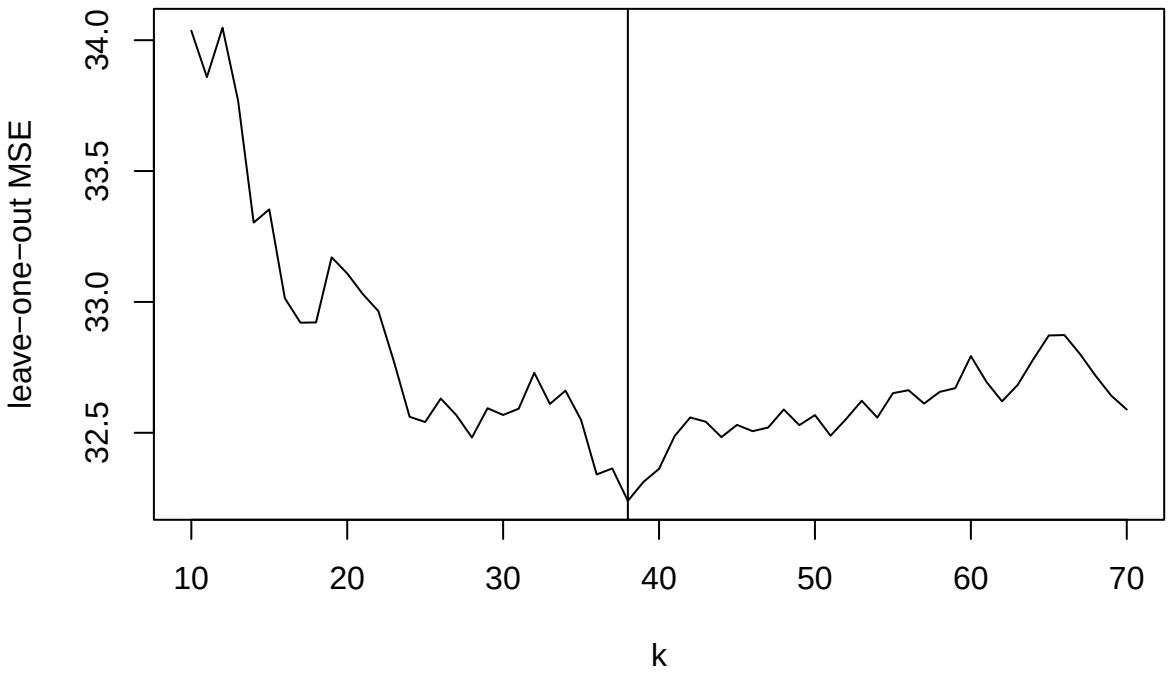


As expected, the optimal bandwidth for local linear regression is larger than for the Nadaraya-Watson estimator.

9.3 *k*-Nearest Neighbour Regression

To conclude this section we use leave-one-out cross-validation to determine the optimal k for k-nearest neighbour regression. Unlike KDE and kernel regression, where clever use of matrix operations allowed us to compute all leave-one-out predictions efficiently, kNN offers no such shortcut: removing observation i can change which points are the k nearest neighbours for every other observation. We therefore resort to a nested loop over both k values and observations, fitting a separate model for each combination. For this reason, the code in this section is much slower to run than the code in the previous sections.

```
k <- 10:70
mse.knn <- numeric(length(k))
for (j in seq_along(k)) {
    y.pred <- numeric(length(x))
    for (i in seq_along(x)) {
        m <- knn.reg(data.frame(x = x[-i]),
                     y = y[-i],
                     test = data.frame(x = x[i]),
                     k = k[j])
        y.pred[i] <- m$pred
    }
    mse.knn[j] <- mean((y - y.pred)^2)
}
plot(k, mse.knn, type = "l",
     ylab = "leave-one-out MSE")
best.k <- k[which.min(mse.knn)]
abline(v = best.k)
```

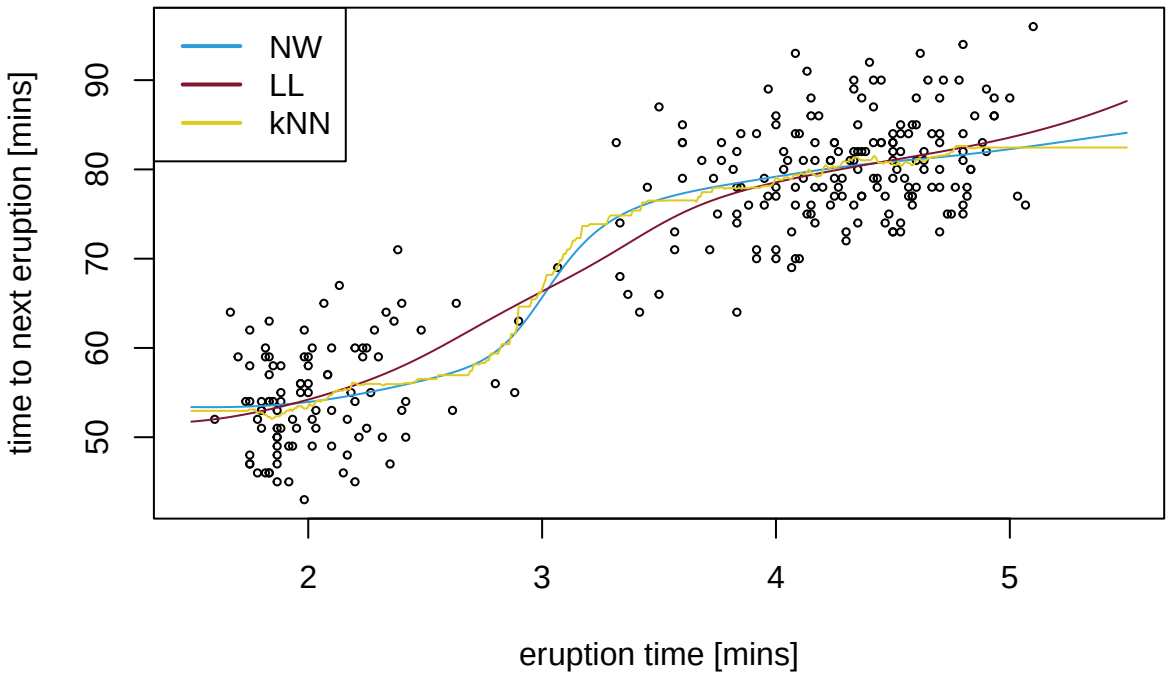


We note that the leave-one-out mean squared error for kNN is smaller than it is for Nadaraya-Watson or local linear regression, in the case of this dataset. Given the structure of the data, with different regions having very different densities of x-values, it makes sense that a method which chooses the bandwidth "adaptively" performs better.

To conclude, we show the optimal regression curves for the three smoothing methods together in one plot.

```
x.tilde <- seq(1.5, 5.5, length.out = 501)

K <- dnorm(outer(x, x.tilde, "-"), sd = best.h.NW)
m.NW <- colSums(K*y) / colSums(K)

dx <- outer(x, x.tilde, "-")
K <- dnorm(dx, sd = best.h.LL)
T1 <- colSums(y*K)
T2 <- colSums(x*dx*K)
T3 <- colSums(x*y*K)
T4 <- colSums(dx*K)
B1 <- colSums(K)
B2 <- colSums(x^2*K)
B3 <- colSums(x*K)
m.LL <- (T1*T2 - T3*T4) / (B1*B2 - B3^2)

m <- knn.reg(data.frame(x),
             y = y,
             test = data.frame(x=x.tilde),
             k = best.k)
m.kNN <- m$pred

colours <- c("#2C9CDA", "#811631", "#E0CA1D")
plot(x, y, xlim = range(x.tilde), cex = .5,
     xlab = "eruption time [mins]",
     ylab = "time to next eruption [mins]")
lines(x.tilde, m.NW, col = colours[1])
lines(x.tilde, m.LL, col = colours[2])
lines(x.tilde, m.kNN, col = colours[3])
legend("topleft", legend = c("NW", "LL", "kNN"), col = colours,
```

```
       lwd = 2)
```



Summary

- Cross-validation for kernel density estimation uses integrated squared error or maximum likelihood criteria to select bandwidth h.
- For kernel regression, leave-one-out cross-validation minimises $\sum_i (y_i - \hat{m}_h^{(i)}(x_i))^2$ where $\hat{m}_h^{(i)}$ excludes observation i.
- Efficient implementation uses matrix operations with `outer()` to compute all pairwise differences and kernel weights simultaneously.
- The `faithful` dataset example shows that k-NN regression can outperform fixed-bandwidth methods when data density varies across the domain.
- Optimal parameters differ between methods: local linear regression typically requires larger bandwidth than Nadaraya-Watson due to reduced boundary bias.