

# MATH5714 Linear Regression, Robustness and Smoothing

Jochen Voss

University of Leeds, Semester 1, 2022/23

## Contents

<b>Preface</b>	<b>2</b>
<b>1 Kernel Density Estimation</b>	<b>3</b>
1.1 Histograms . . . . .	3
1.2 Kernel Density Estimation . . . . .	5
1.3 Kernel Density Estimation in R . . . . .	9
<b>Interlude: Kernel Density Estimation in R</b>	<b>12</b>
Direct Implementation . . . . .	12
Speed Improvements . . . . .	12
<b>2 The Bias of Kernel Density Estimates</b>	<b>16</b>
2.1 A Statistical Model . . . . .	16
2.2 The Bias of the Estimate . . . . .	16
2.3 Moments of Kernels . . . . .	17
2.4 The Bias for Small Bandwidth . . . . .	18
<b>3 The Variance of Kernel Density Estimates</b>	<b>19</b>
3.1 Variance . . . . .	19
3.2 Mean Squared Error . . . . .	21
3.3 Optimal Bandwidth . . . . .	21
<b>4 Kernel Density Estimation in Practice</b>	<b>23</b>
4.1 Integrated Error . . . . .	23
4.2 Choice of Kernel . . . . .	23
4.3 Bandwidth Selection . . . . .	24
4.4 Higher Dimensions . . . . .	26
<b>5 Kernel Smoothing</b>	<b>27</b>
5.1 The Nadaraya-Watson Estimator . . . . .	27
5.2 Estimation Error . . . . .	29
<b>6 Local Polynomial Regression</b>	<b>33</b>
6.1 Linear Regression with Weights . . . . .	33
6.2 Polynomial Regression . . . . .	33
6.3 Polynomial Regression with Weights . . . . .	35
6.4 Special Cases . . . . .	35
<b>7 <math>k</math>-Nearest Neighbour Regression</b>	<b>39</b>
7.1 Definition of the Estimator . . . . .	39
7.2 Properties . . . . .	39
7.3 Numerical Experiment . . . . .	39
7.4 Variants of the Method . . . . .	40
<b>8 Cross-validation</b>	<b>42</b>
8.1 Regression . . . . .	42

8.2	Kernel Density Estimation . . . . .	42
<b>9</b>	<b>Examples</b>	<b>45</b>
9.1	Kernel Density Estimation . . . . .	45
9.2	Kernel Regression . . . . .	47
9.3	$k$ -Nearest Neighbour Regression . . . . .	50

## Preface

From previous modules we know how to fit a regression line through points  $(x_1, y_1), \dots, (x_n, y_n) \in \mathbb{R}^2$ . The underlying model here is described by the equation

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

for all  $i \in \{1, 2, \dots, n\}$ , and the aim is to find values for the intercept  $\alpha$  and the slope  $\beta$  such that the residuals  $\varepsilon_i$  are as small as possible. This procedure, linear regression, and its extensions are discussed in the level 3 component of the module.

In the level 5 component of this module, we will discuss “smoothing” which is a technique which can be used when linear models are no longer appropriate for the data. An example of such a situation is illustrated in figure 1.

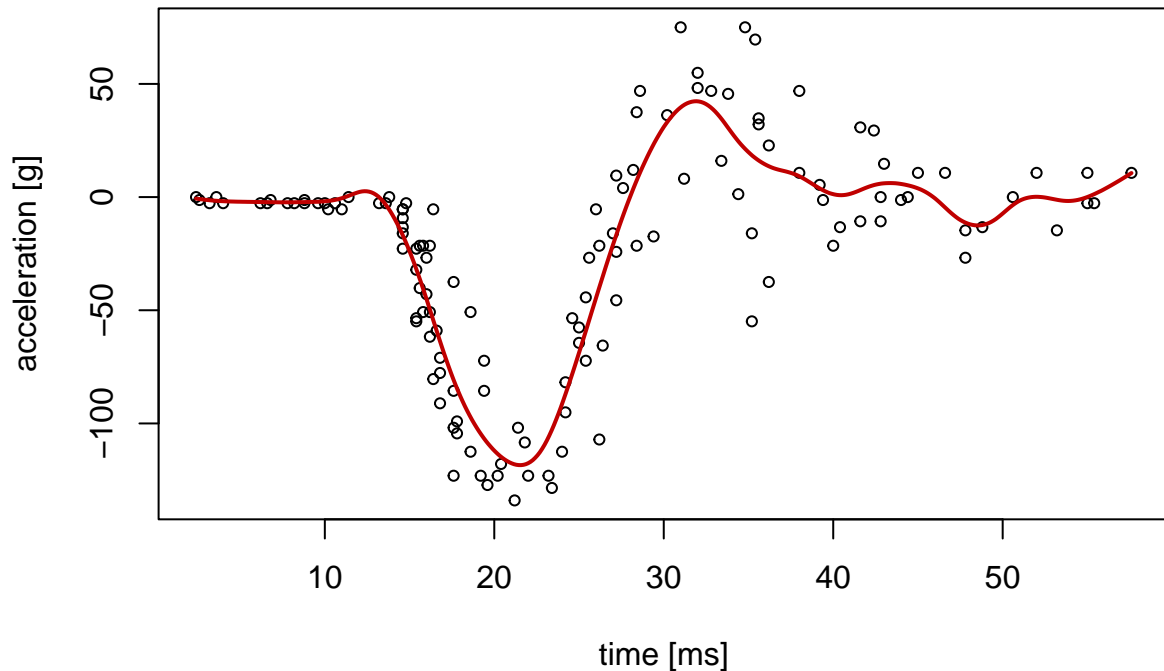


Figure 1: An illustration of a dataset where a linear (straight line) model is not appropriate. The data represents a series of measurements of head acceleration in a simulated motorcycle accident, used to test crash helmets (the `mcycle` dataset from the `MASS` R package).

# 1 Kernel Density Estimation

In this section we discuss the topic of “Kernel Density Estimation”. Here we suppose we are given data  $x_1, \dots, x_n \in \mathbb{R}^d$  from an unknown probability density, say  $f$ . Our objective is to estimate the density  $f$ . This section lays the foundations for many of the following topics.

## 1.1 Histograms

### 1.1.1 Probability Densities

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

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

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

A density  $f$  is large in regions where  $X$  is very likely to take values, and small in regions where  $X$  is less likely to take values. If  $f(x) = 0$  for all  $x$  in a region, that means that  $X$  never takes values there. 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.

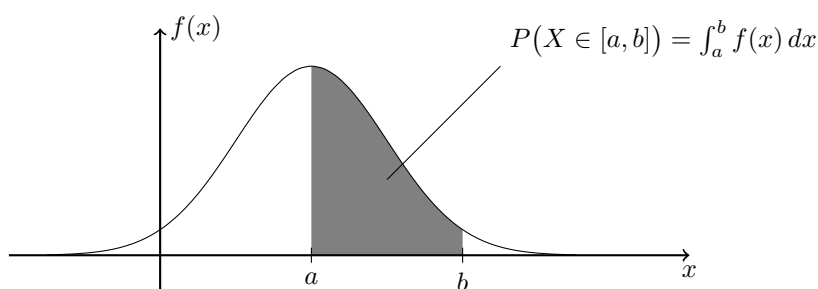


Figure 2: An illustration of how the area under the graph of a density can be interpreted as a probability.

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

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

### 1.1.2 Histograms

In the univariate case, *i.e.* for  $d = 1$ , 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 in 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 data set.) 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 histograms splits the range of the data into “buckets”, and for every bucket  $[a, b]$  it shows a bar where the height is proportional the number of samples in the bucket. I am ignoring the case that a sample may fall exactly on the boundary between two buckets here; in reality all but one buckets need to be half-open intervals, for example  $[40, 45]$ ,  $(45, 50]$ , ...,  $(95, 100]$ .

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

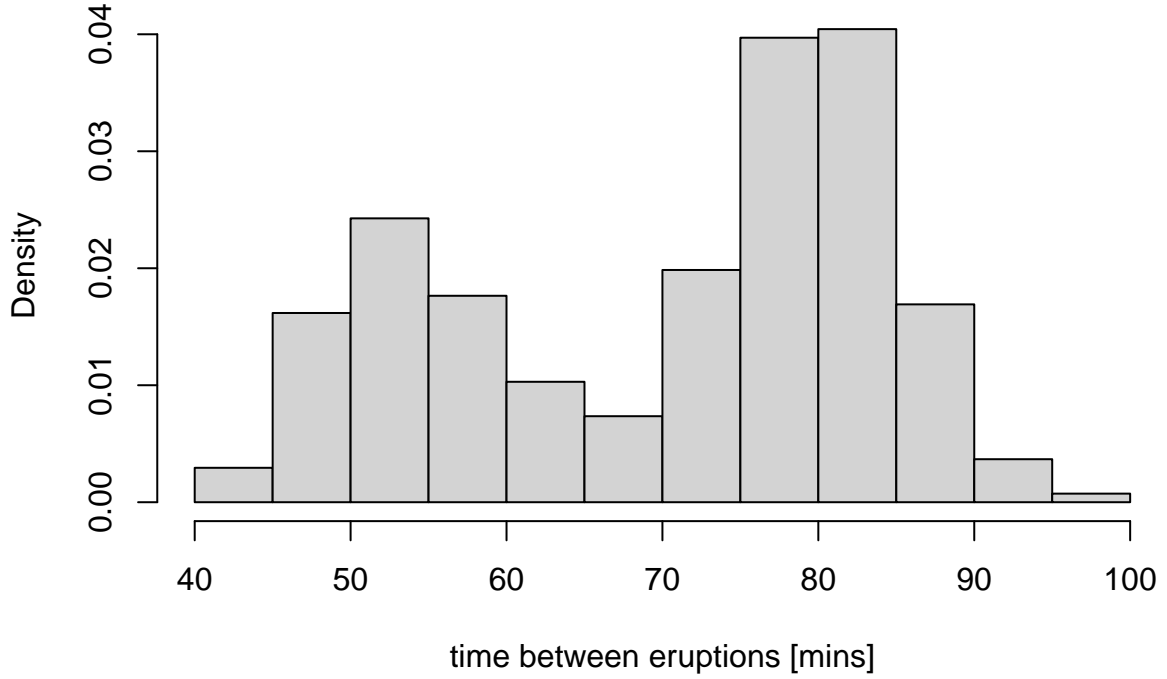


Figure 3: This figure shows how a histogram can be used to approximate a probability density. From the plot one can see that the density of the waiting times distribution seems to be bi-modal with peaks around 53 and 80 minutes.

$h_{a,b}$  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

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

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

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

As expected, the height of the bar for the bucket  $[a, b]$  is proportional to the number  $n_{a,b}$  of samples in the bucket.

### 1.1.3 Choice of Buckets

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  $f$ . This is for example demonstrated in figure 4.

```
set.seed(1)
x <- rnorm(50)
hist(x, probability = TRUE, main = NULL,
     breaks = seq(-2.5, 2.5, length.out = 500))
```

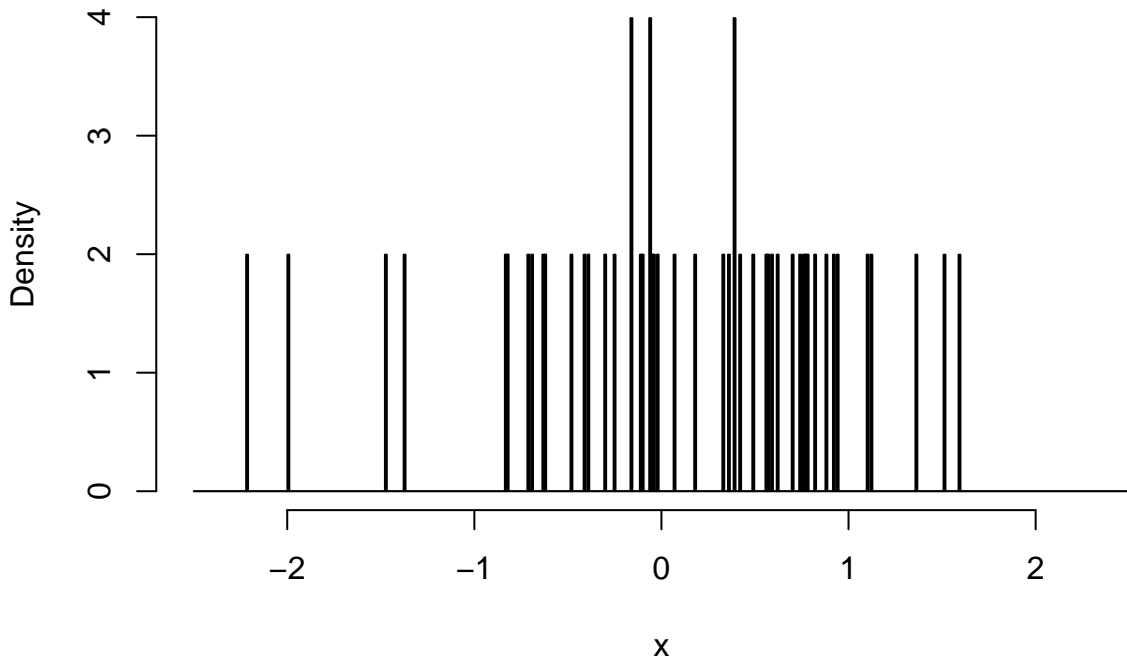


Figure 4: This figure demonstrates the consequences of choosing a too small bucket size.

Finally, even if reasonable bucket sizes are chosen, the result can depend quite strongly on the exact locations of these buckets. To illustrate this effect, we consider a dataset about the annual amount of snow falling in Buffalo, New York for different years. Figures 5 and 6 show the same data in two different ways, allowing to come to different conclusions about the distribution.

```
# downloaded from https://www1.maths.leeds.ac.uk/~voss/data/buffalo/
x <- read.csv("data/buffalo.csv")
snowfall <- x$snowfall
hist(snowfall, probability = TRUE,
     breaks = seq(24.30, 208.22, length.out = 20),
     main = NULL, xlab = "snowfall [in]")

hist(snowfall, probability = TRUE,
     breaks = seq(22.85, 204.92, length.out = 20),
     main = NULL, xlab = "snowfall [in]")
```

As a further illustration of the effect of bucket width, the code in figure 7 shows how histograms with different bucket widths can be generated in R. Here we simply specify numeric values for the `break` argument to `hist()`, which R uses as the *approximate* number of buckets in the plot.

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

for (breaks in c(80, 40, 20, 10)) {
  hist(snowfall,
       prob = TRUE,
       breaks = breaks,
       xlim = c(25, 200),
       ylim = c(0, 0.03),
       xlab = paste("breaks =", breaks),
       main = NULL)
}
```

## 1.2 Kernel Density Estimation

Kernel density estimation allows to estimate the density  $f$  for given data while avoiding some of the disadvantages of histograms. Again, we suppose that we are given data  $x_1, \dots, x_n \in \mathbb{R}$  and that we want

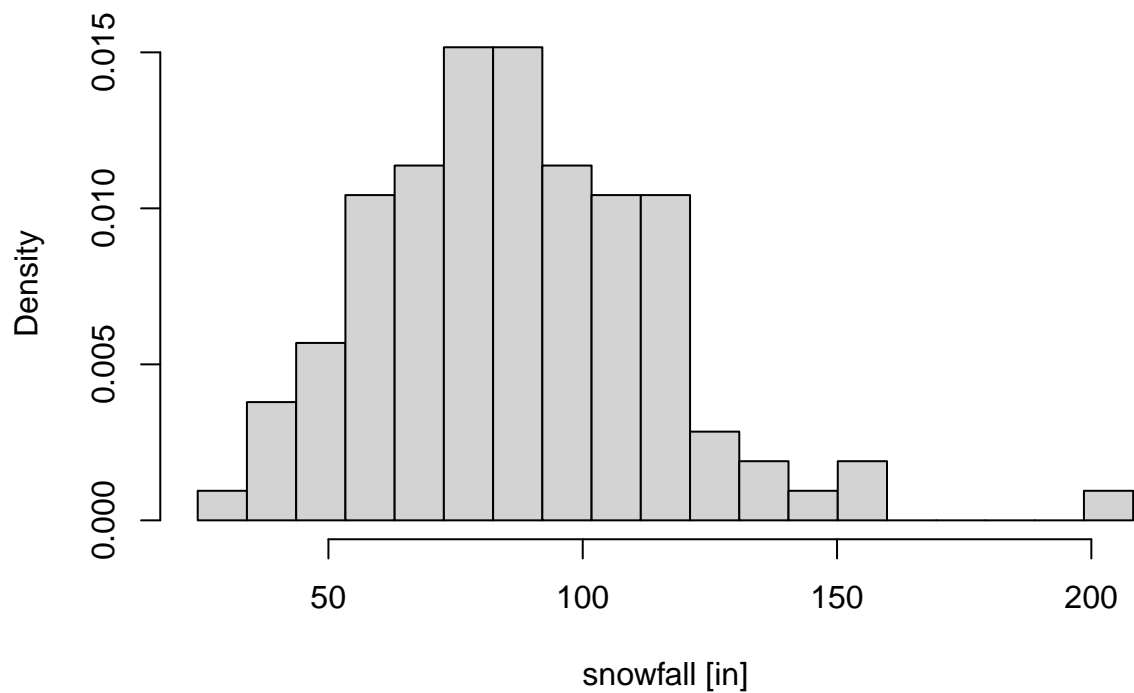


Figure 5: The annual amount of snowfall in Buffalo, New York, in inches. The histogram makes it plausible that there is one main peak in the distribution.

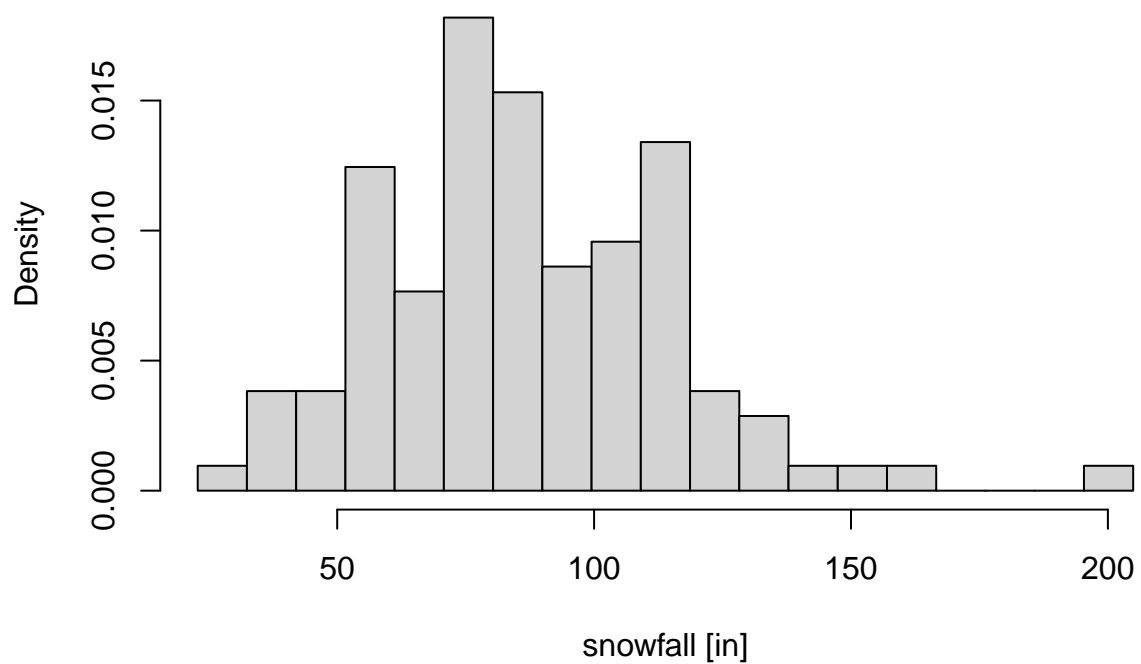


Figure 6: Continued from 5, this histogram shows the dataset in a way that three peaks are visible.



Figure 7: This figure illustrates how the bucket size in a histogram can be controlled in R.

to estimate the density  $f$ .

### 1.2.1 Motivation

Similar to the argument in the previous subsection, for  $x$  in a “small” interval  $[a, b]$  we can estimate  $f(x)$  as

$$f(x) \approx \frac{1}{b-a} \int_a^b f(y) dy = \frac{1}{b-a} P(X \in [a, b]) \approx \frac{1}{b-a} \frac{n_{a,b}}{n},$$

where  $n_{a,b}$  denotes the number of samples in the interval  $[a, b]$ . This equation contains two approximation. The first one,  $f(x) \approx 1/(b-a) \int_a^b f(y) dy$ , is more accurate if the interval is small, because then  $f$  will be nearly constant over the interval. The second approximation will be more accurate if the interval is large, because then the interval  $[a, b]$  covers more samples and the estimate of the probability is based on more data. We will later discuss in detail how these two concerns can be optimally balanced.

A mathematical “trick” to write more clearly how  $n_{a,b}$  depends on the data is to write the value as

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

where

$$I_{[a,b]}(x) = \begin{cases} 1 & \text{if } x \in [a, b], \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

The function  $I_{[a,b]}$  is called the **indicator function** of the set  $[a, b]$ .

Using the indicator function notation, the estimate for  $f(x)$  can be written as

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

whenever  $x \in [a, b]$  and when  $b - a$  is “not too large and not too small”. For symmetry we choose the interval  $[a, b]$  centred around  $x$ , say  $[a, b] = [x - h, x + h]$  where  $h$  can be chosen to control the width of the interval. In this case we have  $b - a = x + h - x - h = 2h$  and thus

$$\begin{aligned} f(x) &\approx \frac{1}{n} \sum_{i=1}^n \frac{1}{2h} I_{[x-h, x+h]}(x_i) \\ &= \frac{1}{n} \sum_{i=1}^n \frac{1}{2h} I_{[-h, +h]}(x_i - x) \\ &= \frac{1}{n} \sum_{i=1}^n \frac{1}{2h} I_{[-1, +1]} \left( \frac{x_i - x}{h} \right) \end{aligned}$$

for all  $x \in \mathbb{R}$ . This is an example of a kernel density estimate. The function  $K(x) = 1/2 I_{[-1, +1]}(x)$  on the right-hand side is called the kernel of the estimate, and the parameter  $h$  is called the **bandwidth** or the smoothing parameter.

### 1.2.2 Definition of a Kernel Density Estimator

The general kernel density estimate is a generalisation of the idea from the previous subsection. We first define the class of functions which we use to replace the function  $1/2 I_{[-1, +1]}$ .

**Definition 1.1.** A **kernel** is a function  $K: \mathbb{R} \rightarrow \mathbb{R}$  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  $x \in \mathbb{R}$ .

Of these three properties, the first one is the most important one. The second condition, symmetry, ensures that  $K$  is centred around 0 and the third definition, positivity, makes  $K$  a probability density. (While most authors list all three properties shown above, sometimes the third condition is omitted.)

It is easy to check that  $K(x) = 1/2 I_{[-1, +1]}(x)$  satisfies all three conditions of definition 1.1. This function  $K$  is sometimes called the “uniform kernel”, because it is the density of the uniform distribution  $\mathcal{U}[-1, +1]$ .

Based on the concept of a kernel, we now can define what a Kernel Density Estimate is.

**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 given by

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

for all  $y \in \mathbb{R}$ .

For  $K(x) = 1/2 I_{[-1, +1]}(x)$  this definition recovers the approximation we discussed in the previous section. In later sections we will see how the kernel  $K$  can be chosen for the estimator  $\hat{f}$  to have “good” properties. As a simple example we note that if  $K$  is continuous, then the rescaled kernel  $K_h$  and thus also the estimate  $\hat{f}_h$  are continuous functions.

Similar to the bucket width in histograms, the bandwidth parameter  $h$  controls how smooth the density estimate  $\hat{f}_h$  is, as a function of  $x$ .

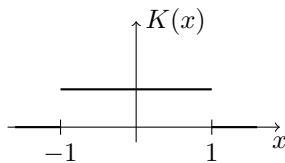
### 1.2.3 Kernels

There are many different kernels in use. Some examples are listed below. A more exhaustive list can, for example, be found on Wikipedia.



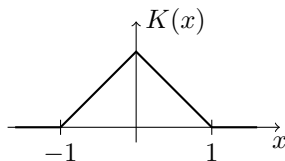
### 1.2.3.1 Uniform Kernel

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



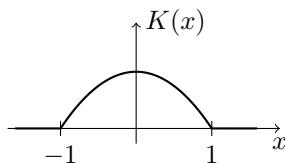
### 1.2.3.2 Triangular Kernel

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



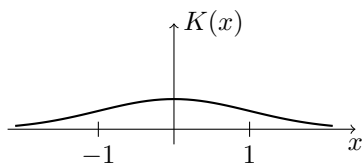
### 1.2.3.3 Epanechnikov Kernel

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



### 1.2.3.4 Gaussian Kernel

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



## 1.3 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. Note that for some kernels, `bw` differs from our  $h$  by a constant factor. The value `bw=1` always corresponds to the case where the probability distribution with density  $K_h$  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".

Details about how to call `density()` can be found by using the command `help(density)` in R.

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

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

```
## List of 7
## $ x      : num [1:512] -4.17 -3.72 -3.26 -2.81 -2.35 ...
## $ y      : num [1:512] 4.32e-06 4.98e-06 5.73e-06 6.56e-06 7.48e-06 ...
## $ bw     : num 9.72
## $ n      : int 109
## $ 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 `density()` with different bandwidth parameter. The result is shown in figure 8.

```
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)
}
```

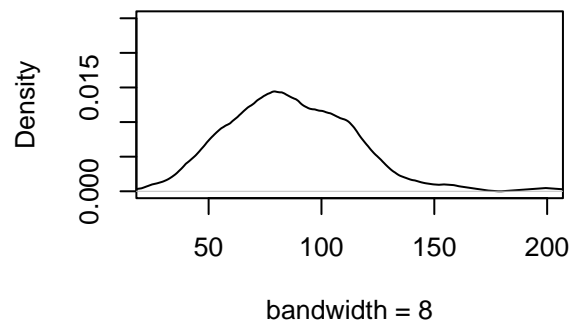
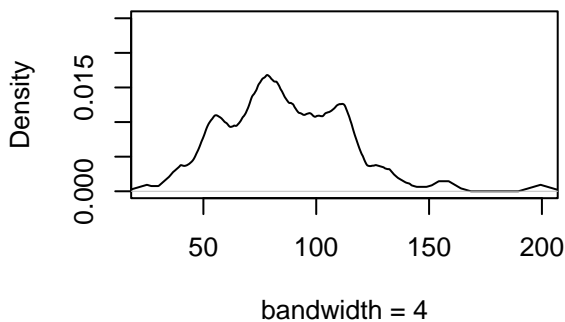
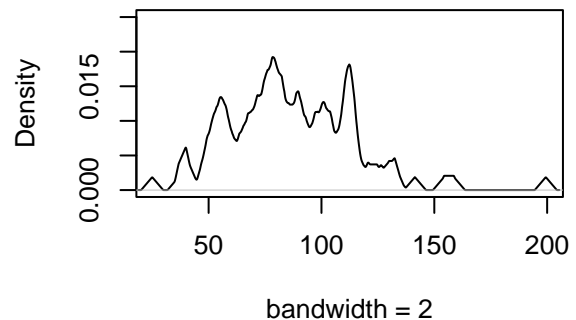
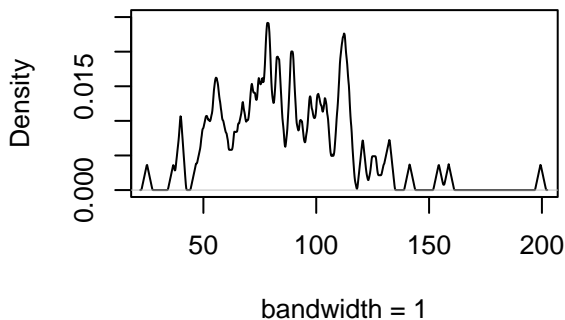


Figure 8: This figure illustrates how the bandwidth of a kernel density estimate can be controlled in R.

## Summary

- Histograms can be scaled so that they approximate densities.
- Some care is needed when choosing buckets for a histogram.
- Kernel density estimates can be used to estimate densities from data.
- A variety of different kernel functions are commonly used.

## Interlude: Kernel Density Estimation in R

In the previous section we have already seen how the function `density()` can be used to compute kernel density estimates in R. In the current section we will compare different methods to compute these estimates “by hand”.

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

```
# downloaded from https://www1.maths.leeds.ac.uk/~voss/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

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

Our aim is to implement this formula in R.

To compute the estimate we need to choose a kernel function. For our experiment we use the triangular kernel from section 1.2.3.2:

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

The use of `pmax()` here allows to evaluate our newly defined function `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. The triangular kernel used by the function `density` is scaled by a factor  $1/\sqrt{6}$  compared to our formula, so here we use  $h = 4\sqrt{6}$  to get output comparable to the density output for `bw=4` (bottom left panel in figure 8).

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

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

Typically we want to evaluate this function on a grid of  $x$ -values, for example to plot the estimated density  $\hat{f}$  as a function. Since  $x_1, \dots, x_n$  is used to denote the input data, we need choose a different name for the  $x$ -values where we evaluate  $\hat{f}_h$ . Here we use the  $\tilde{x}_1, \dots, \tilde{x}_m$  to denote these 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))  
}
```

Plotting `f.hat` as a function `x.tilde` (figure 9) shows that we get a similar result as we did in figure 8.

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

We will see in the next section that the code shown above is slower than the built-in function `density()`, but on the other hand using our own code we can control all details of the computation and there is no uncertainty about what exactly the code does.

### Speed Improvements

In this sub-section we will see how we can speed up our implementation of the kernel density estimate. To allow for easier speed comparisons, we encapsulate the code from above into a function:

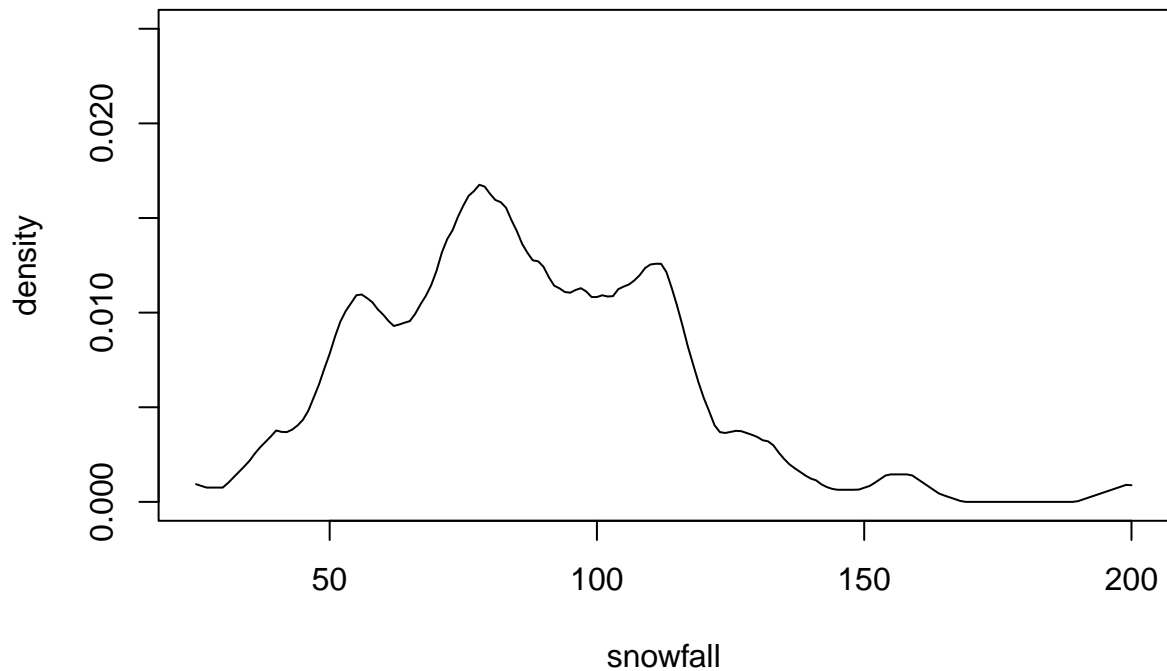


Figure 9: Manually computed kernel density estimate for the snowfall data, corresponding to `bw=4` in `density()`.

```
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 code is the use of loops, like the `for`-loop in the function `KDE1()`. The loop in our code is used to compute the differences  $\tilde{x}_j - x_i$  for all  $j$ . We have already avoided a second loop by using the fact that R interprets `x.tilde[j] - x` a vector operation, by using `pmax()` in our implementation of `K`, and by using `mean()` to evaluate the sum in the formula for  $\hat{f}_h$ .

To also avoid the loop over  $j$  we can use the function `outer()`. This function takes two vectors as inputs and then applies a function to all pairs of elements from the two vectors. The result is a matrix where the rows correspond to the elements of the first vector and the columns to the elements of the second vector. The function to apply can either be an arithmetic operation like `+` or `*`, or an R-function which takes two numbers as arguments. For example, the following code computes all pairwise differences between the elements of two vectors:

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

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

Details about how to call `outer()` can be found by using the command `help(outer)` in R.

We can use `outer(x.tilde, x, "-")` to compute all the  $\tilde{x}_j - x_i$  at once. The result is a matrix with  $m$  rows and  $n$  columns, and our implementation of the function `K()` can be applied to this matrix to compute the kernel values for all values at once:

```
KDE2 <- function(x.tilde, x, K, h) {
  dx <- outer(x.tilde, x, "-")
  rowMeans(1/h * K(dx / h))
}
```

We will see at the end of this section that the function `KDE2()` is significantly faster than `KDE1()`, and we can easily check 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, the new function `KDE3()` returns the same result as the original implementation:

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

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

To verify that our modifications really made the code faster, we measure the execution times using the R package `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 850.914 903.6810 975.1911 919.6915 935.292 6478.574   1000
## KDE2 122.672 169.8835 243.7764 176.1360 183.557 5385.719   1000
## KDE3 110.659 144.9555 230.4161 150.3060 155.800 7954.861   1000
```

The output summarises the execution times for 1000 runs of each of the three functions. The columns correspond to the fastest measured time, lower quartile, mean, median, upper quartile and slowest time, respectively. (Different runs take different times, due to the fact that the computer is busy with other tasks in the background.) We can see that the introduction of `outer()` in `KDE2()` made a big difference, and that the further optimisation in `KDE3()` improved speed further.

Finally, we compare the speed of our implementation to the built-in function `density()`. To get comparable output we use the arguments `from`, `to`, and `n` to specify the same grid  $\tilde{x}$  as we used above.

```
KDE.builtin <- function(x.tilde, x, K, h) {
  density(x,
    kernel = K, 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 96.924 107.461 122.2039 109.757 112.75 3875.156 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 for computing (a good approximation to) the kernel density estimate, based on Fast Fourier Transform. By comparing the output of `KDE.builtin()` we can see that the approximation used by `density()` gives nearly the same result as our exact implementation of the method:

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

```
## [1] 3.228798e-05
```

## 2 The Bias of Kernel Density Estimates

In the previous section we introduced the kernel density estimate

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

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

$$X_1, \dots, X_n \sim f, \quad (2)$$

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  $f: \mathbb{R} \rightarrow \mathbb{R}$  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  $\hat{f}_h: \mathbb{R} \rightarrow \mathbb{R}$ , where  $\hat{f}_h(x)$  is given by (1) for every  $x \in \mathbb{R}$ .

### 2.2 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

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

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 (1) we find

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

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  $z = y - x$ .

### 2.3 Moments of Kernels

To understand how the bias changes as  $h$  varies, we will need to consider properties of  $K$  and  $K_h$  in more detail.

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

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

The second moment  $\mu_2$  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  $K_h$ , given by

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

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

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

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. □

It is easy to check that 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

$$\text{Var}(Y) = \mathbb{E}(Y^2) = \int y^2 K_h(y) dy = \mu_2(K_h) = h^2 \mu_2(K). \quad (3)$$

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

## 2.4 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  $Y \sim K_h$ :

$$\mathbb{E}(\hat{f}_h(x)) = \mathbb{E}(f(x+Y)). \quad (4)$$

Equation (3) 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 qualitative version of this argument, we consider the Taylor approximation of  $f$  around the point  $x$ :

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

Substituting this into equation (4) we find

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

for small  $h$ . Considering the bias again, this gives

$$\text{bias}(\hat{f}_h(x)) = \mathbb{E}(\hat{f}_h(x)) - f(x) \approx \frac{\mu_2(K)f''(x)}{2}h^2 \quad (5)$$

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

In contrast, we will see in the next section that the variance of the estimator *increases* as  $h \downarrow 0$ . We will need to balance these two effects to find the optimal value of  $h$ .

### Summary

- We have introduced a statistical model for density estimation.
- The bias for kernel density estimation can be written as an integral.
- We learned how the moments of a kernel are defined.
- The bias for small bandwidth depends on the second moment of the kernel and the second derivative of the density.

### 3 The Variance of Kernel Density Estimates

In the previous section we considered the bias of the estimator

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

We found

$$\mathbb{E}(\hat{f}_h(x)) = \mathbb{E}(K_h(x - X_i)) \quad (6)$$

for all  $i \in \{1, \dots, n\}$  (we used  $i = 1$ ), and we used this relation to compute the bias. In this section, we will use similar arguments to compute the variance and the mean squared error of the estimator.

#### 3.1 Variance

We use the formula

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

and consider the two terms separately.

##### 3.1.1 Second Moment

For the second moment term in the definition of the variance we get

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

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

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

where we used equation (6) for the last term. Using this equation, we can write the expectation as

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

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

$$\text{Var}(\hat{f}_h(x)) = \frac{1}{n} \left( \mathbb{E}(K_h(x - X_1)^2) - \mathbb{E}(\hat{f}_h(x))^2 \right). \quad (7)$$

##### 3.1.2 The Roughness of a Kernel

Similar to what we did in the last section, 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  $n \rightarrow \infty$  and  $h \rightarrow 0$ . For the first term in equation (7) we find

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

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

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

where the first-order term disappears since  $z \mapsto zK(z)^2$  is an asymmetric function. As a shorthand we use the following definition.

**Definition 3.1.** The **roughness** of a kernel  $K$  is given by

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

This gives the result

$$\mathbb{E}(K_h(x - X_1)^2) \approx \frac{1}{h} f(x) R(K) + \frac{1}{2} h f''(x) \int z^2 K(z)^2 dz \quad (8)$$

for small  $h$ .

### 3.1.3 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 previous section 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

$$\mathbb{E}(\hat{f}_h(x))^2 \approx f(x)^2 + h^2 \mu_2(K) f(x) f''(x) + \frac{1}{4} h^4 \mu_2(K)^2 f''(x)^2 \quad (9)$$

for small  $h$ .

Substituting (8) and (9) into equation (7) we finally find

$$\text{Var}(\hat{f}_h(x)) = \frac{1}{n} \left( \frac{1}{h} f(x) R(K) - f(x)^2 + \dots \right),$$

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

$$\text{Var}(\hat{f}_h(x)) \approx \frac{1}{nh} f(x) R(K) \quad (10)$$

as  $h \downarrow 0$ .

### 3.2 Mean Squared Error

The Mean Squared Error of the estimator  $\hat{f}_h(x)$  for  $f(x)$  is given by

$$\text{MSE}(\hat{f}_h(x)) = \mathbb{E}\left((\hat{f}_h(x) - f(x))^2\right).$$

It is an easy exercise to show that this can equivalently be written as

$$\text{MSE}(\hat{f}_h(x)) = \text{Var}(\hat{f}_h(x)) + \text{bias}(\hat{f}_h(x))^2.$$

Substituting equations (5) and (10) into the formula for the MSE, we get

$$\text{MSE}(\hat{f}_h(x)) \approx \frac{1}{nh}f(x)R(K) + \frac{1}{4}\mu_2(K)^2f''(x)^2h^4. \quad (11)$$

Some care is needed to make sure that the omitted terms from the Taylor approximations really don't make a significant contribution in this formula for the MSE: The additional contributions from the variance have the form  $e_1(h)/n$ , where the error term  $e_1$  does not depend on  $n$  and is negligible compared to  $f(x)R(K)/h$  as  $h \downarrow 0$ . Using little-o notation, This is sometimes denoted by  $e_1(h) = o(1/h)$ , which indicates that  $e_1(h)/(1/h) \rightarrow 0$  as  $h \downarrow 0$ . The additional terms from the squared bias, say  $e_2(h)$ , do not depend on  $n$  and are negligible compared to  $\mu_2(K)^2f''(x)^2h^4$ . We can write  $e_2(h) = o(h^4)$  as  $n \downarrow 0$ , to reflect this fact. We can summarise these results as

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

as  $h \downarrow 0$ , with the understanding that the constants in the definition of  $o(h^4)$  do not depend on  $n$  and that  $o(1/nh)$  really means " $o(1/h)$ , where the constants are proportional to  $1/n$ ."

### 3.3 Optimal Bandwidth

The two terms on the right-hand side of (11) 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

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

and thus the derivative equals zero, if

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

or, equivalently,

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

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 (11), we get

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

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  $(R(K)^2\mu_2(K))^{2/5}$ . Thus we should use kernels where the value  $R(K)^2\mu_2(K)$  is small.
- The error is proportional to  $f(x)^2|f''(x)|$ . We cannot influence this term, but we can see that  $x$  where  $f$  is large or has high curvature have higher estimation error.

### Summary

- We found the variance of the kernel density estimate.
- We studied the mean squared error for small  $h$ .
- We derived a formula for the optimal value of the bandwidth  $h$ .

## 4 Kernel Density Estimation in Practice

In this section we conclude our discussion of kernel density estimation by considering different aspects which are important when using the method in practice.

### 4.1 Integrated Error

From equation (11) we know

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

This 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

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

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

As before, we can use differentiation to find the optimal value of  $h$ . Here we get

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

and the corresponding error is

$$\text{IMSE}_{\text{opt}} = \frac{5}{4} \frac{1}{n^{4/5}} \left( R(K)^2 \mu_2(K) \right)^{2/5} R(f'')^{1/5}. \quad (12)$$

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  $R(K)^2 \mu_2(K)$ .

### 4.2 Choice of Kernel

The integrated error in equation (12) is proportional to  $(R(K)^2 \mu_2(K))^{2/5}$ , 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 4.1.** For the uniform kernel we have

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

From this we find

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

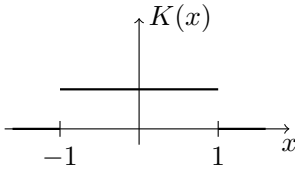
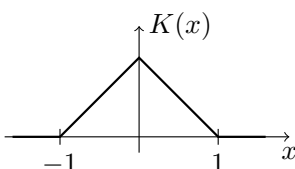
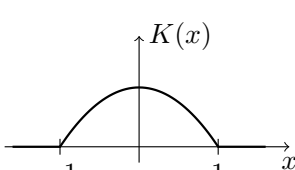
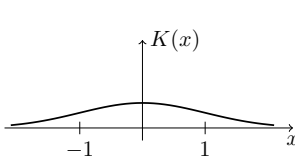
and

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

Thus, for the triangular 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.

### 4.3 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  $\mu$  and variance  $\sigma^2$ . 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

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

Patiently integrating the square of this function gives

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

This can be used as a simple “plug-in rule” with  $\sigma$  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://www1.maths.leeds.ac.uk/~voss/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
```

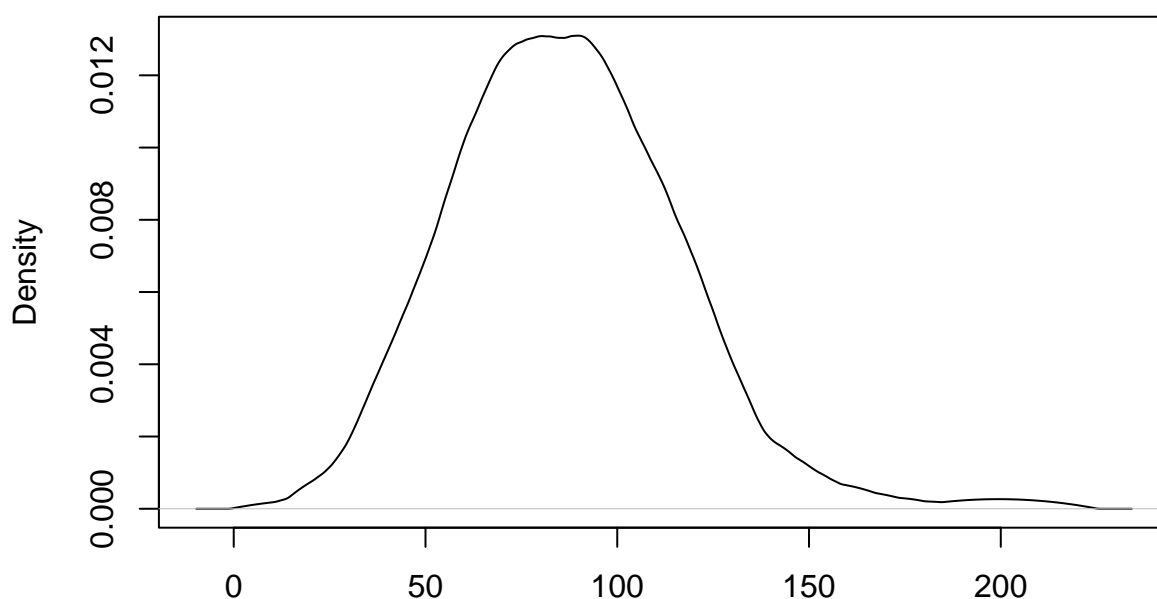
R has a variety of different builtin methods to estimate bandwidths. See `stats/bandwidth` for a description. For comparison to our result, we list here the bandwidths suggested by some of R's algorithms:

```
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 value 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)
```



N = 109 Bandwidth = 11.59

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

mating the bandwidth manually.

#### 4.4 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  $K: \mathbb{R}^p \rightarrow \mathbb{R}$  such that

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

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

**Example 4.2.** If  $K_1, \dots, K_p$  are one-dimensional kernels, then the product

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

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

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

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

$$K_h(x) = \frac{1}{h_1 \dots h_p} K(x_1/h_1, \dots, x_p/h_p)$$

for all  $x \in \mathbb{R}^p$  can be used, where  $h = (h_1, \dots, h_p)$  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.

## 5 Kernel Smoothing

We now consider the statistical model

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

where  $m: \mathbb{R} \rightarrow \mathbb{R}$  is a smooth function and  $\varepsilon_i$  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**.

### 5.1 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 5.1.** 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 are 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

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

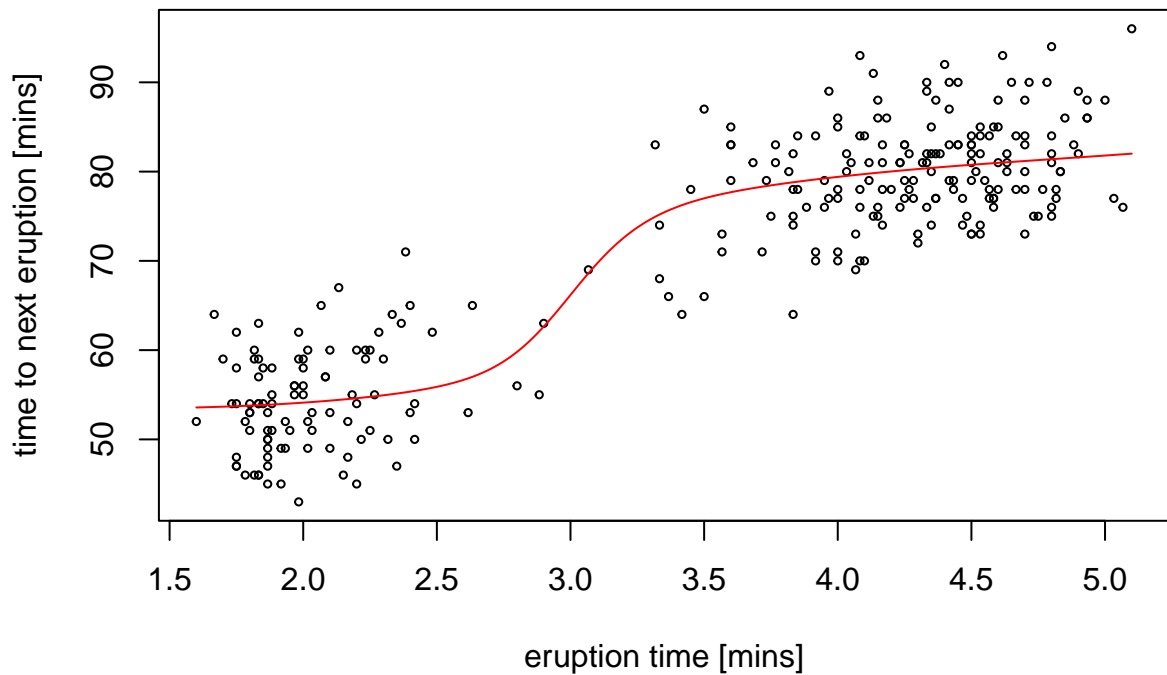
we can write the Nadaraya-Watson Estimator as  $\hat{m}_h(x) = \sum_{i=1}^n w_i(x) y_i$  and since

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

this is indeed a weighted average.

**Example 5.1.** 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")
```

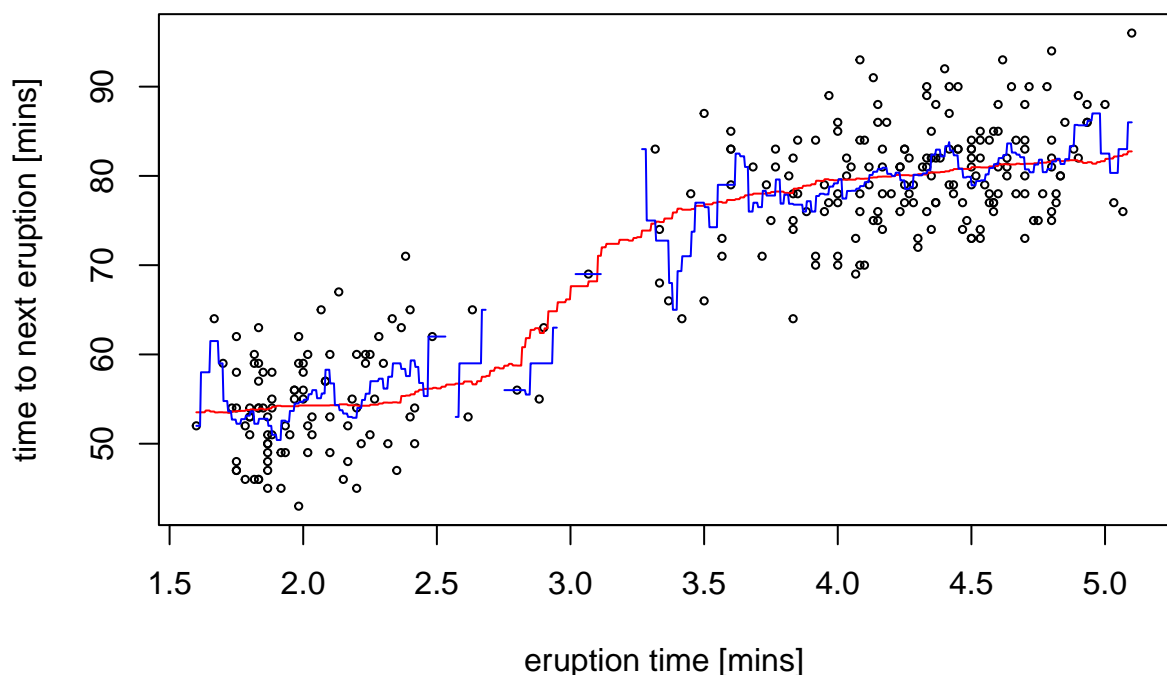


The estimate  $\hat{m}_h$  (red line) smoothly connects the two clusters visible in the scatter plot.

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 5.2.** 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.

## 5.2 Estimation Error

Here we will discuss how fast the estimation error decreases in the limit of  $n \rightarrow \infty$ , *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  $\text{Var}(\eta_i) = 1$ .
- $\varepsilon_i = s(X_i)\eta_i$  for all  $i \in \{1, \dots, n\}$ , where  $s: \mathbb{R} \rightarrow (0, \infty)$  is a smooth function.
- $Y_i = m(X_i) + \varepsilon_i$  where  $m: \mathbb{R} \rightarrow \mathbb{R}$  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  $\text{Var}(\hat{m}_h(x))$  as  $n \rightarrow \infty$  and  $h \downarrow 0$ . We can write  $\hat{m}_h(x)$  as

$$\hat{m}_h(x) = \frac{\frac{1}{n} \sum_{i=1}^n K_h(x - X_i)Y_i}{\frac{1}{n} \sum_{j=1}^n K_h(x - X_j)} = \frac{\frac{1}{n} \sum_{i=1}^n K_h(x - X_i)Y_i}{\hat{f}_h(x)} = \frac{\hat{r}_h(x)}{\hat{f}_h(x)} \quad (13)$$

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 (13) separately

### Denominator

From equations (5) and (10) we know that

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

and

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

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 the section about the variance of kernel density estimates.

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

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

We use integrals to average over the randomness in  $X$  and  $\eta$ , denoting the density of  $\eta$  by  $\varphi$ :

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

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 5.1.** *Let  $g: \mathbb{R} \rightarrow \mathbb{R}$  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 (5). The second statement is proved similarly, as shown in the derivation of equation (8).  $\square$

Using the first part of lemma 5.1 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

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

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 5.1 one can show that

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

Combining these equations we find

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

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

### 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

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

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:

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

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

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

and

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

A similar calculation gives the approximate variance as

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

So finally we have

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

### Notes:

- A more careful calculation will need to take into account that  $\hat{m}(x)$  may be undefined. All expectations and variances are conditional on  $\hat{f}(x) \neq 0$ . One can show that  $P(\hat{f}(x) \neq 0) \rightarrow 1$  as  $n \rightarrow \infty$  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

- We have learned how the Nadaraya-Watson Estimator can be used for smoothing.
- We have considered the mean squared error of this estimator.



## 6 Local Polynomial Regression

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

### 6.1 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:

$$\begin{aligned} r(\beta) &= \sum_{i=1}^n \hat{\varepsilon}_i^2 \\ &= \hat{\varepsilon}^\top \hat{\varepsilon}. \end{aligned}$$

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

$$r_w(\beta) = \sum_{i=1}^n w_i \varepsilon_i^2.$$

In matrix notation, this function can be written as

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

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

$$W = \begin{pmatrix} w_1 & 0 & 0 & \cdots & 0 \\ 0 & w_2 & 0 & \cdots & 0 \\ 0 & 0 & w_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & w_n \end{pmatrix}. \quad (14)$$

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

### 6.2 Polynomial Regression

In these notes we only consider the case of  $p = 1$ . 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

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

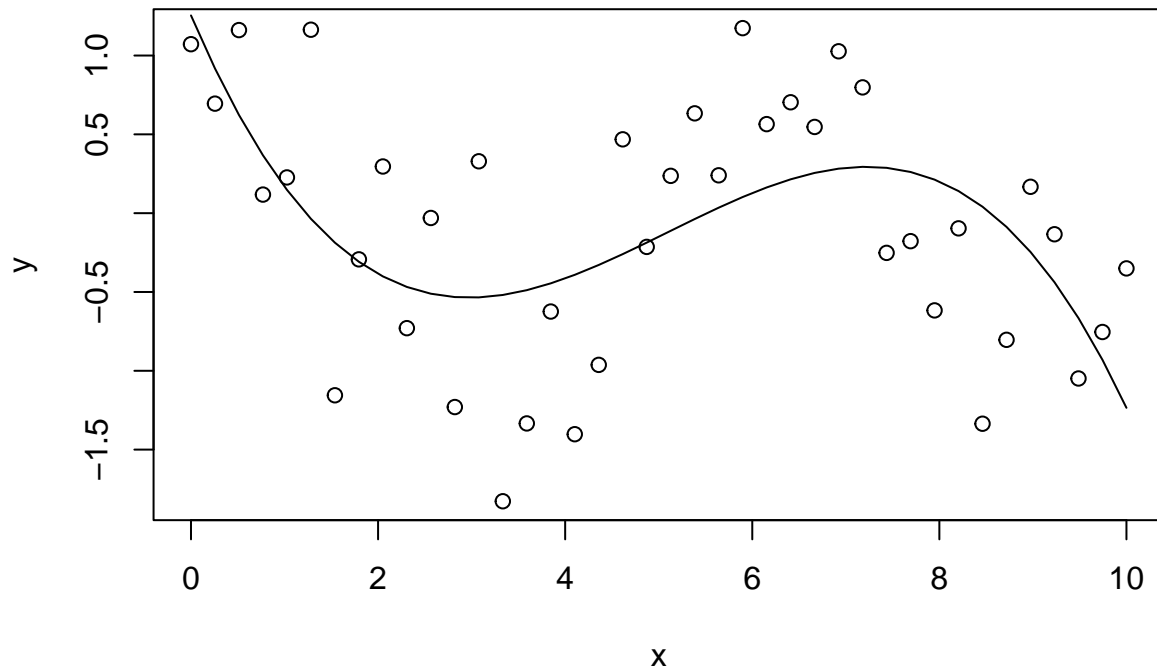
**Example 6.1.** To illustrate polynomial regression, we fit a third-order polynomial to a simple, simulated dataset. Since the operator  $\wedge$  has a special meaning inside `lm()`, we have to use the function `I()` (which disables the special meaning of  $+$  and  $\wedge$  for its arguments) when computing the inputs  $x^2$  and  $x^3$ .

```
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  $\kappa$ . For the example above we get the following value:

```
kappa(m)
```

```
## [1] 2813.242
```

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

$$X = \begin{pmatrix} 1 & (x_1 - \tilde{x}) & (x_1 - \tilde{x})^2 & \cdots & (x_1 - \tilde{x})^p \\ 1 & (x_2 - \tilde{x}) & (x_2 - \tilde{x})^2 & \cdots & (x_2 - \tilde{x})^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (x_n - \tilde{x}) & (x_n - \tilde{x})^2 & \cdots & (x_n - \tilde{x})^p \end{pmatrix}. \quad (15)$$

**Example 6.2.** Continuing from the previous example, we can see that writing the model as in (15) greatly improves the condition number:

```
x.tilde <- 5
m2 <- lm(y ~ I(x-x.tilde) + I((x-x.tilde)^2) + I((x-x.tilde)^3))
kappa(m2)
```

## [1] 65.75674

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

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

### 6.3 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 (14) for the weights and the matrix  $X$  from (15) 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

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

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

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

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

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.

### 6.4 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  $\beta_0$ . In this case, the design matrix  $X$  simplifies to  $X = (1, \dots, 1) \in \mathbb{R}^{n \times 1}$ . Thus we have

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

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 5.1: 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 6.3.** 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(x..., degree = 0, bandwidth = 0.11)` is near identical.

```
set.seed(20211103)

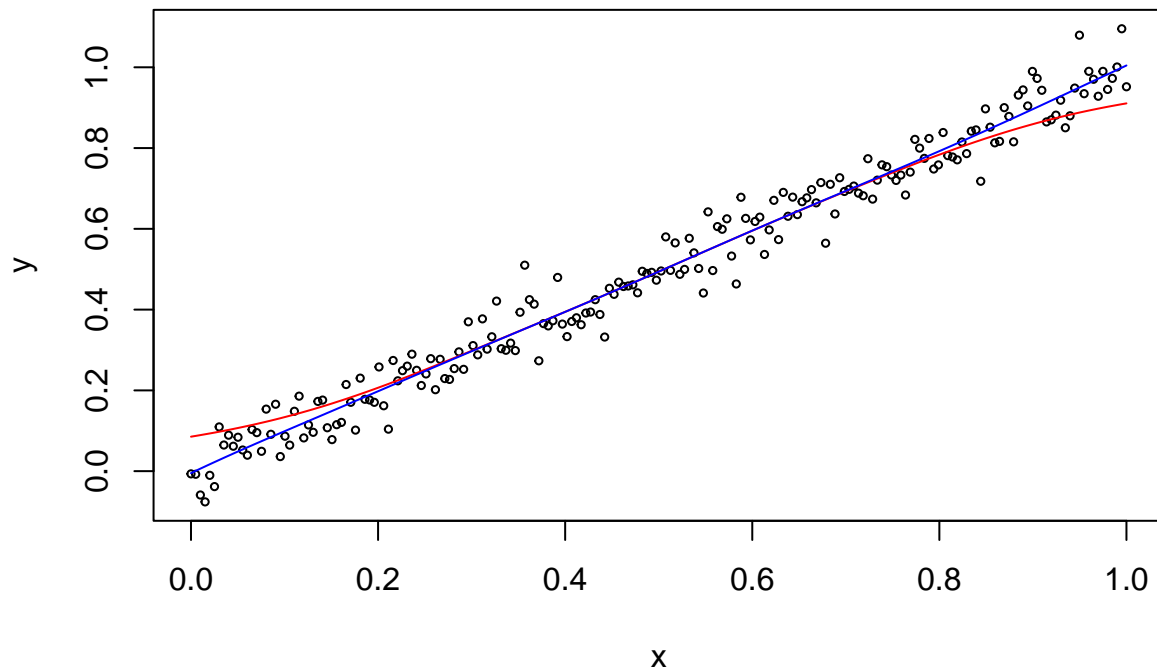
n <- 200
x <- seq(0, 1, length.out = n)
y <- x + rnorm(n, sd = 0.05)
plot(x, y, cex = .5)

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 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 allows sometimes to reduce bias near peaks.

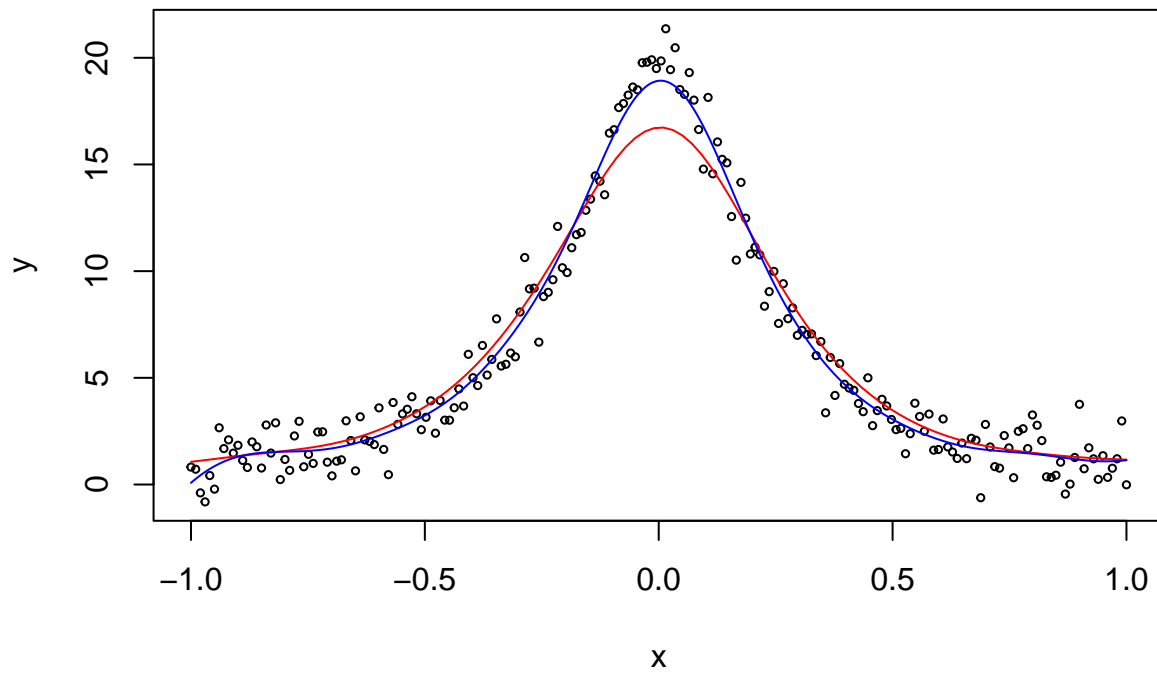
**Example 6.4.** We compare the Nadaraya-Watson estimator to locally polynomial regression with  $p = 2$ , using a simulated dataset which has a peak in the middle of the domain. We choose the same bandwidths as in the previous example.

```
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

- Regression with weights can be used to fit models to the data near a given point.
- A simple application of linear regression can fit polynomials as well as straight lines.
- The local polynomial regression estimator is a generalization of the Nadaraya-Watson estimator.

## 7 $k$ -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 5.2). 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.

### 7.1 Definition of the Estimator

**Definition 7.1.** For  $k \in \{1, \dots, n\}$ , the  **$k$ -nearest neighbour**, or  $k$ -NN estimate for the model mean  $m(x)$  is given by

$$\hat{m}_k(x) := \frac{1}{k} \sum_{i \in J_k(x)} y_i, \quad (16)$$

where

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

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 (16) 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.

### 7.2 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.

In the next section we will learn how  $k$  can be chosen using cross-validation.

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  $x$ .

### 7.3 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 7.1.** Here we compute a  $k$ -NN estimate for the mean of the `faithful` dataset, which we have already encountered in examples 5.1 and 5.2. We start by storing the data in the variables `x` and `y`:

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

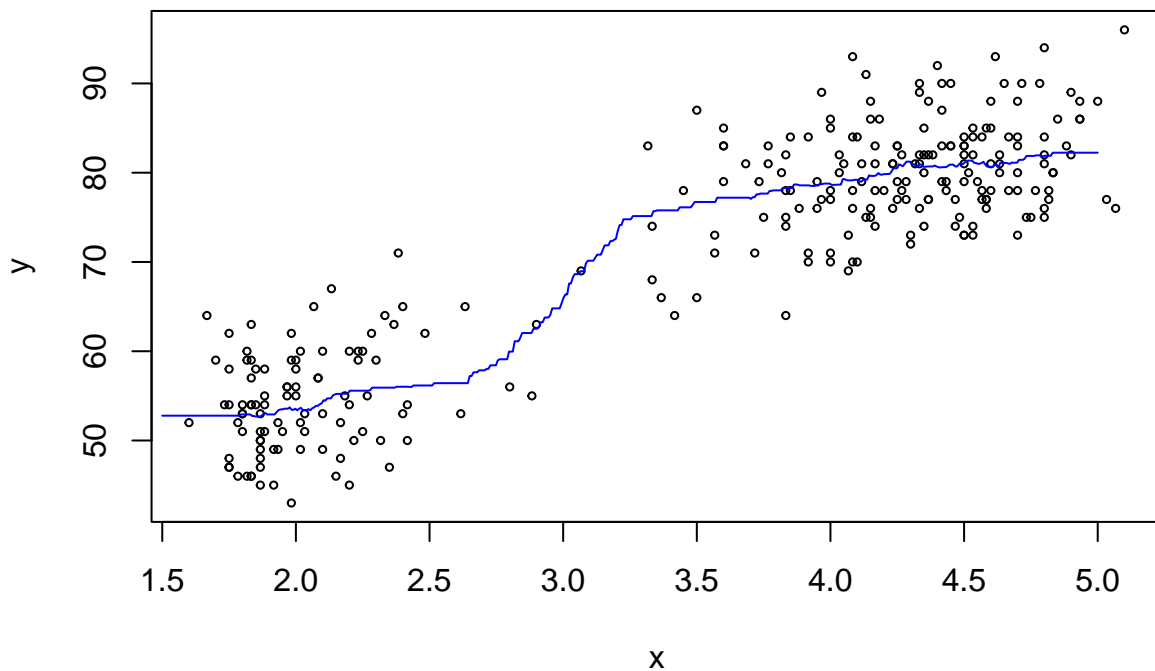
Now we use `knn.reg()` to compute the  $k$ -NN estimate on a grid of values. The help page for `knn.reg()` explains that we need to convert the input to either a matrix or a data frame; here we use data frames. The grid of input values where we want to estimate the  $k$ -NN estimate is passed in via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

## 7.4 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.

## 8 Cross-validation

In this section we will discuss methods for choosing the bandwidth  $h$  for kernel-based methods, and the size  $k$  of the neighbourhood used in  $k$ -nearest neighbour regression. 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  $x_i$ . There are different ways to implement this idea:

- 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 method can be computationally expensive for large  $n$ , since  $n$  models need to be fitted to the data.
- 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.

The idea behind cross-validation is that we always compare  $y_i$  to a fitted value which was computed without using any information about the true value of  $y_i$ . This way, a method cannot achieve a misleadingly good fit by overfitting the data.

### 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  $\hat{m}_h(x_i) \rightarrow y_i$  and thus  $r(h) \rightarrow 0$ . 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.

### 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 estimate of the log likelihood, which is given by

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

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

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

instead of  $L_{\text{LOO}}$ , 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:

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

Unfortunately, the  $h$  which minimises this expression depends on properties of  $h$  and in section 4.3 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  $\hat{h}_h$  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  $\text{IMSE}(\hat{f}_h) = e(h) + \text{const}$ , where  $\text{const}$  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*

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

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

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

$$\begin{aligned}\text{IMSE}(\hat{f}_h) &= \int_{-\infty}^{\infty} \text{MSE}(\hat{f}_h(x)) dx \\ &= \int_{-\infty}^{\infty} \mathbb{E}\left((\hat{f}_h(x) - f(x))^2\right) dx \\ &= \int_{-\infty}^{\infty} \mathbb{E}\left(\hat{f}_h(x)^2 - 2\hat{f}_h(x)f(x) + f(x)^2\right) dx \\ &= \int_{-\infty}^{\infty} \mathbb{E}\left(\hat{f}_h(x)^2\right) dx - 2 \int_{-\infty}^{\infty} \mathbb{E}\left(\hat{f}_h(x)f(x)\right) dx \\ &\quad + \int_{-\infty}^{\infty} \mathbb{E}\left(f(x)^2\right) dx \\ &= \int_{-\infty}^{\infty} \mathbb{E}\left(\hat{f}_h(x)^2\right) dx - 2 \int_{-\infty}^{\infty} \mathbb{E}(\hat{f}_h(x))f(x) dx \\ &\quad + \int_{-\infty}^{\infty} f(x)^2 dx,\end{aligned}$$

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

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

For the first term in the definition of CV we get

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

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

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

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  $\mathbb{E}\left(\hat{f}_h^{(1)}(x)\right)$  as a function of  $x$ , then using  $X_1$  in place of  $x$ , and computing the expectation over  $X_1$  afterwards. This gives

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

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.  $\square$

Using the result from the lemma we see that we can choose  $h$  to minimise  $\text{CV}(h)$  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

- Cross-validation can be used to avoid overfitting the data when we choose  $h$ .
- We have seen how to estimate  $h$  for kernel-based methods, and  $k$  for  $k$ -NN regression estimates.

## 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

$$CV(h) = \int \hat{f}_h(x)^2 dx - \frac{2}{n} \sum_{i=1}^n \hat{f}_h^{(i)}(x_i) =: A - B. \quad (17)$$

We will consider the snow fall dataset again:

```
# data from https://www1.maths.leeds.ac.uk/~voss/data/buffalo/
buffalo <- read.csv("data/buffalo.csv")
x <- buffalo$snowfall
n <- length(x)
```

In order to speed up the computation of the  $f_h^{(i)}$ , 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 (17) can now be approximate 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 (17), 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 `cv(h)` in R as follows:

```

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)
}

```

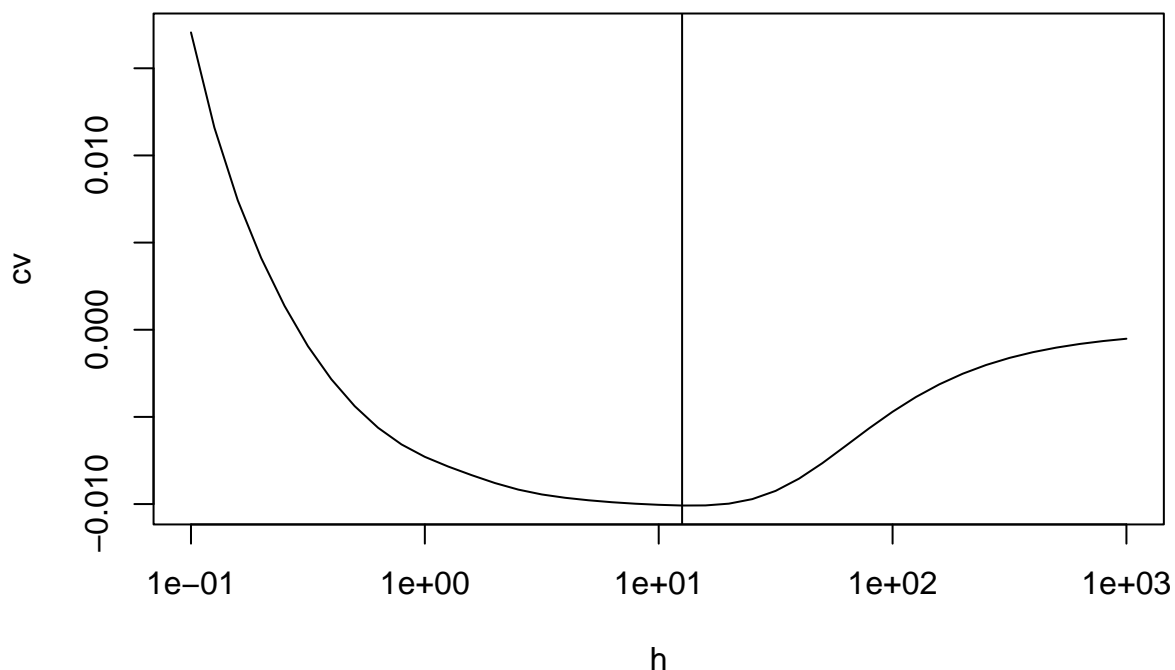
Finally, we evaluate the function `cv.h()` on a grid of  $h$ -values to find a good value of  $h$ :

```

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.

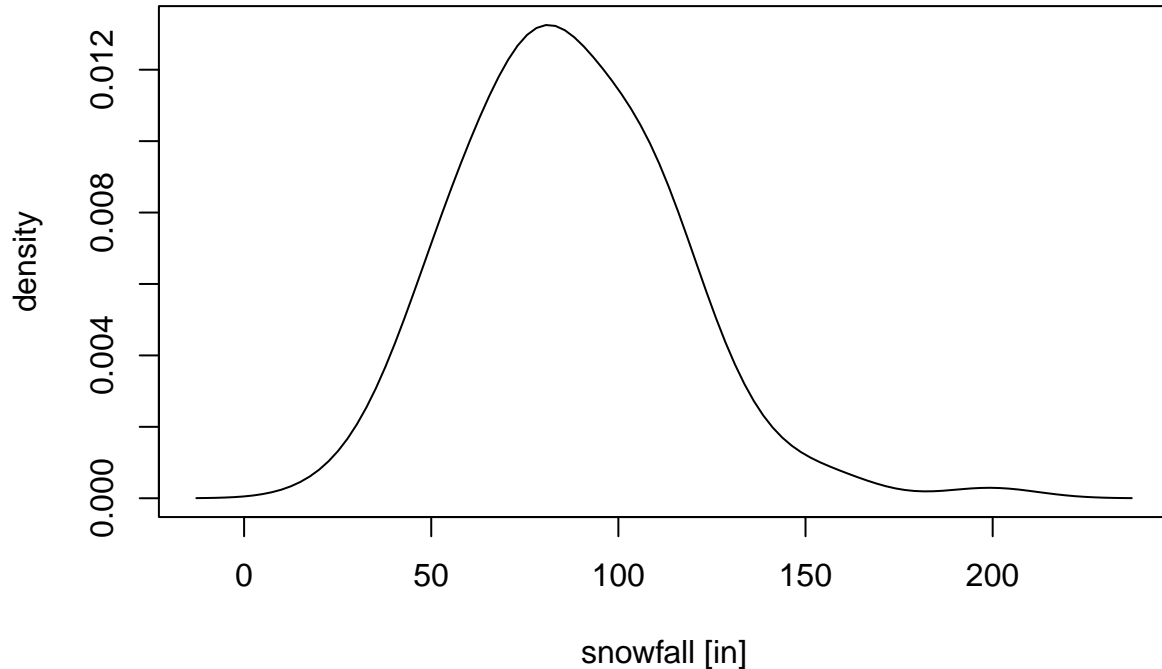
```

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)

```

```
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.

```
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  $i \in \{1, \dots, n\}$ . 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 results 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 (15) and (14). Here we use the “linear” case ( $p = 1$ ) instead of the polynomial case ( $p \geq 1$ ). 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  $2 \times 2$ -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)
}

```

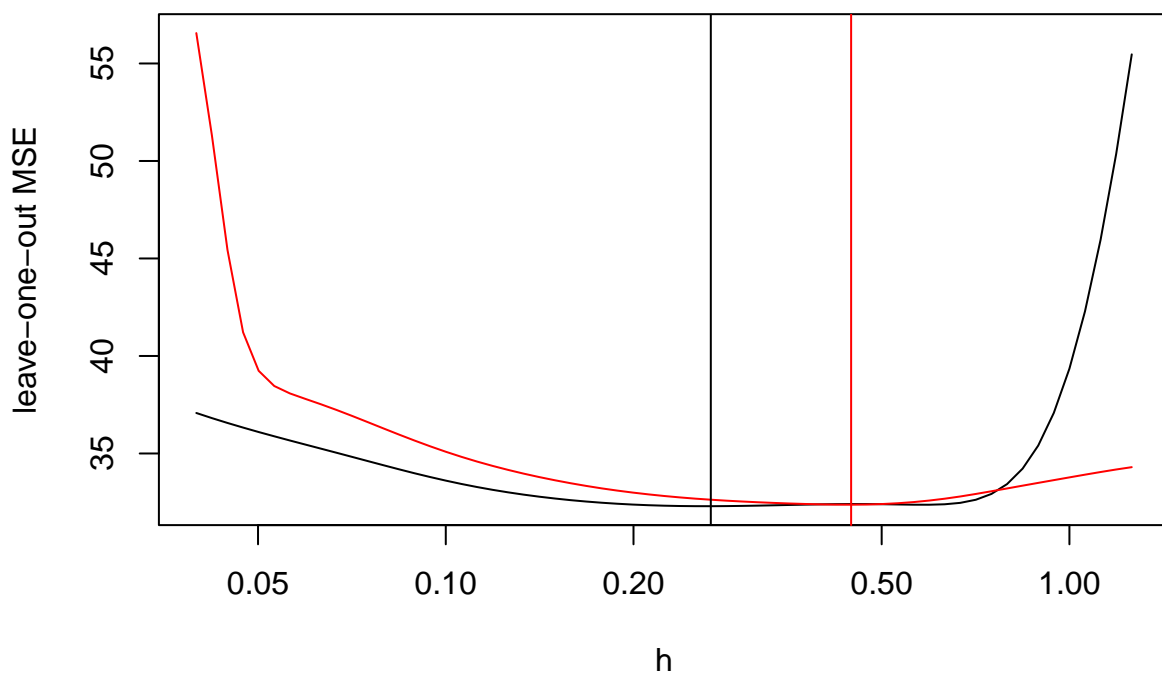
Now we evaluate the function `r.NW()` and `r.LL()` on a grid of  $h$ -values to find the optimal  $h$  for each method.

```

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")

```

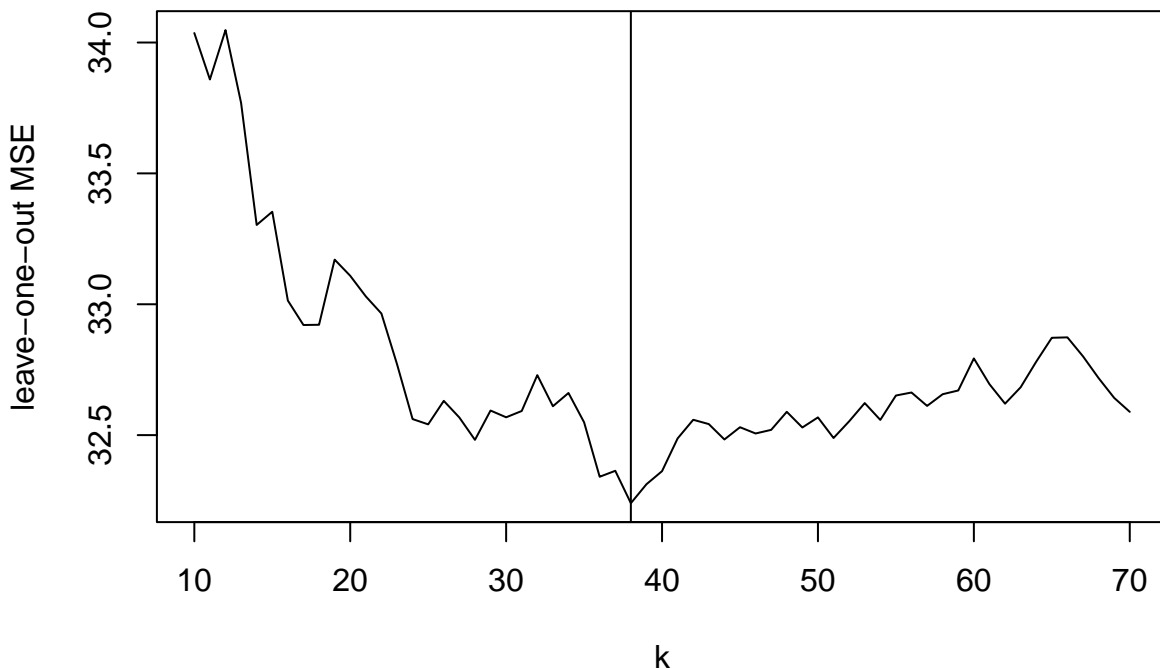


As expected, the optimal bandwidth for local linear regression is larger than for the Nadaraya Watson estimator.

### 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. Here it seems difficult to make any savings, so we resort to simply fitting  $n$  different models in the naive way. 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. 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[-i],
              test = data.frame(x=x.tilde),
              k = best.k)
m.kNN <- m$pred

colours <- c("#2C9CDA", "#811631", "#EOCA1D")
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)

```

