Notes for Nonparametric Statistics

Eduardo García-Portugués

Last updated: 2023-07-31, v6.9.0

ISBN 978-84-09-29537-1

AVAILABLE AT

BOOKDOWN.ORG/EGARPOR/NP-UC3M

First printing, July 2023

Contents

Preface

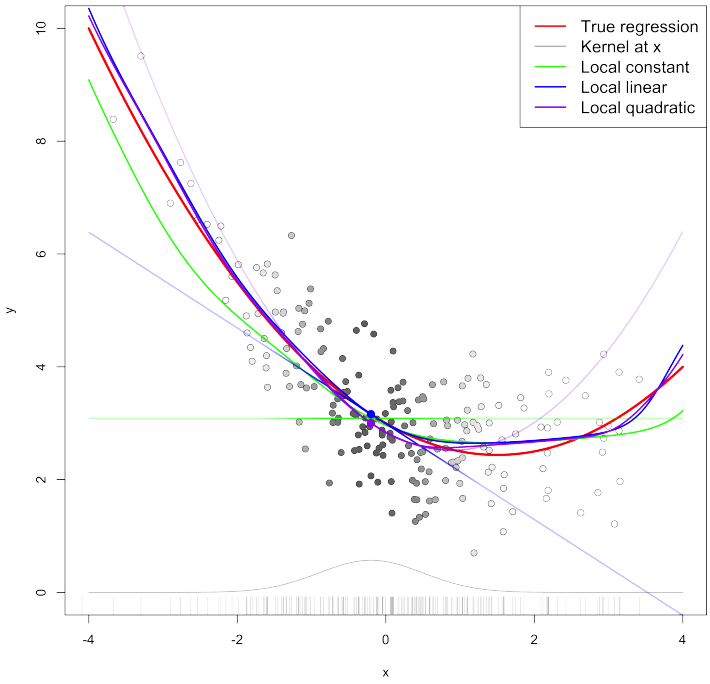


Figure 1: Illustration of nonparametric kernel estimators of the regression function.

Welcome

Welcome to the notes for *Nonparametric Statistics*. The course is part of the MSc in Statistics for Data Science from Carlos III University of Madrid.

The course is designed to have, roughly, **one session per each main topic** in the syllabus. The schedule is tight due to time constraints, which will inevitably make the treatment of certain methods somehow superficial. Nevertheless, the course will hopefully give you a respectable panoramic view of different available topics on nonparametric statistics. A broad view of the syllabus and its planning is:

1. Introduction (first session)
2. Kernel density estimation I (first/second session)
3. Kernel density estimation II (second/third session)
4. Kernel regression estimation I (fourth/fifth session)

5. Kernel regression estimation II (fifth/sixth session)
6. Nonparametric tests (seventh session)

Some logistics for the development of the course follow:

- **Office hours** are described in the Aula Global (right panel).
- **Questions and comments** during lectures are most welcome. Particularly if these are clarifications, comments, or alternative perspectives that may help the rest of the class. So just go ahead and fire!
- Detailed **course evaluation** guidelines can be found in the Aula Global. Recall that participation in lessons is positively evaluated.

Main references and credits

Several great reference books have been used for preparing these notes. The following list presents the books that have been consulted:

- D'Agostino and Stephens (1986) (Section 6.1).
- Chacón and Duong (2018) (Sections 3.1, 3.3, 3.4, and 3.5).
- DasGupta (2008) (Sections 1.1, 1.3, 1.4, and 6.1).
- Fan and Gijbels (1996) (Sections 4.1, 4.2, and 4.3).
- Li and Racine (2007) (Section 5.1).
- Loader (1999) (Section 5.4).
- Nelsen (2006) (Section 6.3.1).
- Scott (2015) (Sections 2.1, 2.2, and 2.4)).
- Sheskin (2011) (Section 6.2.2).
- Silverman (1986) (Sections 2.2 and 2.4).
- van der Vaart (1998) (Sections 1.3 and 1.4).
- Wand and Jones (1995) (Sections 2.2, 2.4, 2.4, 2.5, and 5.4).
- Wasserman (2004) (Sections 1.3, 5.4, and B).
- Wasserman (2006) (Sections 1.1, 4.3, and A).

These notes are possible due to the existence of the incredible pieces of software by Xie (2016), Xie (2020), Allaire et al. (2020), Xie and Allaire (2020), and R Core Team (2020). Also, certain hacks to improve the design layout have been possible due to the outstanding work by Úcar (2018). The icons used in the notes were designed by madebyoliver, freepik, and roundicons from Flaticon.

Last but not least, the notes have benefited from contributions from the following people:

- Roberto Jesús Alcaraz Molina (fixed a typo)
- Dimitar Aleksandrov Terziev (fixed nine typos)
- Irina Antich Moreno (identified a typo)
- Rafaela Becerra Robalino (performed a thorough revision of the course materials fixing more than twenty typos)
- Germán Blanco Blanco (fixed two typos)
- Xavier Bryant (indicated a bug and fixed one typo)

- Cynthia Bueno Macedo Medeiros (fixed seven typos)
- Ilán Francisco Carretero Juchnowicz (fixed two typos)
- César Conejo Villalobos (fixed two typos)
- Marta Cortés Ocaña (fixed two typos)
- David Crespo Acero (fixed four typos)
- Álvaro Díaz Pérez (fixed five typos)
- Alba Diego Velarde (fixed one typo)
- Mauricio Marcos Fajgenbaun (fixed two typos)
- Alberto Fernández de Marcos Giménez-Galanes (fixed three typos)
- David de la Fuente López (fixed three typos and three bugs)
- Javier Gámez Sanz (fixed two typos)
- Manuel García Corbí (indicated two typos)
- María Del Pilar González Barquero (indicated a typo)
- Luis González-Conde Sánchez-Crespo (fixed two bugs)
- Fabian Guignard (fixed three typos)
- Hongfei Guo (fixed a bug)
- Marta Ilundain Martínez (fixed a typo)
- Amalia Jiménez Toledano (indicated a typo)
- Mike Knecht (fixed one typo)
- Julen Leo Gilete (fixed a bug)
- Javier López Fernández (fixed a bug)
- David Enrique Merchán Cano (fixed a typo)
- Rafael Monsalve Roquero (indicated a typo)
- Juan Montero (fixed a typo)
- Pablo Morala Miguélez (fixed two typos)
- Manuel Navarro García (performed a thorough revision of the course materials fixing fifteen typos)
- Miguel Novillo Arana (indicated a typo)
- Berta de Pablo Brito (fixed two typos)
- Sergio Palacio Vega (fixed a typo)
- Guendalina Palmirotta (fixed a typo)
- Georgia Papadogeorgou (indicated a typo)
- Raquel Parra Suazo (indicated two typos)
- David Parrón Duce (fixed one bug)
- María del Carmen Paternina Die (indicated two typos)
- Eloy Pérez Gómez (fixed one typo)
- Pilar Pérez Piedra (fixed one typo)
- David Pérez Ros (fixed one typo)
- Jingye Qian (indicated two bugs)
- Paloma Romero Palop (fixed a typo)
- Jorge Sánchez Polo (fixed three typos)
- Camila San José (fixed three typos)
- Diego Serrano Ortega (fixed three typos and one bug)
- Kendal Raymond William Smith (performed a thorough revision of the course materials fixing more than twenty typos)
- Anna Subirós de Arriaga (fixed three typos)
- He Sun (indicated two typos)
- Adrián Torres Núñez (fixed eleven typos)

- Jaime Ugarte Abollado (indicated a typo)
- Adrian Bijan White (fixed a typo)
- Du Zhang (indicated a bug and a typo)
- Shenbin Zheng (indicated a typo)

Contributions

Contributions, reporting of typos, and feedback on the notes are very welcome. Just send an email to `edgarcia@est-econ.uc3m.es` and give me a good reason for writing your name in the list of contributors!

License

Citation

You may use the following BibTeX entry when citing these notes:

```
@book{Garcia-Portugues2023,
    title        = {Notes for Nonparametric Statistics},
    author       = {Garc\'ia-Portugu\'es, E.},
    year         = {2023},
    note         = {Version 6.9.0. ISBN 978-84-09-29537-1},
    url          = {https://bookdown.org/egarpor/NP-UC3M/}
}
```

You may also want to use the following template:

> García-Portugués, E. (2023). *Notes for Nonparametric Statistics*. Version 6.9.0. ISBN 978-84-09-29537-1. Available at `https://bookdown.org/egarpor/NP-UC3M/`.

Course overview

The notes contain a substantial amount of snippets of code that are fully self-contained within the chapter in which they are included. This allows understanding of how the methods and theory translate neatly to the practice. The **software** employed in the course is *the* statistical language R and its most common IDE (Integrated Development Environment) nowadays, RStudio. Prior basic knowledge of both is assumed.[1]

[1] Among others: basic programming in R, ability to work with objects and data structures, ability to produce graphics, knowledge of the main statistical functions, and ability to run scripts in RStudio.

The **Shiny interactive apps** on the notes can be downloaded and run locally, which in particular allows inspection of their codes. Check out this GitHub repository for the sources.

We will employ several packages that are not included within R by default. These can be installed as:

```
# Installation of required packages
packages <- c("ks", "mvtnorm", "nor1mix", "rgl", "misc3d", "viridis",
              "manipulate", "geometry", "numDeriv", "OceanView", "ISLR",
              "emstreeR", "circular", "maps", "MASS", "microbenchmark",
              "np", "locfit", "latex2exp", "dgof", "goftest", "nortest",
              "boot", "energy")
install.packages(packages)
```

The notes make explicit mention of the package to which a function belongs by using the operator `::`, unless when the use of the functions of a package is very repetitive and that package is loaded. You can load all the packages by running:

```
# Load packages
lapply(packages, library, character.only = TRUE)
```

Scripts

The snippets of code of the notes are conveniently collected in the following scripts. To download them, simply **save the link as a file** in your browser.

- Chapter 1: `01-intro.R`.
- Chapter 2: `02-kde-i.R`.
- Chapter 3: `03-kde-ii.R`.
- Chapter 4: `04-kre-i.R`.
- Chapter 5: `05-kre-ii.R`
- Chapter 6: `06-nptests.R`

1
Introduction

We begin by reviewing some elementary results that will be employed during the course and which will also serve to introduce notation.

1.1 Probability review

1.1.1 Random variables

A triple $(\Omega, \mathcal{A}, \mathbb{P})$ is called a *probability space*. Ω represents the *sample space*, the set of all possible individual outcomes of a random experiment. $\mathcal{A}$ is a *σ-field*, a class of subsets of Ω that is closed under complementation and numerable unions, and such that $\Omega \in \mathcal{A}$. $\mathcal{A}$ represents the collection of possible *events* (combinations of individual outcomes) that are assigned a probability by the *probability measure* $\mathbb{P}$. A *random variable* is a map $X : \Omega \longrightarrow \mathbb{R}$ such that $X^{-1}((-\infty, x]) = \{\omega \in \Omega : X(\omega) \leq x\} \in \mathcal{A}, \forall x \in \mathbb{R}$ (the set $X^{-1}((-\infty, x])$ of possible outcomes of X is said *measurable*).

1.1.2 Cumulative distribution and probability density functions

The *cumulative distribution function* (cdf) of a random variable X is $F(x) := \mathbb{P}[X \leq x]$. When an *independent and identically distributed* (iid) sample $X_1, \ldots, X_n$ is given, the cdf can be estimated by the *empirical distribution function* (ecdf)

$$F_n(x) = \frac{1}{n} \sum_{i=1}^{n} 1_{\{X_i \leq x\}}, \tag{1.1}$$

where $1_A := \begin{cases} 1, & A \text{ is true,} \\ 0, & A \text{ is false} \end{cases}$ is an *indicator function*.[1]

[1] Inspiration for (1.1) comes from realizing that $F(x) = \mathbb{E}[1_{\{X \leq x\}}]$.

Continuous random variables are characterized by either the cdf F or the *probability density function* (pdf) $f = F'$, the latter representing the *infinitesimal relative probability* of X per unit of length. We write $X \sim F$ (or $X \sim f$) to denote that X has a cdf F (or a pdf f). If two random variables X and Y have the same distribution, we write $X \overset{d}{=} Y$.

1.1.3 Expectation

The *expectation* operator is constructed using the Lebesgue–Stieljes "$\mathrm{d}F(x)$" integral. Hence, for $X \sim F$, the expectation of $g(X)$ is

$$\begin{aligned}\mathbb{E}[g(X)] &:= \int g(x)\,\mathrm{d}F(x)\\ &= \begin{cases}\displaystyle\int g(x)f(x)\,\mathrm{d}x, & \text{if } X \text{ is continuous,}\\ \displaystyle\sum_{\{x\in\mathbb{R}:\mathbb{P}[X=x]>0\}} g(x)\mathbb{P}[X=x], & \text{if } X \text{ is discrete.}\end{cases}\end{aligned}$$

Unless otherwise stated, the integration limits of any integral are $\mathbb{R}$ or $\mathbb{R}^p$. The variance operator is defined as $\mathbb{V}\mathrm{ar}[X] := \mathbb{E}[(X - \mathbb{E}[X])^2]$.

1.1.4 Random vectors, marginals, and conditionals

We employ bold face to denote vectors, assumed to be *column* matrices, and matrices. A *p-random vector* is a map $\mathbf{X} : \Omega \longrightarrow \mathbb{R}^p$, $\mathbf{X}(\omega) := (X_1(\omega), \ldots, X_p(\omega))'$, such that each X_i is a random variable. The (joint) cdf of $\mathbf{X}$ is $F(\mathbf{x}) := \mathbb{P}[\mathbf{X} \leq \mathbf{x}] := \mathbb{P}[X_1 \leq x_1, \ldots, X_p \leq x_p]$ and, if $\mathbf{X}$ is continuous, its (joint) pdf is $f := \frac{\partial^p}{\partial x_1 \cdots \partial x_p} F$.

The *marginals* of F and f are the cdfs and pdfs of X_i, $i = 1, \ldots, p$, respectively. They are defined as

$$\begin{aligned}F_{X_i}(x_i) &:= \mathbb{P}[X_i \leq x_i] = F(\infty, \ldots, \infty, x_i, \infty, \ldots, \infty),\\ f_{X_i}(x_i) &:= \frac{\partial}{\partial x_i} F_{X_i}(x_i) = \int_{\mathbb{R}^{p-1}} f(\mathbf{x})\,\mathrm{d}\mathbf{x}_{-i},\end{aligned}$$

where $\mathbf{x}_{-i} := (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_p)'$. The definitions can be extended analogously to the marginals of the cdf and pdf of different subsets of $\mathbf{X}$.

The *conditional* cdf and pdf of $X_1|(X_2, \ldots, X_p)$ are defined, respectively, as

$$\begin{aligned}F_{X_1|\mathbf{X}_{-1}=\mathbf{x}_{-1}}(x_1) &:= \mathbb{P}[X_1 \leq x_1 | \mathbf{X}_{-1} = \mathbf{x}_{-1}],\\ f_{X_1|\mathbf{X}_{-1}=\mathbf{x}_{-1}}(x_1) &:= \frac{f(\mathbf{x})}{f_{\mathbf{X}_{-1}}(\mathbf{x}_{-1})}.\end{aligned}$$

The *conditional expectation* of $Y|X$ is the following random variable[2]

$$\mathbb{E}[Y|X] := \int y\,\mathrm{d}F_{Y|X}(y|X).$$

[2] Recall that the X-part of $\mathbb{E}[Y|X]$ is *random*. However, $\mathbb{E}[Y|X = x]$ is *deterministic*.

The *conditional variance* of $Y|X$ is defined as

$$\mathbb{V}\mathrm{ar}[Y|X] := \mathbb{E}[(Y - \mathbb{E}[Y|X])^2|X] = \mathbb{E}[Y^2|X] - \mathbb{E}[Y|X]^2.$$

Proposition 1.1 (Laws of total expectation and variance)**.** *Let X and Y be two random variables.*

- *Total expectation: if* $\mathbb{E}[|Y|] < \infty$, *then* $\mathbb{E}[Y] = \mathbb{E}[\mathbb{E}[Y|X]]$.
- *Total variance: if* $\mathbb{E}[Y^2] < \infty$, *then* $\mathbb{V}\mathrm{ar}[Y] = \mathbb{E}[\mathbb{V}\mathrm{ar}[Y|X]] + \mathbb{V}\mathrm{ar}[\mathbb{E}[Y|X]]$.

Exercise 1.1. Prove the law of total variance from the law of total expectation.

Figure 1.1 graphically summarizes the concepts of joint, marginal, and conditional distributions within the context of a 2-dimensional normal.

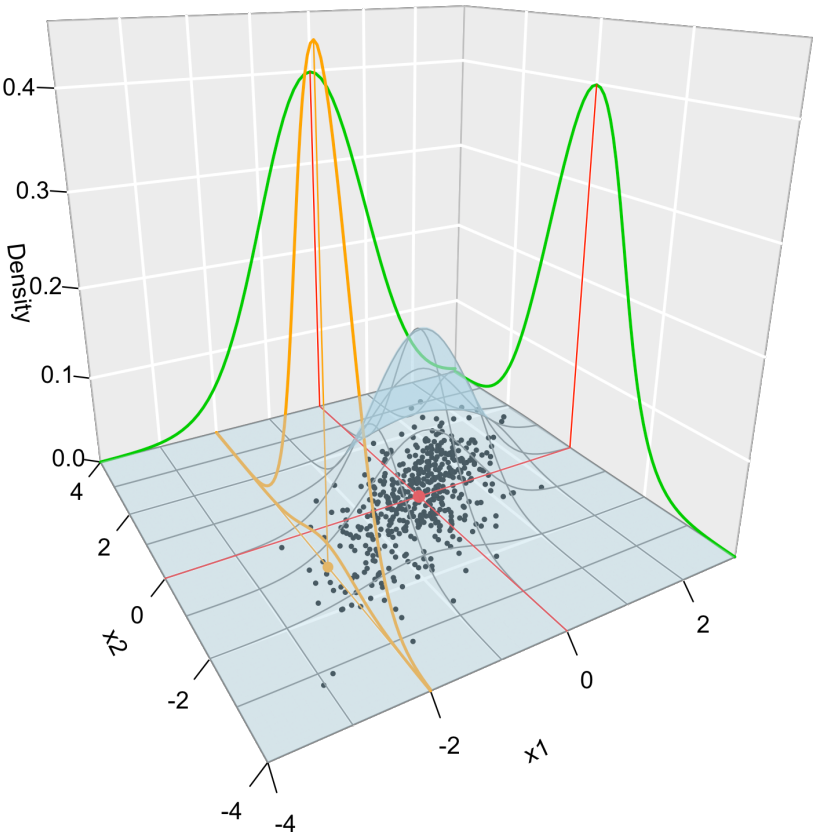


Figure 1.1: Visualization of the joint pdf (in blue), marginal pdfs (green), conditional pdf of $X_2|X_1 = x_1$ (orange), expectation (red point), and conditional expectation $\mathbb{E}[X_2|X_1 = x_1]$ (orange point) of a 2-dimensional normal. The conditioning point of X_1 is $x_1 = -2$. Note the different scales of the densities, as they have to integrate one over different supports. Note how the conditional density (upper orange curve) is *not* the joint pdf $f(x_1, x_2)$ (lower orange curve) with $x_1 = -2$ but its rescaling by $\frac{1}{f_{X_1}(x_1)}$. The parameters of the 2-dimensional normal are $\mu_1 = \mu_2 = 0$, $\sigma_1 = \sigma_2 = 1$ and $\rho = 0.75$ (see Exercise 1.9). 500 observations sampled from the distribution are shown in black.

Exercise 1.2. Consider the random vector (X, Y) with joint pdf

$$f(x,y) = \begin{cases} ye^{-axy}, & x > 0,\ y \in (0,b), \\ 0, & \text{else.} \end{cases}$$

a. Determine the value of $b > 0$ that makes f a valid pdf.
b. Compute $\mathbb{E}[X]$ and $\mathbb{E}[Y]$.
c. Verify the law of the total expectation.
d. Verify the law of the total variance.

Exercise 1.3. Consider the continuous random vector (X_1, X_2) with joint pdf given by

$$f(x_1,x_2) = \begin{cases} 2, & 0 < x_1 < x_2 < 1, \\ 0, & \text{else.} \end{cases}$$

a. Check that f is a proper pdf.
b. Obtain the joint cdf of (X_1, X_2).
c. Obtain the marginal pdfs of X_1 and X_2.
d. Obtain the marginal cdfs of X_1 and X_2.
e. Obtain the conditional pdfs of $X_1|X_2 = x_2$ and $X_2|X_1 = x_1$.

1.1.5 *Variance-covariance matrix*

For two random variables X_1 and X_2, the *covariance* between them is defined as

$$\mathrm{Cov}[X_1, X_2] := \mathbb{E}[(X_1 - \mathbb{E}[X_1])(X_2 - \mathbb{E}[X_2])] = \mathbb{E}[X_1X_2] - \mathbb{E}[X_1]\mathbb{E}[X_2],$$

and the *correlation* between them, as

$$\mathrm{Cor}[X_1, X_2] := \frac{\mathrm{Cov}[X_1, X_2]}{\sqrt{\mathbb{V}\mathrm{ar}[X_1]\mathbb{V}\mathrm{ar}[X_2]}}.$$

The variance and the covariance are extended to a random vector $\mathbf{X} = (X_1, \ldots, X_p)'$ by means of the so-called *variance-covariance matrix*:

$$\begin{aligned}\mathbb{V}\mathrm{ar}[\mathbf{X}] :=& \mathbb{E}[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])'] \\ =& \mathbb{E}[\mathbf{X}\mathbf{X}'] - \mathbb{E}[\mathbf{X}]\mathbb{E}[\mathbf{X}]' \\ =& \begin{pmatrix} \mathbb{V}\mathrm{ar}[X_1] & \mathrm{Cov}[X_1, X_2] & \cdots & \mathrm{Cov}[X_1, X_p] \\ \mathrm{Cov}[X_2, X_1] & \mathbb{V}\mathrm{ar}[X_2] & \cdots & \mathrm{Cov}[X_2, X_p] \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}[X_p, X_1] & \mathrm{Cov}[X_p, X_2] & \cdots & \mathbb{V}\mathrm{ar}[X_p] \end{pmatrix},\end{aligned}$$

where $\mathbb{E}[\mathbf{X}] := (\mathbb{E}[X_1], \ldots, \mathbb{E}[X_p])'$ is just the componentwise expectation. As in the univariate case, the expectation is a linear operator, which now means that

$$\mathbb{E}[\mathbf{A}\mathbf{X} + \mathbf{b}] = \mathbf{A}\mathbb{E}[\mathbf{X}] + \mathbf{b}, \quad \text{for a } q \times p \text{ matrix } \mathbf{A} \text{ and } \mathbf{b} \in \mathbb{R}^q. \tag{1.2}$$

It follows from (1.2) that

$$\mathbb{V}\mathrm{ar}[\mathbf{A}\mathbf{X} + \mathbf{b}] = \mathbf{A}\mathbb{V}\mathrm{ar}[\mathbf{X}]\mathbf{A}', \quad \text{for a } q \times p \text{ matrix } \mathbf{A} \text{ and } \mathbf{b} \in \mathbb{R}^q. \tag{1.3}$$

1.1.6 *Inequalities*

We conclude this section by reviewing some useful probabilistic inequalities.

Proposition 1.2 (Markov's inequality)**.** *Let X be a non-negative random variable with $\mathbb{E}[X] < \infty$. Then*

$$\mathbb{P}[X \geq t] \leq \frac{\mathbb{E}[X]}{t}, \quad \forall t > 0.$$

Proposition 1.3 (Chebyshev's inequality)**.** *Let X be a random variable with $\mu = \mathbb{E}[X]$ and $\sigma^2 = \mathbb{V}\mathrm{ar}[X] < \infty$. Then*

$$\mathbb{P}[|X - \mu| \geq t] \leq \frac{\sigma^2}{t^2}, \quad \forall t > 0.$$

Exercise 1.4. Prove Markov's inequality using $X = X1_{\{X \geq t\}} + X1_{\{X < t\}}$.

Exercise 1.5. Prove Chebyshev's inequality using Markov's.

Remark. Chebyshev's inequality gives a quick and handy way of computing confidence intervals for the values of *any* random variable X with finite variance:

$$\mathbb{P}[X \in (\mu - t\sigma, \mu + t\sigma)] \geq 1 - \frac{1}{t^2}, \quad \forall t > 0. \tag{1.4}$$

That is, for any $t > 0$, the interval $(\mu - t\sigma, \mu + t\sigma)$ has, at least, a probability $1 - 1/t^2$ of containing a random realization of X. The intervals are conservative, but extremely general. The table below gives the guaranteed coverage probability $1 - 1/t^2$ for common values of t.

t	2	3	4	5	6
Guaranteed coverage	0.75	0.8889	0.9375	0.96	0.9722

Exercise 1.6. Prove (1.4) from Chebyshev's inequality.

Proposition 1.4 (Cauchy–Schwartz inequality)**.** *Let X and Y such that $\mathbb{E}[X^2] < \infty$ and $\mathbb{E}[Y^2] < \infty$. Then*

$$|\mathbb{E}[XY]| \leq \sqrt{\mathbb{E}[X^2]\mathbb{E}[Y^2]}.$$

Exercise 1.7. Prove Cauchy–Schwartz inequality "pulling a rabbit out of a hat": consider the polynomial $p(t) = \mathbb{E}[(tX + Y)^2] = At^2 + 2Bt + C \geq 0$, $\forall t \in \mathbb{R}$.

Exercise 1.8. Does $\mathbb{E}[|XY|] \leq \sqrt{\mathbb{E}[X^2]\mathbb{E}[Y^2]}$ hold? Observe that, due to the next proposition, $|\mathbb{E}[XY]| \leq \mathbb{E}[|XY|]$.

Proposition 1.5 (Jensen's inequality)**.** *If g is a convex function, then*

$$g(\mathbb{E}[X]) \leq \mathbb{E}[g(X)].$$

Example 1.1. Jensen's inequality has interesting derivations. For example:

1. Take $h = -g$. Then h is a concave function and $h(\mathbb{E}[X]) \geq \mathbb{E}[h(X)]$.
2. Take $g(x) = x^r$ for $r \geq 1$, which is a convex function. Then $\mathbb{E}[X]^r \leq \mathbb{E}[X^r]$. If $0 < r < 1$, then $g(x) = x^r$ is concave and $\mathbb{E}[X]^r \geq \mathbb{E}[X^r]$.
3. The previous results hold considering $g(x) = |x|^r$. In particular, $|\mathbb{E}[X]| \leq \mathbb{E}[|X|]$ for $r \geq 1$.
4. Consider $0 \leq r \leq s$. Then $g(x) = x^{s/r}$ is convex (since $s/r \geq 1$) and $g(\mathbb{E}[|X|^r]) \leq \mathbb{E}[g(|X|^r)] = \mathbb{E}[|X|^s]$. As a consequence, $\mathbb{E}[|X|^s] < \infty \implies \mathbb{E}[|X|^r] < \infty$ for $0 \leq r \leq s$.[3]
5. The exponential (logarithm) function is convex (concave). Consequently, $\exp(\mathbb{E}[X]) \leq \mathbb{E}[\exp(X)]$ and $\log(\mathbb{E}[|X|]) \geq \mathbb{E}[\log(|X|)]$.

[3] "Finite moments of higher order imply finite moments of lower order".

1.2 *Facts about distributions*

We will make use of certain parametric distributions. Some notation and facts are introduced as follows.

1.2.1 *Normal distribution*

The *normal distribution* with mean μ and variance σ^2 is denoted by $\mathcal{N}(\mu, \sigma^2)$. Its pdf is $\phi_\sigma(x-\mu) := \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, $x \in \mathbb{R}$, and satisfies that $\phi_\sigma(x-\mu) = \frac{1}{\sigma}\phi\left(\frac{x-\mu}{\sigma}\right)$ (if $\sigma = 1$ the dependence on σ is omitted). Its cdf is denoted by $\Phi_\sigma(x-\mu)$. The upper α-quantile of a $\mathcal{N}(0,1)$ is denoted by z_α, so it satisfies that $z_\alpha = \Phi^{-1}(1-\alpha)$.[4] The shortest interval that contains $1-\alpha$ probability of a $X \sim \mathcal{N}(\mu, \sigma^2)$ is $(\mu - z_{\alpha/2}\sigma, \mu + z_{\alpha/2}\sigma)$, i.e., $\mathbb{P}[X \in (\mu \pm z_{\alpha/2}\sigma)] = 1-\alpha$. Some uncentered moments of $X \sim \mathcal{N}(\mu, \sigma^2)$ are

[4] A particular useful value for computing confidence intervals is $z_{0.05/2} = z_{0.025} \approx 1.96 \approx 2$.

$$\begin{aligned}
\mathbb{E}[X] &= \mu, \\
\mathbb{E}[X^2] &= \mu^2 + \sigma^2, \\
\mathbb{E}[X^3] &= \mu^3 + 3\mu\sigma^2, \\
\mathbb{E}[X^4] &= \mu^4 + 6\mu^2\sigma^2 + 3\sigma^4.
\end{aligned}$$

Remark. It is interesting to compare the length of $(\mu \pm z_{\alpha/2}\sigma)$ for $\alpha = 1/t^2$ with the one in (1.4), as this gives direct insight into how larger the Chebyshev confidence interval (1.4) is when $X \sim \mathcal{N}(\mu, \sigma^2)$. The table below gives the length increment factor $t/z_{(0.5/t^2)}$ of the Chebyshev confidence interval.

t	2	3	4	5	6
Guaranteed coverage	0.75	0.8889	0.9375	0.96	0.9722
Increment factor	1.7386	1.883	2.1474	2.4346	2.7268

Balancing between the guaranteed coverage and increment factor, it seems reasonable to define the "3σ-rule" for *any* random variable as: "almost 90% of the values of a random variable X lie on $(\mu - 3\sigma, \mu + 3\sigma)$, if $\mathbb{E}[X] = \mu$ and $\mathbb{V}\text{ar}[X] = \sigma^2 < \infty$".

The *multivariate normal* is represented by $\mathcal{N}_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu}$ is a p-vector and $\boldsymbol{\Sigma}$ is a $p \times p$ symmetric and positive matrix. The pdf of a $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is $\phi_{\boldsymbol{\Sigma}}(\mathbf{x}-\boldsymbol{\mu}) := \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}$ and satisfies that $\phi_{\boldsymbol{\Sigma}}(\mathbf{x}-\boldsymbol{\mu}) = |\boldsymbol{\Sigma}|^{-1/2}\phi\left(\boldsymbol{\Sigma}^{-1/2}(\mathbf{x}-\boldsymbol{\mu})\right)$ (if $\boldsymbol{\Sigma} = \mathbf{I}$, the dependence on $\boldsymbol{\Sigma}$ is omitted). The multivariate normal has an appealing linear property that stems from (1.2) and (1.3):

$$\mathbf{A}\mathcal{N}_p(\boldsymbol{\mu}, \boldsymbol{\Sigma}) + \mathbf{b} \overset{d}{=} \mathcal{N}_q(\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}'). \tag{1.5}$$

Exercise 1.9. The pdf of a bivariate normal ($p = 2$, see Figure 1.1) can be also expressed as

$$\begin{aligned}
\phi(x_1, x_2; \mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \rho) &:= \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \\
&\times \exp\left\{-\frac{1}{2(1-\rho^2)}\left[\frac{(x_1-\mu_1)^2}{\sigma_1^2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2}\right]\right\}
\end{aligned} \tag{1.6}$$

[5] Note that this is an immediate parametrization of a 2×2 covariance matrix. The parametrization becomes cumbersome when $p > 2$.

where $\mu_1, \mu_2 \in \mathbb{R}$, $\sigma_1, \sigma_2 > 0$, and $-1 < \rho < 1$. The parametrization uses $\boldsymbol{\mu} = (\mu_1, \mu_2)'$ and $\boldsymbol{\Sigma} = (\sigma_1^2, \rho\sigma_1\sigma_2; \rho\sigma_1\sigma_2, \sigma_2^2)$.[5]

a. Derive the pdf of X_1: $\phi(x_1; \mu_1, \sigma_1^2)$.
b. Derive the pdf of $X_1|X_2 = x_2$: $\phi\left(x_1; \mu_1 + \rho\frac{\sigma_1}{\sigma_2}(x_2 - \mu_2), (1-\rho^2)\sigma_1^2\right)$.
c. Derive $\mathbb{E}[X_1|X_2 = x_2]$ and $\mathbb{V}\text{ar}[X_1|X_2 = x_2]$.

1.2.2 *Other distributions*

- The *lognormal distribution* is denoted by $\mathcal{LN}(\mu, \sigma^2)$ and is such that $\mathcal{LN}(\mu, \sigma^2) \overset{d}{=} \exp(\mathcal{N}(\mu, \sigma^2))$. Its pdf is $f(x; \mu, \sigma) = \frac{1}{x}\phi_\sigma(\log x - \log \mu) = \frac{1}{\sqrt{2\pi}\sigma x} e^{-\frac{(\log x - \log \mu)^2}{2\sigma^2}}$, $x > 0$. Note that $\mathbb{E}[\mathcal{LN}(\mu, \sigma^2)] = e^{\mu + \frac{\sigma^2}{2}}$ and $\mathbb{V}\text{ar}[\mathcal{LN}(\mu, \sigma^2)] = \left(e^{\sigma^2} - 1\right) e^{2\mu + \sigma^2}$.

- The *exponential distribution* is denoted by $\text{Exp}(\lambda)$ and has pdf $f(x; \lambda) = \lambda e^{-\lambda x}$, $\lambda, x > 0$.

- The *gamma distribution* is denoted by $\Gamma(a, p)$ and has pdf $f(x; a, p) = \frac{a^p}{\Gamma(p)} x^{p-1} e^{-ax}$, $a, p, x > 0$, where $\Gamma(p) = \int_0^\infty x^{p-1} e^{-ax}\, \mathrm{d}x$. The parameter a is the *rate* and p is the *shape*. It is known that $\mathbb{E}[\Gamma(a, p)] = \frac{p}{a}$ and $\mathbb{V}\text{ar}[\Gamma(a, p)] = \frac{p}{a^2}$.

- The *inverse gamma distribution*, $\text{IG}(a, p) \overset{d}{=} \Gamma(a, p)^{-1}$, has pdf $f(x; a, p) = \frac{a^p}{\Gamma(p)} x^{-p-1} e^{-\frac{a}{x}}$, $a, p, x > 0$. It is known that $\mathbb{E}[\text{IG}(a, p)] = \frac{a}{p-1}$ and $\mathbb{V}\text{ar}[\text{IG}(a, p)] = \frac{a^2}{(p-1)^2(p-2)}$.

- The *binomial distribution* is denoted by $\text{B}(n, p)$. Recall that $\mathbb{E}[\text{B}(n, p)] = np$ and $\mathbb{V}\text{ar}[\text{B}(n, p)] = np(1-p)$. A $\text{B}(1, p)$ is a *Bernoulli distribution*, denoted by $\text{Ber}(p)$.

- The *beta distribution* is denoted by $\beta(a, b)$ and its pdf is $f(x; a, b) = \frac{1}{\beta(a,b)} x^{a-1}(1-x)^{1-b}$, $0 < x < 1$, where $\beta(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$. When $a = b = 1$, the *uniform distribution* $\mathcal{U}(0, 1)$ arises.

- The *Poisson distribution* is denoted by $\text{Pois}(\lambda)$ and has probability mass function $\mathbb{P}[X = x] = \frac{x^\lambda e^{-\lambda}}{x!}$, $x = 0, 1, 2, \ldots$ Recall that $\mathbb{E}[\text{Pois}(\lambda)] = \mathbb{V}\text{ar}[\text{Pois}(\lambda)] = \lambda$.

1.3 *Stochastic convergence review*

Let X_n be a sequence of random variables defined in a common probability space $(\Omega, \mathcal{A}, \mathbb{P})$. The four most common types of convergence of X_n to a random variable in $(\Omega, \mathcal{A}, \mathbb{P})$ are the following.

Definition 1.1 (Convergence in distribution)**.** *X_n converges in distribution* to X, written $X_n \xrightarrow{d} X$, if $\lim_{n\to\infty} F_n(x) = F(x)$ for all x for which F is continuous, where $X_n \sim F_n$ and $X \sim F$.

"Convergence in distribution" is also referred to as *weak convergence*. Proposition 1.6 justifies why this terminology.

Definition 1.2 (Convergence in probability)**.** X_n *converges in probability* to X, written $X_n \xrightarrow{\mathbb{P}} X$, if $\lim_{n\to\infty} \mathbb{P}[|X_n - X| > \varepsilon] = 0$, $\forall \varepsilon > 0$.

Definition 1.3 (Convergence almost surely)**.** X_n *converges almost surely* (as) to X, written $X_n \xrightarrow{\text{as}} X$, if $\mathbb{P}[\{\omega \in \Omega : \lim_{n\to\infty} X_n(\omega) = X(\omega)\}] = 1$.

Definition 1.4 (Convergence in r-mean)**.** For $r \geq 1$, X_n *converges in r-mean* to X, written $X_n \xrightarrow{r} X$, if $\lim_{n\to\infty} \mathbb{E}[|X_n - X|^r] = 0$.

Remark. The previous definitions can be extended to a sequence of p-random vectors $\mathbf{X}_n$. For Definitions 1.2 and 1.4, replace $|\cdot|$ with the Euclidean norm $||\cdot||$. Alternatively, Definition 1.2 can be extended *marginally*: $\mathbf{X}_n \xrightarrow{\mathbb{P}} \mathbf{X} :\iff X_{j,n} \xrightarrow{\mathbb{P}} X_j$, $\forall j = 1, \ldots, p$. For Definition 1.1, replace F_n and F by the joint cdfs of $\mathbf{X}_n$ and $\mathbf{X}$, respectively. Definition 1.3 also extends marginally.

The 2-mean convergence plays a remarkable role in defining a consistent estimator $\hat{\theta}_n = T(X_1, \ldots, X_n)$ of a parameter θ. We say that the *estimator is consistent* if its *Mean Squared Error* (MSE),

$$\begin{aligned} \text{MSE}[\hat{\theta}_n] &:= \mathbb{E}[(\hat{\theta}_n - \theta)^2] \\ &= (\mathbb{E}[\hat{\theta}_n] - \theta)^2 + \mathbb{V}\text{ar}[\hat{\theta}_n] \\ &=: \text{Bias}[\hat{\theta}_n]^2 + \mathbb{V}\text{ar}[\hat{\theta}_n], \end{aligned}$$

goes to zero as $n \to \infty$. Equivalently written, if $\hat{\theta}_n \xrightarrow{2} \theta$.

If $X_n \xrightarrow{d,\mathbb{P},r,\text{as}} X$ and X is a *degenerate* random variable such that $\mathbb{P}[X = c] = 1$, $c \in \mathbb{R}$, then we write $X_n \xrightarrow{d,\mathbb{P},r,\text{as}} c$ (the list-notation $\xrightarrow{d,\mathbb{P},r,\text{as}}$ is used to condensate four different convergence results in the same line).

The relations between the types of convergences are conveniently summarized in the following proposition.

Proposition 1.6. *Let X_n be a sequence of random variables and X a random variable. Then the following implication diagram is satisfied:*

$$\begin{array}{ccccc} X_n \xrightarrow{r} X & \implies & X_n \xrightarrow{\mathbb{P}} X & \impliedby & X_n \xrightarrow{\text{as}} X \\ & & \Downarrow & & \\ & & X_n \xrightarrow{d} X & & \end{array}$$

Also, if $s \geq r \geq 1$, then $X_n \xrightarrow{s} X \implies X_n \xrightarrow{r} X$.

None of the converses holds in general. However, there are some notable exceptions:

i. If $X_n \xrightarrow{d} c$, then $X_n \xrightarrow{\mathbb{P}} c$, $c \in \mathbb{R}$.

ii. If $\forall \varepsilon > 0$, $\lim_{n\to\infty} \sum_n \mathbb{P}[|X_n - X| > \varepsilon] < \infty$ (implies[6] $X_n \xrightarrow{\mathbb{P}} X$), then $X_n \xrightarrow{\text{as}} X$.

iii. If $X_n \xrightarrow{\mathbb{P}} X$ and $\mathbb{P}[|X_n| \leq M] = 1, \forall n \in \mathbb{N}$ and $M > 0$[7], then $X_n \xrightarrow{r} X$ for $r \geq 1$.

iv. If $S_n = \sum_{i=1}^n X_i$ with $X_1, \ldots, X_n$ iid, then $S_n \xrightarrow{\mathbb{P}} S \iff S_n \xrightarrow{\text{as}} S$.

[6] Intuitively: if convergence in probability is fast enough, then we have almost surely convergence.

[7] "Uniformly bounded random variables".

The weak *Law of Large Numbers* (LLN) and its strong version are the two most representative results of convergence in probability and almost surely.

Theorem 1.1 (Weak and strong LLN)**.** *Let X_n be an iid sequence with $\mathbb{E}[X_i] = \mu$, $i \geq 1$. Then: $\frac{1}{n}\sum_{i=1}^n X_i \xrightarrow{\mathbb{P},\text{as}} \mu$.*

The corner stone limit result in probability is the *Central Limit Theorem* (CLT). One of its simpler versions has the following form.

Theorem 1.2 (CLT)**.** *Let X_n be a sequence of iid random variables with $\mathbb{E}[X_i] = \mu$ and $\mathbb{V}\text{ar}[X_i] = \sigma^2 < \infty$, $i \in \mathbb{N}$. Then:*

$$\frac{\sqrt{n}(\bar{X} - \mu)}{\sigma} \xrightarrow{d} \mathcal{N}(0,1).$$

We will later use the following CLT for random variables that are independent but *not* identically distributed due to its easy-to-check moment conditions.

Theorem 1.3 (Lyapunov's CLT)**.** *Let X_n be a sequence of independent random variables with $\mathbb{E}[X_i] = \mu_i$ and $\mathbb{V}\text{ar}[X_i] = \sigma_i^2 < \infty$, $i \in \mathbb{N}$, and such that for some $\delta > 0$*

$$\frac{1}{s_n^{2+\delta}} \sum_{i=1}^n \mathbb{E}\left[|X_i - \mu_i|^{2+\delta}\right] \longrightarrow 0 \text{ as } n \to \infty,$$

with $s_n^2 = \sum_{i=1}^n \sigma_i^2$. Then:

$$\frac{1}{s_n} \sum_{i=1}^n (X_i - \mu_i) \xrightarrow{d} \mathcal{N}(0,1).$$

Finally, the following results will be useful ($'$ denotes transposition). In particular, Slutsky's theorem allows mixing the LLNs and the CLT with additions and products.

Theorem 1.4 (Cramér–Wold device)**.** *Let $\mathbf{X}_n$ be a sequence of p-dimensional random vectors. Then:*

$$\mathbf{X}_n \xrightarrow{d} \mathbf{X} \iff \mathbf{c}'\mathbf{X}_n \xrightarrow{d} \mathbf{c}'\mathbf{X}, \quad \forall \mathbf{c} \in \mathbb{R}^p.$$

Theorem 1.5 (Continuous mapping theorem)**.** *If $\mathbf{X}_n \xrightarrow{d,\mathbb{P},\text{as}} \mathbf{X}$, then*

$$g(\mathbf{X}_n) \xrightarrow{d,\mathbb{P},\text{as}} g(\mathbf{X})$$

for any continuous function g.

Theorem 1.6 (Slutsky's theorem)**.** *Let X_n and Y_n be sequences of random variables and $c \in \mathbb{R}$. Then:*

i. If $X_n \xrightarrow{d} X$ and $Y_n \xrightarrow{\mathbb{P}} c$, then $X_n Y_n \xrightarrow{d} cX$.
ii. If $X_n \xrightarrow{d} X$ and $Y_n \xrightarrow{\mathbb{P}} c$, $c \neq 0$, then $\frac{X_n}{Y_n} \xrightarrow{d} \frac{X}{c}$.
iii. If $X_n \xrightarrow{d} X$ and $Y_n \xrightarrow{\mathbb{P}} c$, then $X_n + Y_n \xrightarrow{d} X + c$.

Theorem 1.7 (Limit algebra for $(\mathbb{P}, r, \text{as})$-convergence)**.** *Let X_n and Y_n be sequences of random variables, and $a_n \to a$ and $b_n \to b$ two sequences.*

i. If $X_n \xrightarrow{\mathbb{P},r,\text{as}} X$ *and* $Y_n \xrightarrow{\mathbb{P},r,\text{as}} Y$, *then* $a_nX_n + b_nY_n \xrightarrow{\mathbb{P},r,\text{as}} aX + bY$.
ii. If $X_n \xrightarrow{\mathbb{P},\text{as}} X$ *and* $Y_n \xrightarrow{\mathbb{P},\text{as}} Y$, *then* $X_nY_n \xrightarrow{\mathbb{P},\text{as}} XY$.

Remark. Recall the absence of the analogous results for convergence in distribution. In general, there are no such results!

- Particularly, it is **false that, in general,** $X_n \xrightarrow{d} X$ **and** $Y_n \xrightarrow{d} Y$ **imply** $X_n + Y_n \xrightarrow{d} X + Y$ **or** $X_nY_n \xrightarrow{d} XY$.
- It is true, however, that $(X_n, Y_n) \xrightarrow{d} (X, Y)$ (a much stronger premise) implies both $X_n + Y_n \xrightarrow{d} X + Y$ and $X_nY_n \xrightarrow{d} XY$, as Theorem 1.5 indicates. Note that $X_n + Y_n \xrightarrow{d} X + Y$ is also implied by $(X_n, Y_n) \xrightarrow{d} (X, Y)$ by Theorem 1.4 with $\mathbf{c} = (1, 1)'$.
- Consequently, it is also true that, under **independence** of X_n and Y_n, $X_n \xrightarrow{d} X$ and $Y_n \xrightarrow{d} Y$ imply $X_n + Y_n \xrightarrow{d} X + Y$ and $X_nY_n \xrightarrow{d} XY$

Theorem 1.8 (Delta method)**.** *If* $\sqrt{n}(X_n - \mu) \xrightarrow{d} \mathcal{N}(0, \sigma^2)$, *then*

$$\sqrt{n}(g(X_n) - g(\mu)) \xrightarrow{d} \mathcal{N}\left(0, (g'(\mu))^2\sigma^2\right)$$

for any function g *that is differentiable at* μ *and such that* $g'(\mu) \neq 0$.

Example 1.2. It is well known that, given a parametric density f_θ with parameter $\theta \in \Theta$ and iid $X_1, \ldots, X_n \sim f_\theta$, then the *Maximum Likelihood* (ML) estimator $\hat{\theta}_{\text{ML}} := \arg\max_{\theta\in\Theta} \sum_{i=1}^n \log f_\theta(X_i)$ (the parameter that maximizes the *"probability" of the data based on the model*) converges to a normal under certain regularity conditions:

$$\sqrt{n}(\hat{\theta}_{\text{ML}} - \theta) \xrightarrow{d} \mathcal{N}\left(0, I(\theta)^{-1}\right),$$

where $I(\theta) := -\mathbb{E}_\theta\left[\frac{\partial^2 \log f_\theta(x)}{\partial\theta^2}\right]$ is known as the *Fisher information*. Then, it is satisfied that

$$\sqrt{n}(g(\hat{\theta}_{\text{ML}}) - g(\theta)) \xrightarrow{d} \mathcal{N}\left(0, (g'(\theta))^2 I(\theta)^{-1}\right).$$

Note that, had we applied the continuous mapping theorem for g, we would have obtained a different result:

$$g(\sqrt{n}(\hat{\theta}_{\text{ML}} - \theta)) \xrightarrow{d} g\left(\mathcal{N}\left(0, I(\theta)^{-1}\right)\right).$$

Exercise 1.10. Let's dig further into the differences between the delta method and the continuous mapping theorem when applied to $\sqrt{n}(X_n - \mu) \xrightarrow{d} \mathcal{N}(0, \sigma^2)$:

a. Under what kind of maps g the results $\sqrt{n}(g(X_n) - g(\mu)) \xrightarrow{d} \mathcal{N}(0, (g'(\mu))^2\sigma^2)$ and $g(\sqrt{n}(X_n - \mu)) \xrightarrow{d} g(\mathcal{N}(0, \sigma^2))$ are equivalent?
b. Take $g(x) = e^x$. What two results do you obtain with the delta method and the continuous mapping theorem when applied to $\sqrt{n}\bar{X} \xrightarrow{d} \mathcal{N}(0, \sigma^2)$?

1.4 $O_{\mathbb{P}}$ and $o_{\mathbb{P}}$ notation

1.4.1 Deterministic versions

In computer science the O notation is used to measure the complexity of algorithms. For example, when an algorithm is $O(n^2)$, it is said that it is *quadratic* in time and we know that is going to take *on the order of* n^2 operations to process an input of size n. We do not care about the specific amount of computations; rather, we focus on the *big picture* by looking for an upper bound for the sequence of computation times in terms of n. This upper bound disregards constants. For example, the dot product between two vectors of size n is an $O(n)$ operation, although it takes n multiplications and $n-1$ sums, hence $2n-1$ operations.

In mathematical analysis, O-related notation is mostly used to bound sequences that *shrink to zero*. The technicalities are however the same.

Definition 1.5 (Big-O)**.** Given two strictly positive sequences a_n and b_n,

$$a_n = O(b_n) : \iff \limsup_{n\to\infty} \frac{a_n}{b_n} \leq C, \text{ for a } C > 0.$$

If $a_n = O(b_n)$, then we say that a_n *is big-O of* b_n. To indicate that a_n is bounded, we write $a_n = O(1)$.[8]

Definition 1.6 (Little-o)**.** Given two strictly positive sequences a_n and b_n,

$$a_n = o(b_n) : \iff \lim_{n\to\infty} \frac{a_n}{b_n} = 0.$$

If $a_n = o(b_n)$, then we say that a_n *is little-o of* b_n. To indicate that $a_n \to 0$, we write $a_n = o(1)$.

Exercise 1.11. Show the following statements by directly applying the previous definitions.

a. $n^{-2} = o(n^{-1})$.
b. $\log n = O(n)$.
c. $n^{-1} = o((\log n)^{-1})$.
d. $n^{-4/5} = o(n^{-2/3})$.
e. $3\sin(n) = O(1)$.
f. $n^{-2} - n^{-3} + n^{-1} = O(n^{-1})$.
g. $n^{-2} - n^{-3} = o(n^{-1/2})$.

The interpretation of these two definitions is simple:

- $a_n = O(b_n)$ means that a_n is **"not larger than"** b_n asymptotically. If $a_n, b_n \to 0$, then it means that a_n **"does not decrease more slowly"** than b_n, i.e., that a_n either decreases as fast as b_n or faster than b_n.
- $a_n = o(b_n)$ means that a_n is **"smaller than"** b_n asymptotically. If $a_n, b_n \to 0$, then it means that a_n **"decreases faster"** than b_n.

[8] For a deterministic sequence a_n, $\limsup_{n\to\infty} a_n := \lim_{n\to\infty} \left(\sup_{k\geq n} a_k\right)$ is the largest limit of the subsequences of a_n. It can be defined even if $\lim_{n\to\infty} a_n$ does not exist (e.g., in trigonometric functions). If $\lim_{n\to\infty} a_n$ exists, as in most of the common usages of the big-O notation, then $\limsup_{n\to\infty} a_n = \lim_{n\to\infty} a_n$.

Obviously, **little-o implies big-O** (take any $C > 0$ in Definition 1.5). Playing with limits we can get a list of useful facts.

Proposition 1.7. *Consider two strictly positive sequences $a_n, b_n \to 0$. The following properties hold:*

i. $kO(a_n) = O(a_n)$, $ko(a_n) = o(a_n)$, $k \in \mathbb{R}$.
ii. $o(a_n) + o(b_n) = o(a_n + b_n)$, $O(a_n) + O(b_n) = O(a_n + b_n)$.
iii. $o(a_n)o(b_n) = o(a_n b_n)$, $O(a_n)O(b_n) = O(a_n b_n)$.
iv. $o(a_n) + O(b_n) = O(a_n + b_n)$, $o(a_n)O(b_n) = o(a_n b_n)$.
v. $o(1)O(a_n) = o(a_n)$.
vi. $a_n^r = o(a_n^s)$, *for* $r > s \geq 0$.
vii. $a_n b_n = o(a_n^2 + b_n^2)$.
viii. $a_n b_n = o(a_n + b_n)$.
ix. $(a_n + b_n)^k = O(a_n^k + b_n^k)$.

The last result is a consequence of a useful inequality.

Lemma 1.1 (C_p inequality)**.** *Given $a, b \in \mathbb{R}$ and $p > 0$,*

$$|a + b|^p \leq C_p(|a|^p + |b|^p), \quad C_p = \begin{cases} 1, & p \leq 1, \\ 2^{p-1}, & p > 1. \end{cases}$$

Exercise 1.12. Illustrate the properties of Proposition 1.7 considering $a_n = n^{-1}$ and $b_n = n^{-1/2}$.

1.4.2 *Stochastic versions*

The previous notation is purely deterministic. Let's add some stochastic flavor by establishing the stochastic analogous of little-o.

Definition 1.7 (Little-$o_{\mathbb{P}}$)**.** Given a strictly positive sequence a_n and a sequence of random variables X_n,

$$\begin{aligned} X_n = o_{\mathbb{P}}(a_n) : &\iff \frac{|X_n|}{a_n} \xrightarrow{\mathbb{P}} 0 \\ &\iff \lim_{n\to\infty} \mathbb{P}\left[\frac{|X_n|}{a_n} > \varepsilon\right] = 0, \ \forall \varepsilon > 0. \end{aligned}$$

If $X_n = o_{\mathbb{P}}(a_n)$, then we say that X_n *is little-$o_{\mathbb{P}}$ of* a_n. To indicate that $X_n \xrightarrow{\mathbb{P}} 0$, we write $X_n = o_{\mathbb{P}}(1)$.

Therefore, little-$o_{\mathbb{P}}$ allows us to easily quantify the speed at which a sequence of random variables converges to zero in probability.

Example 1.3. Let $Y_n = o_{\mathbb{P}}\left(n^{-1/2}\right)$ and $Z_n = o_{\mathbb{P}}(n^{-1})$. Then Z_n converges faster to zero in probability than Y_n. To visualize this, recall that $X_n = o_{\mathbb{P}}(a_n)$ and that limit definitions entail that

$$\forall \varepsilon, \delta > 0, \ \exists\, n_0 = n_0(\varepsilon, \delta) \in \mathbb{N} : \forall n \geq n_0(\varepsilon, \delta), \ \mathbb{P}\left[|X_n| > a_n \varepsilon\right] < \delta.$$

Therefore, for fixed $\varepsilon, \delta > 0$ and a fixed $n \geq \max(n_{0,Y}, n_{0,Z})$, then $\mathbb{P}\left[Y_n \in \left(-n^{-1/2}\varepsilon, n^{-1/2}\varepsilon\right)\right] > 1 - \delta$ and $\mathbb{P}\left[Z_n \in (-n^{-1}\varepsilon, n^{-1}\varepsilon)\right] > 1 - \delta$, but the latter interval is much shorter, hence Z_n is forced to be more tightly concentrated about 0.

Big-$O_{\mathbb{P}}$ allows us to bound a sequence of random variables in probability, in the sense that we can state that the probability of being above an arbitrarily large threshold C converges to zero. As with its deterministic versions o and O, a **little-$o_{\mathbb{P}}$ is more restrictive than a big-$O_{\mathbb{P}}$**, and the former implies the latter.

Definition 1.8 (Big-$O_{\mathbb{P}}$)**.** Given a strictly positive sequence a_n and a sequence of random variables X_n,

$$\begin{aligned} X_n = O_{\mathbb{P}}(a_n) : \iff & \forall \varepsilon > 0, \exists C_\varepsilon > 0, n_0(\varepsilon) \in \mathbb{N} : \\ & \forall n \geq n_0(\varepsilon), \mathbb{P}\left[\frac{|X_n|}{a_n} > C_\varepsilon\right] < \varepsilon \\ \iff & \lim_{C \to \infty} \limsup_{n \to \infty} \mathbb{P}\left[\frac{|X_n|}{a_n} > C\right] = 0. \end{aligned}$$

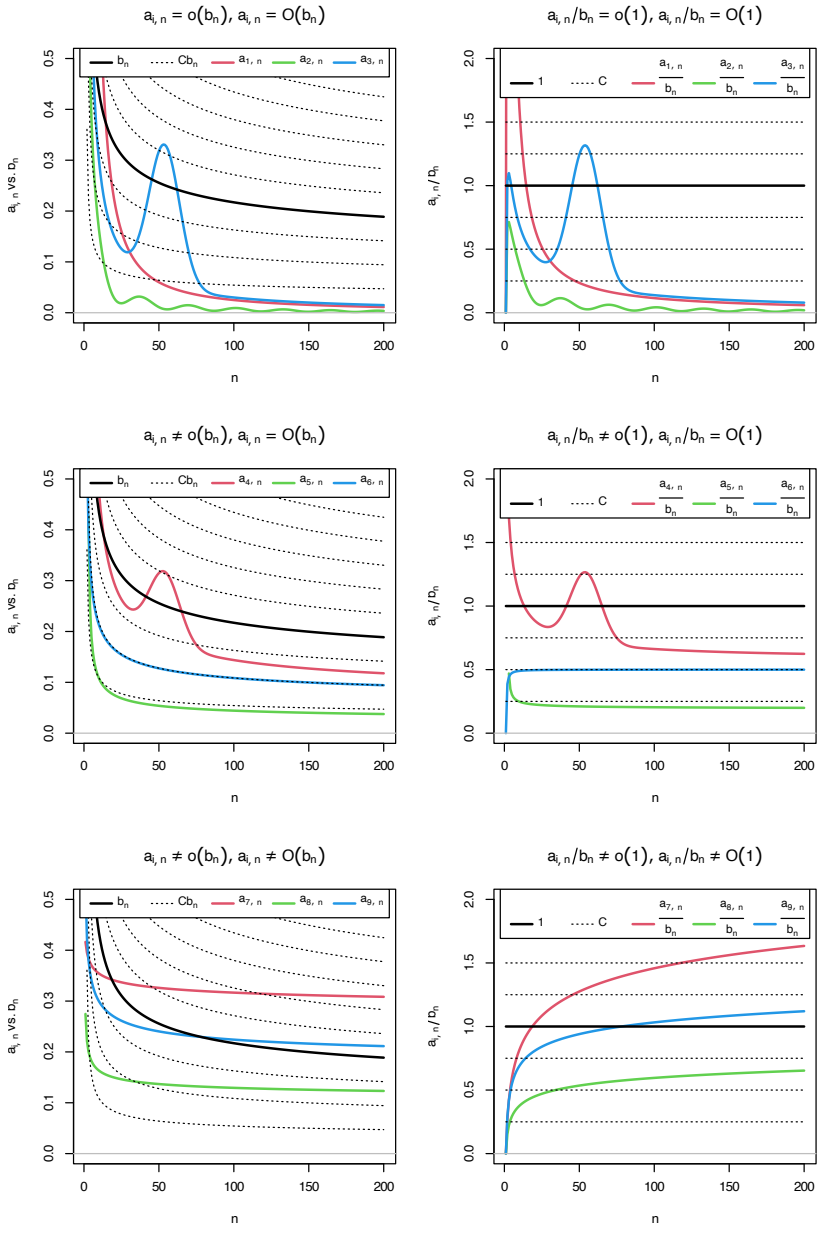


Figure 1.2: Differences and similarities between little-o and big-O, illustrated for the dominating sequence $b_n = 1/\log(n)$ (black solid curve) and sequences $a_{i,n}$, $j = 1, \ldots, 9$ (colored curves). The dashed lines represent the sequences Cb_n, for a grid of constants C. The plots on the left column compare $a_{i,n}$ against b_n, whereas the right column plots show the equivalent view in terms of the ratios $a_{i,n}/b_n$ (recall *iii* in Proposition 1.7). Sequences $a_{1,n} = 2/n + 50/n^2$, $a_{2,n} = (\sin(n/5) + 2)/n^{5/4}$, and $a_{3,n} = 3(1 + 5\exp(-(n - 55.5)^2/200))/n$ are $o(b_n)$ (hence also $O(b_n)$). Sequences $a_{4,n} = (2\log_{10}(n)((n+3)/(2n)))^{-1} + a_{3,n}/2$, $a_{5,n} = (4\log_2(n/2))^{-1}$, and $a_{6,n} = (\log(n^2 + n))^{-1}$ are $O(b_n)$, but not $o(b_n)$. Finally, sequences $a_{7,n} = \log(5n + 3)^{-1/4}/2$, $a_{8,n} = (4\log(\log(10n + 2)))^{-1}$, and $a_{9,n} = (2\log(\log(n^2 + 10n + 2)))^{-1}$ are not $O(b_n)$ (hence neither $o(b_n)$).

If $X_n = O_{\mathbb{P}}(a_n)$, then we say that X_n *is big-*$O_{\mathbb{P}}$ *of* a_n.

Example 1.4. Chebyshev inequality entails that $\mathbb{P}[|X_n - \mathbb{E}[X_n]| \geq t] \leq \mathbb{V}\text{ar}[X_n]/t^2$, $\forall t > 0$. Setting $\varepsilon := \mathbb{V}\text{ar}[X_n]/t^2$ and $C_\varepsilon := 1/\sqrt{\varepsilon}$, then $\mathbb{P}\left[|X_n - \mathbb{E}[X_n]| \geq \sqrt{\mathbb{V}\text{ar}[X_n]}C_\varepsilon\right] \leq \varepsilon$. Therefore,

$$X_n - \mathbb{E}[X_n] = O_{\mathbb{P}}\left(\sqrt{\mathbb{V}\text{ar}[X_n]}\right). \tag{1.7}$$

This is a very useful result, as it gives an efficient way of deriving the big-$O_{\mathbb{P}}$ form of a sequence of random variables X_n with finite variances.

An application of Example 1.4 shows that $X_n = O_{\mathbb{P}}(n^{-1/2})$ for $X_n \overset{d}{=} \mathcal{N}(0, 1/n)$. The nature of this statement and its relation with little-$o_{\mathbb{P}}$ is visualized with Figure 1.3, which shows a particular realization $X_n(\omega)$ of the sequence of random variables.

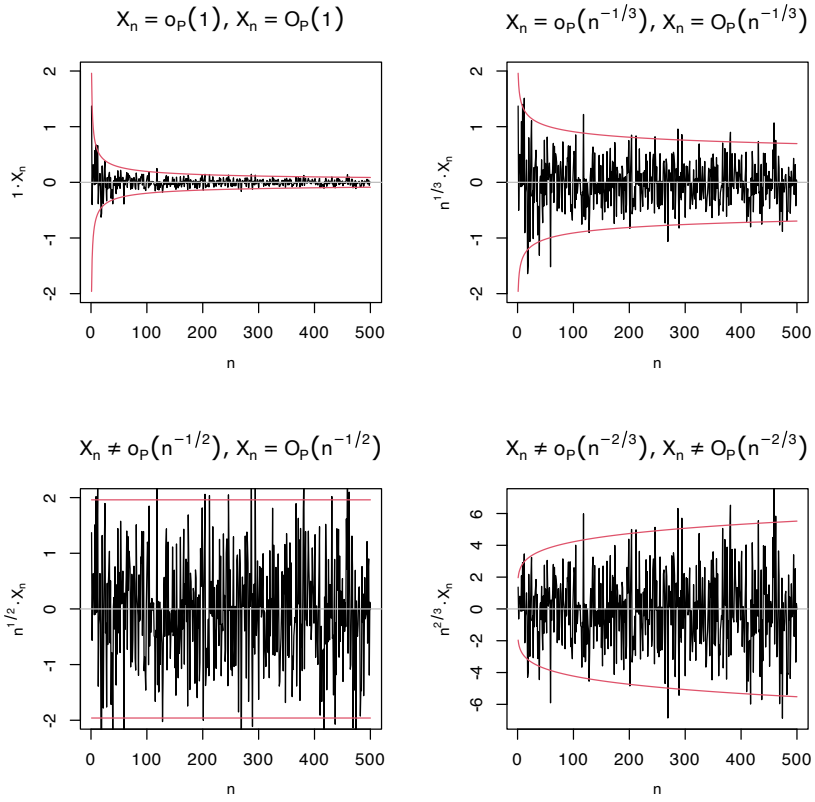


Figure 1.3: Differences and similarities between little-$o_{\mathbb{P}}$ and big-$O_{\mathbb{P}}$, illustrated for the sequence of random variables $X_n \overset{d}{=} \mathcal{N}(0, 1/n)$. Since $X_n \xrightarrow{\mathbb{P}} 0$, $X_n = o_{\mathbb{P}}(1)$, as evidenced in the upper left plot. The next plots check if $X_n = o_{\mathbb{P}}(a_n)$ by evaluating if $X_n/a_n \xrightarrow{\mathbb{P}} 0$, for $a_n = n^{-1/3}, n^{-1/2}, n^{-2/3}$. Clearly, $X_n = o_{\mathbb{P}}(n^{-1/3})$ ($n^{1/3}X_n \xrightarrow{\mathbb{P}} 0$) but $X_n \neq o_{\mathbb{P}}(n^{-1/2})$ ($n^{1/2}X_n \xrightarrow{\mathbb{P}} \mathcal{N}(0,1)$) and $X_n \neq o_{\mathbb{P}}(n^{-2/3})$ ($n^{2/3}X_n$ diverges). In the first three cases, $X_n = O_{\mathbb{P}}(a_n)$; the fourth is $X_n \neq O_{\mathbb{P}}(n^{-2/3})$, $n^{2/3}X_n$ is not bounded in probability. The red lines represent the 95% confidence intervals $\left(-z_{0.025}/(a_n\sqrt{n}), z_{0.025}/(a_n\sqrt{n})\right)$ of the random variable X_n/a_n, and help evaluating graphically the convergence in probability towards zero.

Exercise 1.13. As illustrated in Figure 1.3, prove that it is actually true that:

a. $X_n \xrightarrow{\mathbb{P}} 0$.
b. $n^{1/3}X_n \xrightarrow{\mathbb{P}} 0$.
c. $n^{1/2}X_n \xrightarrow{\mathbb{P}} \mathcal{N}(0,1)$.

Exercise 1.14. Prove that if $X_n \xrightarrow{d} X$, then $X_n = O_{\mathbb{P}}(1)$. *Hint*: use the double-limit definition and note that $X = O_{\mathbb{P}}(1)$.

Example 1.5 (Example 1.18 in DasGupta (2008))**.** Suppose that $a_n(X_n - c_n) \xrightarrow{d} X$ for deterministic sequences a_n and c_n such that $c_n \to c$. Then, if $a_n \to \infty$, $X_n - c = o_{\mathbb{P}}(1)$. The argument is simple:

$$\begin{aligned} X_n - c &= X_n - c_n + c_n - c \\ &= \frac{1}{a_n} a_n(X_n - c_n) + o(1) \\ &= \frac{1}{a_n} O_{\mathbb{P}}(1) + o(1). \end{aligned}$$

Exercise 1.15. Using the previous example, derive the weak law of large numbers as a consequence of the CLT, both for id and non-id independent random variables.

Proposition 1.8. *Consider two strictly positive sequences $a_n, b_n \to 0$. The following properties hold:*

i. $o_{\mathbb{P}}(a_n) = O_{\mathbb{P}}(a_n)$ *(little-$o_{\mathbb{P}}$ implies big-$O_{\mathbb{P}}$).*
ii. $o(1) = o_{\mathbb{P}}(1)$, $O(1) = O_{\mathbb{P}}(1)$ *(deterministic implies probabilistic).*
iii. $kO_{\mathbb{P}}(a_n) = O_{\mathbb{P}}(a_n)$, $ko_{\mathbb{P}}(a_n) = o_{\mathbb{P}}(a_n)$, $k \in \mathbb{R}$.
iv. $o_{\mathbb{P}}(a_n) + o_{\mathbb{P}}(b_n) = o_{\mathbb{P}}(a_n + b_n)$, $O_{\mathbb{P}}(a_n) + O_{\mathbb{P}}(b_n) = O_{\mathbb{P}}(a_n + b_n)$.
v. $o_{\mathbb{P}}(a_n)o_{\mathbb{P}}(b_n) = o_{\mathbb{P}}(a_n b_n)$, $O_{\mathbb{P}}(a_n)O_{\mathbb{P}}(b_n) = O_{\mathbb{P}}(a_n b_n)$.
vi. $o_{\mathbb{P}}(a_n) + O_{\mathbb{P}}(b_n) = O_{\mathbb{P}}(a_n + b_n)$, $o_{\mathbb{P}}(a_n)O_{\mathbb{P}}(b_n) = o_{\mathbb{P}}(a_n b_n)$.
vii. $o_{\mathbb{P}}(1)O_{\mathbb{P}}(a_n) = o_{\mathbb{P}}(a_n)$.
viii. $(1 + o_{\mathbb{P}}(1))^{-1} = O_{\mathbb{P}}(1)$.

Example 1.6. Example 1.4 allows us to obtain the $O_{\mathbb{P}}$-part of a sequence of random variables X_n with finite variances using (1.7). As a result of *ii* and *iv* in Proposition 1.8, we can also further simplify the coarse-grained description of X_n as

$$\begin{aligned} X_n &= O(\mathbb{E}[X_n]) + O_{\mathbb{P}}\left(\sqrt{\mathbb{V}\mathrm{ar}[X_n]}\right) \\ &= O_{\mathbb{P}}\left(\mathbb{E}[X_n] + \sqrt{\mathbb{V}\mathrm{ar}[X_n]}\right). \end{aligned}$$

Exercise 1.16. Consider the following sequences of random variables:

a. $X_n \overset{d}{=} \Gamma(2n, n+1) - 2$.
b. $X_n \overset{d}{=} \mathcal{LN}(0, n^{-1/3}) - 1$.
c. $X_n \overset{d}{=} \mathrm{B}(n, 1/n) - 1$.

For each X_n, obtain, if possible, two different positive sequences $a_n, b_n \to 0$ such that:

1. $X_n = o_{\mathbb{P}}(a_n)$.
2. $X_n \neq o_{\mathbb{P}}(b_n)$, $X_n = O_{\mathbb{P}}(b_n)$.

Use (1.7) and check your results by performing an analogous analysis to that in Figure 1.3.

1.5 *Review of basic analytical tools*

We will make use of the following well-known analytical results.

Theorem 1.9 (Mean value theorem)**.** *Let $f : [a, b] \longrightarrow \mathbb{R}$ be a continuous function and differentiable in (a, b). Then there exists $c \in (a, b)$ such that $f(b) - f(a) = f'(c)(b - a)$.*

Theorem 1.10 (Integral mean value theorem)**.** *Let $f : [a, b] \longrightarrow \mathbb{R}$ be a continuous function over (a, b). Then there exists $c \in (a, b)$ such that $\int_a^b f(x)\,\mathrm{d}x = f(c)(b - a)$.*

Theorem 1.11 (Taylor's theorem)**.** *Let $f : \mathbb{R} \longrightarrow \mathbb{R}$ and $x \in \mathbb{R}$. Assume that f has p continuous derivatives in an interval $(x - \delta, x + \delta)$ for a $\delta > 0$. Then, for any $|h| < \delta$,*

$$f(x + h) = \sum_{j=0}^{p} \frac{f^{(j)}(x)}{j!} h^j + R_n, \quad R_n = o(h^p).$$

Remark. The remainder R_n *depends on* $x \in \mathbb{R}$. Explicit control of R_n is possible if f is further assumed to be $(p + 1)$ differentiable in $(x - \delta, x + \delta)$. In that case, $R_n = \frac{f^{(p+1)}(\xi_x)}{(p+1)!} h^{p+1} = o(h^p)$ for a certain $\xi_x \in (x - \delta, x + \delta)$. Then, if $f^{(p+1)}$ is bounded in $(x - \delta, x + \delta)$, $\sup_{y \in (x-\delta, x+\delta)} \frac{R_n}{h^p} \to 0$, i.e., the remainder is $o(h^p)$ uniformly in $(x - \delta, x + \delta)$.

Theorem 1.12 (Dominated Convergence Theorem; DCT)**.** *Let $f_n : S \subset \mathbb{R} \longrightarrow \mathbb{R}$ be a sequence of Lebesgue measurable functions such that $\lim_{n\to\infty} f_n(x) = f(x)$ and $|f_n(x)| \leq g(x)$, $\forall x \in S$ and $\forall n \in \mathbb{N}$, where $\int_S |g(x)|\,\mathrm{d}x < \infty$. Then*

$$\lim_{n\to\infty} \int_S f_n(x)\,\mathrm{d}x = \int_S f(x)\,\mathrm{d}x < \infty.$$

Remark. Note that if S is bounded and $|f_n(x)| \leq M$, $\forall x \in S$ and $\forall n \in \mathbb{N}$, then limit interchangeability with integral is always possible.

1.6 *Why* Nonparametric Statistics*?*

The aim of statistical inference is to use data to *infer* an unknown quantity. In the game of inference, there is usually a trade-off between *efficiency* and *generality*, and this trade-off is controlled by the strength of assumptions that are made on the data generating process.

Parametric inference favors **efficiency**. Given a model (a strong assumption on the data generating process), parametric inference delivers a set of methods (point estimation, confidence intervals, hypothesis testing, etc) tailored for such model. All of these methods are the most efficient inferential procedures ***if*** the model matches the reality, in other words, ***if*** the data generating process truly satisfies the assumptions. Otherwise the methods may be inconsistent.

Nonparametric inference favors **generality**. Given a set of *minimal and weak* assumptions (e.g., certain smoothness of a density or existence of moments of a random variable), it provides inferential methods that are consistent for broad situations, in exchange for losing efficiency for small or moderate sample sizes. Broadly speaking, a statistical technique qualifies as "nonparametric" if it does not rely on parametric assumptions, these typically having a finite-dimensional nature.[9]

Hence, for any specific data generation process there is a parametric method that dominates its nonparametric counterpart in efficiency. But knowledge of the data generation process is rarely the case in practice. That is the appeal of a nonparametric method: it will **perform adequately no matter what the data generation process is**. For that reason, nonparametric methods are useful:

1. When we have no clue on what could be a good parametric model.
2. For creating goodness-of-fit tests employed to validate parametric models.

The following example aims to illustrate the first advantage, the most useful in practice.

Example 1.7. Assume we have a sample $X_1, \dots, X_n$ from a random variable X and we want to estimate its distribution function F. Without any assumption, we know that the ecdf in (1.1) is an estimate for $F(x) = \mathbb{P}[X \leq x]$. It is indeed a *nonparametric estimate* for F. Its expectation and variance are

$$\mathbb{E}[F_n(x)] = F(x), \quad \mathbb{V}\mathrm{ar}[F_n(x)] = \frac{F(x)(1-F(x))}{n}.$$

From the squared bias and variance, we can get the MSE:

$$\mathrm{MSE}[F_n(x)] = \frac{F(x)(1-F(x))}{n}.$$

Assume now that $X \sim \mathrm{Exp}(\lambda)$. By maximum likelihood, it is possible to estimate λ as $\hat{\lambda}_{\mathrm{ML}} = \bar{X}^{-1}$. Then, we have the following estimate for $F(x)$:

$$F(x; \hat{\lambda}_{\mathrm{ML}}) = 1 - e^{-\hat{\lambda}_{\mathrm{ML}} x}. \tag{1.8}$$

Obtaining the exact MSE for (1.8) is not so simple, even if it is easy to prove that $\hat{\lambda}_{\mathrm{ML}} \sim \mathrm{IG}(\lambda^{-1}, n)$. Approximations are possible using Exercise 1.2. However, the MSE can be easily approximated by Monte Carlo.

What happens when the data is generated from an $\mathrm{Exp}(\lambda)$? Then (1.8) uniformly dominates (1.1) in performance. But, even for small deviations from $\mathrm{Exp}(\lambda)$ given by $\Gamma(\lambda, p)$, $p \neq 1$, the parametric estimator (1.8) shows major problems in terms of bias, while the performance of the nonparametric estimator (1.1) is completely unaltered. The animation in Figure 1.4 illustrates precisely this behavior.

[9] For example, the exact two-sided t-test for the mean of a random variable X, i.e., the test $H_0 : \mu = \mu_0$ vs. $H_1 : \mu \neq \mu_0$, assumes that $X \sim \mathcal{N}(\mu, \sigma^2)$. This is an assumption indexed by the two parameters $(\mu, \sigma^2) \in \mathbb{R} \times \mathbb{R}^+$.

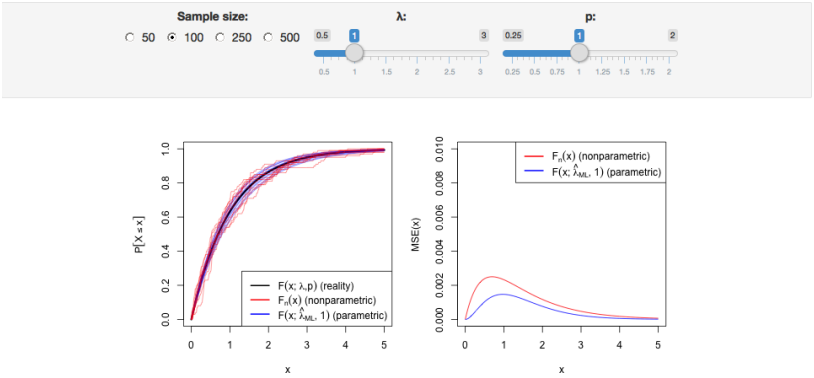


Figure 1.4: A simplified example of parametric and nonparametric estimation. The objective is to estimate the distribution function F of a random variable. The data is generated from a $\Gamma(\lambda, p)$. The parametric method assumes that $p = 1$, that is, that the data comes from a $\text{Exp}(\lambda)$. The nonparametric method does not assume anything on the data generation process. The left plot shows the true distribution function and ten estimates of each method from samples of size n. The right plot shows the MSE of each method on estimating $F(x)$. Application available here.

2
Kernel density estimation I

A random variable X is completely characterized by its cdf. Hence, an estimation of the cdf yields estimates for different characteristics of X as side-products by plugging, in these characteristics, the ecdf F_n instead of the F. For example[1], the mean $\mu = \mathbb{E}[X] = \int x \, \mathrm{d}F(x)$ can be estimated by $\int x \, \mathrm{d}F_n(x) = \frac{1}{n}\sum_{i=1}^n X_i = \bar{X}$.

Despite their usefulness and many advantages, cdfs are hard to visualize and interpret, a consequence of their cumulative-based definition.

Densities, on the other hand, are easy to visualize and interpret, making them *ideal tools for data exploration* of continuous random variables. They provide immediate graphical information about the highest-density regions, modes, and shape of the support of X. In addition, densities also completely characterize *continuous* random variables. Yet, even though a pdf follows from a cdf by the relation $f = F'$, density estimation does not follow immediately from the ecdf F_n, since this function is not differentiable[2]. Hence the need for specific procedures for estimating f that we will see in this chapter.

[1] Another example is the p-quantile $x_p := F^{(-1)}(p)$, where $F^{(-1)}(u) := \inf\{x \in \mathbb{R} : F(x) \geq u\}$ is the pseudo-inverse of the cdf F. Then, x_p can be estimated by the sample p-quantile $\hat{x}_p := F_n^{(-1)}(p)$.

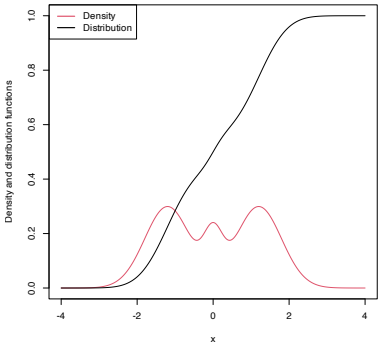


Figure 2.1: The pdf and cdf of a mixture of three normals. The pdf yields better insights into the structure of the continuous random variable X than the cdf does.

[2] Is not even continuous!

2.1 Histograms

2.1.1 Histogram

The simplest method to estimate a density f from an iid sample $X_1, \ldots, X_n$ is the *histogram*. From an analytical point of view, the idea is to aggregate the data in intervals of the form $[x_0, x_0 + h)$ and then use their relative frequency to approximate the density at $x \in [x_0, x_0 + h)$, $f(x)$, by the estimate of[3]

$$\begin{aligned} f(x_0) &= F'(x_0) \\ &= \lim_{h \to 0^+} \frac{F(x_0 + h) - F(x_0)}{h} \\ &= \lim_{h \to 0^+} \frac{\mathbb{P}[x_0 < X < x_0 + h]}{h}. \end{aligned}$$

More precisely, given an origin t_0 and a *bandwidth* $h > 0$, the histogram builds a piecewise constant function in the intervals $\{B_k := [t_k, t_{k+1}) : t_k = t_0 + hk, k \in \mathbb{Z}\}$ by counting the number of sample points inside each of them. These constant-length intervals are also called *bins*. The fact that they are of constant length h

[3] Note that we estimate $f(x)$ by means of an estimate for $f(x_0)$, where x is at most $h > 0$ units above x_0. Thus, we do *not* estimate directly $f(x)$ with the histogram.

is important: we can easily standardize the counts on any bin by h in order to have relative frequencies *per length*[4] in the bins. The histogram at a point x is defined as

$$\hat{f}_{\mathrm{H}}(x; t_0, h) := \frac{1}{nh} \sum_{i=1}^{n} 1_{\{X_i \in B_k : x \in B_k\}}. \tag{2.1}$$

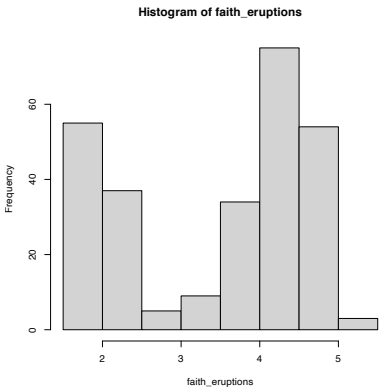
Equivalently, if we denote the number of observations $X_1, \ldots, X_n$ in B_k as v_k, then the histogram can be written as

$$\hat{f}_{\mathrm{H}}(x; t_0, h) = \frac{v_k}{nh}, \quad \text{if } x \in B_k \text{ for a certain } k \in \mathbb{Z}.$$

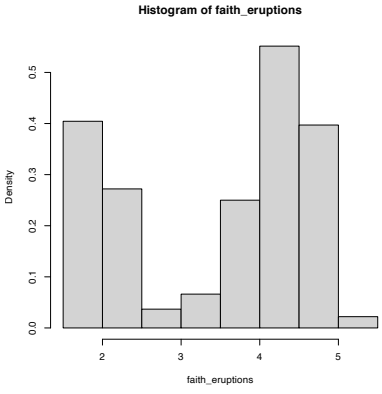
The computation of histograms is straightforward in R. As an example, we consider the old-but-gold `faithful` dataset. This dataset contains the duration of the eruption and the waiting time between eruptions for the Old Faithful geyser in Yellowstone National Park (USA).

[4] Recall that, with this standardization, we approach to the probability *density* concept.

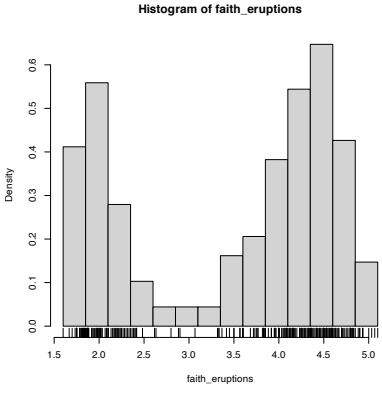
```
# The faithful dataset is included in R
head(faithful)
##   eruptions waiting
## 1     3.600      79
## 2     1.800      54
## 3     3.333      74
## 4     2.283      62
## 5     4.533      85
## 6     2.883      55

# Duration of eruption
faith_eruptions <- faithful$eruptions

# Default histogram: automatically chosen bins and absolute frequencies!
histo <- hist(faith_eruptions)
```




```
# List that contains several objects
str(histo)
## List of 6
##  $ breaks  : num [1:9] 1.5 2 2.5 3 3.5 4 4.5 5 5.5
##  $ counts  : int [1:8] 55 37 5 9 34 75 54 3
##  $ density : num [1:8] 0.4044 0.2721 0.0368 0.0662 0.25 ...
##  $ mids    : num [1:8] 1.75 2.25 2.75 3.25 3.75 4.25 4.75 5.25
##  $ xname   : chr "faith_eruptions"
##  $ equidist: logi TRUE
##  - attr(*, "class")= chr "histogram"

# With relative frequencies
hist(faith_eruptions, probability = TRUE)
```




```
# Choosing the breaks
# t0 = min(faithE), h = 0.25
Bk <- seq(min(faith_eruptions), max(faith_eruptions), by = 0.25)
hist(faith_eruptions, probability = TRUE, breaks = Bk)
rug(faith_eruptions) # Plotting the sample
```



Exercise 2.1. For `iris$Sepal.Length`, compute:

a. The histogram of relative frequencies with five bins.
b. The histogram of absolute frequencies with $t_0 = 4.3$ and $h = 1$.

Add the rug of the data for both histograms.

The analysis of $\hat{f}_{\mathrm{H}}(x; t_0, h)$ as a random variable is simple, once one recognizes that the bin counts v_k are distributed as $\mathrm{B}(n, p_k)$, with $p_k := \mathbb{P}[X \in B_k] = \int_{B_k} f(t)\,\mathrm{d}t$.[5] If f is continuous, then by the mean value theorem, $p_k = hf(\xi_{k,h})$ for a $\xi_{k,h} \in (t_k, t_{k+1})$. Assume that $k \in \mathbb{Z}$ is such that $x \in B_k = [t_k, t_{k+1})$.[6] Therefore:

$$\begin{aligned} \mathbb{E}[\hat{f}_{\mathrm{H}}(x; t_0, h)] &= \frac{np_k}{nh} = f(\xi_{k,h}), \\ \mathbb{V}\mathrm{ar}[\hat{f}_{\mathrm{H}}(x; t_0, h)] &= \frac{np_k(1-p_k)}{n^2h^2} = \frac{f(\xi_{k,h})(1 - hf(\xi_{k,h}))}{nh}. \end{aligned} \tag{2.2}$$

The results above yield interesting insights:

1. If $h \to 0$, then $\xi_{k,h} \to x$[7], resulting in $f(\xi_{k,h}) \to f(x)$, and thus (2.1) becomes an *asymptotically* (when $h \to 0$) unbiased estimator of $f(x)$.
2. But if $h \to 0$, the variance increases. For decreasing the variance, $nh \to \infty$ is required.
3. The variance is directly dependent on $f(\xi_{k,h})(1 - hf(\xi_{k,h})) \to f(x)$ (as $h \to 0$), hence there is more variability at regions with higher density.

A more detailed analysis of the histogram can be seen in Section 3.2.2 in Scott (2015). We skip it since the detailed asymptotic analysis for the more general kernel density estimator will be given in Section 2.2.

Exercise 2.2. Given (2.2), obtain $\mathrm{MSE}[\hat{f}_{\mathrm{H}}(x; t_0, h)]$. What should happen in order to have $\mathrm{MSE}[\hat{f}_{\mathrm{H}}(x; t_0, h)] \to 0$?

Clearly, the shape of the histogram depends on:

- t_0, since the separation between bins happens at $t_0 + kh$, $k \in \mathbb{Z}$;
- h, which controls the bin size and the effective number of bins for aggregating the sample.

We focus first on exploring the dependence on t_0, as it serves for motivating the next density estimator.

```
# Sample from a U(0, 1)
set.seed(1234567)
u <- runif(n = 100)

# Bins for t0 = 0, h = 0.2
Bk1 <- seq(0, 1, by = 0.2)

# Bins for t0 = -0.1, h = 0.2
Bk2 <- seq(-0.1, 1.1, by = 0.2)

# Comparison of histograms for different t0's
hist(u, probability = TRUE, breaks = Bk1, ylim = c(0, 1.5),
     main = "t0 = 0, h = 0.2")
rug(u)
abline(h = 1, col = 2)
hist(u, probability = TRUE, breaks = Bk2, ylim = c(0, 1.5),
     main = "t0 = -0.1, h = 0.2")
rug(u)
abline(h = 1, col = 2)
```

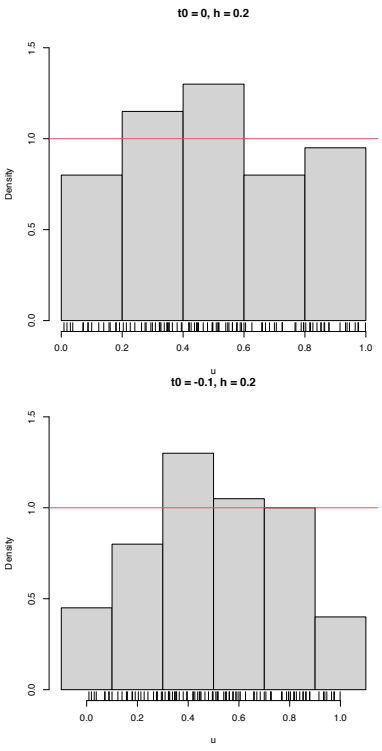


Figure 2.2: The dependence of the histogram on the origin t_0. The red curve represents the uniform pdf.

[5] Note that it is key that the $\{B_k\}$ are *deterministic* (and not sample-dependent) for this result to hold.

[6] This is an important point. Notice also that this k depends on h because $t_k = t_0 + kh$, therefore the k for which $x \in [t_k, t_{k+1})$ will change when, for example, $h \to 0$.

[7] Because $x \in [t_k, t_{k+1})$ with k changing as $h \to 0$ (see the previous footnote) and the interval ends up collapsing in x, so any point in $[t_k, t_{k+1})$ converges to x.

High dependence on t_0 also happens when estimating densities that are not compactly supported. The following snippet of code points towards it.

```
# Sample 50 points from a N(0, 1) and 25 from a N(3.25, 0.25)
set.seed(1234567)
samp <- c(rnorm(n = 50, mean = 0, sd = 1),
          rnorm(n = 25, mean = 3.25, sd = sqrt(0.5)))

# min and max for choosing Bk1 and Bk2
range(samp)
## [1] -2.082486  4.344547

# Comparison
Bk1 <- seq(-2.5, 5, by = 0.5)
Bk2 <- seq(-2.25, 5.25, by = 0.5)
hist(samp, probability = TRUE, breaks = Bk1, ylim = c(0, 0.5),
     main = "t0 = -2.5, h = 0.5")
curve(2/3 * dnorm(x, mean = 0, sd = 1) +
        1/3 * dnorm(x, mean = 3.25, sd = sqrt(0.5)), col = 2, add = TRUE,
      n = 200)
rug(samp)
hist(samp, probability = TRUE, breaks = Bk2, ylim = c(0, 0.5),
     main = "t0 = -2.25, h = 0.5")
curve(2/3 * dnorm(x, mean = 0, sd = 1) +
        1/3 * dnorm(x, mean = 3.25, sd = sqrt(0.5)), col = 2, add = TRUE,
      n = 200)
rug(samp)
```

Clearly, the subjectivity introduced by the dependence of t_0 is something that we would like to get rid of. We can do so by allowing the bins to be dependent on x (the point at which we want to estimate $f(x)$), rather than fixing them beforehand.

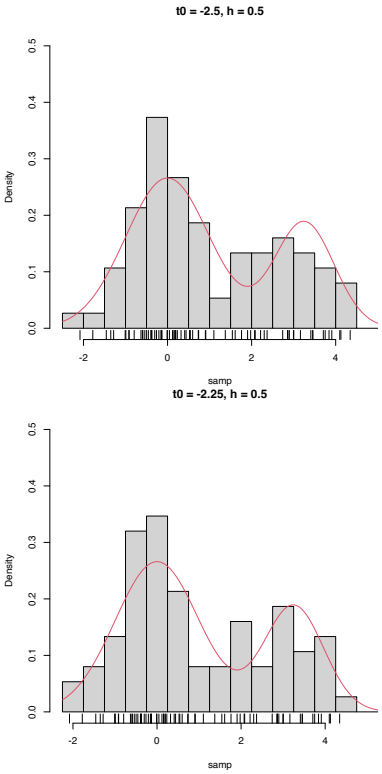


Figure 2.3: The dependence of the histogram on the origin t_0 for non-compactly supported densities. The red curve represents the underlying pdf, a mixture of two normals.

2.1.2 *Moving histogram*

An alternative to avoid the dependence on t_0 is the *moving histogram*, also known as the *naive density estimator*.[8] The idea is to aggregate the sample $X_1, \ldots, X_n$ in intervals of the form $(x-h, x+h)$ and then use its relative frequency in $(x-h, x+h)$ to approximate the density at x, which can be expressed as

$$\begin{aligned} f(x) &= F'(x) \\ &= \lim_{h\to 0^+} \frac{F(x+h)-F(x-h)}{2h} \\ &= \lim_{h\to 0^+} \frac{\mathbb{P}[x-h<X<x+h]}{2h}. \end{aligned}$$

Recall the differences with the histogram: the intervals depend on the evaluation point x and are centered about it. This allows us to directly estimate $f(x)$ (without the proxy $f(x_0)$) using an estimate based on the symmetric derivative of F at x, instead of employing an estimate based on the forward derivative of F at x_0.

More precisely, given a bandwidth $h > 0$, the naive density estimator builds a piecewise constant function by considering the relative frequency of $X_1, \ldots, X_n$ inside $(x-h, x+h)$:

$$\hat{f}_{\mathrm{N}}(x;h) := \frac{1}{2nh}\sum_{i=1}^{n} 1_{\{x-h<X_i<x+h\}}. \tag{2.3}$$

[8] The motivation for this terminology will be apparent in Section 2.2.

Figure 2.4 shows the moving histogram for the same sample used in Figure 2.3, clearly revealing the remarkable improvement with respect to the histograms shown when estimating the underlying density.

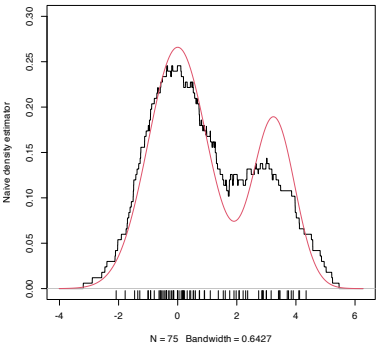


Figure 2.4: The naive density estimator $\hat{f}_{\mathrm{N}}(\cdot; h)$ (black curve). The red curve represents the underlying pdf, a mixture of two normals.

Exercise 2.3. Is $\hat{f}_{\mathrm{N}}(\cdot; h)$ continuous in general? Justify your answer. If the answer is negative, then:

a. What is the maximum number of discontinuities it may have?
b. What should happen to have fewer discontinuities than its maximum?

Exercise 2.4. Implement your own version of the moving histogram in R. It must be a function that takes as inputs:

1. a vector with the evaluation points x;
2. sample $X_1, \ldots, X_n$;
3. bandwidth h.

The function must return (2.3) evaluated for *each* x. Test the implementation by comparing the density of a $\mathcal{N}(0,1)$ when estimated with $n = 100$ observations.

Analogously to the histogram, the analysis of $\hat{f}_{\mathrm{N}}(x; h)$ as a random variable follows from realizing that

$$\sum_{i=1}^{n} 1_{\{x-h<X_i<x+h\}} \sim \mathrm{B}(n, p_{x,h}),$$
$$p_{x,h} := \mathbb{P}[x-h<X<x+h] = F(x+h) - F(x-h).$$

Then:

$$\mathbb{E}[\hat{f}_{\mathrm{N}}(x; h)] = \frac{F(x+h) - F(x-h)}{2h}, \tag{2.4}$$
$$\mathbb{V}\mathrm{ar}[\hat{f}_{\mathrm{N}}(x; h)] = \frac{F(x+h) - F(x-h)}{4nh^2} - \frac{(F(x+h) - F(x-h))^2}{4nh^2}. \tag{2.5}$$

Exercise 2.5. Derive expressions (2.4) and (2.5) from the binomial relation indicated above.

Results (2.4) and (2.5) provide interesting insights into the effect of h:

1. If $h \to 0$, then:
   - $\mathbb{E}[\hat{f}_{\mathrm{N}}(x; h)] \to f(x)$ and (2.3) is an asymptotically unbiased estimator of $f(x)$.
   - $\mathbb{V}\mathrm{ar}[\hat{f}_{\mathrm{N}}(x; h)] \approx \frac{f(x)}{2nh} - \frac{f(x)^2}{n} \to \infty$.
2. If $h \to \infty$, then:
   - $\mathbb{E}[\hat{f}_{\mathrm{N}}(x; h)] \to 0$.
   - $\mathbb{V}\mathrm{ar}[\hat{f}_{\mathrm{N}}(x; h)] \to 0$.

[9] Or, in other words, if $h^{-1} = o(n)$, i.e., if h^{-1} grows more slowly than n does.

3. The variance shrinks to zero if $nh \to \infty$.[9] So both the bias and the variance can be reduced if $n \to \infty$, $h \to 0$, and $nh \to \infty$, simultaneously.

4. The variance is *almost proportional*[10] to $f(x)$.

[10] Why so?

The animation in Figure 2.5 illustrates the previous points and gives insights into how the performance of (2.3) varies smoothly with h.

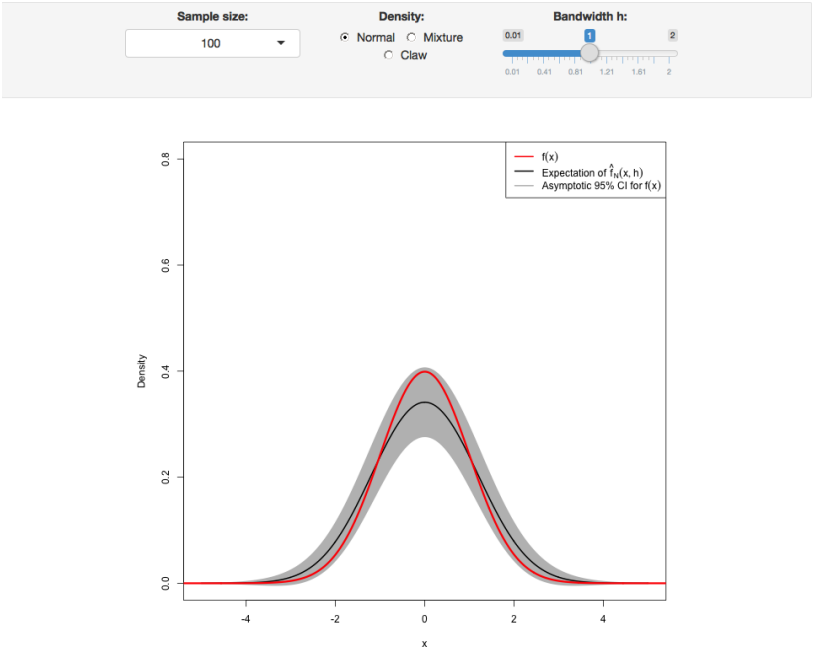


Figure 2.5: Bias and variance for the moving histogram. The animation shows how for small bandwidths the bias of $\hat{f}_N(x;h)$ on estimating $f(x)$ is small, but the variance is high, and how for large bandwidths the bias is large and the variance is small. The variance is visualized through the asymptotic 95% confidence intervals for $\hat{f}_N(x;h)$. Application available here.

The estimator (2.3) raises an important question:

> Why give the same weight to all $X_1, \ldots, X_n$ in $(x-h, x+h)$ for approximating $\frac{\mathbb{P}[x-h<X<x+h]}{2h}$?

We are estimating $f(x) = F'(x)$ by estimating $\frac{F(x+h)-F(x-h)}{2h}$ through the relative frequency of $X_1, \ldots, X_n$ in the interval $(x-h, x+h)$. Therefore, it seems reasonable that the **data points closer to x are more important** to assess the *infinitesimal probability* of x than the ones further away. This observation shows that (2.3) is indeed a particular case of a wider and more sensible class of density estimators, which we will see next.

2.2 *Kernel density estimation*

The moving histogram (2.3) can be equivalently written as

$$\begin{aligned}\hat{f}_{\mathrm{N}}(x;h) &= \frac{1}{2nh}\sum_{i=1}^{n} 1_{\{x-h<X_i<x+h\}} \\ &= \frac{1}{nh}\sum_{i=1}^{n} \frac{1}{2} 1_{\left\{-1<\frac{x-X_i}{h}<1\right\}} \\ &= \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-X_i}{h}\right), \end{aligned} \tag{2.6}$$

with $K(z) = \frac{1}{2}1_{\{-1<z<1\}}$. Interestingly, K is a uniform *density* in $(-1,1)$! This means that, when approximating

$$\mathbb{P}[x-h<X<x+h] = \mathbb{P}\left[-1<\frac{x-X}{h}<1\right]$$

by (2.6), we are giving *equal weight* to all the points $X_1,\ldots,X_n$. The generalization of (2.6) to non-uniform weighting is now obvious: replace K with an arbitrary density! Then K is known as a *kernel*. As it is commonly[11] assumed, we consider K to be a *density that is symmetric and unimodal at zero*. This generalization provides the definition of the **kernel density estimator** (kde)[12]:

$$\hat{f}(x;h) := \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-X_i}{h}\right). \tag{2.7}$$

A common notation is $K_h(z) := \frac{1}{h}K\left(\frac{z}{h}\right)$, the so-called *scaled kernel*, so that the kde is written as $\hat{f}(x;h) = \frac{1}{n}\sum_{i=1}^{n} K_h(x-X_i)$.

[11] In greater generality, the kernel K might only be assumed to be an integrable function with unit integral.

[12] Also known as the *Parzen–Rosemblatt estimator* to honor the proposals by Parzen (1962) and Rosenblatt (1956).

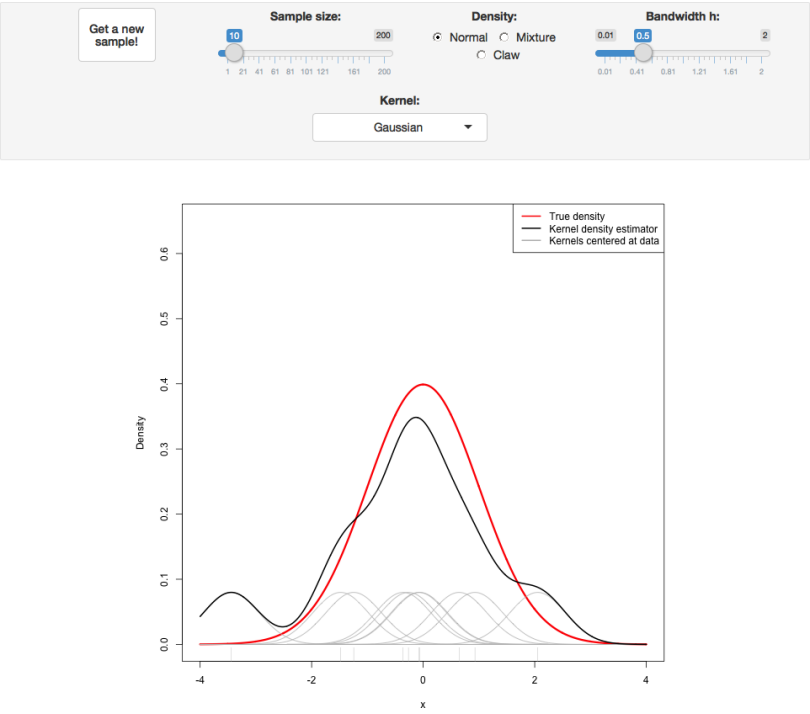


Figure 2.6: Construction of the kernel density estimator. The animation shows how bandwidth and kernel affect the density estimate, and how the kernels are rescaled densities with modes at the data points. Application available here.

It is useful to recall (2.7) with the normal kernel. If that is the case, then $K_h(x - X_i) = \phi_h(x - X_i)$ and the kernel is the density of a $\mathcal{N}(X_i, h^2)$. Thus the bandwidth h can be thought of as the *standard deviation* of a normal density with mean X_i, and the kde (2.7) as a data-driven mixture of those densities. Figure 2.6 illustrates the construction of the kde and the bandwidth and kernel effects.

Several types of kernels are possible. Figure 2.7 illustrates the form of several implemented in R's `density` function. The most popular is the *normal kernel* $K(z) = \phi(z)$, although the *Epanechnikov kernel*, $K(z) = \frac{3}{4}(1 - z^2)1_{\{|z|<1\}}$, is the most efficient.[13] The *rectangular kernel* $K(z) = \frac{1}{2}1_{\{|z|<1\}}$ yields the moving histogram as a particular case. Importantly, the kde inherits the smoothness properties of the kernel. That means, for example, that (2.7) with a normal kernel is infinitely differentiable. But with an Epanechnikov kernel, (2.7) is not differentiable, and with a rectangular kernel is not even continuous. However, if a certain smoothness is guaranteed (continuity at least), the *choice of the kernel has little importance in practice*, at least in comparison with the much critical choice of the bandwidth h.

The computation of the kde in R is done through the `density` function. The function automatically chooses the bandwidth h using a data-driven criterion[14] referred to as a *bandwidth selector*. Bandwidth selectors will be studied in detail in Section 2.4.

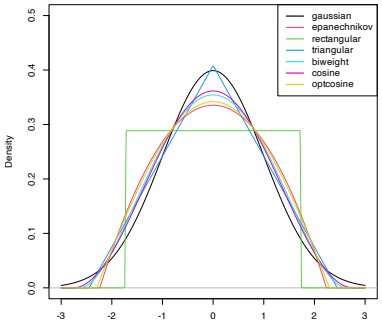
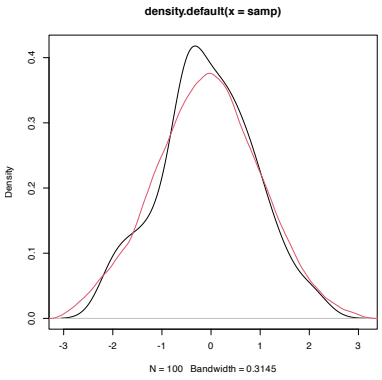


Figure 2.7: Several scaled kernels from R's `density` function. See (2.9) in the remark below for the different parametrization.

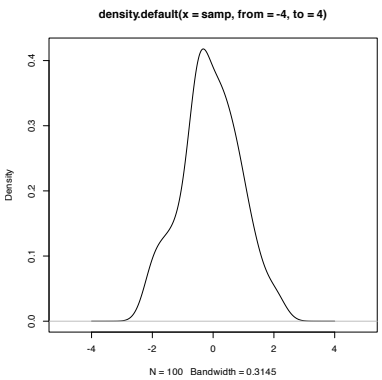
[13] Although the efficiency of the normal kernel, with respect to the Epanechnikov kernel, is roughly 0.95.

[14] Precisely, the *rule-of-thumb* given by `bw.nrd`.

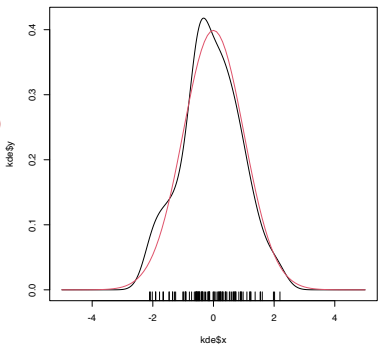
```
# Sample 100 points from a N(0, 1)
set.seed(1234567)
samp <- rnorm(n = 100, mean = 0, sd = 1)

# Quickly compute a kde and plot the density object
# Automatically chooses bandwidth and uses normal kernel
plot(density(x = samp))

# Select a particular bandwidth (0.5) and kernel (Epanechnikov)
lines(density(x = samp, bw = 0.5, kernel = "epanechnikov"), col = 2)
```




```
# density() automatically chooses the interval for plotting the kde
# (observe that the black line goes to roughly between -3 and 3)
# This can be tuned using "from" and "to"
plot(density(x = samp, from = -4, to = 4), xlim = c(-5, 5))
```




```
# The density object is a list
kde <- density(x = samp, from = -5, to = 5, n = 1024)
str(kde)
## List of 7
##  $ x        : num [1:1024] -5 -4.99 -4.98 -4.97 -4.96 ...
##  $ y        : num [1:1024] 5.98e-17 3.46e-17 2.33e-17 3.40e-17 4.29e-17 ...
##  $ bw       : num 0.315
##  $ n        : int 100
##  $ call     : language density.default(x = samp, n = 1024, from = -5, to = 5)
##  $ data.name: chr "samp"
##  $ has.na   : logi FALSE
##  - attr(*, "class")= chr "density"
# Note that the evaluation grid "x" is not directly controlled, only through
# "from, "to", and "n" (better use powers of 2)
plot(kde$x, kde$y, type = "l")
curve(dnorm(x), col = 2, add = TRUE) # True density
rug(samp)
```



Exercise 2.6. Employing the normal kernel:

1. Estimate and plot the density of `faithful$eruptions`.
2. Create a new plot and superimpose different density estimations with bandwidths equal to 0.1, 0.5, and 1.
3. Get the density estimate at *exactly* the point $x = 3.1$ using $h = 0.15$.

We next introduce an important remark about the use of the `density` function. Before that, we need some notation. From now on, we consider the integrals to be over $\mathbb{R}$ if not stated otherwise. In addition, we denote

$$\mu_2(K) := \int z^2 K(z)\,\mathrm{d}z$$

to the *second-order moment*[15] of K.

[15] The variance, since $\int zK(z)\,\mathrm{d}z = 0$ because the kernel is symmetric with respect to zero.

Remark. The kernel, say $\tilde{K}$, employed in `density` uses a parametrization that guarantees that $\mu_2(\tilde{K}) = 1$. This implies that the variance of the *scaled* kernel is h^2, that is, that $\mu_2(\tilde{K}_h) = h^2$. These *normalized kernels* may be different from the ones we have considered in the exposition. For example, the uniform kernel $K(z) = \frac{1}{2}1_{\{-1<z<1\}}$, for which $\mu_2(K) = \frac{1}{3}$, is implemented in R as $\tilde{K}(z) = \frac{1}{2\sqrt{3}}1_{\{-\sqrt{3}<z<\sqrt{3}\}}$ (observe in Figure 2.7 that the rectangular kernel takes the value $\frac{1}{2\sqrt{3}} \approx 0.29$).

The `density`'s normalized kernel $\tilde{K}$ can be obtained from K, and vice versa, with a straightforward derivation. On the one hand, since $\mu_2(K) = \int z^2 K(z)$,

$$\begin{aligned} 1 &= \int \frac{z^2}{\mu_2(K)} K(z)\,\mathrm{d}z \\ &= \int t^2 K\left(\mu_2(K)^{1/2} t\right) \mu_2(K)^{1/2}\,\mathrm{d}t \end{aligned} \tag{2.8}$$

by making the change of variables $t = \frac{z}{\mu_2(K)^{1/2}}$ in the second equality. Now, based on (2.8), define

$$\tilde{K}(z) := \mu_2(K)^{1/2} K\left(\mu_2(K)^{1/2} z\right) \tag{2.9}$$

and this is a kernel that satisfies $\mu_2(\tilde{K}) = \int t^2 \tilde{K}(t)\,\mathrm{d}t = 1$ and is a density unimodal about 0. It is indeed the kernel employed by `density`.

Therefore, if we consider a bandwidth h for the normalized kernel $\tilde{K}$ (resulting $\tilde{K}_h$), we have that

$$\tilde{K}_h(z) = \frac{\mu_2(K)^{1/2}}{h} K\left(\frac{\mu_2(K)^{1/2} z}{h}\right) = K_{\tilde{h}}(z)$$

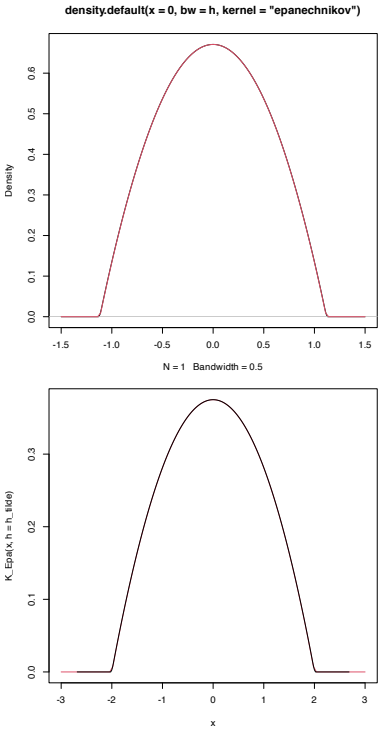
where

$$\tilde{h} := \mu_2(K)^{-1/2} h. \tag{2.10}$$

As a consequence, a normalized kernel $\tilde{K}$ with a given bandwidth h (for example, the one considered in `density`'s `bw`) has an associated scaled bandwidth $\tilde{h}$ for the unnormalized kernel K.

The following code numerically illustrates the difference between $\tilde{K}$ and K for the Epanechnikov kernel.

```
# Implementation of the Epanechnikov based on the theory
K_Epa <- function(z, h = 1) 3 / (4 * h) * (1 - (z / h)^2) * (abs(z) < h)
mu2_K_Epa <- integrate(function(z) z^2 * K_Epa(z), lower = -1, upper = 1)$value

# Epanechnikov kernel by R
h <- 0.5
plot(density(0, kernel = "epanechnikov", bw = h))

# Build the equivalent bandwidth
h_tilde <- h / sqrt(mu2_K_Epa)
curve(K_Epa(x, h = h_tilde), add = TRUE, col = 2)
```




```
# The other way around
h_tilde <- 2
h <- h_tilde * sqrt(mu2_K_Epa)
curve(K_Epa(x, h = h_tilde), from = -3, to = 3, col = 2)
lines(density(0, kernel = "epanechnikov", bw = h))
```

Exercise 2.7. Obtain the kernel $\tilde{K}$ for:

a. the normal kernel $K(z) = \phi(z)$;
b. the Epanechnikov kernel $K(z) = \frac{3}{4}(1-z^2)1_{\{|z|<1\}}$;
c. the kernel $K(z) = (1-|z|)1_{\{|z|<1\}}$.

Exercise 2.8. Given h, obtain $\tilde{h}$ such that $\tilde{K}_h = K_{\tilde{h}}$ for:

a. the uniform kernel $K(z) = \frac{1}{2}1_{\{|z|<1\}}$;
b. the kernel $K(z) = \frac{1}{2\pi}(1+\cos(z))1_{\{|z|<\pi\}}$;
c. the kernel $K(z) = (1-|z|)1_{\{|z|<1\}}$.

Exercise 2.9. Repeat Exercise 2.6, but now considering the Epanechnikov kernel in its unnormalized form. To do so, find the adequate bandwidths to input in `density`'s `bw`.

2.3 *Asymptotic properties*

Asymptotic results serve the purpose of establishing the large-sample ($n \to \infty$) properties of an estimator. One might question why they are useful, since in practice we only have *finite* sample sizes. Apart from purely theoretical reasons, asymptotic results usually give *highly valuable insights* into the properties of the estimator, typically much simpler to grasp than those obtained from finite-sample results.[16]

Throughout this section we will make the following assumptions:

- **A1**.[17] The density f is square integrable, twice continuously differentiable, and the second derivative is square integrable.
- **A2**.[18] The kernel K is a symmetric and bounded pdf with finite second moment and square integrable.
- **A3**.[19] $h = h_n$ is a deterministic sequence of positive scalars[20] such that, when $n \to \infty$, $h \to 0$ and $nh \to \infty$.

[16] Finite-sample results might be either analytically unfeasible or dependent on simulation studies that are necessarily limited in their scope.

[17] This assumption requires certain smoothness of the pdf, allowing thus for Taylor expansions to be performed on f.

[18] Mild assumption that makes the first term of the Taylor expansion of f negligible and the second one bounded.

[19] The key assumption for reducing the bias and variance of $\hat{f}(\cdot; h)$ *simultaneously*.

[20] $h = h_n$ *always* depends on n from now on, although the subscript is dropped for the ease of notation.

We need to introduce some notation. The squared integral of a function f is denoted by $R(f) := \int f(x)^2 \, \mathrm{d}x$. The *convolution* between two real functions f and g, $f * g$, is the function

$$(f * g)(x) := \int f(x-y)g(y) \, \mathrm{d}y = (g * f)(x). \tag{2.11}$$

We are now ready to obtain the bias and variance of $\hat{f}(x;h)$. Recall that is not possible to apply the "binomial trick" we used previously for the histogram and moving histogram: now the estimator is not piecewise constant. Instead, we use the linearity of the kde and the convolution definition. For the expectation, we have

$$\begin{aligned} \mathbb{E}[\hat{f}(x;h)] &= \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}[K_h(x - X_i)] \\ &= \int K_h(x-y)f(y) \, \mathrm{d}y \\ &= (K_h * f)(x). \end{aligned} \tag{2.12}$$

Similarly, the variance is obtained as

$$\mathbb{V}\mathrm{ar}[\hat{f}(x;h)] = \frac{1}{n}((K_h^2 * f)(x) - (K_h * f)^2(x)). \tag{2.13}$$

[21] Since $\mathrm{Bias}[\hat{f}(x;h)] = (K_h * f)(x) - f(x) \neq 0$. An example of this bias in given in Section A.

These two expressions are exact, but hard to interpret. Equation (2.12) indicates that the estimator is *biased*[21], but it does not explicitly differentiate the effects of kernel, bandwidth, and density on the bias. The same happens with (2.13), yet more emphasized. Clarity is why the following *asymptotic* expressions are preferred.

Theorem 2.1. *Under **A1–A3**, the bias and variance of the kde at x are*

$$\mathrm{Bias}[\hat{f}(x;h)] = \frac{1}{2}\mu_2(K)f''(x)h^2 + o(h^2), \tag{2.14}$$

$$\mathbb{V}\mathrm{ar}[\hat{f}(x;h)] = \frac{R(K)}{nh}f(x) + o((nh)^{-1}). \tag{2.15}$$

Proof. For the *bias* we consider the change of variables $z = \frac{x-y}{h}$, $y = x - hz$, $\mathrm{d}y = -h \, \mathrm{d}z$. The integral limits flip and we have

$$\begin{aligned} \mathbb{E}[\hat{f}(x;h)] &= \int K_h(x-y)f(y) \, \mathrm{d}y \\ &= \int K(z)f(x-hz) \, \mathrm{d}z. \end{aligned} \tag{2.16}$$

Since $h \to 0$, an application of a second-order Taylor expansion gives

$$\begin{aligned} f(x-hz) = f(x) - f'(x)hz + \frac{f''(x)}{2}h^2z^2 \\ + o(h^2z^2). \end{aligned} \tag{2.17}$$

Substituting (2.17) in (2.16), and bearing in mind that K is a sym-

metric density about 0, we have

$$\begin{aligned}\int K(z)f(x-hz)\,\mathrm{d}z &= \int K(z)\Big\{f(x)-f'(x)hz+\frac{f''(x)}{2}h^2z^2 \\ &\quad +o(h^2z^2)\Big\}\,\mathrm{d}z \\ &= f(x)+\frac{1}{2}\mu_2(K)f''(x)h^2+o(h^2),\end{aligned}$$

which provides (2.14).

For the *variance*, first note that

$$\begin{aligned}\mathbb{V}\mathrm{ar}[\hat{f}(x;h)] &= \frac{1}{n^2}\sum_{i=1}^{n}\mathbb{V}\mathrm{ar}[K_h(x-X_i)] \\ &= \frac{1}{n}\left\{\mathbb{E}[K_h^2(x-X)]-\mathbb{E}[K_h(x-X)]^2\right\}. \qquad (2.18)\end{aligned}$$

The second term of (2.18) is already computed, so we focus on the first. Using the previous change of variables and a first-order Taylor expansion, we have

$$\begin{aligned}\mathbb{E}[K_h^2(x-X)] &= \frac{1}{h}\int K^2(z)f(x-hz)\,\mathrm{d}z \\ &= \frac{1}{h}\int K^2(z)\left\{f(x)+O(hz)\right\}\,\mathrm{d}z \\ &= \frac{R(K)}{h}f(x)+O(1). \qquad (2.19)\end{aligned}$$

Plugging (2.18) into (2.19) gives

$$\begin{aligned}\mathbb{V}\mathrm{ar}[\hat{f}(x;h)] &= \frac{1}{n}\left\{\frac{R(K)}{h}f(x)+O(1)-O(1)\right\} \\ &= \frac{R(K)}{nh}f(x)+O(n^{-1}) \\ &= \frac{R(K)}{nh}f(x)+o((nh)^{-1}),\end{aligned}$$

since $n^{-1}=o((nh)^{-1})$. □

Remark. Integrating little-o's is a tricky issue. In general, integrating a $o^x(1)$ quantity, possibly dependent on x (this is emphasized with the superscript), does not provide an $o(1)$. In other words: $\int o^x(1)\,\mathrm{d}x \neq o(1)$. If the previous inequality becomes an equality, then the limits and integral will be interchangeable. But this is not always true – only if certain conditions are met, recall the DCT (Theorem 1.12). If one wants to be completely rigorous on the two implicit commutations of integrals and limits that took place in the proof, it is necessary to have explicit control of the remainder via Taylor's theorem (Theorem 1.11) and then apply the DCT. This has been skipped for simplicity in the exposition.

The bias and variance expressions (2.14) and (2.15) yield interesting insights (see Figure 2.5 for visualizations thereof):

- The bias decreases with h *quadratically*. In addition, the bias at x is directly proportional to $f''(x)$. This has an interesting interpretation:

  - The bias is negative where f is concave, i.e., $\{x \in \mathbb{R} : f''(x) < 0\}$. These regions correspond to *peaks and modes of* f, where the kde *underestimates* f (it tends to be below f).
  - Conversely, the bias is positive where f is convex, i.e., $\{x \in \mathbb{R} : f''(x) > 0\}$. These regions correspond to *valleys and tails of* f, where the kde *overestimates* f (it tends to be above f).
  - The wilder the curvature f'', the harder to estimate f. Flat density regions are easier to estimate than wiggling regions with high curvature (e.g., with several modes).

- The variance depends directly on $f(x)$. The higher the density, the more variable is the kde. Interestingly, the variance decreases as a factor of $(nh)^{-1}$, a consequence of nh playing the role of the *effective sample size* for estimating $f(x)$. The effective sample size can be thought of as the amount of data[22] in the neighborhood of x that is employed for estimating $f(x)$.

[22] The variance of an unweighted mean is reduced by a factor n^{-1} when n observations are employed. For computing $\hat{f}(x;h)$, n observations are used but in a *weighted* fashion that roughly amounts to considering nh *unweighted* observations.

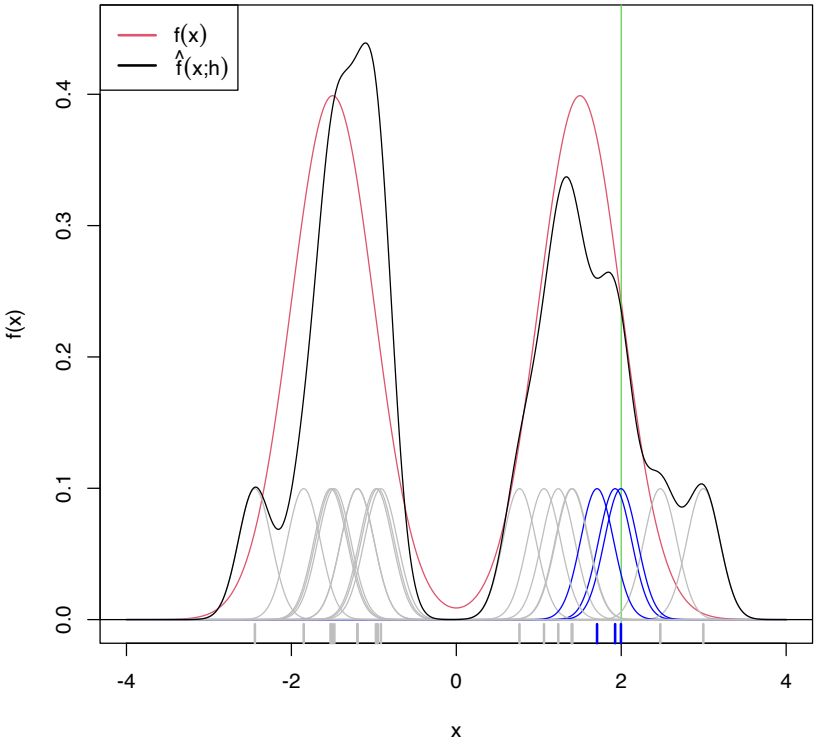


Figure 2.8: Illustration of the effective sample size for estimating $f(x)$ at $x = 2$. In blue, the kernels with contribution to the kde larger than 0.01. In gray, rest of the kernels. Only 3 observations are effectively employed for estimating $f(2)$ by $\hat{f}(2;0.2)$, despite the sample size being $n = 20$.

Exercise 2.10. Using Example 1.4 and Theorem 2.1, show that:

a. $\hat{f}(x;h) = f(x) + O(h^2) + O_{\mathbb{P}}\left((nh)^{-1/2}\right)$.
b. $\hat{f}(x;h) = f(x)(1 + o_{\mathbb{P}}(1))$.

The MSE of the kde is trivial to obtain from the bias and variance.

Corollary 2.1. *Under* ***A1–A3****, the MSE of the kde at* x *is*

$$\mathrm{MSE}[\hat{f}(x;h)] = \frac{\mu_2^2(K)}{4}(f''(x))^2 h^4 + \frac{R(K)}{nh} f(x) + o(h^4 + (nh)^{-1}). \tag{2.20}$$

Therefore, the kde is pointwise consistent in MSE, i.e., $\hat{f}(x;h) \xrightarrow{2} f(x)$.

Exercise 2.11. Prove Corollary 2.1 using Theorem 2.1 and Proposition 1.7.

Note that, due to the MSE-consistency of $\hat{f}(x;h)$,

$$\hat{f}(x;h) \xrightarrow{2} f(x) \implies \hat{f}(x;h) \xrightarrow{\mathbb{P}} f(x) \implies \hat{f}(x;h) \xrightarrow{d} f(x)$$

under **A1**–**A2**. However, these results are *not useful for quantifying the randomness* of $\hat{f}(x;h)$: the convergence is towards the degenerate random variable $f(x)$, for a given $x \in \mathbb{R}$.[23] For that reason, we now turn our attention to the asymptotic normality of the estimator.

[23] That is, towards the random variable that always takes the value $f(x)$.

Theorem 2.2. *Assume that* $\int K^{2+\delta}(z)\,\mathrm{d}z < \infty$ [24] *for some* $\delta > 0$. *Then, under* ***A1***–***A3***,

$$\sqrt{nh}\left(\hat{f}(x;h) - \mathbb{E}[\hat{f}(x;h)]\right) \xrightarrow{d} \mathcal{N}(0, R(K)f(x)). \tag{2.21}$$

[24] This is satisfied, for example, if the kernel decreases exponentially, i.e., if $\exists\, \alpha,\ M > 0 : K(z) \leq e^{-\alpha|z|},\ \forall |z| > M$.

Additionally, if $nh^5 = O(1)$, *then*

$$\sqrt{nh}\left(\hat{f}(x;h) - f(x) - \frac{1}{2}\mu_2(K)f''(x)h^2\right) \xrightarrow{d} \mathcal{N}(0, R(K)f(x)). \tag{2.22}$$

Proof. First note that $K_h(x - X_n)$ is a sequence of independent but *not* identically distributed random variables: $h = h_n$ depends on n. Therefore, we look forward to applying Theorem 1.3.

We first prove (2.21). For simplicity, denote $K_i := K_h(x - X_i)$, $i = 1, \ldots, n$. From the proof of Theorem 2.1 we know that $\mathbb{E}[K_i] = \mathbb{E}[\hat{f}(x;h)] = f(x) + o(1)$ and

$$\begin{aligned} s_n^2 &= \sum_{i=1}^{n} \mathbb{V}\mathrm{ar}[K_i] \\ &= n^2 \mathbb{V}\mathrm{ar}[\hat{f}(x;h)] \\ &= n\frac{R(K)}{h} f(x)(1 + o(1)). \end{aligned}$$

An application of the C_p inequality of Lemma 1.1 (first) and Jensen's inequality (second), gives

$$\begin{aligned} \mathbb{E}\left[|K_i - \mathbb{E}[K_i]|^{2+\delta}\right] &\leq C_{2+\delta}\left(\mathbb{E}\left[|K_i|^{2+\delta}\right] + |\mathbb{E}[K_i]|^{2+\delta}\right) \\ &\leq 2C_{2+\delta}\mathbb{E}\left[|K_i|^{2+\delta}\right] \\ &= O\left(\mathbb{E}\left[|K_i|^{2+\delta}\right]\right). \end{aligned}$$

In addition, due to a Taylor expansion after $z = \frac{x-y}{h}$ and using that $\int K^{2+\delta}(z)\,\mathrm{d}z < \infty$,

$$\begin{aligned} \mathbb{E}\left[|K_i|^{2+\delta}\right] &= \frac{1}{h^{2+\delta}} \int K^{2+\delta}\left(\frac{x-y}{h}\right) f(y)\,\mathrm{d}y \\ &= \frac{1}{h^{1+\delta}} \int K^{2+\delta}(z) f(x - hz)\,\mathrm{d}z \\ &= \frac{1}{h^{1+\delta}} \int K^{2+\delta}(z)(f(x) + o(1))\,\mathrm{d}z \\ &= O\left(h^{-(1+\delta)}\right). \end{aligned}$$

Then,

$$\begin{aligned}
&\frac{1}{s_n^{2+\delta}} \sum_{i=1}^{n} \mathbb{E}\left[|K_i - \mathbb{E}[K_i]|^{2+\delta}\right] \\
&\quad = \left(\frac{h}{nR(K)f(x)}\right)^{1+\frac{\delta}{2}} (1+o(1))O\left(nh^{-(1+\delta)}\right) \\
&\quad = O\left((nh)^{-\frac{\delta}{2}}\right)
\end{aligned}$$

and the Lyapunov's condition is satisfied. As a consequence, by Lyapunov's CLT and Slutsky's theorem,

$$\begin{aligned}
&\sqrt{\frac{nh}{R(K)f(x)}}(\hat{f}(x;h) - \mathbb{E}[\hat{f}(x;h)]) \\
&\qquad = (1+o(1))\frac{1}{s_n}\sum_{i=1}^{n}(K_i - \mathbb{E}[K_i]) \\
&\qquad \xrightarrow{d} \mathcal{N}(0,1)
\end{aligned}$$

and (2.21) is proved.

To prove (2.22), we consider

$$\begin{aligned}
&\sqrt{nh}\left(\hat{f}(x;h) - f(x) - \frac{1}{2}\mu_2(K)f''(x)h^2\right) \\
&\qquad = \sqrt{nh}(\hat{f}(x;h) - \mathbb{E}[\hat{f}(x;h)] + o(h^2)).
\end{aligned}$$

Therefore, Slutsky's theorem and the assumption $nh^5 = O(1)$ give

$$\begin{aligned}
&\sqrt{nh}(\hat{f}(x;h) - \mathbb{E}[\hat{f}(x;h)] + o(h^2)) \\
&\quad = \sqrt{nh}\left(\hat{f}(x;h) - \mathbb{E}[\hat{f}(x;h)] + o(\sqrt{nh^5})\right) \\
&\quad = \sqrt{nh}\left(\hat{f}(x;h) - \mathbb{E}[\hat{f}(x;h)]\right) + o(1). \qquad \xrightarrow{d} \mathcal{N}(0, R(K)f(x)).
\end{aligned}$$

□

Remark. Note the rate $\sqrt{nh}$ in the asymptotic normality results. This is different from the standard CLT rate $\sqrt{n}$ (see Theorem 1.2). Indeed, $\sqrt{nh}$ is *slower* than $\sqrt{n}$: the variance of the limiting normal distribution decreases as $O((nh)^{-1})$ and not as $O(n^{-1})$. The phenomenon is related to the effective sample size previously illustrated with Figure 2.8.

Exercise 2.12. Using (2.22) and Example 1.5, show that $\hat{f}(x;h) = f(x) + o_{\mathbb{P}}(1)$.

2.4 *Bandwidth selection*

As we saw in the previous sections, the kde critically depends on the bandwidth employed. The purpose of this section is to introduce objective and automatic bandwidth selectors that attempt to minimize the estimation error of the target density f.

The first step is to define a global, rather than local, error criterion. The *Integrated Squared Error* (ISE),

$$\mathrm{ISE}[\hat{f}(\cdot;h)] := \int (\hat{f}(x;h) - f(x))^2 \,\mathrm{d}x,$$

is the squared distance between the kde and the target density. The ISE is a random quantity, since it depends directly on the sample $X_1, \dots, X_n$. As a consequence, looking for an optimal-ISE bandwidth is a hard task, since the optimality is dependent on the sample itself and not only on the population and n. To avoid this problem, it is usual to compute the *Mean Integrated Squared Error* (MISE):

$$\begin{aligned}\operatorname{MISE}[\hat{f}(\cdot;h)] &:= \mathbb{E}\left[\operatorname{ISE}[\hat{f}(\cdot;h)]\right] \\ &= \mathbb{E}\left[\int (\hat{f}(x;h) - f(x))^2 \,\mathrm{d}x\right] \\ &= \int \mathbb{E}\left[(\hat{f}(x;h) - f(x))^2\right] \,\mathrm{d}x \\ &= \int \operatorname{MSE}[\hat{f}(x;h)] \,\mathrm{d}x.\end{aligned}$$

The MISE is convenient due to its mathematical tractability and its natural relation to the MSE. There are, however, other error criteria that present attractive properties, such as the *Mean Integrated Absolute Error* (MIAE):

$$\begin{aligned}\operatorname{MIAE}[\hat{f}(\cdot;h)] &:= \mathbb{E}\left[\int |\hat{f}(x;h) - f(x)| \,\mathrm{d}x\right] \\ &= \int \mathbb{E}\left[|\hat{f}(x;h) - f(x)|\right] \,\mathrm{d}x.\end{aligned}$$

The MIAE, unlike the MISE, has the appeal of being *invariant* with respect to monotone transformations of the density. For example, if $g(x) = f(t^{-1}(x))(t^{-1})'(x)$ is the density of $Y = t(X)$ and $X \sim f$, then the change of variables[25] $y = t(x)$ gives

$$\begin{aligned}\int |\hat{f}(x;h) - f(x)| \,\mathrm{d}x &= \int |\hat{f}(t^{-1}(y);h) - f(t^{-1}(y))|(t^{-1})'(y) \,\mathrm{d}y \\ &= \int |\hat{g}(y;h) - g(y)| \,\mathrm{d}y.\end{aligned}$$

[25] For defining $\hat{g}(y;h)$ we consider the transformed kde seen in Section 2.5.1.

Despite this attractive invariance property, the analysis of MIAE is substantially more complicated than the MISE. We refer to Devroye and Györfi (1985) for a comprehensive treatment of absolute value metrics for kde.

Once the MISE is set as the error criterion to be minimized, our aim is to find

$$h_{\operatorname{MISE}} := \arg\min_{h>0} \operatorname{MISE}[\hat{f}(\cdot;h)]. \tag{2.23}$$

For that purpose, we need an explicit expression of the MISE which we can attempt to minimize. The following asymptotic expansion for the MISE solves this issue.

Corollary 2.2. *Under **A1–A3**,*

$$\begin{aligned}\operatorname{MISE}[\hat{f}(\cdot;h)] = \frac{1}{4}\mu_2^2(K)R(f'')h^4 + \frac{R(K)}{nh} \\ + o(h^4 + (nh)^{-1}).\end{aligned} \tag{2.24}$$

Therefore, $\operatorname{MISE}[\hat{f}(\cdot;h)] \to 0$ *when* $n \to \infty$.

Exercise 2.13. Prove Corollary 2.2 by integrating the MSE of $\hat{f}(x;h)$.

The dominating part of the MISE is denoted by AMISE, which stands for *Asymptotic MISE*:

$$\text{AMISE}[\hat{f}(\cdot;h)] = \frac{1}{4}\mu_2^2(K)R(f'')h^4 + \frac{R(K)}{nh}.$$

The closed-form expression of the AMISE allows obtaining a bandwidth that minimizes this error.

Corollary 2.3. *The bandwidth that minimizes the AMISE is*

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

The optimal AMISE is:

$$\inf_{h>0} \text{AMISE}[\hat{f}(\cdot;h)] = \frac{5}{4}(\mu_2^2(K)R(K)^4)^{1/5}R(f'')^{1/5}n^{-4/5}. \tag{2.25}$$

Exercise 2.14. Prove Corollary 2.3 by solving $\frac{\mathrm{d}}{\mathrm{d}h}\text{AMISE}[\hat{f}(\cdot;h)] = 0$.

The AMISE-optimal order deserves some further inspection. It can be seen in Section 3.2 in Scott (2015) that the AMISE-optimal order for the *histogram* of Section 2.1.1 (not the moving histogram) is $(3/4)^{2/3}\,R(f')^{1/3}n^{-2/3}$. Two aspects are interesting when comparing this result with (2.25):

- First, the MISE of the histogram is asymptotically larger than the MISE of the kde.[26] This is a quantification of the quite apparent visual improvement of the kde over the histogram.
- Second, $R(f')$ appears instead of $R(f'')$, evidencing that the histogram is affected by *how fast* f *varies* and not only by the *curvature* of the target density f.

[26] Since $n^{-4/5} = o(n^{-2/3})$.

Unfortunately, the AMISE bandwidth depends on $R(f'') = \int (f''(x))^2\,\mathrm{d}x$, which measures the *curvature* of the density. As a consequence, it can not be readily applied in practice, as $R(f'')$ is unknown! In the next subsection we will see how to plug-in estimates for $R(f'')$.

2.4.1 *Plug-in rules*

A simple solution to estimate $R(f'')$ is to assume that f is the density of a $\mathcal{N}(\mu,\sigma^2)$ and then plug-in the form of the curvature for such density,[27]

$$R(\phi_\sigma''(\cdot-\mu)) = \frac{3}{8\pi^{1/2}\sigma^5}.$$

[27] We only use a parametric assumption for estimating the curvature of f in present h_{AMISE}, not for directly estimating f itself.

While doing so, we approximate the curvature of an arbitrary density by means of the curvature of a normal and we have that

$$h_{\text{AMISE}} = \left[\frac{8\pi^{1/2}R(K)}{3\mu_2^2(K)n}\right]^{1/5}\sigma.$$

Interestingly, the bandwidth is directly proportional to the standard deviation of the target density. Replacing σ by an estimate yields the **normal scale bandwidth selector**, which we denote $\hat{h}_{\text{NS}}$ to emphasize its randomness:

$$\hat{h}_{\text{NS}} = \left[\frac{8\pi^{1/2}R(K)}{3\mu_2^2(K)n}\right]^{1/5}\hat{\sigma}.$$

The estimate $\hat{\sigma}$ can be chosen as the standard deviation s, or, in order to avoid the effects of potential outliers, as the standardized interquantile range

$$\hat{\sigma}_{\text{IQR}} := \frac{X_{([0.75n])} - X_{([0.25n])}}{\Phi^{-1}(0.75) - \Phi^{-1}(0.25)}$$

or as

$$\hat{\sigma} = \min(s, \hat{\sigma}_{\text{IQR}}). \tag{2.26}$$

When combined with a normal kernel, for which $\mu_2(K) = 1$ and $R(K) = \frac{1}{2\sqrt{\pi}}$, this particularization of $\hat{h}_{\text{NS}}$ featuring (2.26) gives the famous **rule-of-thumb** for bandwidth selection:

$$\hat{h}_{\text{RT}} = \left(\frac{4}{3}\right)^{1/5} n^{-1/5}\hat{\sigma} \approx 1.06n^{-1/5}\hat{\sigma}.$$

[28] Not to confuse with `bw.nrd0`!

$\hat{h}_{\text{RT}}$ is implemented in R through the function `bw.nrd`[28].

```
# Data
set.seed(667478)
n <- 100
x <- rnorm(n)

# Rule-of-thumb
bw.nrd(x = x)
## [1] 0.4040319
# bwd.nrd employs 1.34 as an approximation for diff(qnorm(c(0.25, 0.75)))

# Same as
iqr <- diff(quantile(x, c(0.25, 0.75))) / diff(qnorm(c(0.25, 0.75)))
1.06 * n^(-1/5) * min(sd(x), iqr)
## [1] 0.4040319
```

The previous selector is an example of a **zero-stage plug-in** selector, a terminology inspired by the fact that the scalar $R(f'')$ was estimated by plugging a parametric assumption directly, without attempting a nonparametric estimation first. Another possibility could have been to estimate $R(f'')$ nonparametrically and then to plug-in the estimate into h_{AMISE}. Let's explore this possibility in more detail next.

First, note the useful equality

$$\int f^{(s)}(x)^2\,\mathrm{d}x = (-1)^s \int f^{(2s)}(x)f(x)\,\mathrm{d}x.$$

This equality follows by an iterative application of integration by parts. For example, for $s = 2$, take $u = f''(x)$ and $\mathrm{d}v = f''(x)\,\mathrm{d}x$. It

gives

$$\begin{aligned}\int f''(x)^2\,\mathrm{d}x &= [f''(x)f'(x)]_{-\infty}^{+\infty} - \int f'(x)f'''(x)\,\mathrm{d}x \\ &= -\int f'(x)f'''(x)\,\mathrm{d}x\end{aligned}$$

under the assumption that the derivatives vanish at infinity. Applying again integration by parts with $u = f'''(x)$ and $\mathrm{d}v = f'(x)\,\mathrm{d}x$ gives the result. This simple derivation has an important consequence: for estimating the functionals $R(f^{(s)})$ it suffices to estimate the functionals

$$\psi_r := \int f^{(r)}(x)f(x)\,\mathrm{d}x = \mathbb{E}[f^{(r)}(X)] \tag{2.27}$$

for $r = 2s$. In particular, $R(f'') = \psi_4$.

Thanks to the expression (2.27), a possible way to estimate ψ_r nonparametrically is

$$\begin{aligned}\hat{\psi}_r(g) &= \frac{1}{n}\sum_{i=1}^{n}\hat{f}^{(r)}(X_i;g) \\ &= \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}L_g^{(r)}(X_i - X_j),\end{aligned} \tag{2.28}$$

where $\hat{f}^{(r)}(\cdot;g)$ is the r-th derivative of a kde with bandwidth g and kernel L, i.e.,

$$\hat{f}^{(r)}(x;g) = \frac{1}{ng^{r+1}}\sum_{i=1}^{n}L^{(r)}\left(\frac{x - X_i}{g}\right).$$

Note that g and L can be different from h and K, respectively. It turns out that estimating ψ_r involves the adequate selection of a bandwidth g. The agenda is analogous to the one for h_{AMISE}, but now taking into account that both $\hat{\psi}_r(g)$ and ψ_r are *scalar* quantities:

1. Under certain regularity assumptions[29], the asymptotic bias and variance of $\hat{\psi}_r(g)$ are obtained. With them, we can compute the asymptotic expansion of the MSE[30] and obtain the *Asymptotic Mean Squared Error* AMSE:

$$\begin{aligned}\mathrm{AMSE}[\hat{\psi}_r(g)] = &\left\{\frac{L^{(r)}(0)}{ng^{r+1}} + \frac{\mu_2(L)\psi_{r+2}g^2}{4}\right\} + \frac{2R(L^{(r)})\psi_0}{n^2g^{2r+1}} \\ &+ \frac{4}{n}\left\{\int f^{(r)}(x)^2 f(x)\,\mathrm{d}x - \psi_r^2\right\}.\end{aligned}$$

   Note: k is the highest integer such that $\mu_k(L) > 0$. In these notes we have restricted the exposition to the case $k = 2$ for the kernels K, but there are theoretical gains if one allows *high-order kernels* L with vanishing even moments larger than 2 for estimating ψ_r.

2. Obtain the AMSE-optimal bandwidth:

$$g_{\mathrm{AMSE}} = \left[-\frac{k!L^{(r)}(0)}{\mu_k(L)\psi_{r+k}n}\right]^{1/(r+k+1)}$$

[29] See Section 3.5 in Wand and Jones (1995) for full details.

[30] Recall there is no "I", since we are estimating a scalar, not a function.

The order of the optimal AMSE is

$$\inf_{g>0} \text{AMSE}[\hat{\psi}_r(g)] = \begin{cases} O(n^{-(2k+1)/(r+k+1)}), & k < r, \\ O(n^{-1}), & k \geq r, \end{cases}$$

which shows that a parametric-like rate of convergence can be achieved with high-order kernels. If we consider $L = K$ and $k = 2$, then

$$g_{\text{AMSE}} = \left[-\frac{2K^{(r)}(0)}{\mu_2(L)\psi_{r+2}n} \right]^{1/(r+3)}.$$

The result above has a major problem: it depends on ψ_{r+2}! Thus, if we want to estimate $R(f'') = \psi_4$ by $\hat{\psi}_4(g_{\text{AMSE}})$ we will need to estimate ψ_6. But $\hat{\psi}_6(g_{\text{AMSE}})$ will depend on ψ_8, and so on! The solution to this convoluted problem is to stop estimating the functional ψ_r after a given number ℓ of *stages*, hence the terminology **ℓ-stage plug-in selector**. At the ℓ-th stage, the functional $\psi_{2\ell+4}$ inside the AMSE-optimal bandwidth for estimating $\psi_{2\ell+2}$ nonparametrically[31] is computed assuming that the density is a $\mathcal{N}(\mu, \sigma^2)$ [32], for which

$$\psi_r = \frac{(-1)^{r/2} r!}{(2\sigma)^{r+1}(r/2)!\sqrt{\pi}}, \quad \text{for } r \text{ even}.$$

Typically, **two stages** are considered a good trade-off between bias (mitigated when ℓ increases) and variance (increases with ℓ) of the plug-in selector. This is the method proposed by Sheather and Jones (1991), where they consider $L = K$ and $k = 2$, yielding what we call the **Direct Plug-In** (DPI). The algorithm is:

1. Estimate ψ_8 using $\hat{\psi}_8^{\text{NS}} := \frac{105}{32\sqrt{\pi}\hat{\sigma}^9}$, where $\hat{\sigma}$ is given in (2.26).
2. Estimate ψ_6 using $\hat{\psi}_6(g_1)$ from (2.28), where
$$g_1 := \left[-\frac{2K^{(6)}(0)}{\mu_2(K)\hat{\psi}_8^{\text{NS}} n} \right]^{1/9}.$$
3. Estimate ψ_4 using $\hat{\psi}_4(g_2)$ from (2.28), where
$$g_2 := \left[-\frac{2K^{(4)}(0)}{\mu_2(K)\hat{\psi}_6(g_1) n} \right]^{1/7}.$$
4. The selected bandwidth is
$$\hat{h}_{\text{DPI}} := \left[\frac{R(K)}{\mu_2^2(K)\hat{\psi}_4(g_2) n} \right]^{1/5}.$$

Remark. The derivatives $K^{(r)}$ for the normal kernel can be obtained using the (probabilists') *Hermite polynomials*: $\phi^{(r)}(x) = \phi(x)H_r(x)$. For $r = 0, \ldots, 6$, these are: $H_0(x) = 1$, $H_1(x) = x$, $H_2(x) = x^2 - 1$, $H_3(x) = x^3 - 3x$, $H_4(x) = x^4 - 6x^2 + 3$, $H_5(x) = x^5 - 10x^3 + 15x$, and $H_6(x) = x^6 - 15x^4 + 45x^2 - 15$. Hermite polynomials satisfy the recurrence relation $H_{\ell+1}(x) = xH_\ell(x) - \ell H_{\ell-1}(x)$ for $\ell \geq 1$.

[31] For the rule-of-thumb selector, $\ell = 0$ and we estimate ψ_4 parametrically. If $\ell = 2$, we estimate ψ_8 parametrically, which is then required for estimating ψ_6 nonparametrically.

[32] Hence, what we are doing is "sweeping under ℓ nonparametric carpets" this parametric estimate on the characteristic $\psi_{\ell+4}$ of the density, ultimately required for computing the bandwidth selector.

$\hat{h}_{\text{DPI}}$ is implemented in R through the function `bw.SJ` (use `method = "dpi"`). An alternative and faster implementation is `ks::hpi`, which also accounts for more flexibility and has a somehow more complete documentation.

```
# Data
set.seed(672641)
x <- rnorm(100)

# DPI selector
bw.SJ(x = x, method = "dpi")
## [1] 0.5006905

# Similar to
ks::hpi(x) # Default is two-stage
## [1] 0.4999456
```

Exercise 2.15. Apply and inspect the kde of `airquality$Ozone` using the DPI selector (both `bw.SJ` and `ks::hpi`). What can you conclude from the estimate?

2.4.2 *Cross-validation*

We now turn our attention to a different philosophy of bandwidth estimation. Instead of trying to minimize the AMISE by plugging estimates for the unknown curvature term, we directly attempt to minimize the MISE. The idea is to use the sample twice: one for computing the kde and other for *evaluating* its performance on estimating f. To avoid the clear dependence on the sample, we do this evaluation in a *cross-validatory* way: the data used for computing the kde is *not* used for its evaluation.

We begin by expanding the square in the MISE expression:

$$\begin{aligned}\text{MISE}[\hat{f}(\cdot;h)] &= \mathbb{E}\left[\int (\hat{f}(x;h)-f(x))^2\,\mathrm{d}x\right]\\ &= \mathbb{E}\left[\int \hat{f}(x;h)^2\,\mathrm{d}x\right] - 2\mathbb{E}\left[\int \hat{f}(x;h)f(x)\,\mathrm{d}x\right]\\ &\quad + \int f(x)^2\,\mathrm{d}x.\end{aligned}$$

Since the last term does not depend on h, minimizing $\text{MISE}[\hat{f}(\cdot;h)]$ is equivalent to minimizing

$$\mathbb{E}\left[\int \hat{f}(x;h)^2\,\mathrm{d}x\right] - 2\mathbb{E}\left[\int \hat{f}(x;h)f(x)\,\mathrm{d}x\right]. \tag{2.29}$$

This quantity is unknown, but it can be estimated unbiasedly by

$$\text{LSCV}(h) := \int \hat{f}(x;h)^2\,\mathrm{d}x - 2n^{-1}\sum_{i=1}^{n}\hat{f}_{-i}(X_i;h), \tag{2.30}$$

where $\hat{f}_{-i}(\cdot;h)$ is the *leave-one-out* kde and is based on the sample with the X_i removed:

$$\hat{f}_{-i}(x;h) = \frac{1}{n-1}\sum_{\substack{j=1\\ j\neq i}}^{n} K_h(x-X_j).$$

Exercise 2.16. Prove that (2.30) is indeed an unbiased estimator of (2.29).

Exercise 2.17 (Exercise 3.3 in Wand and Jones (1995))**.** Prove that $\mathbb{E}[\text{LSCV}(h)] = \text{MISE}[\hat{f}(\cdot;h)] - R(f)$.

The motivation for (2.30) is the following. The first term is unbiased by design.[33] The second arises from approximating $\int \hat{f}(x;h)f(x)\,\mathrm{d}x$ by Monte Carlo from the sample $X_1,\ldots,X_n \sim f$; in other words, by replacing $f(x)\,\mathrm{d}x = \mathrm{d}F(x)$ with $\mathrm{d}F_n(x)$. This gives

$$\int \hat{f}(x;h)f(x)\,\mathrm{d}x \approx \frac{1}{n}\sum_{i=1}^{n}\hat{f}(X_i;h)$$

and, in order to mitigate the dependence of the sample, we replace $\hat{f}(X_i;h)$ with $\hat{f}_{-i}(X_i;h)$ above. In that way, we use the sample for estimating the integral involving $\hat{f}(\cdot;h)$, but for each X_i we compute the kde on the *remaining* points.

[33] Why so? Because X is unbiased for estimating $\mathbb{E}[X]$.

The **Least Squares Cross-Validation** (LSCV) selector, also denoted **Unbiased Cross-Validation** (UCV) selector, is defined as

$$\hat{h}_{\text{LSCV}} := \arg\min_{h>0} \text{LSCV}(h).$$

Numerical optimization is required for obtaining $\hat{h}_{\text{LSCV}}$, contrary to the previous plug-in selectors, and there is little control on the shape of the objective function. This will also be the case for the subsequent bandwidth selectors. The following remark warns about the dangers of numerical optimization in this context.

Remark. Numerical optimization of the LSCV function can be challenging. In practice, several local minima are possible, and the roughness of the objective function can vary notably depending on n and f. As a consequence, optimization routines may get trapped in spurious solutions. To be on the safe side, it is always advisable to check (when possible) the solution by plotting $\text{LSCV}(h)$ for a range of h, or to perform a search in a bandwidth grid: $\hat{h}_{\text{LSCV}} \approx \arg\min_{h_1,\ldots,h_G} \text{LSCV}(h)$.

$\hat{h}_{\text{LSCV}}$ is implemented in R through the function `bw.ucv`. This function uses R's `optimize`, which is quite sensitive to the selection of the search interval.[34] Therefore, some care is needed; that is why the `bw.ucv.mod` function is presented below.

[34] Long intervals containing the solution may lead to unsatisfactory termination of the search; short intervals might not contain the minimum.

```
# Data
set.seed(123456)
x <- rnorm(100)

# UCV gives a warning
bw.ucv(x = x)
## [1] 0.4499177

# Extend search interval
bw.ucv(x = x, lower = 0.01, upper = 1)
## [1] 0.5482419

# bw.ucv.mod replaces the optimization routine of bw.ucv with an exhaustive
# search on "h_grid" (chosen adaptatively from the sample) and optionally
```

```
# plots the LSCV curve with "plot_cv"
bw.ucv.mod <- function(x, nb = 1000L,
                       h_grid = 10^seq(-3, log10(1.2 * sd(x) *
                                                 length(x)^(-1/5)), l = 200),
                       plot_cv = FALSE) {
  if ((n <- length(x)) < 2L)
    stop("need at least 2 data points")
  n <- as.integer(n)
  if (is.na(n))
    stop("invalid length(x)")
  if (!is.numeric(x))
    stop("invalid 'x'")
  nb <- as.integer(nb)
  if (is.na(nb) || nb <= 0L)
    stop("invalid 'nb'")
  storage.mode(x) <- "double"
  hmax <- 1.144 * sqrt(var(x)) * n^(-1/5)
  Z <- .Call(stats:::C_bw_den, nb, x)
  d <- Z[[1L]]
  cnt <- Z[[2L]]
  fucv <- function(h) .Call(stats:::C_bw_ucv, n, d, cnt, h)
  ## Original
  # h <- optimize(fucv, c(lower, upper), tol = tol)$minimum
  # if (h < lower + tol | h > upper - tol)
  #   warning("minimum occurred at one end of the range")
  ## Modification
  obj <- sapply(h_grid, function(h) fucv(h))
  h <- h_grid[which.min(obj)]
  if (h %in% range(h_grid))
    warning("minimum occurred at one end of h_grid")
  if (plot_cv) {
    plot(h_grid, obj, type = "o")
    rug(h_grid)
    abline(v = h, col = 2, lwd = 2)
  }
  h
}

# Compute the bandwidth and plot the LSCV curve
bw.ucv.mod(x = x, plot_cv = TRUE, h_grid = 10^seq(-1.25, 0.5, l = 200))
## [1] 0.5431561

# We can compare with the default bw.ucv output
abline(v = bw.ucv(x = x), col = 3)
```

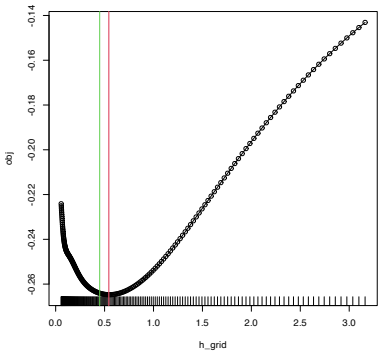


Figure 2.9: LSCV curve evaluated for a grid of bandwidths, with a clear global minimum corresponding to $\hat{h}_{\text{LSCV}}$.

The next cross-validation selector is based on **Biased Cross-Validation** (BCV). The BCV selector presents a *hybrid strategy* that combines plug-in and cross-validation ideas. It starts by considering the AMISE expression in (2.24)

$$\text{AMISE}[\hat{f}(\cdot;h)] = \frac{1}{4}\mu_2^2(K)R(f'')h^4 + \frac{R(K)}{nh}$$

and then plugs-in an estimate for $R(f'')$ based on a modification of $R(\hat{f}''(\cdot;h))$. The modification is

$$\begin{aligned}\widetilde{R(f'')} &:= R(\hat{f}''(\cdot;h)) - \frac{R(K'')}{nh^5} \\ &= \frac{1}{n^2}\sum_{i=1}^{n}\sum_{\substack{j=1 \\ j\neq i}}^{n}(K_h'' * K_h'')(X_i - X_j), \end{aligned} \tag{2.31}$$

a *leave-out-diagonals* estimate of $R(f'')$. It is designed to reduce the bias of $R(\hat{f}''(\cdot;h))$, since $\mathbb{E}\left[R(\hat{f}''(\cdot;h))\right] = R(f'') + \frac{R(K'')}{nh^5} + O(h^2)$

(Scott and Terrell, 1987). Plugging (2.31) into the AMISE expression yields the BCV objective function and the BCV bandwidth selector:

$$\begin{aligned}\text{BCV}(h) &:= \frac{1}{4}\mu_2^2(K)\widetilde{R(f'')}h^4 + \frac{R(K)}{nh},\\ \hat{h}_{\text{BCV}} &:= \arg\operatorname{locmin}_{h>0}\text{BCV}(h),\end{aligned}$$

where $\arg\operatorname{locmin}_{h>0}\text{BCV}(h)$ stands for the smallest local minimizer of $\text{BCV}(h)$. The consideration of the local minimum is because, by design, $\text{BCV}(h) \to 0$ as $h \to \infty$ for fixed n: $\widetilde{R(f'')} \to 0$, at a faster rate than $O(h^{-4})$. Therefore, when minimizing $\text{BCV}(h)$, some care is required, as one is actually interested in obtaining the *smallest local minimizer*. Consequently, bandwidth grids with an upper extreme that is too large[35] are to be avoided, as these will miss the local minimum in favor of the global one at $h \to \infty$.

[35] The precise point at which a bandwidth is "too big" can be formalized with the *maximal smoothing* principle, as elaborated in Section 3.2.2 in Wand and Jones (1995).

The most appealing property of $\hat{h}_{\text{BCV}}$ is that it has a considerably smaller variance than $\hat{h}_{\text{LSCV}}$. This reduction in variance comes at the price of an increased bias, which tends to make $\hat{h}_{\text{BCV}}$ larger than h_{MISE}.

$\hat{h}_{\text{BCV}}$ is implemented in R through the function `bw.bcv`. Again, `bw.bcv` uses R's `optimize` so the `bw.bcv.mod` function is presented to have better guarantees on finding the first local minimum. Quite some care is needed with the range of bandwidth grid, though, to avoid the global minimum for large bandwidths.

```
# Data
set.seed(123456)
x <- rnorm(100)

# BCV gives a warning
bw.bcv(x = x)
## [1] 0.4500924

# Extend search interval
args(bw.bcv)
## function (x, nb = 1000L, lower = 0.1 * hmax, upper = hmax, tol = 0.1 *
##     lower)
## NULL
bw.bcv(x = x, lower = 0.01, upper = 1)
## [1] 0.5070129

# bw.bcv.mod replaces the optimization routine of bw.bcv with an exhaustive
# search on "h_grid" (chosen adaptatively from the sample) and optionally
# plots the BCV curve with "plot_cv"
bw.bcv.mod <- function(x, nb = 1000L,
                       h_grid = 10^seq(-3, log10(1.2 * sd(x) *
                                                 length(x)^(-1/5)), l = 200),
                       plot_cv = FALSE) {
  if ((n <- length(x)) < 2L)
    stop("need at least 2 data points")
  n <- as.integer(n)
  if (is.na(n))
    stop("invalid length(x)")
  if (!is.numeric(x))
    stop("invalid 'x'")
  nb <- as.integer(nb)
  if (is.na(nb) || nb <= 0L)
    stop("invalid 'nb'")
  storage.mode(x) <- "double"
  hmax <- 1.144 * sqrt(var(x)) * n^(-1/5)
```

```
  Z <- .Call(stats:::C_bw_den, nb, x)
  d <- Z[[1L]]
  cnt <- Z[[2L]]
  fbcv <- function(h) .Call(stats:::C_bw_bcv, n, d, cnt, h)
  ## Original code
  # h <- optimize(fbcv, c(lower, upper), tol = tol)$minimum
  # if (h < lower + tol | h > upper - tol)
  #   warning("minimum occurred at one end of the range")
  ## Modification
  obj <- sapply(h_grid, function(h) fbcv(h))
  h <- h_grid[which.min(obj)]
  if (h %in% range(h_grid))
    warning("minimum occurred at one end of h_grid")
  if (plot_cv) {
    plot(h_grid, obj, type = "o")
    rug(h_grid)
    abline(v = h, col = 2, lwd = 2)
  }
  h
}

# Compute the bandwidth and plot the BCV curve
bw.bcv.mod(x = x, plot_cv = TRUE, h_grid = 10^seq(-1.25, 0.5, l = 200))
## [1] 0.5111433

# We can compare with the default bw.bcv output
abline(v = bw.bcv(x = x), col = 3)
```

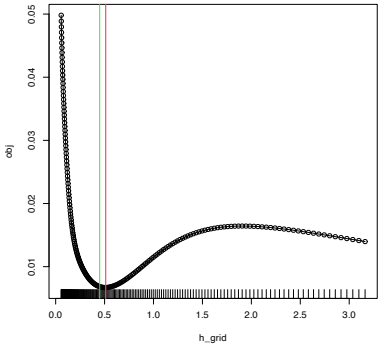


Figure 2.10: BCV curve evaluated for a grid of bandwidths, with a clear local minimum corresponding to $\hat{h}_{\text{BCV}}$. Observe the decreasing pattern when $h \to \infty$.

2.4.3 *Comparison of bandwidth selectors*

Next, we state some insights from the convergence results of the DPI, LSCV, and BCV selectors. All of them are based on results of the kind

$$n^{\nu}(\hat{h}/h_{\text{MISE}} - 1) \xrightarrow{d} \mathcal{N}(0, \sigma^2), \tag{2.32}$$

where σ^2 depends on K and f only, and measures how variable the selector is. The rate n^{ν} serves to quantify how fast[36] the relative error $\hat{h}/h_{\text{MISE}} - 1$ decreases (the larger the ν, the faster the convergence).

Under certain regularity conditions, we have:

- $n^{1/10}(\hat{h}_{\text{LSCV}}/h_{\text{MISE}} - 1) \xrightarrow{d} \mathcal{N}(0, \sigma^2_{\text{LSCV}})$ and $n^{1/10}(\hat{h}_{\text{BCV}}/h_{\text{MISE}} - 1) \xrightarrow{d} \mathcal{N}(0, \sigma^2_{\text{BCV}})$. Both cross-validation selectors have a slow rate of convergence.[37] Inspection of the variances of both selectors reveals that, for the normal kernel, $\sigma^2_{\text{LSCV}}/\sigma^2_{\text{BCV}} \approx 15.7$. Therefore, **LSCV is considerably more variable than BCV**.

- $n^{5/14}(\hat{h}_{\text{DPI}}/h_{\text{MISE}} - 1) \xrightarrow{d} \mathcal{N}(0, \sigma^2_{\text{DPI}})$. Thus, the **DPI selector** has a **convergence rate much faster** than the **cross-validation selectors**. There is an appealing explanation for this phenomenon. Recall that $\hat{h}_{\text{BCV}}$ minimizes the slightly modified version of $\text{BCV}(h)$ given by

$$\frac{1}{4}\mu_2^2(K)\tilde{\psi}_4(h)h^4 + \frac{R(K)}{nh}$$

[36] Recall that another way of writing (2.32) is as the relative error $\frac{\hat{h}-h_{\text{MISE}}}{h_{\text{MISE}}}$ being asymptotically distributed as a $\mathcal{N}\left(0, \frac{\sigma^2}{n^{2\nu}}\right)$.

[37] Compare it with the $n^{1/2}$ rate of the CLT!

and

$$\begin{aligned}\tilde{\psi}_4(h) :=& \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} (K_h'' * K_h'')(X_i - X_j) \\ =& \frac{n}{n-1} \widetilde{R(f'')}.\end{aligned}$$

$\tilde{\psi}_4$ is a leave-out-diagonals estimate of ψ_4. Despite being different from $\hat{\psi}_4$, it serves for building a DPI analogous to BCV that points towards the precise fact that drags down the performance of BCV. The modified version of the DPI minimizes

$$\frac{1}{4}\mu_2^2(K)\tilde{\psi}_4(g)h^4 + \frac{R(K)}{nh},$$

where g is independent of h. The two methods differ in **the way g is chosen**: BCV sets $g = h$ and the modified DPI looks for the best g in terms of the $\text{AMSE}[\tilde{\psi}_4(g)]$. It can be seen that $g_{\text{AMSE}} = O(n^{-2/13})$, whereas the h used in BCV is asymptotically $O(n^{-1/5})$. This suboptimality on the choice of g is the reason of the asymptotic deficiency of BCV.

We now focus on exploring the empirical performance of bandwidth selectors. The workhorse for doing that is simulation. A popular collection of simulation scenarios was given by Marron and Wand (1992) and is conveniently available through the package `nor1mix`. This collection is formed by normal r-mixtures of the form

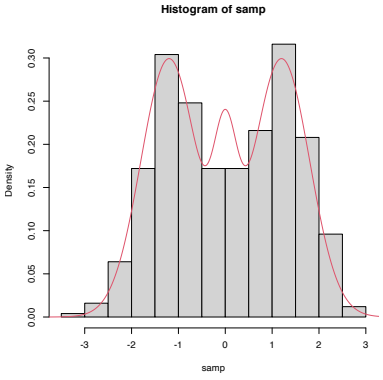
$$f(x; \boldsymbol{\mu}, \boldsymbol{\sigma}, \mathbf{w}) := \sum_{j=1}^{r} w_j \phi_{\sigma_j}(x - \mu_j),$$

where $w_j \geq 0$, $j = 1, \ldots, r$ and $\sum_{j=1}^{r} w_j = 1$. Densities of this form are especially attractive since they allow for arbitrary flexibility and, if the normal kernel is employed, they allow for *explicit and exact* MISE expressions:

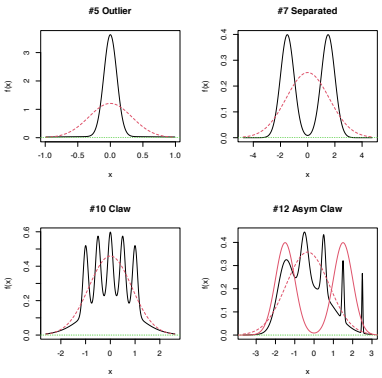
$$\begin{aligned}\text{MISE}_r[\hat{f}(\cdot; h)] &= (2\sqrt{\pi}nh)^{-1} + \mathbf{w}'\{(1 - n^{-1})\boldsymbol{\Omega}_2 - 2\boldsymbol{\Omega}_1 + \boldsymbol{\Omega}_0\}\mathbf{w}, \\ (\boldsymbol{\Omega}_a)_{ij} &= \phi_{(ah^2 + \sigma_i^2 + \sigma_j^2)^{1/2}}(\mu_i - \mu_j), \quad i, j = 1, \ldots, r.\end{aligned} \tag{2.33}$$

These exact MISE expressions are highly convenient for computing h_{MISE}, as defined in (2.23), by minimizing (2.33) numerically.

```
# Available models
?nor1mix::MarronWand

# Simulating
samp <- nor1mix::rnorMix(n = 500, obj = nor1mix::MW.nm9)
# MW object in the second argument
hist(samp, freq = FALSE)

# Density evaluation
x <- seq(-4, 4, length.out = 400)
lines(x, nor1mix::dnorMix(x = x, obj = nor1mix::MW.nm9), col = 2)
```



```
# Plot a MW object directly
# A normal with the same mean and variance is plotted in dashed lines --
# you can remove it with argument p.norm = FALSE
par(mfrow = c(2, 2))
plot(nor1mix::MW.nm5)
plot(nor1mix::MW.nm7)
plot(nor1mix::MW.nm10)
plot(nor1mix::MW.nm12)
lines(nor1mix::MW.nm7, col = 2) # Also possible
```



Exercise 2.18. Implement the h_{MISE} using (2.23) and (2.33) for model `nor1mix::MW.nm6`. Then, compute by Monte Carlo the densities of $\hat{h}_{\text{DPI}}/h_{\text{MISE}} - 1$, $\hat{h}_{\text{LSCV}}/h_{\text{MISE}} - 1$, and $\hat{h}_{\text{BCV}}/h_{\text{MISE}} - 1$. Compare them for $n = 100, 200, 500$, adding a vertical line to represent the h_{MISE} bandwidth. Describe in detail the results and the major takeaways.

Exercise 2.19. Compare the MISE and AMISE criteria in three densities in `nor1mix` of your choice. To that purpose, code (2.33) and the AMISE expression for the normal kernel, and compare the two error curves. Compare them for $n = 100, 200, 500$, adding a vertical line to represent the h_{MISE} and h_{AMISE} bandwidths. Describe in detail the results and the major takeaways.

A key practical issue that emerges after discussing several bandwidth selectors is the following:

Which bandwidth selector is the most adequate for a given dataset?

Unfortunately, there is no simple and universal answer to this question. There are, however, a series of useful facts and suggestions:

- Trying several selectors and inspecting the results may help to determine which one is estimating the density better.
- The **DPI selector has a convergence rate much faster than the cross-validation selectors**. Therefore, in theory it is expected to perform better than LSCV and BCV. For this reason, it tends to be among the **preferred bandwidth selectors** in the literature.
- Cross-validatory selectors may be better suited for highly non-normal and rough densities, in which plug-in selectors may end up oversmoothing.
- LSCV tends to be considerably more variable than BCV.
- The RT is a quick, simple, and inexpensive selector. However, it tends to give bandwidths that are too large for non-normal-like data.

Figure 2.11 presents a visualization of the performance of the kde with different bandwidth selectors, carried out in the family of mixtures by Marron and Wand (1992).

2.5 *Practical issues*

In this section we discuss several practical issues for kernel density estimation.

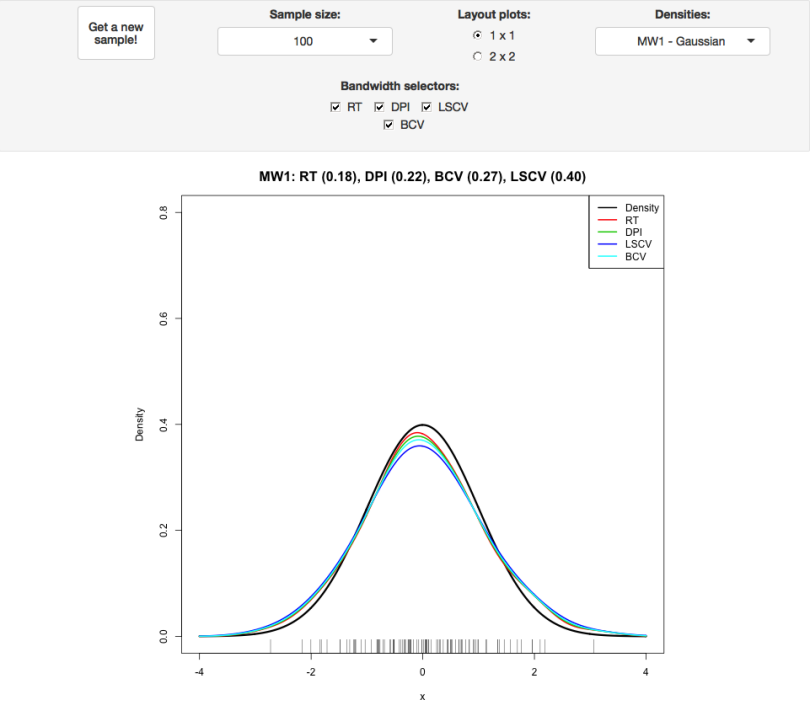
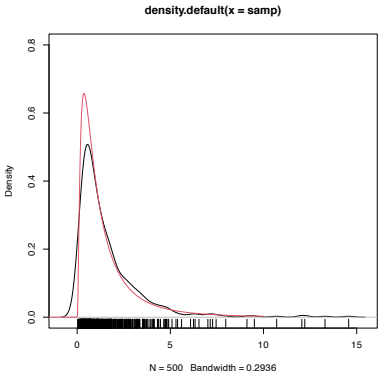


Figure 2.11: Performance comparison of bandwidth selectors. The RT, DPI, LSCV, and BCV are computed for each sample for a normal mixture density. For each sample, the animation computes the ISEs of the selectors and sorts them from best to worst. Changing the scenarios gives insight into the adequacy of each selector to hard- and simple-to-estimate densities. Application available here.

2.5.1 *Boundary issues and transformations*

In Section 2.3 we assumed certain regularity conditions for f. Assumption **A1** stated that f should be twice continuously differentiable (on $\mathbb{R}$!). It is simple to think a counterexample to that: take any density with a support with boundary, for example a $\mathcal{LN}(0,1)$ in $(0,\infty)$, as seen below. The kde will run into trouble!

```
# Sample from a LN(0, 1)
set.seed(123456)
samp <- rlnorm(n = 500)

# kde and density
plot(density(samp), ylim = c(0, 0.8))
curve(dlnorm(x), from = -2, to = 10, n = 500, col = 2, add = TRUE)
rug(samp)
```



What is happening is clear: the kde is spreading probability mass outside the support of $\mathcal{LN}(0,1)$, because the kernels are functions defined in $\mathbb{R}$. Since the kde places probability mass at negative values, it takes it from the positive side, resulting in a severe negative bias about the right-hand side of 0. As a consequence, the kde does not integrate one in the support of the data. No matter what the sample size considered is, the kde will always have a **negative bias of $O(h)$ at the boundary**, instead of the standard $O(h^2)$.

A simple approach to deal with the *boundary bias* is to map a non-real-supported density f into a real-supported density g, which is simpler to estimate, by means of a transformation t:

$$f(x) = g(t(x))t'(x).$$

The *transformation kde* is obtained by replacing g with the usual kde, but acting on the transformed data $t(X_1), \ldots, t(X_n)$:

$$\hat{f}_{\mathrm{T}}(x; h, t) := \frac{1}{n} \sum_{i=1}^{n} K_h(t(x) - t(X_i)) t'(x). \tag{2.34}$$

Note that h is in the scale of $t(X_i)$, not X_i. Hence, another benefit of this approach, in addition to its simplicity, is that bandwidth selection can be done transparently[38] in terms of the already discussed bandwidth selectors: **select a bandwidth from $t(X_1), \ldots, t(X_n)$ and use it in (2.34)**. A table with some common transformations[39] is the following.

Transform.	Data in	$t(x)$	$t'(x)$
Log	(a, ∞)	$\log(x-a)$	$\frac{1}{x-a}$
Probit	(a, b)	$\Phi^{-1}\left(\frac{x-a}{b-a}\right)$	$(b-a)\phi\left(\frac{\Phi^{-1}(x)-a}{b-a}\right)^{-1}$
Shifted power	$(-\lambda_1, \infty)$ (data skewed)	$(x+\lambda_1)^{\lambda_2}\mathrm{sign}(\lambda_2)$ if $\lambda_2 \neq 0$	$\lambda_2(x+\lambda_1)^{\lambda_2-1}\mathrm{sign}(\lambda_2)$ if $\lambda_2 \neq 0$

[38] However, note that the optimality on selecting a bandwidth for estimating g from $t(X_1), \ldots, t(X_n)$ may not translate into an optimality for estimating f.

[39] If $\lambda_2 \to 0$ in the shifted power, then the log transformation follows with $a = -\lambda_1$.

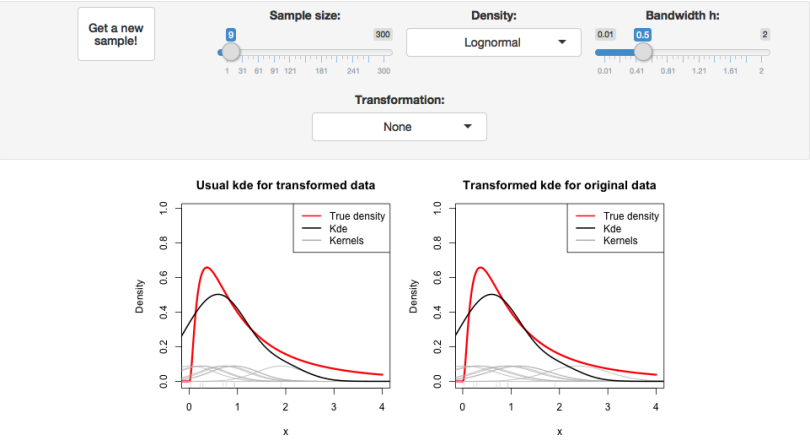
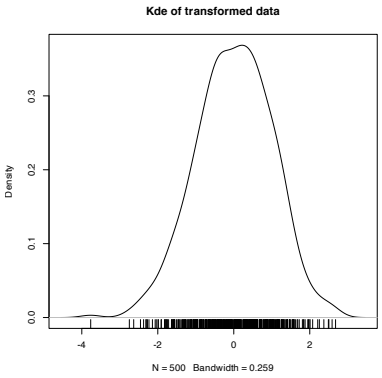


Figure 2.12: Construction of the transformation kde for the log and probit transformations. The left panel shows the kde (2.7) applied to the transformed data. The right plot shows the transformed kde (2.34) applied to the original data. Application available here.

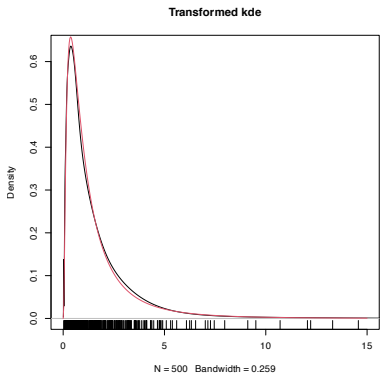
The code below illustrates how to compute a transformation kde in practice from `density`.

```
# kde with log-transformed data
kde <- density(log(samp))
plot(kde, main = "Kde of transformed data")
rug(log(samp))
```




```
# Careful: kde$x is in the reals!
range(kde$x)
## [1] -4.542984  3.456035

# Untransform kde$x so the grid is in (0, infty)
kde_transf <- kde
kde_transf$x <- exp(kde_transf$x)

# Transform the density using the chain rule
```

```
kde_transf$y <- kde_transf$y * 1 / kde_transf$x

# Transformed kde
plot(kde_transf, main = "Transformed kde", xlim = c(0, 15))
curve(dlnorm(x), col = 2, add = TRUE, n = 500)
rug(samp)
```



Exercise 2.20. Consider the data given by `set.seed(12345); x <- rbeta(n = 500, shape1 = 2, shape2 = 2)`. Compute:

a. The untransformed kde employing the DPI and LSCV selectors. Overlay the true density.
b. The transformed kde employing a probit transformation and using the DPI and LSCV selectors on the transformed data.

Exercise 2.21 (Exercise 2.23 in Wand and Jones (1995))**.** Show that the bias and variance for the transformation kde (2.34) are

$$\begin{aligned}\mathrm{Bias}[\hat{f}_{\mathrm{T}}(x;h,t)] &= \frac{1}{2}\mu_2(K)g''(t(x))t'(x)h^2 + o(h^2),\\ \mathbb{V}\mathrm{ar}[\hat{f}_{\mathrm{T}}(x;h,t)] &= \frac{R(K)}{nh}g(t(x))t'(x)^2 + o((nh)^{-1}),\end{aligned}$$

where g is the density of $t(X)$.

Exercise 2.22. Using the results from Exercise 2.21, prove that

$$\begin{aligned}\mathrm{AMISE}[\hat{f}_{\mathrm{T}}(\cdot;h,t)] = \frac{1}{4}\mu_2^2(K)\int (g''(x))^2 t'(t^{-1}(x))\,\mathrm{d}x h^4\\ + \frac{R(K)}{nh}\mathbb{E}[t'(X)],\end{aligned}$$

where g is the density of $t(X)$.

2.5.2 *Sampling*

Sampling a kde is relatively straightforward. The trick is to recall

$$\hat{f}(x;h) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - X_i)$$

as a mixture density made of n mixture components in which each of them is sampled independently. The only part that might require special treatment is sampling from the density K, although R contains specific sampling functions for the most popular kernels.

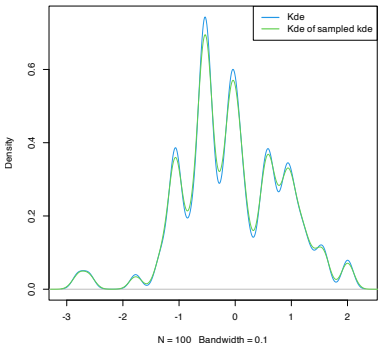
The algorithm for sampling N points goes as follows:

1. Choose $i \in \{1,\ldots,n\}$ at random.
2. Sample from $K_h(\cdot - X_i) = \frac{1}{h}K\left(\frac{\cdot - X_i}{h}\right)$.
3. Repeat the previous steps N times.

Let's see a quick example.

```
# Sample the claw
n <- 100
set.seed(23456)
samp <- nor1mix::rnorMix(n = n, obj = nor1mix::MW.nm10)
```

```r
# Kde
h <- 0.1
plot(density(samp, bw = h), main = "", col = 4)

# # Naive sampling algorithm
# N <- 1e6
# samp_kde <- numeric(N)
# for (k in 1:N) {
#
#   i <- sample(x = 1:n, size = 1)
#   samp_kde[k] <- rnorm(n = 1, mean = samp[i], sd = h)
#
# }

# Sample N points from the kde
N <- 1e6
i <- sample(x = n, size = N, replace = TRUE)
samp_kde <- rnorm(N, mean = samp[i], sd = h)

# Add kde of the sampled kde -- almost equal
lines(density(samp_kde), col = 3)
legend("topright", legend = c("Kde", "Kde of sampled kde"),
       lwd = 2, col = 4:3)
```



Exercise 2.23. Sample data points from the kde of `iris$Petal.Width` that is computed with the NS selector.

Exercise 2.24. The dataset `sunspot.year` contains the yearly numbers of sunspots from 1700 to 1988 (rounded to one digit). Employing a log-transformed kde with DPI bandwidth, sample new sunspots observations. Check by simulation that the sampling is done appropriately by comparing the log-transformed kde of the sampled data with the original kde.

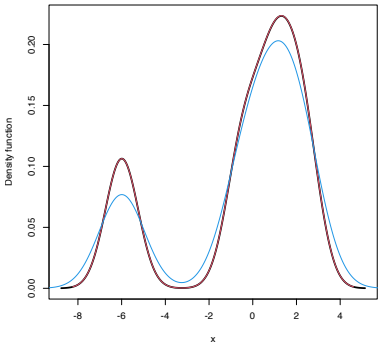
2.6 *Kernel density estimation with `ks`*

The `density` function in R presents certain limitations, such as the impossibility of evaluating the kde at arbitrary points, the unavailability of built-in transformations, and the lack of a multivariate extension. The `ks` package (Duong, 2020) delivers the `ks::kde` function, providing these and other functionalities. It will be the workhorse for carrying out multivariate kde (Section 3.1). The only drawback of `ks::kde` to be aware of is that it only considers normal kernels – though these are by far the most popular.

The following chunk of code shows that `density` and `ks::kde` give the same outputs. It also presents some of the extra flexibility on the evaluation that `ks::kde` provides.

```r
# Sample
n <- 5
set.seed(123456)
samp_t <- rt(n, df = 2)

# Comparison: same output and same parametrization for bandwidth
bw <- 0.75
plot(kde <- ks::kde(x = samp_t, h = bw), lwd = 3) # ?ks::plot.kde for options
lines(density(x = samp_t, bw = bw), col = 2)
# Beware: there is no lines() method for ks::kde objects
```

```r
# The default h is the DPI obtained by ks::hpi
kde <- ks::kde(x = samp_t)

# Manual plot -- recall $eval.points and $estimate
lines(kde$eval.points, kde$estimate, col = 4)
```



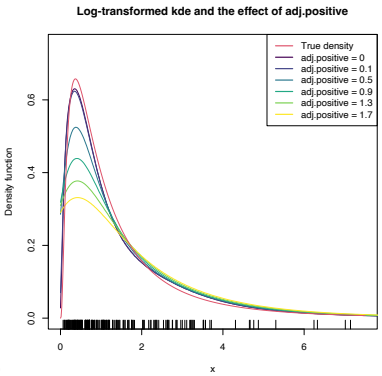

```r
# Evaluating the kde at specific points, e.g., the first 2 sample points
ks::kde(x = samp_t, h = bw, eval.points = samp_t[1:2])
## $x
## [1]  1.4650470 -0.4971902  0.6701367  2.3647267 -5.9918352
##
## $eval.points
## [1]  1.4650470 -0.4971902
##
## $estimate
## [1] 0.2223029 0.1416113
##
## $h
## [1] 0.75
##
## $H
## [1] 0.5625
##
## $gridtype
## [1] "linear"
##
## $gridded
## [1] TRUE
##
## $binned
## [1] TRUE
##
## $names
## [1] "x"
##
## $w
## [1] 1 1 1 1 1
##
## $type
## [1] "kde"
##
## attr(,"class")
## [1] "kde"

# By default ks::kde() computes the *binned* kde (much faster) and then employs
# an interpolation to evaluate the kde at the given grid; if the exact kde is
# desired, this can be specified with binned = FALSE
ks::kde(x = samp_t, h = bw, eval.points = samp_t[1:2], binned = FALSE)
## $x
## [1]  1.4650470 -0.4971902  0.6701367  2.3647267 -5.9918352
##
## $eval.points
## [1]  1.4650470 -0.4971902
##
## $estimate
## [1] 0.2223316 0.1416132
##
## $h
## [1] 0.75
##
## $H
## [1] 0.5625
##
## $gridded
## [1] FALSE
##
## $binned
```

```
## [1] FALSE
##
## $names
## [1] "x"
##
## $w
## [1] 1 1 1 1 1
##
## $type
## [1] "kde"
##
## attr(,"class")
## [1] "kde"
```

```
# Changing the size of the evaluation grid
length(ks::kde(x = samp_t, h = bw, gridsize = 1e3)$estimate)
## [1] 1000
```

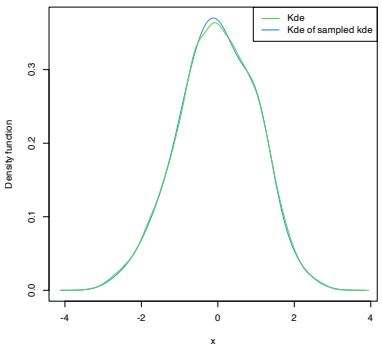
The computation of log-transformed kde is straightforward using the `positive` and `adj.positive`.

```
# Sample from a LN(0, 1)
set.seed(123456)
samp_ln <- rlnorm(n = 200)

# Log-kde without shifting
a <- seq(0.1, 2, by = 0.4) # Sequence of shifts
col <- viridis::viridis(length(a) + 1)
plot(ks::kde(x = samp_ln, positive = TRUE), col = col[1],
     main = "Log-transformed kde and the effect of adj.positive",
     xlim = c(0, 7.5), ylim = c(0, 0.75))
# If h is not provided, then ks::hpi() is called on the transformed data

# Shifting: larger a increases the bias
for (i in seq_along(a)) {
  plot(ks::kde(x = samp_ln, positive = TRUE, adj.positive = a[i]),
       add = TRUE, col = col[i + 1])
}
curve(dlnorm(x), col = 2, add = TRUE, n = 500)
rug(samp_ln)
legend("topright", legend = c("True density", paste("adj.positive =", c(0, a))),
       col = c(2, col), lwd = 2)
```

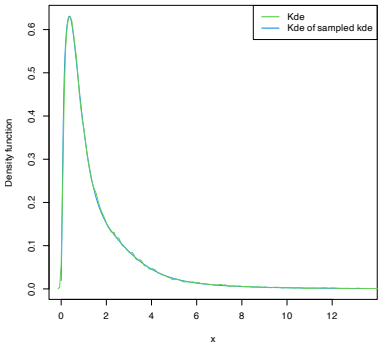


Finally, sampling from the kde and log-transformed kde is easily achieved by the use of `ks::rkde`.

```
# Untransformed kde
plot(kde <- ks::kde(x = log(samp_ln)), col = 4)
samp_kde <- ks::rkde(n = 5e4, fhat = kde)
plot(ks::kde(x = samp_kde), add = TRUE, col = 3)
legend("topright", legend = c("Kde", "Kde of sampled kde"),
       lwd = 2, col = 3:4)
```




```
# Transformed kde
plot(kde_transf <- ks::kde(x = samp_ln, positive = TRUE), col = 4)
samp_kde_transf <- ks::rkde(n = 5e4, fhat = kde_transf, positive = TRUE)
plot(ks::kde(x = samp_kde_transf), add = TRUE, col = 3)
legend("topright", legend = c("Kde", "Kde of sampled kde"),
       lwd = 2, col = 3:4)
```



Exercise 2.25. Consider the `MNIST` dataset in the `MNIST-tSNE.RData` file. It contains the first 60,000 images of the MNIST database. Each observation is a grayscale image made of 28×28 pixels that

is recorded as a vector of length $28^2 = 784$ by concatenating the pixel columns. The entries of these vectors are numbers in $[0, 1]$ indicating the level of grayness of the pixel: 0 for white, 1 for black. These vectorised images are stored in the $60,000 \times 784$ matrix in `MNIST$x`. The 0–9 labels of the digits are stored in `MNIST$labels`.

a. Compute the average gray level, `av_gray_one`, for each image of the digit "1".
b. Compute and plot the kde of `av_gray_one`. Consider taking into account that it is a positive variable.
c. Overlay the lognormal distribution density, with parameters estimated by maximum likelihood (use `MASS::fitdistr`).
d. Repeat c for the Weibull density.
e. Which parametric model seems more adequate?

3
Kernel density estimation II

Like in the univariate case, any random vector $\mathbf{X}$ supported in $\mathbb{R}^p$ is completely characterized by its cdf. However, cdfs are even harder to visualize and interpret when $p > 1$, as the accumulation of probability happens simultaneously in several directions. As a consequence, densities become highly valuable tools for data exploration, especially for dimensions $p = 2, 3$.

Densities characterize continuous random vectors $\mathbf{X}$ and are key building blocks for a variety of highly successful multivariate methods. Indeed, many of them may be seen as connected to the omnipresent normal distribution.

As we will see in this chapter, the concepts associated with multivariate density estimation extend quite straightforwardly from the univariate situation. However, despite this conceptual simplicity, and as often happens with any problem in statistics, density estimation in $\mathbb{R}^p$ becomes more and more complex as the dimension p increases.

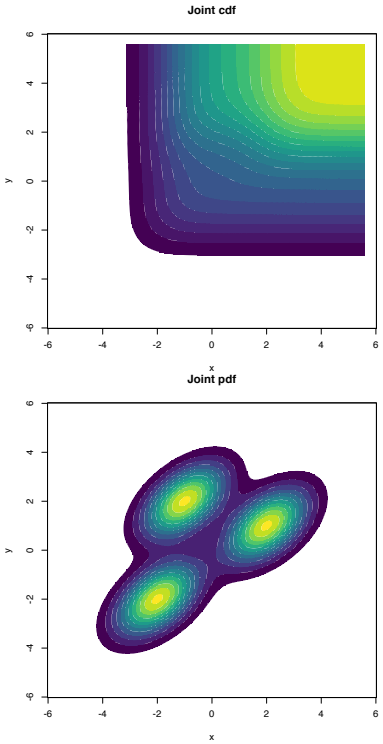


Figure 3.1: The contour levels of the joint cdf and pdf for a mixture of three bivariate normals. Clearly, the pdf surface yields insights into the structure of the continuous random vector $\mathbf{X}$, whereas the cdf gives hardly any.

3.1 *Multivariate kernel density estimation*

Kernel density estimation can be extended to estimate multivariate densities f in $\mathbb{R}^p$ based on the same principle: perform an average of densities "centered" at the data points. For a sample $\mathbf{X}_1, \ldots, \mathbf{X}_n$ in $\mathbb{R}^p$, the kde of f evaluated at $\mathbf{x} \in \mathbb{R}^p$ is defined as

$$\hat{f}(\mathbf{x}; \mathbf{H}) := \frac{1}{n|\mathbf{H}|^{1/2}} \sum_{i=1}^{n} K\left(\mathbf{H}^{-1/2}(\mathbf{x} - \mathbf{X}_i)\right), \tag{3.1}$$

where K is *multivariate kernel*, a p-variate density that is (typically) symmetric and unimodal at $\mathbf{0}$, and that depends on the *bandwidth matrix*[1] $\mathbf{H}$, a $p \times p$ symmetric and *positive definite*[2] matrix.

A common notation is $K_{\mathbf{H}}(\mathbf{z}) := |\mathbf{H}|^{-1/2} K(\mathbf{H}^{-1/2}\mathbf{z})$, the so-called *scaled kernel*, so the kde can be compactly written as $\hat{f}(\mathbf{x}; \mathbf{H}) := \frac{1}{n}\sum_{i=1}^{n} K_{\mathbf{H}}(\mathbf{x} - \mathbf{X}_i)$. The most employed multivariate kernel is the normal kernel $K(\mathbf{z}) = \phi(\mathbf{z}) = (2\pi)^{-p/2} e^{-\frac{1}{2}\mathbf{z}'\mathbf{z}}$, for which $K_{\mathbf{H}}(\mathbf{x} - \mathbf{X}_i) = \phi_{\mathbf{H}}(\mathbf{z} - \mathbf{X}_i)$. Then, the bandwidth $\mathbf{H}$ can be thought of as the *variance-covariance matrix* of a multivariate normal density with mean $\mathbf{X}_i$ and the kde (3.1) can be regarded as a data-driven mixture of those densities.

[1] Observe that, if $p = 1$, then $\mathbf{H}$ will equal the *square* of the bandwidth h, that is, $\mathbf{H} = h^2$.

[2] A positive definite matrix is a real symmetric matrix with positive eigenvalues. Recall that this is of key importance in order to guarantee that $|\mathbf{H}|^{1/2}$ ($|\mathbf{H}|$ must be non-negative!) and $\mathbf{H}^{-1/2}$ are well-defined ($\mathbf{H}^{-1/2} := \mathbf{P}\mathbf{\Lambda}^{1/2}\mathbf{P}'$ where $\mathbf{H} = \mathbf{P}\mathbf{\Lambda}\mathbf{P}'$ is the spectral decomposition of $\mathbf{H}$ with $\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_p)$ and $\mathbf{\Lambda}^{1/2} := \text{diag}(\lambda_1^{1/2}, \ldots, \lambda_p^{1/2})$).

The interpretation of (3.1) is analogous to the one of (2.7): build a mixture of densities with each density centered at each data point. As a consequence, and roughly speaking, most of the concepts and ideas seen in univariate kernel density estimation extend to the multivariate situation, although some of them have considerable technical complications. For example, bandwidth selection inherits the same cross-validatory ideas (LSCV and BCV selectors) and plug-in methods (NS and DPI) seen before, but with increased complexity for the BCV and DPI selectors.

Exercise 3.1. Using that $|\mathbf{H}| = \prod_{i=1}^{p} \lambda_i$ and $\text{tr}(\mathbf{H}) = \sum_{i=1}^{p} \lambda_i$, where $\lambda_1, \ldots, \lambda_p$ are the eigenvalues of $\mathbf{H}$, find a simple check for a symmetric matrix $\mathbf{H}$ that guarantees its positive definiteness (and hence its adequacy as a bandwidth for (3.1)) when $p = 2$.

Recall that considering a *full* bandwidth matrix $\mathbf{H}$ gives more flexibility to the kde, but also quadratically increases the amount of bandwidth parameters that need to be chosen – precisely $\frac{p(p+1)}{2}$ – which notably complicates bandwidth selection as the dimension p grows, and increases the variance of the kde. A common simplification is to consider a diagonal bandwidth matrix $\mathbf{H} = \text{diag}(h_1^2, \ldots, h_p^2)$, which yields the kde employing product kernels:

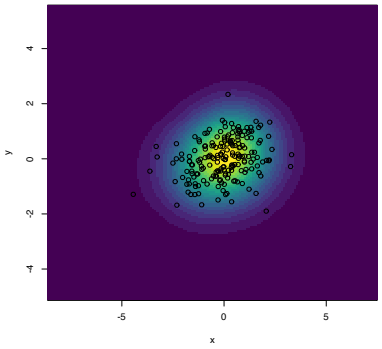
$$\hat{f}(\mathbf{x}; \mathbf{h}) = \frac{1}{n} \sum_{i=1}^{n} K_{h_1}(x_1 - X_{i,1}) \times \overset{p}{\cdots} \times K_{h_p}(x_p - X_{i,p}), \tag{3.2}$$

where $\mathbf{X}_i = (X_{i,1}, \ldots, X_{i,p})'$ and $\mathbf{h} = (h_1, \ldots, h_p)'$ is the vector of bandwidths. If the variables $X_1, \ldots, X_p$ are standardized (so that they have the same scale), then a simple choice is to consider $h = h_1 = \cdots = h_p$. Diagonal bandwidth matrices will be thoroughly employed when performing kernel regression estimation in Chapter 4.

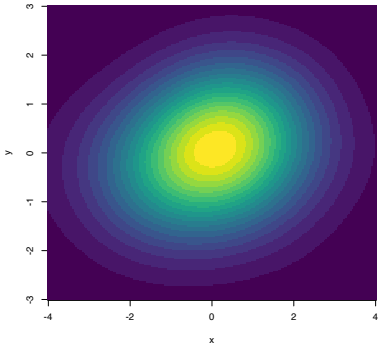
Multivariate kernel density estimation and bandwidth selection is not supported in base R, but `ks::kde` implements both for $p \leq 6$. In the following code snippet, the functionalities of `ks::kde` are illustrated for data in $\mathbb{R}^2$.

```
# Simulated data from a bivariate normal
n <- 200
set.seed(35233)
x <- mvtnorm::rmvnorm(n = n, mean = c(0, 0),
                      sigma = rbind(c(1.5, 0.25), c(0.25, 0.5)))

# Compute kde for a diagonal bandwidth matrix (trivially positive definite)
H <- diag(c(1.25, 0.75))
kde <- ks::kde(x = x, H = H)

# The eval.points slot contains the grids on x and y
str(kde$eval.points)
## List of 2
##  $ : num [1:151] -8.58 -8.47 -8.37 -8.26 -8.15 ...
##  $ : num [1:151] -5.1 -5.03 -4.96 -4.89 -4.82 ...

# The grids in kde$eval.points are crossed in order to compute a grid matrix
# where to evaluate the estimate
dim(kde$estimate)
## [1] 151 151
```

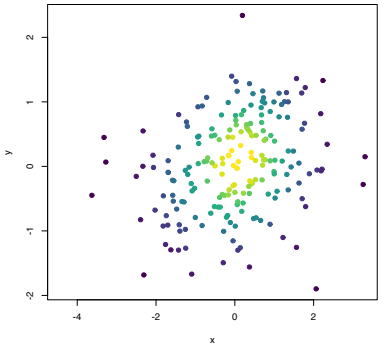
```r
# Manual plotting using the kde object structure
image(kde$eval.points[[1]], kde$eval.points[[2]], kde$estimate,
      col = viridis::viridis(20), xlab = "x", ylab = "y")
points(kde$x) # The data is returned in $x
```



```r
# Changing the grid size to compute the estimates to be 200 x 200 and in the
# rectangle (-4, 4) x c(-3, 3)
kde <- ks::kde(x = x, H = H, gridsize = c(200, 200), xmin = c(-4, -3),
               xmax = c(4, 3))
image(kde$eval.points[[1]], kde$eval.points[[2]], kde$estimate,
      col = viridis::viridis(20), xlab = "x", ylab = "y")
```



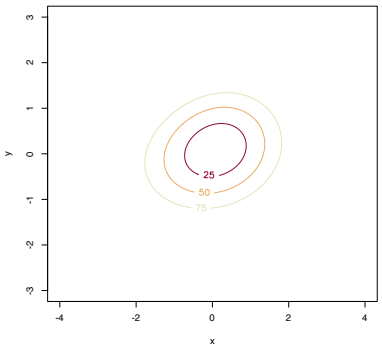
```r
dim(kde$estimate)
## [1] 200 200

# Do not confuse "gridsize" with "bgridsize". The latter controls the internal
# grid size for binning the data and speeding up the computations (compare
# with binned = FALSE for a large sample size), and is not recommended to
# modify unless you know what you are doing. The binning takes place if
# binned = TRUE or if "binned" is not specified and the sample size is large

# Evaluating the kde at specific points can be done with "eval.points"
kde_sample <- ks::kde(x = x, H = H, eval.points = x)
str(kde_sample$estimate)
##  num [1:200] 0.0803 0.0332 0.0274 0.0739 0.0411 ...

# Assign colors automatically from quantiles to have an idea the densities of
# each one
n_cols <- 20
quantiles <- quantile(kde_sample$estimate, probs = seq(0, 1, l = n_cols + 1))
col <- viridis::viridis(n_cols)[cut(kde_sample$estimate, breaks = quantiles)]
plot(x, col = col, pch = 19, xlab = "x", ylab = "y")
```



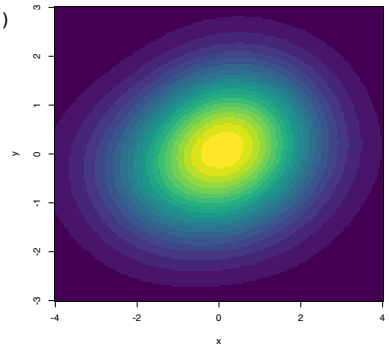
```r
# Binning vs. not binning
abs(max(ks::kde(x = x, H = H, eval.points = x, binned = TRUE)$estimate -
          ks::kde(x = x, H = H, eval.points = x, binned = FALSE)$estimate))
## [1] 2.189159e-05
```

There are specific, more sophisticated, plot methods for `ks::kde` objects via `ks::plot.kde`.

```r
# Contourplot
plot(kde, display = "slice", cont = c(25, 50, 75), xlab = "x", ylab = "y")
```

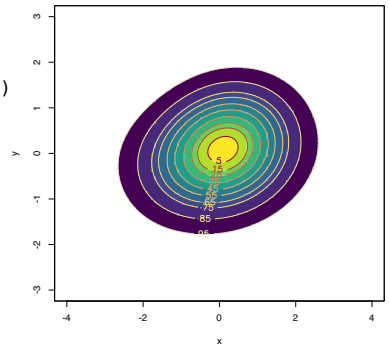



```r
# "cont" specifies the density contours, which are upper percentages of the
# highest density regions. The default contours are at 25%, 50%, and 75%

# Raw image with custom colors
plot(kde, display = "image", xlab = "x", ylab = "y", col = viridis::viridis(20))
```



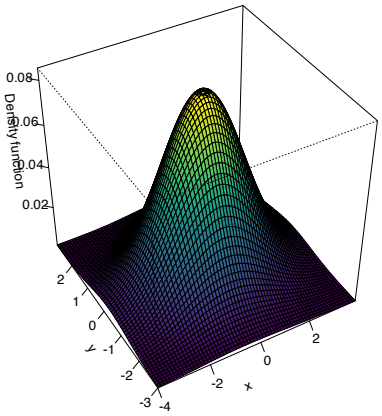
```r
# Filled contour with custom color palette in "col.fun"
plot(kde, display = "filled.contour2", cont = seq(5, 95, by = 10),
     xlab = "x", ylab = "y", col.fun = viridis::viridis)
# Alternatively: col = viridis::viridis(length(cont) + 1)

# Add contourlevels
plot(kde, display = "filled.contour", cont = seq(5, 95, by = 10),
     xlab = "x", ylab = "y", col.fun = viridis::viridis)
plot(kde, display = "slice", cont = seq(5, 95, by = 10), add = TRUE)
```

```
# Perspective plot
plot(kde, display = "persp", col.fun = viridis::viridis, xlab = "x", ylab = "y")
```



Kernel density estimation in $\mathbb{R}^3$ can be visualized via 3D contours (to be discussed in more detail in Section 3.5.1) that represent the level surfaces.

```
# Simulated data from a trivariate normal
n <- 500
set.seed(213212)
x <- mvtnorm::rmvnorm(n = n, mean = c(0, 0, 0),
                      sigma = rbind(c(1.5, 0.25, 0.5),
                                    c(0.25, 0.75, 1),
                                    c(0.5, 1, 2)))

# Show nested contours of high-density regions
plot(ks::kde(x = x, H = diag(c(rep(1.25, 3)))), drawpoints = TRUE, col.pt = 1)

# Beware! Incorrect (not symmetric or positive definite) bandwidths do not
# generate an error, but they return a non-sense kde
head(ks::kde(x = x, H = diag(c(1, 1, -1)), eval.points = x)$estimate)
head(ks::kde(x = x, H = diag(c(1, 1, 0)), eval.points = x)$estimate)

# H not positive definite
H <- rbind(c(1.5, 0.25, 0.5),
           c(0.25, 0.75, -1.5),
           c(0.5, -1.5, 2))
eigen(H)$values
head(ks::kde(x = x, H = H, eval.points = x)$estimate)

# H semipositive definite but not positive definite
H <- rbind(c(1.5, 0.25, 0.5),
           c(0.25, 0.5, 1),
           c(0.5, 1, 2))
eigen(H)$values
head(ks::kde(x = x, H = H, eval.points = x)$estimate) # Numerical instabilities
```



The kde can be computed in higher dimensions (up to $p \leq 6$, the maximum supported by `ks`) with a little care to avoid a bug in versions of `ks` *prior* to 1.11.4. For these outdated versions, the bug was present in the `ks::kde` function for dimensions $p \geq 4$, as illustrated in the example below.

```
# Sample test data
p <- 4
data <- mvtnorm::rmvnorm(n = 10, mean = rep(0, p))
kde <- ks::kde(x = data, H = diag(rep(1, p))) # Error on the verbose argument
```

The bug resided in the default arguments of the internal function `ks:::kde.points`, and as a consequence made `ks::kde` not immediately usable. Although the bug has been patched since the 1.11.4 version of `ks`, it is interesting to observe that this and other bugs one may encounter in any function within an R package (even internal functions) can be patched in-session by means of the following code, which simply replaces a function in the environment of the loaded package.

```
# Create the replacement function. In this case, we just set the default
# argument of ks:::kde.points to F (FALSE)
kde.points.fixed <- function (x, H, eval.points, w, verbose = FALSE) {
  n <- nrow(x)
```

```
  d <- ncol(x)
  ne <- nrow(eval.points)
  Hs <- replicate(n, H, simplify = FALSE)
  Hs <- do.call(rbind, Hs)
  fhat <- dmvnorm.mixt(x = eval.points, mus = x, Sigmas = Hs,
                       props = w / n, verbose = verbose)
  return(list(x = x, eval.points = eval.points, estimate = fhat,
              H = H, gridded = FALSE))
}

# Assign package environment to the replacement function
environment(kde.points.fixed) <- environment(ks:::kde.points)

# Overwrite original function with replacement (careful -- you will have to
# restart session to come back to the original object)
assignInNamespace(x = "kde.points", value = kde.points.fixed, ns = "ks",
                  pos = 3)
# ns = "ks" to indicate the package namespace, pos = 3 to indicate :::

# Check the result
ks:::kde.points
```

Another peculiarity of `ks::kde` to be aware of is that it does not implement binned kde for dimension $p > 4$, so it is necessary to set the flag `binned = FALSE`[3] when calling `ks::kde`.

[3] Beware! For the package version 1.11.3 the error message has a typo and asks to set `binned = TRUE`, precisely what generated the error. This has been fixed since version 1.11.4.

3.2 *Density derivative estimation*

As we will see in Section 3.5, sometimes it is interesting to estimate the derivatives of the density, particularly the gradient and the Hessian, rather than the density itself.

For a density $f : \mathbb{R}^p \longrightarrow \mathbb{R}$ its gradient $\mathrm{D}f : \mathbb{R}^p \longrightarrow \mathbb{R}^p$ is defined as

$$\mathrm{D}f(\mathbf{x}) := \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_p} \end{pmatrix},$$

which can be regarded as the result of applying the differential operator $\mathrm{D} := \left(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_p}\right)'$ to f. The Hessian of f is the matrix

$$\mathrm{H}f(\mathbf{x}) := \left(\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}\right), \quad i, j = 1, \ldots, p.$$

The Hessian is a well-known object in multivariate calculus, but in the way it is defined somehow closes the way to the construction of high-order derivatives that admit a simple algebraic disposition.[4] Thus, an alternative arrangement of the Hessian is imperative, and this can by achieved by its *columnwise stacking* performed by the vec **operator**. The vec operator stacks the columns of any matrix into a long vector[5] and makes the Hessian operator $\mathrm{H} := \left(\frac{\partial^2}{\partial x_i \partial x_j}\right)$ expressible as the vector

$$\operatorname{vec} \mathrm{H} = \begin{pmatrix} \frac{\partial^2}{\partial x_1 \partial x_1} \\ \vdots \\ \frac{\partial^2}{\partial x_p \partial x_p} \end{pmatrix} \in \mathbb{R}^{p^2}. \tag{3.3}$$

[4] Which is the algebraic object containing all the *third* derivatives? And all the *fourth* derivatives?

[5] Precisely, given $\mathbf{A}_{n\times m} = (a_{ij})$, then $\operatorname{vec} \mathbf{A} := (\underbrace{a_{11}, \ldots, a_{n1}}_{\text{Column 1}}, \ldots, \underbrace{a_{1m}, \ldots, a_{nm}}_{\text{Column } m})'$ is a column vector in $\mathbb{R}^{nm}$.

Interestingly, (3.3) coincides with the *Kronecker product*[6] of D with itself: $\mathrm{D}^{\otimes 2} := \mathrm{D} \otimes \mathrm{D}$ (a column vector with p^2 entries). That is,

$$\operatorname{vec}(\mathrm{H}f(\mathbf{x})) = \mathrm{D}^{\otimes 2} f(\mathbf{x}). \tag{3.4}$$

The previous relation may seem to be just a mathematical formalism without a clear practical relevance. However, it is *key* for simplifying the consideration of second and high-order derivatives of a function f. Indeed, the r-th derivatives of f, for $r \geq 0$[7], can now be collected in terms of the ***r-fold Kronecker product of*** D:

$$\mathrm{D}^{\otimes r} f(\mathbf{x}) := \begin{pmatrix} \frac{\partial^r f(\mathbf{x})}{\partial x_1 \cdots \partial x_1} \\ \vdots \\ \frac{\partial^r f(\mathbf{x})}{\partial x_1 \cdots \partial x_p} \\ \vdots \\ \frac{\partial^r f(\mathbf{x})}{\partial x_p \cdots \partial x_p} \end{pmatrix} \in \mathbb{R}^{p^r}. \tag{3.5}$$

This formalism is extremely useful, for example, to obtain a compact expansion of the univariate Taylor's theorem (Theorem 1.11).

Theorem 3.1 (Multivariate Taylor's Theorem)**.** *Let $f : \mathbb{R}^p \longrightarrow \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^p$. Assume that f has r continuous derivatives in $B(\mathbf{x}, \delta) := \{\mathbf{y} \in \mathbb{R}^p : \|\mathbf{y} - \mathbf{x}\| < \delta\}$. Then, for any $\|\mathbf{h}\| < \delta$,*

$$f(\mathbf{x} + \mathbf{h}) = \sum_{j=0}^{r} \frac{1}{j!} (\mathrm{D}^{\otimes j} f(\mathbf{x}))' \mathbf{h}^{\otimes j} + R_r, \quad R_r = o(\|\mathbf{h}\|^r). \tag{3.6}$$

Observe the compactness and scalability of (3.6), featuring multiplications between vectors of lengths p^j for $j = 0, \ldots, r$. This contrasts with the traditional second-order[8] multivariate Taylor expansion:

$$f(\mathbf{x} + \mathbf{h}) = f(\mathbf{x}) + (\mathrm{D}f(\mathbf{x}))' \mathbf{h} + \frac{1}{2} \mathbf{h}' \mathrm{H}f(\mathbf{x}) \mathbf{h} + o(\|\mathbf{h}\|^2).$$

Exercise 3.2. Let $f(x, y) := x^3 + \cos(y)$. Compute:

1. $\mathrm{D}f$ and $\mathrm{H}f$.
2. $\mathrm{D}^{\otimes 2} f$. Check that $\operatorname{vec}(\mathrm{H}f) = \mathrm{D}^{\otimes 2} f$.
3. $\mathrm{D}^{\otimes 3} f = \mathrm{D} \otimes \mathrm{D}^{\otimes 2} f$.
4. The Taylor expansion of f at $\mathbf{x} = (1, 0)'$ for the first, second, and third orders.

Exercise 3.3. Considering the setting in Exercise 3.2:

1. Plot the surface for f about $\mathbf{x} = (1, 0)'$.
2. Plot the surfaces for the three Taylor expansions at $\mathbf{x} = (1, 0)'$ and verify their increasing accuracy on approximating f.

For visualizing the different surfaces, rely on several `image` panels with the same vertical scale or use `rgl::surface3d`.

Exercise 3.4. Perform Exercises 3.2 and 3.3 using the function $f(x, y) = \phi_{\sigma_1}(x)\phi_{\sigma_2}(y)$ and $\mathbf{x} = (0, 0)'$. Consider $\sigma_1 = \sigma_2 = 1$, and $\sigma_1 = 1.25$ and $\sigma_2 = 0.75$.

[6] The Kronecker product of two matrices $\mathbf{A}_{n \times m} = (a_{ij})$ and $\mathbf{B}_{p \times q} = (b_{ij})$ is defined as the $(np) \times (mq)$ matrix $\mathbf{A} \otimes \mathbf{B} := \begin{pmatrix} a_{11}\mathbf{B} & \cdots & a_{1m}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{n1}\mathbf{B} & \cdots & a_{nm}\mathbf{B} \end{pmatrix}$. Of course, $\mathbf{A} \otimes \mathbf{B} \neq \mathbf{B} \otimes \mathbf{A}$.

[7] If $r = 0$, $\mathrm{D}^{\otimes 0} f := f$.

[8] Often truncated at two terms because it becomes too cumbersome (without the (3.5) operator!) for larger orders.

[9] To be precise, this is the collection of p^r mixed r-th derivatives (if $r = 1$, the gradient; if $r = 2$, the Hessian in its vectorized form).

Employing the (3.5) operator, the kernel estimator for the *r-th derivative*[9] $\mathrm{D}^{\otimes r} f$ is defined as the application of the operator $\mathrm{D}^{\otimes r}$ to the kde $\hat{f}(\cdot; \mathbf{H})$:

$$\widehat{\mathrm{D}^{\otimes r} f}(\mathbf{x}; \mathbf{H}) := \mathrm{D}^{\otimes r} \hat{f}(\mathbf{x}; \mathbf{H}) = \frac{1}{n} \sum_{i=1}^{n} (\mathrm{D}^{\otimes r} K_{\mathbf{H}})(\mathbf{x} - \mathbf{X}_i), \tag{3.7}$$

where $\mathrm{D}^{\otimes r} K_{\mathbf{H}}$ can be computed as

$$\mathrm{D}^{\otimes r} K_{\mathbf{H}} = |\mathbf{H}|^{-1/2} (\mathbf{H}^{-1/2})^{\otimes r} (\mathrm{D}^{\otimes r} K)(\mathbf{H}^{-1/2}(\mathbf{x} - \mathbf{X}_i)). \tag{3.8}$$

The gradient estimator has a key relevance in Section 3.5.2 and is the simplest derivative estimator. Its form follows either by replacing $r = 1$ in (3.7) and (3.8) or by directly differentiating (3.1):

$$\widehat{\mathrm{D}f}(\mathbf{x}; \mathbf{H}) = \frac{\mathbf{H}^{-1/2}}{n|\mathbf{H}|^{1/2}} \sum_{i=1}^{n} (\mathrm{D}K)(\mathbf{H}^{-1/2}(\mathbf{x} - \mathbf{X}_i)).$$

If we set $(\mathrm{D}K)_{\mathbf{H}}(\mathbf{x} - \mathbf{X}_i) := |\mathbf{H}|^{-1/2} (\mathrm{D}K)(\mathbf{H}^{-1/2}(\mathbf{x} - \mathbf{X}_i))$, then it can clearly be seen that

$$\widehat{\mathrm{D}f}(\mathbf{x}; \mathbf{H}) = \frac{\mathbf{H}^{-1/2}}{n} \sum_{i=1}^{n} (\mathrm{D}K)_{\mathbf{H}}(\mathbf{x} - \mathbf{X}_i)$$

and how the **extra factor** $\mathbf{H}^{-1/2}$ appears. This extra factor explains why the optimal bandwidth selectors for (3.1) and (3.7) are different. The estimator of the Hessian will be employed in Section 3.5.4.

The estimator (3.7) can be computed using `ks::kdde`. The following code presents how to do so for $p = 1$ and $r = 1, 2$.

```
# Simulated univariate data
n <- 1e3
set.seed(324178)
x <- nor1mix::rnorMix(n = n, obj = nor1mix::MW.nm8)

# Location of relative extrema
dens <- function(x) nor1mix::dnorMix(x, obj = nor1mix::MW.nm8)
minus_dens <- function(x) -dens(x)
extrema <- c(nlm(f = minus_dens, p = 0)$estimate,
             nlm(f = dens, p = 0.75)$estimate,
             nlm(f = minus_dens, p = 1.5)$estimate)

# Plot target density
par(mfrow = c(2, 2))
plot(nor1mix::MW.nm8, p.norm = FALSE)
rug(x)
abline(v = extrema, col = c(3, 2, 3))

# Density estimation (automatically chosen bandwidth)
kdde_0 <- ks::kdde(x = x, deriv.order = 0)
plot(kdde_0, xlab = "x", main = "Density estimation")
abline(v = extrema, col = c(3, 2, 3))

# Density derivative estimation (automatically chosen bandwidth, but different
# from kdde_0!)
kdde_1 <- ks::kdde(x = x, deriv.order = 1)
plot(kdde_1, xlab = "x", main = "Density derivative estimation")
abline(v = extrema, col = c(3, 2, 3))

# Density second derivative estimation
kdde_2 <- ks::kdde(x = x, deriv.order = 2)
plot(kdde_2, xlab = "x", main = "Density second derivative estimation")
abline(v = extrema, col = c(3, 2, 3))
```

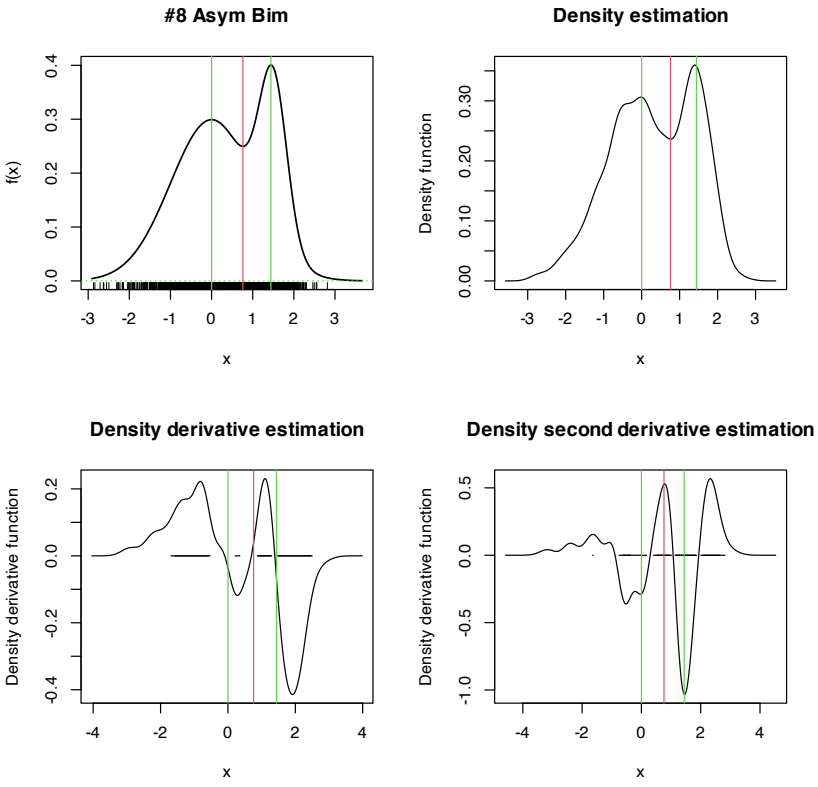
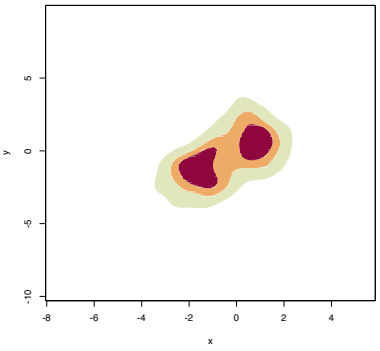


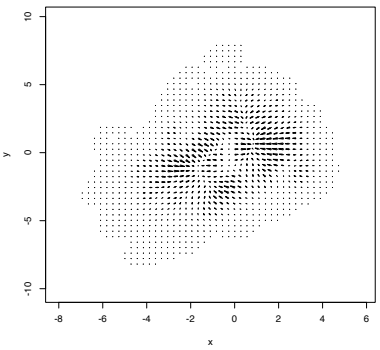
Figure 3.2: Univariate density derivative estimation for $r = 0, 1, 2$. The green vertical bars represent the modes (local maxima) and the red vertical bar the *antimode* (local minimum) of `nor1mix::MW8`. Observe how the density derivative estimation vanishes at the local extrema and how the second derivative estimation captures the sign of the extrema, thus indicating the presence of a mode or an antimode.

The following code presents how to do so for $p = 2$ and $r = 1, 2$.

```
# Simulated bivariate data
n <- 1e3
mu_1 <- rep(1, 2)
mu_2 <- rep(-1.5, 2)
Sigma_1 <- matrix(c(1, -0.75, -0.75, 3), nrow = 2, ncol = 2)
Sigma_2 <- matrix(c(2, 0.75, 0.75, 3), nrow = 2, ncol = 2)
w <- 0.45
set.seed(324178)
x <- ks::rmvnorm.mixt(n = n, mus = rbind(mu_1, mu_2),
                      Sigmas = rbind(Sigma_1, Sigma_2), props = c(w, 1 - w))

# Density estimation
kdde_0 <- ks::kdde(x = x, deriv.order = 0)
plot(kdde_0, display = "filled.contour2", xlab = "x", ylab = "y")
```

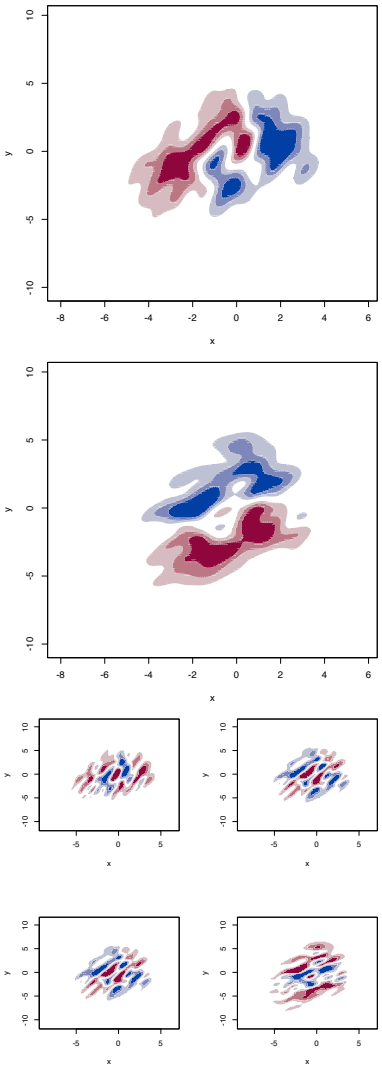



```
# Density derivative estimation
kdde_1 <- ks::kdde(x = x, deriv.order = 1)
str(kdde_1$estimate)
## List of 2
##  $ : num [1:151, 1:151] -4.76e-19 4.18e-19 1.18e-19 -6.27e-20 -3.74e-19 ...
##  $ : num [1:151, 1:151] -7.66e-19 -1.18e-18 -4.19e-19 -6.70e-19 -8.30e-19 ...
# $estimate is now a list of two matrices with each of the derivatives

# Plot of the gradient field - arrows pointing towards the modes
plot(kdde_1, display = "quiver", xlab = "x", ylab = "y")
```




```
# Plot of the two components of the gradient field
for(i in 1:2) {
  plot(kdde_1, display = "filled.contour2", which.deriv.ind = i,
       xlab = "x", ylab = "y")
}
```

```
# Second density derivative estimation
kdde_2 <- ks::kdde(x = x, deriv.order = 2)
str(kdde_2$estimate)
## List of 4
##  $ : num [1:151, 1:151] -2.53e-19 -6.84e-19 -2.33e-19 -1.33e-18 -4.44e-19 ...
##  $ : num [1:151, 1:151] -1.55e-20 9.60e-20 3.56e-19 7.44e-20 1.11e-19 ...
##  $ : num [1:151, 1:151] -1.55e-20 9.60e-20 3.56e-19 7.44e-20 1.11e-19 ...
##  $ : num [1:151, 1:151] -5.53e-19 -3.68e-19 6.63e-21 -7.58e-19 -1.28e-19 ...
# $estimate is now a list of four matrices with each of the derivatives

# Plot of the two components of the gradient field ("which.deriv.ind" indicates
# the index in the Kronecker product)
par(mfcol = c(2, 2))
for(i in 1:4) {
  plot(kdde_2, display = "filled.contour2", which.deriv.ind = i,
       xlab = "x", ylab = "y")
}
```



Exercise 3.5. Inspecting the second derivatives of a bivariate kde can be challenging. For that reason, the *summary curvature* $s(\mathbf{x}) = -1_{\{\mathrm{H}f(\mathbf{x}) \text{ is negative definite}\}}||\mathrm{H}f(\mathbf{x})||$ is often considered, as it serves for indicating the presence and "strength" of a local mode. Implement, from the output of `ks::kdde`, the kernel estimator for s. Then, compare your implementation with the function `ks::kcurv` and use both to determine the presence of modes in the previous example.

To evaluate the performance of kernel estimators of the derivatives, it is very useful to recall that the gradient and Hessian of a multivariate normal density $\phi_{\boldsymbol{\Sigma}}(\cdot - \boldsymbol{\mu})$ are:[10]

$$\mathrm{D}\phi_{\boldsymbol{\Sigma}}(\mathbf{x}-\boldsymbol{\mu}) = -\phi_{\boldsymbol{\Sigma}}(\mathbf{x}-\boldsymbol{\mu})\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}), \tag{3.9}$$

$$\mathrm{H}\phi_{\boldsymbol{\Sigma}}(\mathbf{x}-\boldsymbol{\mu}) = \phi_{\boldsymbol{\Sigma}}(\mathbf{x}-\boldsymbol{\mu})\boldsymbol{\Sigma}^{-1}\left((\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})' - \boldsymbol{\Sigma}\right)\boldsymbol{\Sigma}^{-1}. \tag{3.10}$$

These functions can be implemented as follows.

[10] Check equations (347) and (348) in Petersen and Pedersen (2012), for example.

```
# Gradient of a N(mu, Sigma) density (vectorized on x)
grad_norm <- function(x, mu, Sigma) {

  # Check dimensions
  x <- rbind(x)
  p <- length(mu)
  stopifnot(ncol(x) == p & nrow(Sigma) == p & ncol(Sigma) == p)

  # Gradient
  grad <- -mvtnorm::dmvnorm(x = x, mean = mu, sigma = Sigma) *
    t(t(x) - mu) %*% solve(Sigma)
  return(grad)

}

# Hessian of a N(mu, Sigma) density (vectorized on x)
Hess_norm <- function(x, mu, Sigma) {

  # Check dimensions
  x <- rbind(x)
  p <- length(mu)
  stopifnot(ncol(x) == p & nrow(Sigma) == p & ncol(Sigma) == p)

  # Hessian
  Sigma_inv <- solve(Sigma)
```

```
  H <- apply(x, 1, function(y) {
    mvtnorm::dmvnorm(x = y, mean = mu, sigma = Sigma) *
      (Sigma_inv %*% tcrossprod(y - mu) %*% Sigma_inv - Sigma_inv)
  })

  # As an array
  return(array(data = c(H), dim = c(p, p, nrow(x))))

}
```

Obviously, in the case of a mixture $\sum_{j=1}^r w_j \phi_{\boldsymbol{\Sigma}_j}(\mathbf{x} - \boldsymbol{\mu}_j)$, the gradient becomes $\sum_{j=1}^r w_j \mathrm{D}\phi_{\boldsymbol{\Sigma}_j}(\mathbf{x} - \boldsymbol{\mu}_j)$ and the Hessian, $\sum_{j=1}^r w_j \mathrm{H}\phi_{\boldsymbol{\Sigma}_j}(\mathbf{x} - \boldsymbol{\mu}_j)$.

Exercise 3.6. Graphically evaluate the accuracy on estimating the density gradient and Hessian of the estimators considered in the previous code chunks. For that aim:

1. Use (3.9) and (3.10) to compute the gradients and Hessians of the mixtures of normal densities considered for $p = 1, 2$.
2. Compare these density derivatives with their estimators.

3.3 *Asymptotic properties*

The asymptotic results for the multivariate kde are very similar to those of the univariate kde, but with an increasing notational complexity. Hopefully, the vec operator, (3.5), and Theorem 3.1 allow for expression simplification and yield a clear connection with, for example, the expressions for the asymptotic bias and variance obtained in Theorem 2.1. As before, the insights obtained from these expressions will be highly valuable to select $\mathbf{H}$ optimally in practice by means of the derivation of the MISE and AMISE errors.

We need to make the following assumptions:

- **A1**.[11] The density f is square integrable, twice continuously differentiable, and all the second order partial derivatives are square integrable.
- **A2**.[12] The kernel K is a *spherically symmetric*[13] and bounded pdf with finite second moment and square integrable.
- **A3**.[14] $\mathbf{H} = \mathbf{H}_n$ is a deterministic sequence of positive definite symmetric matrices such that, when $n \to \infty$, $\operatorname{vec} \mathbf{H} \to \mathbf{0}$ and $n|\mathbf{H}|^{1/2} \to \infty$.

[11] This assumption requires certain smoothness of f, allowing thus for Theorem 3.1 to be applied.

[12] Mild assumption that makes the first term of the Taylor expansion of f negligible and the second bounded.

[13] This is the extension of the symmetry requirement for a univariate kernel to $\mathbb{R}^p$. The spherical symmetry of K implies that $\int \mathbf{z} K(\mathbf{z})\, \mathrm{d}\mathbf{z} = \mathbf{0}$ and that $\int \mathbf{z}\mathbf{z}' K(\mathbf{z})\, \mathrm{d}\mathbf{z} = \mu_2(K)\mathbf{I}_p$ (the covariances are zero), where $\mu_2(K) := \int z_j^2 K(\mathbf{z})\, \mathrm{d}\mathbf{z} = \int z_k^2 K(\mathbf{z})\, \mathrm{d}\mathbf{z}$ for all $j, k = 1, \ldots, p$. Equivalently, $\int \mathbf{z}^{\otimes 2} K(\mathbf{z})\, \mathrm{d}\mathbf{z} = \mu_2(K) \operatorname{vec} \mathbf{I}_p$.

[14] The key assumption for reducing the bias and variance of $\hat{f}(\cdot; \mathbf{H})$ *simultaneously*.

The convolution between two functions $f, g : \mathbb{R}^p \longrightarrow \mathbb{R}$ is defined analogously as in the univariate case as $(f * g)(\mathbf{x}) := \int f(\mathbf{x} - \mathbf{y}) g(\mathbf{y})\, \mathrm{d}\mathbf{y}$. Thus, we readily obtain that

$$\begin{aligned} \mathbb{E}[\hat{f}(\mathbf{x}; \mathbf{H})] &= \int K_{\mathbf{H}}(\mathbf{x} - \mathbf{y}) f(\mathbf{y})\, \mathrm{d}\mathbf{y} \\ &= (K_{\mathbf{H}} * f)(\mathbf{x}), \\ \mathbb{V}\mathrm{ar}[\hat{f}(\mathbf{x}; \mathbf{H})] &= \frac{1}{n}((K_{\mathbf{H}}^2 * f)(\mathbf{x}) - (K_{\mathbf{H}} * f)^2(\mathbf{x})). \end{aligned} \tag{3.11}$$

Again, although these two expressions are exact, they are hard to interpret. The only immediate insight that we are able to get is that, by equation (3.11), the kde is biased. But neither expression differentiates the effects of kernel, bandwidth, and density, the reason why asymptotic expressions are preferred. In what follows, we denote $R(g) = \int g(\mathbf{z})^2 \, d\mathbf{z}$ for any function $g : \mathbb{R}^p \longrightarrow \mathbb{R}$.

Theorem 3.2. *Under A1–A3, the bias and variance of the kde at* $\mathbf{x}$ *are*

$$\text{Bias}[\hat{f}(\mathbf{x}; \mathbf{H})] = \frac{1}{2}\mu_2(K)(\mathrm{D}^{\otimes 2} f(\mathbf{x}))' \text{vec}\, \mathbf{H} + o(\|\text{vec}\, \mathbf{H}\|), \tag{3.12}$$

$$\mathbb{V}\text{ar}[\hat{f}(\mathbf{x}; \mathbf{H})] = \frac{R(K)}{n|\mathbf{H}|^{1/2}} f(\mathbf{x}) + o((n|\mathbf{H}|^{1/2})^{-1}). \tag{3.13}$$

Proof. The proof follows the lines of the proof of Theorem 2.1 and we only provide a sketch. First, consider the change of variables $\mathbf{z} = \mathbf{H}^{-1/2}(\mathbf{x} - \mathbf{y})$, $\mathbf{y} = \mathbf{x} - \mathbf{H}^{1/2}\mathbf{z}$, $d\mathbf{y} = -|\mathbf{H}|^{1/2} \, d\mathbf{z}$. The integral limits flip and we have

$$\begin{aligned} \mathbb{E}[\hat{f}(\mathbf{x}; \mathbf{H})] &= \int K_{\mathbf{H}}(\mathbf{x} - \mathbf{y}) f(\mathbf{y}) \, d\mathbf{y} \\ &= \int K(\mathbf{z}) f(\mathbf{x} - \mathbf{H}^{1/2}\mathbf{z}) \, d\mathbf{z}. \end{aligned}$$

Since $\mathbf{H} \to \mathbf{0}$, we can apply (3.6) to $f(\mathbf{x} - \mathbf{H}^{1/2}\mathbf{z})$ and then use the properties of the kernel to arrive to (3.12). Therefore, (3.13) is obtained by adapting the steps of the bias and replicating the arguments in the proof of Theorem 2.1. □

Exercise 3.7. Detail, elaborate, and conclude the proof above.

The bias and variance expressions (3.12) and (3.13) give important insights:

- The bias decreases with $\mathbf{H}$.[15] By observing that $(\mathrm{D}^{\otimes 2} f(\mathbf{x}))' \text{vec}\, \mathbf{H} = \text{tr}((\mathrm{H} f(\mathbf{x}))' \mathbf{H})$ [16] we have interesting interpretations:
  - The bias is negative whenever $\mathrm{H} f(\mathbf{x})$ is negative definite.[17] These regions correspond to the *modes (or local maxima) of* f, where the kde *underestimates* f (it tends to be below f).
  - Conversely, the bias is positive whenever $\mathrm{H} f(\mathbf{x})$ is positive definite, which happens in the *antimodes (or local minima)* of f, where the kde *overestimates* f (it tends to be above f).
  - The wilder the curvature $\mathrm{D}^{\otimes 2} f$, the harder to estimate f. Flat density regions are easier to estimate than wiggling regions with high curvature (e.g., with several modes).
- The variance depends directly on $f(\mathbf{x})$. The higher the density, the more variable the kde is. The variance decreases as a factor of $(n|\mathbf{H}|^{1/2})^{-1}$, a consequence of $n|\mathbf{H}|^{1/2}$'s playing the role of the effective sample size for estimating $f(\mathbf{x})$.

[15] If $\mathbf{H} = \text{diag}(h_1^2, \ldots, h_p^2)$, the reduction is clearly seen to be quadratic on the marginal bandwidths.

[16] By (3.4), $\mathrm{D}^{\otimes 2} f(\mathbf{x}) = \text{vec}(\mathrm{H} f(\mathbf{x}))$. On the other hand, $(\text{vec}\, \mathbf{A})' \text{vec}\, \mathbf{B} = \text{tr}(\mathbf{A}'\mathbf{B})$ for any matrices $\mathbf{A}$ and $\mathbf{B}$.

[17] For any two symmetric matrices $\mathbf{A}$ and $\mathbf{B}$ of size $p \times p$ having as sorted eigenvalues $\alpha_1, \ldots, \alpha_p$ and $\beta_1, \ldots, \beta_p$, respectively, it is satisfied that $\sum_{i=1}^p \alpha_i \beta_{p-i} \leq \text{tr}(\mathbf{AB}) \leq \sum_{i=1}^p \alpha_i \beta_i$. Taking $\mathbf{B} = \mathbf{H}$, we know that its eigenvalues are positive because of the positive definiteness of $\mathbf{H}$. If $\mathbf{A} = (\mathrm{H} f(\mathbf{x}))'$ is negative definite, then all its eigenvalues are negative and, as a consequence, $\text{tr}(\mathbf{AB}) < 0$.

3.4 Bandwidth selection

The construction of bandwidth selectors for the multivariate kde is analogous to the univariate case. The first step is to consider the multivariate MISE

$$\begin{aligned}\mathrm{MISE}[\hat{f}(\cdot;\mathbf{H})] &:= \mathbb{E}\left[\int (\hat{f}(\mathbf{x};\mathbf{H}) - f(\mathbf{x}))^2 \,\mathrm{d}\mathbf{x}\right] \\ &= \mathbb{E}\left[\mathrm{ISE}[\hat{f}(\cdot;\mathbf{H})]\right] \\ &= \int \mathrm{MSE}[\hat{f}(\mathbf{x};\mathbf{H})] \,\mathrm{d}\mathbf{x}\end{aligned}$$

and define the optimal MISE bandwidth matrix as

$$\mathbf{H}_{\mathrm{MISE}} := \arg\min_{\mathbf{H}\in \mathrm{SPD}_p} \mathrm{MISE}[\hat{f}(\cdot;\mathbf{H})], \tag{3.14}$$

where SPD_p is the set of positive definite matrices[18] of size p.

In practice, obtaining (3.14) is unfeasible, and the first step towards constructing a usable selector is to derive a more practical (but close to the MISE) error criterion, such as the AMISE. The following result provides it from the bias and variance given in Theorem 3.2.

Corollary 3.1. *Under **A1–A3**,*

$$\mathrm{AMISE}[\hat{f}(\cdot;\mathbf{H})] = \frac{1}{4}\mu_2^2(K) R\left(\mathrm{tr}(\mathbf{H}\mathrm{H}f(\cdot))\right) + \frac{R(K)}{n|\mathbf{H}|^{1/2}}.$$

Therefore, $\mathrm{AMISE}[\hat{f}(\cdot;\mathbf{H})] \to 0$ *when* $n \to \infty$.

Exercise 3.8. Prove Corollary 3.1 by deriving the MSE of $\hat{f}(\mathbf{x};\mathbf{H})$ and integrating it.

Differently to what happened in the univariate case, and despite the closed expression of the AMISE, it is *not* possible to obtain a general bandwidth matrix that minimizes the AMISE, that is, to obtain

$$\mathbf{H}_{\mathrm{AMISE}} := \arg\min_{\mathbf{H}\in \mathrm{SPD}_p} \mathrm{AMISE}[\hat{f}(\cdot;\mathbf{H})] \tag{3.15}$$

explicitly. It is possible in the special case in which $\mathbf{H} = h^2\mathbf{I}_p$, since $R\left(\mathrm{tr}(\mathbf{H}\mathrm{H}f(\cdot))\right) = h^4 R\left(\mathrm{tr}(\mathrm{H}f(\cdot))\right)$ and, differentiating with respect to h, it follows that

$$h_{\mathrm{AMISE}} = \left[\frac{pR(K)}{\mu_2^2(K) R\left(\mathrm{tr}(\mathrm{H}f(\cdot))\right) n}\right]^{1/(p+4)}. \tag{3.16}$$

Exercise 3.9. Show that the expression for h_{AMISE} minimizes $\mathrm{AMISE}[\hat{f}(\cdot;h^2\mathbf{I}_p)]$.

Even if (3.16) has been obtained by means of an important simplification, it gives a very important insight: $h_{\mathrm{AMISE}} = O(n^{-1/(p+4)})$. Therefore, **the larger the dimension p, the larger the optimal bandwidth** needs to be.[19] In addition, it can be seen that, for un-

[18] Observe that implementing the optimization of (3.14) is not trivial, since it is required to enforce the constrain $\mathbf{H} \in \mathrm{SPD}_p$. A neat way of parametrizing $\mathbf{H}$ that induces the positive definiteness constrain is through the (unique) Cholesky decomposition of $\mathbf{H} \in \mathrm{SPD}_p$: $\mathbf{H} = \mathbf{R}'\mathbf{R}$, where $\mathbf{R}$ is a *triangular* matrix with *positive* entries on the diagonal (but the remaining entries unconstrained). Therefore, optimization of (3.14) (if f was known) can be done through the $\frac{p(p+1)}{2}$ entries of $\mathbf{R}$.

[19] Intuitively, this fact can be regarded as the necessity of h_{AMISE} to account for larger neighborhoods about $\mathbf{x}$, as the emptiness of the space $\mathbb{R}^p$ grows with p.

[20] Recall that, restricting to specific bandwidths like $\mathbf{H} = h^2\mathbf{I}_p$, then $\mathbf{H}_{\mathrm{AMISE}} = h^2_{\mathrm{AMISE}}\mathbf{I}_p$, so the expressions $h_{\mathrm{AMISE}} = O(n^{-1/(p+4)})$ and $\mathbf{H}_{\mathrm{AMISE}} = O(n^{-2/(p+4)})$ are perfectly coherent.

constrained bandwidth matrices, $\mathbf{H}_{\text{AMISE}} = O(n^{-2/(p+4)})$ [20] for (3.15), so exactly the same insight holds without imposing $\mathbf{H} = h^2\mathbf{I}_p$ for obtaining (3.16).

However, neither the computation of (3.15) (through numerical optimization over SPD_p) nor of (3.16) can be performed in practice, as both depend on functionals of f.

3.4.1 *Plug-in rules*

The **normal scale bandwidth selector**, denoted by $\hat{\mathbf{H}}_{\text{NS}}$, escapes the dependence of the AMISE on f by replacing the latter with $\phi_{\boldsymbol{\Sigma}}(\cdot - \boldsymbol{\mu})$, for which the curvature term in (3.15) can be computed (using (3.10)). Conveniently, this replacement also allows us to solve *explicitly* (3.15) and, with the normal kernel, results in

$$\mathbf{H}_{\text{NS}} = (4/(p+2))^{2/(p+4)} n^{-2/(p+4)} \boldsymbol{\Sigma}. \tag{3.17}$$

Replacing $\boldsymbol{\Sigma}$ by the sample covariance matrix[21] $\mathbf{S}$ in (3.17) gives $\hat{\mathbf{H}}_{\text{NS}}$, which can be straightforwardly computed in practice.

[21] Or by a robust estimator of $\boldsymbol{\Sigma}$.

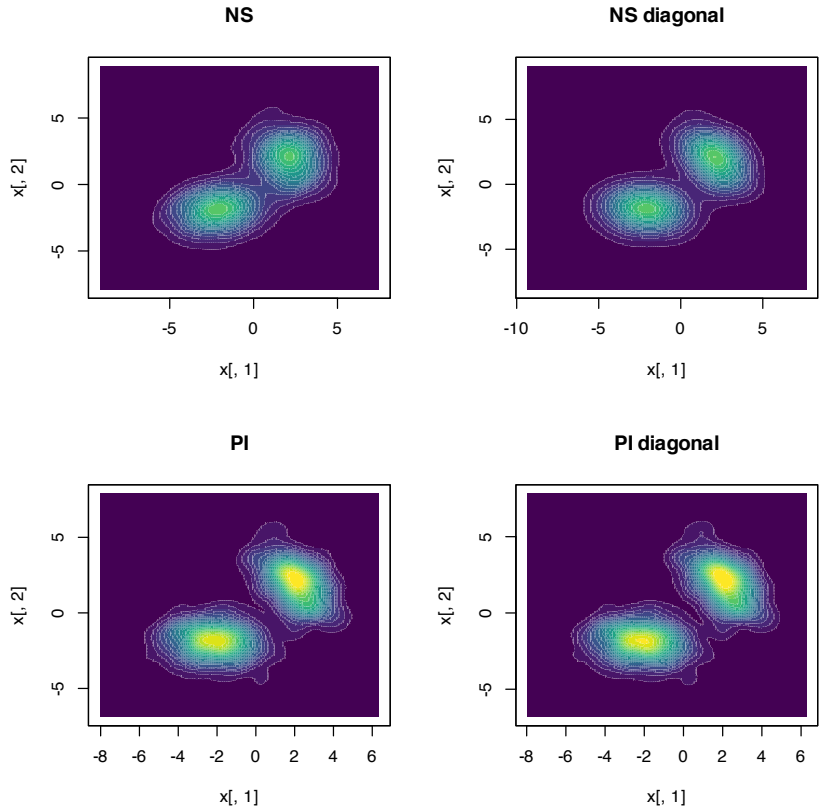
$\hat{\mathbf{H}}_{\text{NS}}$ is a *zero-stage plug-in* selector, a concept that was introduced in Section 2.4. The DPI selector, or henceforth simply **Plug-In selector** (PI), follows the same steps that were employed in Section 2.4.1 to build up an **ℓ-stage plug-in selector**. This selector successively estimates a series of bandwidths, each of them associated with a chain of optimality problems, until "giving up" at stage ℓ (usually, $\ell = 2$) and "sweeping under the carpet" a parametric assumption for f that allows a functional of f to be estimated at the latest optimal expression. In the multivariate context, the conceptual idea is the same, but the technicalities are more demanding, with the extra nuisance of the lack of explicit expressions for the optimal bandwidths in each of the PI steps. We do not go into details[22] and we just denote $\hat{\mathbf{H}}_{\text{PI}}$ to the bandwidth selector that extends $\hat{h}_{\text{DPI}}$ to the multivariate case.

[22] The interested reader is referred to Section 3.6 in Chacón and Duong (2018) for an excellent exposition.

Plug-in selectors are implemented by `ks::Hns` (NS), `ks::Hpi` (PI with full bandwidth matrix) and `ks::Hpi.diag` (PI with diagonal bandwidth matrix). The next chunk of code shows how to use them.

```
# Simulated data
n <- 500
Sigma_1 <- matrix(c(1, -0.75, -0.75, 2), nrow = 2, ncol = 2)
Sigma_2 <- matrix(c(2, -0.25, -0.25, 1), nrow = 2, ncol = 2)
set.seed(123456)
samp <- ks::rmvnorm.mixt(n = n, mus = rbind(c(2, 2), c(-2, -2)),
                         Sigmas = rbind(Sigma_1, Sigma_2),
                         props = c(0.5, 0.5))

# Normal scale bandwidth
(Hns <- ks::Hns(x = samp))
##           [,1]      [,2]
## [1,] 0.7508991 0.4587521
## [2,] 0.4587521 0.6582366

# PI bandwidth unconstrained
(Hpi <- ks::Hpi(x = samp))
##            [,1]       [,2]
```

```
## [1,] 0.26258868 0.03375926
## [2,] 0.03375926 0.23619016

# PI bandwidth diagonal
(Hpi_diag <- ks::Hpi.diag(x = samp))
##           [,1]      [,2]
## [1,] 0.2416352 0.0000000
## [2,] 0.0000000 0.2172413

# Compare kdes
par(mfrow = c(2, 2))
cont <- seq(0, 0.05, l = 20)
col <- viridis::viridis
plot(ks::kde(x = samp, H = Hns), display = "filled.contour2",
     abs.cont = cont, col.fun = col, main = "NS")
plot(ks::kde(x = samp, H = diag(diag(Hns))), display = "filled.contour2",
     abs.cont = cont, col.fun = col, main = "NS diagonal")
plot(ks::kde(x = samp, H = Hpi), display = "filled.contour2",
     abs.cont = cont, col.fun = col, main = "PI")
plot(ks::kde(x = samp, H = Hpi_diag), display = "filled.contour2",
     abs.cont = cont, col.fun = col, main = "PI diagonal")
```



Density derivative estimation

The normal scale selector expands in a simple fashion to the problem of selecting the optimal bandwidth for estimating the *r-th derivative* of f. The solution to this problem follows a path completely parallel to the density case. To begin with, the MISE of the density derivative estimator $\widehat{\mathrm{D}^{\otimes r} f}(\cdot; \mathbf{H})$ is defined as[23]

$$\mathrm{MISE}[\widehat{\mathrm{D}^{\otimes r} f}(\cdot; \mathbf{H})] := \mathbb{E}\left[\int \|\widehat{\mathrm{D}^{\otimes r} f}(\mathbf{x}; \mathbf{H}) - \mathrm{D}^{\otimes r} f(\mathbf{x})\|^2 \, \mathrm{d}\mathbf{x}\right].$$

[23] Note the presence of the squared norm $\|\cdot\|^2$ in the integrand, as $\widehat{\mathrm{D}^{\otimes r} f}(\mathbf{x}; \mathbf{H}) - \mathrm{D}^{\otimes r} f(\mathbf{x})$ is a vector in $\mathbb{R}^{p^r}$.

After obtaining an explicit form for $\mathrm{AMISE}[\widehat{\mathrm{D}^{\otimes r} f}(\cdot; \mathbf{H})]$, the assumption that $f = \phi_{\boldsymbol{\Sigma}}(\cdot - \boldsymbol{\mu})$ for computing the functional depending on f, and the use of the normal kernel, the **normal scale bandwidth selector for the r-th derivative** of f follows:

$$\mathbf{H}_{\mathrm{NS},r} = (4/(p + 2r + 2))^{2/(p+2r+4)} n^{-2/(p+2r+4)} \boldsymbol{\Sigma}. \tag{3.18}$$

Replacing $\boldsymbol{\Sigma}$ by the sample covariance matrix $\mathbf{S}$ in (3.18) gives $\hat{\mathbf{H}}_{\mathrm{NS},r}$, which is remarkably simple to compute in practice.

Observe that $\mathbf{H}_{\mathrm{NS},0} = \mathbf{H}_{\mathrm{NS}}$ and that the factor $f_{n,p,r} := (4/(p + 2r + 2))^{2/(p+2r+4)} n^{-2/(p+2r+4)}$ increases as r increases (see Figure 3.3), indicating that **optimal bandwidths for derivative estimation are *larger*** than optimal bandwidths for density estimation ($r = 0$). Indeed, it can be seen that $\mathbf{H}_{\mathrm{AMISE},r} = O(n^{-2/(p+2r+4)})$. Therefore, employing an optimal bandwidth for density estimation to estimate a derivative will result in an undersmoothing of the derivative.

The PI selector also admits a derivative version, $\hat{\mathbf{H}}_{\mathrm{PI},r}$, which sacrifices the simplicity of (3.18) to gain performance on estimating $\mathbf{H}_{\mathrm{AMISE},r}$ for non-normal-like densities.

```
# Normal scale bandwidth (compare with Hns)
(Hns1 <- ks::Hns(x = samp, deriv.order = 1))
##          [,1]      [,2]
## [1,] 1.138867 0.6957760
## [2,] 0.695776 0.9983282

# PI bandwidth unconstrained (compare with Hpi)
(Hpi1 <- ks::Hpi(x = samp, deriv.order = 1))
##           [,1]      [,2]
## [1,] 0.3248011 0.1125982
## [2,] 0.1125982 0.3017212

# PI bandwidth diagonal (compare with Hpi_diag)
(Hpi_diag1 <- ks::Hpi.diag(x = samp, deriv.order = 1))
##           [,1]      [,2]
## [1,] 0.2854801 0.0000000
## [2,] 0.0000000 0.2545556

# Compare kdes
par(mfrow = c(2, 2))
cont <- seq(-0.02, 0.02, l = 21)
plot(ks::kde(x = samp, H = Hns1, deriv.order = 1),
     display = "filled.contour2", main = "NS", abs.cont = cont)
plot(ks::kde(x = samp, H = diag(diag(Hns1)), deriv.order = 1),
     display = "filled.contour2", main = "NS diagonal", abs.cont = cont)
plot(ks::kde(x = samp, H = Hpi1, deriv.order = 1),
     display = "filled.contour2", main = "PI", abs.cont = cont)
plot(ks::kde(x = samp, H = Hpi_diag1, deriv.order = 1),
     display = "filled.contour2", main = "PI diagonal", abs.cont = cont)
```

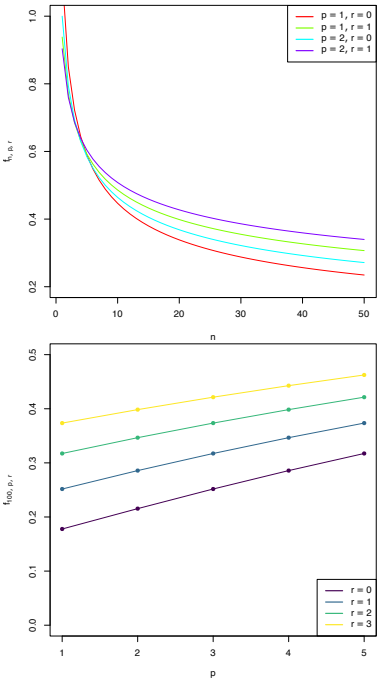
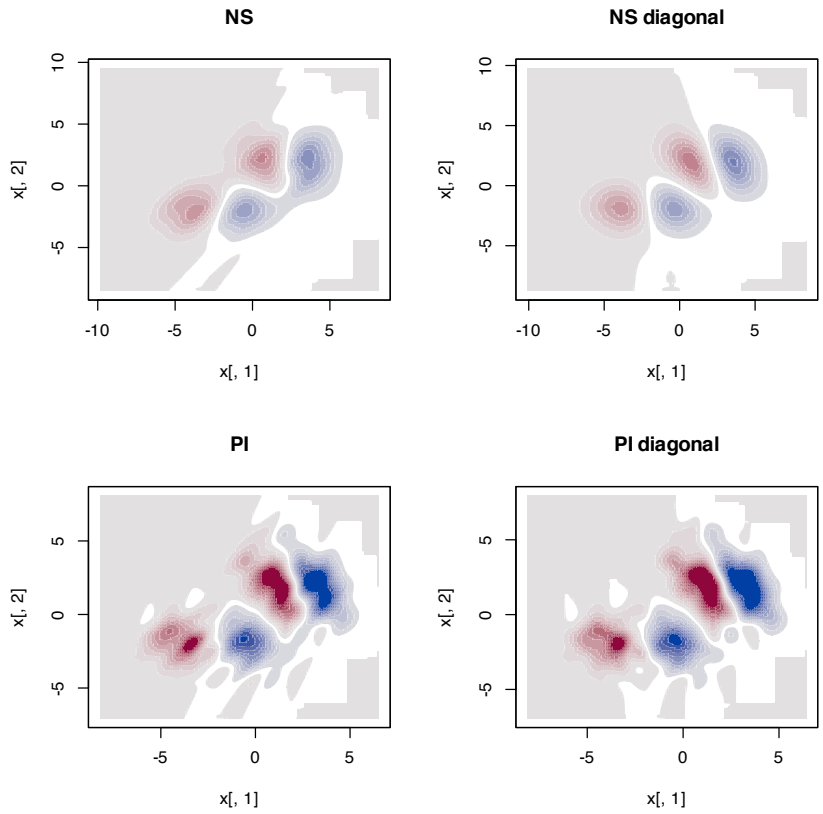


Figure 3.3: Graphs of $n \mapsto f_{n,p,r}$ (for varying p and r) and $p \mapsto f_{n,p,r}$ (for $n = 100$ and varying r).



Exercise 3.10. The `data(sunspots_births, package = "rotasym")` dataset contains the recorded sunspots births during 1872–2018 from the Debrecen Photoheliographic Data (DPD) catalog. The dataset presents 51,303 sunspot records, featuring their positions in spherical coordinates (`theta` and `phi`), sizes (`total_area`), and distances to the center of the solar disk (`dist_sun_disc`).

a. Compute and plot the kde for `phi` using the DPI selector. Describe the result.
b. Compute and plot the kernel density derivative estimator for `phi` using the adequate DPI selector. Determine approximately the location of the main mode(s).
c. Compute the log-transformed kde with `adj.positive = 1` for `total_area` using the NS selector.
d. Draw the histogram of $M = 10,000$ samples simulated from the kde obtained in a.

3.4.2 *Cross-validation*

The **Least Squares Cross-Validation** (LSCV) selector directly extends from the univariate case and attempts to minimize $\text{MISE}[\hat{f}(\cdot;\mathbf{H})]$ by estimating it unbiasedly with

$$\text{LSCV}(\mathbf{H}) := \int \hat{f}(\mathbf{x};\mathbf{H})^2\,\mathrm{d}\mathbf{x} - 2n^{-1}\sum_{i=1}^{n}\hat{f}_{-i}(\mathbf{X}_i;\mathbf{H})$$

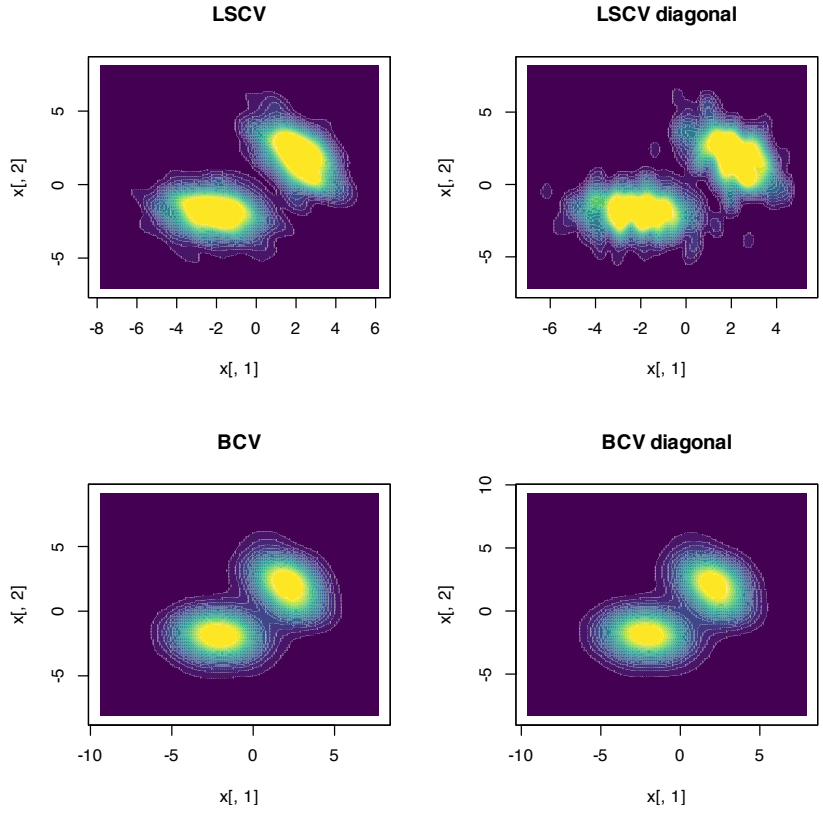
and then minimizing that loss:

$$\hat{\mathbf{H}}_{\text{LSCV}} := \arg\min_{\mathbf{H}\in\text{SPD}_p} \text{LSCV}(\mathbf{H}).$$

Biased Cross-Validation (BCV) combines the cross-validation ideas from $\hat{\mathbf{H}}_{\text{LSCV}}$ with the plug-in ideas behind $\hat{\mathbf{H}}_{\text{PI}}$ to build, in a way analogous to the univariate case (see (2.31)), the $\hat{\mathbf{H}}_{\text{BCV}}$ selector. This selector trades the unbiasedness of $\hat{\mathbf{H}}_{\text{LSCV}}$ for a reduction in its variance.

Cross-validatory selectors, allowing for full or diagonal bandwidth matrices, are implemented by `ks::Hlscv` and `ks::Hlscv.diag` (LSCV), and `ks::Hbcv` and `ks::Hbcv.diag` (BCV). The following chunk of code shows how to use them.

```
# LSCV bandwidth unconstrained
Hlscv <- ks::Hlscv(x = samp)

# LSCV bandwidth diagonal
Hlscv_diag <- ks::Hlscv.diag(x = samp)

# BCV bandwidth unconstrained
Hbcv <- ks::Hbcv(x = samp)

# BCV bandwidth diagonal
Hbcv_diag <- ks::Hbcv.diag(x = samp)

# Compare kdes
par(mfrow = c(2, 2))
cont <- seq(0, 0.03, l = 20)
col <- viridis::viridis
plot(ks::kde(x = samp, H = Hlscv), display = "filled.contour2",
     abs.cont = cont, col.fun = col, main = "LSCV")
plot(ks::kde(x = samp, H = Hlscv_diag), display = "filled.contour2",
     abs.cont = cont, col.fun = col, main = "LSCV diagonal")
plot(ks::kde(x = samp, H = Hbcv), display = "filled.contour2",
     abs.cont = cont, col.fun = col, main = "BCV")
plot(ks::kde(x = samp, H = Hbcv_diag), display = "filled.contour2",
     abs.cont = cont, col.fun = col, main = "BCV diagonal")
```



Exercise 3.11. Consider the normal mixture

$$w\mathcal{N}_2(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) + (1-w)\mathcal{N}_2(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2),$$

where $w = 0.3$, $\boldsymbol{\mu}_1 = (1,1)'$, $\boldsymbol{\mu}_2 = (-1,-1)'$, $\boldsymbol{\Sigma}_i = \begin{pmatrix} \sigma_{i1}^2 & \sigma_{i1}\sigma_{i2}\rho_i \\ \sigma_{i1}\sigma_{i2}\rho_i & \sigma_{i2}^2 \end{pmatrix}$, $i = 1,2$, and $\sigma_{11}^2 = \sigma_{21}^2 = 1$, $\sigma_{12}^2 = \sigma_{22}^2 = 2$, $\rho_1 = 0.5$, and $\rho_2 = -0.5$.

Perform the following simulation exercise:

1. Plot the density of the mixture using `ks::dmvnorm.mixt` and overlay points simulated employing `ks::rmvnorm.mixt`. You may want to use `ks::contourLevels` to produce density plots comparable to the kde plots performed in the next step.
2. Compute the kde employing $\hat{\mathbf{H}}_{\text{PI}}$, both for full and diagonal bandwidth matrices. Are there any gains on considering full bandwidths? What if $\rho_2 = 0.7$?
3. Consider the previous point with $\hat{\mathbf{H}}_{\text{LSCV}}$ instead of $\hat{\mathbf{H}}_{\text{PI}}$. Are the conclusions the same?

3.5 *Applications of kernel density estimation*

Once we are able to adequately estimate the multivariate density f of a random vector $\mathbf{X}$ by $\hat{f}(\cdot; \mathbf{H})$, we can perform a series of interesting applications that go beyond the mere visualization and graphical description of the estimated density. These applications are intimately related to multivariate analysis methods rooted on the *normality assumption*:

- **Density level set estimation**. The level set of the density f at level $c \geq 0$ is defined as $\mathcal{L}(f; c) := \{\mathbf{x} \in \mathbb{R}^p : f(\mathbf{x}) \geq c\}$. An estimation of $\mathcal{L}(f; c)$ is useful to visualize the highest density regions, which give a concise view of the most likely values for $\mathbf{X}$ and therefore summarize the structure of $\mathbf{X}$. In addition, this summary is produced without needing to restrict to connected sets,[24] as the *box plot*[25] does,[26] and with the additional benefit of determining the approximate probability contained in them. Level sets also are useful to estimate the support of $\mathbf{X}$ and to detect multivariate outliers without the need to employ the *Mahalanobis distance*.[27]
- **Clustering or unsupervised learning**. The aim of clustering techniques, such as *hierarchical clustering* or *k-means*,[28] is to find homogeneous *clusters* (subgroups) within the sample (group). Despite their usefulness, these two techniques have important drawbacks: they do *not* automatically determine the number of clusters and they lack a simple interpretation in terms of the population, relying entirely on the sample for its construction and performance evaluation. Based on a density f, the *population clusters* of the domain of $\mathbf{X}$ can be precisely defined as the "domains of attraction of the density modes". The *mean shift clustering* algorithm replaces f with $\hat{f}(\cdot; \mathbf{H})$ and gives a neat clustering recipe that automatically determines the number of clusters.

[24] Such as, e.g., intervals in $\mathbb{R}$.

[25] The kde is not exogenous to box plots. Several kde-based variants of box plots have been produced, such as the popular violin plots by Hintze and Nelson (1998); see Benjamini (1988) and Wickham and Stryjewski (2011).

[26] The construction of the box plot by Tukey (1977) seems to be influenced by the normal distribution. For a $\mathcal{N}(0, \sigma^2)$, the "1.5 × IQR rule" that flags an observation as an outlier has a special interpretation. Since $1.5 \times \text{IQR} = 1.5 \times (\Phi^{-1}(0.75) - \Phi^{-1}(0.25))\sigma \approx 2.0235\sigma$, an observation is flagged as an outlier if it is below $\text{Q1} - 1.5 \times \text{IQR} = \Phi^{-1}(0.25)\sigma - 2.0235\sigma \approx -2.6980\sigma$ or above $\text{Q3} + 1.5 \times \text{IQR} \approx 2.6980\sigma$. This is slightly more liberal than the "3σ rule" that accounts for 99.73% of the probability, which seems too restrictive for outlier-hunting. Tukey's 1.5 × IQR rule precisely determines that 99.3% of the points are *not* outliers. The closeness of this percentage to 99% might explain the magic behind the 1.5 factor.

[27] Which is based on the assumption that $\mathbf{X} \sim \mathcal{N}_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

[28] Which can be seen as a special case of fitting a mixture of k normals with isotropic covariance matrices $\boldsymbol{\Sigma} = c\mathbf{I}_p$, $c > 0$.

- **Classification or supervised learning**. Suppose $\mathbf{X}$ has a categorical random variable Y as a companion, the *labels* of which indicate several classes within the population $\mathbf{X}$. Then, the task of assigning a point $\mathbf{x} \in \mathbb{R}^p$ to a class of Y, based on a sample $(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)$, is a classification problem. Classic multivariate techniques such as *Linear Discriminant Analysis* (LDA) and *Quadratic Discriminant Analysis* (QDA)[29] tackle this task by confronting normals associated with each class, say $\mathcal{N}_p(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}_p(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, and then assigning $\mathbf{x} \in \mathbb{R}^p$ to the class that maximizes $\phi_{\hat{\boldsymbol{\Sigma}}_i}(\mathbf{x} - \hat{\boldsymbol{\mu}}_i)$, $i = 1, 2$. The very same principle can be followed by replacing the estimated normal densities with the kdes of each class.
- **Description of the main features of the data**. *Principal Component Analysis* (PCA) is a very well-known linear dimension-reduction technique that is closely related to the normal distribution.[30] PCA helps to describe the main features of a dataset by estimating the *principal directions* on $\mathbb{R}^p$, $p \geq 2$, of the maximum projected variance of $\mathbf{X}$. These principal directions are highly related to the *density ridges* of $\phi_{\boldsymbol{\Sigma}}(\cdot - \boldsymbol{\mu})$, the latter concept generating flexible principal *curves* for any density f. As a consequence, density ridges of $\hat{f}(\cdot; \mathbf{H})$ give a flexible nonlinear description of the structure of the data.

[29] The difference between LDA and QDA is that the former assumes $\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2$ when confronting $\mathcal{N}_p(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ against $\mathcal{N}_p(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, whereas the latter does not. As a consequence, LDA produces a separating hyperplane between both classes, whereas QDA produces a separating quadric surface (such as a hyperboloid or paraboloid in $\mathbb{R}^3$).

[30] Indeed, the principal components can be regarded as the maximum likelihood estimators of the eigenvectors of $\boldsymbol{\Sigma}$ when $\mathbf{X} \sim \mathcal{N}_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

Next, we review the details associated with these applications, which are excellently described in Chapters 6 and 7 in Chacón and Duong (2018).

3.5.1 *Level set estimation*

The estimation of the level set $\mathcal{L}(f; c) = \{\mathbf{x} \in \mathbb{R}^p : f(\mathbf{x}) \geq c\}$, useful to determine high-density regions, can be straightforwardly done by plugging the kde of f and considering

$$\mathcal{L}(\hat{f}(\cdot; \mathbf{H}); c) = \{\mathbf{x} \in \mathbb{R}^p : \hat{f}(\mathbf{x}; \mathbf{H}) \geq c\}.$$

Obtaining the representation of $\mathcal{L}(\hat{f}(\cdot; \mathbf{H}); c)$ in practice involves considering a grid in $\mathbb{R}^p$ in order to evaluate the condition $\hat{f}(\mathbf{x}; \mathbf{H}) \geq c$ and to determine the region of $\mathbb{R}^p$ in which it is satisfied.

We illustrate the computation of $\mathcal{L}(\hat{f}(\cdot; \mathbf{H}); c)$ for a couple of simulated examples in $\mathbb{R}$. Notice, in the code below, how the function `kde_level_set` automatically deals with the annoyance that $\mathcal{L}(\hat{f}(\cdot; \mathbf{H}); c)$ **may not be a connected region**, that is, that $\mathcal{L}(\hat{f}(\cdot; \mathbf{H}); c) = \bigcup_{i=1}^{k} [a_i, b_i]$ with k unknown beforehand.

```
# Simulated sample
n <- 100
set.seed(12345)
samp <- rnorm(n = n)

# Kde as usual, but force to evaluate it at seq(-4, 4, length = 4096)
bw <- bw.nrd(x = samp)
kde <- density(x = samp, bw = bw, n = 4096, from = -4, to = 4)
```

```
# For a given c, what is the theoretical level set? Since we know that the
# real density is symmetric and unimodal, then the level set is an interval
# of the form [-x_c, x_c]
c <- 0.2
x_c <- tryCatch(uniroot(function(x) dnorm(x) - c, lower = 0, upper = 4)$root,
                error = function(e) NA)

# Show theoretical level set
x <- seq(-4, 4, by = 0.01)
plot(x, dnorm(x), type = "l", ylim = c(0, 0.5), ylab = "Density")
rug(samp)
polygon(x = c(-x_c, -x_c, x_c, x_c), y = c(0, c, c, 0),
        col = rgb(0, 0, 0, alpha = 0.5), density = 10)

# Function to compute and plot a kde level set. Observe that kde stands for an
# object containing the output of density(), although obvious modifications
# could be done to the function to receive a ks::kde object
# as the main argument
kde_level_set <- function(kde, c, add_plot = FALSE, ...) {

  # Begin and end index for the potentially many intervals in the level sets
  # of the kde
  kde_larger_c <- kde$y >= c
  run_length_kde <- rle(kde_larger_c) # Trick to compute the length of the
  # sequence of TRUEs that indicates an interval for which kde$y >= c
  begin <- which(diff(kde_larger_c) > 0) + 1 # Trick to search for the beginning
  # of each of the intervals
  end <- begin + run_length_kde$lengths[run_length_kde$values] - 1 # Compute
  # the end of the intervals from begin + length

  # Add polygons to a density plot? If so, ... are the additional parameters
  # for polygon()
  if (add_plot) {

    apply(cbind(begin, end), 1, function(ind) {
      polygon(x = c(kde$x[ind[1]], kde$x[ind[1]],
                    kde$x[ind[2]], kde$x[ind[2]]),
              y = c(0, kde$y[ind[1]],
                    kde$y[ind[2]], 0), ...)
      })

  }

  # Return the [a_i, b_i], i = 1, ..., K in the K rows
  return(cbind(kde$x[begin], kde$x[end]))

}

# Add kde and level set
lines(kde, col = 2)
kde_level_set(kde = kde, c = c, add_plot = TRUE,
              col = rgb(1, 0, 0, alpha = 0.5))
##          [,1]     [,2]
## [1,] -1.01685 1.444689
abline(h = c, col = 4) # Level
legend("topright", legend = c("True density", "Kde", "True level set",
                              "Kde level set", "Level c"),
       lwd = 2, col = c(1, 2, rgb(0:1, 0, 0, alpha = 0.5), 4))
```

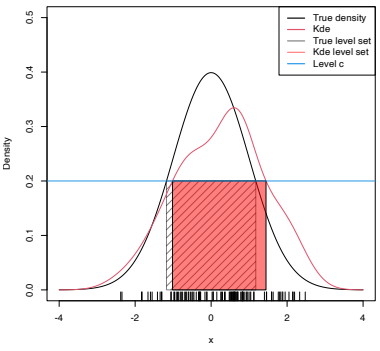


Figure 3.4: Level set $\mathcal{L}(f;c)$ and its estimation by $\mathcal{L}(\hat{f}(\cdot;h);c)$ for $c = 0.2$ and $f = \phi$.

The following code chunk illustrates how changing the bandwidth h and the level of the set c affects the connectedness of $\mathcal{L}(\hat{f}(\cdot;h);c)$.

```
# Simulated sample
n <- 100
set.seed(12345)
samp <- rnorm(n = n)
```

```
# Interactive visualization
x <- seq(-4, 4, by = 0.01)
manipulate::manipulate({

  # Show theoretical level set
  plot(x, dnorm(x), type = "l", ylim = c(0, 0.5), ylab = "Density")
  rug(samp)
  x_c <- tryCatch(uniroot(function(x) dnorm(x) - c, lower = 0, upper = 4)$root,
                  error = function(e) NA) # tryCatch() to bypass errors
  polygon(x = c(-x_c, -x_c, x_c, x_c), y = c(0, c, c, 0),
          col = rgb(0, 0, 0, alpha = 0.5), density = 10)

  # Add estimation
  kde <- density(x = samp, bw = bw, n = 1e5, from = -4, to = 4)
  lines(kde, col = 2)
  kde_level_set(kde = kde, c = c, add_plot = TRUE,
                col = rgb(1, 0, 0, alpha = 0.5))
  abline(h = c, col = 4) # Level
  legend("topright", legend = c("True density", "Kde", "True level set",
                                "Kde level set", "Level c"),
         lwd = 2, col = c(1, 2, rgb(0:1, 0, 0, alpha = 0.5), 4))

}, c = manipulate::slider(min = 0.01, max = 0.5, initial = 0.2, step = 0.01),
bw = manipulate::slider(min = 0.01, max = 1, initial = 0.25, step = 0.01))
```

Exercise 3.12. Consider the *bimodal* density f given in `nor1mix::MW.nm6`. Consider $c = 0.15, 0.25$.

- Compute $\mathcal{L}(f; c)$.
- From a sample of size $n = 200$, compute $\mathcal{L}(\hat{f}(\cdot; \hat{h}_{\text{DPI}}); c)$.

Highest density regions for a given probability

The level c in $\mathcal{L}(f; c) = \{\mathbf{x} \in \mathbb{R}^p : f(\mathbf{x}) \geq c\}$ may be difficult to interpret: its effective scale depends on the dimension[31] p of the random vector $\mathbf{X}$ and on the units in which their components are measured. A more convenient parametrization of the level set is attained if we state the *probability* that is intended to contain, rather than the minimum value of the density attained in the region. For that purpose, it is usually considered the *largest*[32] c_α such that

$$\int_{\mathcal{L}(f; c_\alpha)} f(\mathbf{x}) \, \mathrm{d}\mathbf{x} \geq 1 - \alpha, \quad \alpha \in (0, 1).$$

This formulation raises a key interpretation of $\mathcal{L}(f; c_\alpha)$:

> $\mathcal{L}(f; c_\alpha)$ is the ***smallest***[33] **region** of $\mathbb{R}^p$ that **contains at least** $1 - \alpha$ **of the probability** of $\mathbf{X}$.

As defined, c_α depends on f, which is of course unknown. However, it can be estimated in a very elegant and computationally efficient way. To see it, first recall that

$$\begin{aligned} \int_{\mathcal{L}(f; c_\alpha)} f(\mathbf{x}) \, \mathrm{d}\mathbf{x} &= \mathbb{P}[\mathbf{X} \in \mathcal{L}(f; c_\alpha)] \\ &= \mathbb{P}[f(\mathbf{X}) \geq c_\alpha] \geq 1 - \alpha, \end{aligned} \tag{3.19}$$

which amounts to find the largest c_α such that $\mathbb{P}[f(\mathbf{X}) \leq c_\alpha] \leq \alpha$. Therefore,

> c_α is precisely the lower α-quantile of the random variable $f(\mathbf{X})$!

[31] Since $\int f(\mathbf{x}) \, \mathrm{d}\mathbf{x} = 1$, the scale for a pdf f decreases as the dimension p increases, as the volume in $\mathbb{R}^p$ grows exponentially with p.

[32] We consider the *largest* c_α such that $\int_{\mathcal{L}(f; c_\alpha)} f(\mathbf{x}) \, \mathrm{d}\mathbf{x} \geq 1 - \alpha$, instead of *the* c_α such that $\int_{\mathcal{L}(f; c_\alpha)} f(\mathbf{x}) \, \mathrm{d}\mathbf{x} = 1 - \alpha$, because the function $c \in [0, \infty) \mapsto \int_{\mathcal{L}(f; c)} f(\mathbf{x}) \, \mathrm{d}\mathbf{x} \in [0, 1]$ may not be continuous: think about $f(x) = 1_{\{0 < x < 1\}}$ and $c = 1$. What is c_α for $\alpha = 1/2$ in this case?

[33] Smallest in terms of volume in $\mathbb{R}^p$. Since f is measuring the density of probability per volume in $\mathbb{R}^p$, the highest density regions minimize the amount of volume required for allocating at least $1 - \alpha$ probability. If $p = 1$, then the union of intervals that conform $\mathcal{L}(f; c_\alpha)$ has the shortest length possible to allocate $1 - \alpha$ probability.

From this insight, it follows that, if we knew f, then from the sample $\mathbf{X}_1,\dots,\mathbf{X}_n$ we could estimate c_α by the (lower) sample α-quantile of $f(\mathbf{X}_1),\dots,f(\mathbf{X}_n)$. Since this is not the case, we can opt to replace f with the kde and define

$\hat{c}_\alpha$ as the sample α-quantile of $\hat{f}(\mathbf{X}_1;\mathbf{H}),\dots,\hat{f}(\mathbf{X}_n;\mathbf{H})$.

Then, the estimator of $\mathcal{L}(f;c_\alpha)$, the highest density region that accumulates $1-\alpha$ of the probability, follows by considering the kde twice, one for obtaining $\hat{c}_\alpha$ and another for replacing f with $\hat{f}(\cdot;\mathbf{H})$, resulting in $\mathcal{L}(\hat{f}(\cdot;\mathbf{H}),\hat{c}_\alpha)$.

The following code illustrates how to estimate $\mathcal{L}(f,c_\alpha)$ by $\mathcal{L}(\hat{f}(\cdot;\mathbf{H}),\hat{c}_\alpha)$. Since we need to evaluate the kde at $\mathbf{X}_1,\dots,\mathbf{X}_n$, we employ `ks::kde` instead of `density`, as the latter allows evaluating the kde at a uniformly spaced grid only.

```
# Simulate sample
n <- 200
set.seed(12345)
samp <- rnorm(n = n)

# We want to estimate the highest density region containing 0.75 probability
alpha <- 0.25

# For the N(0, 1), we know that this region is the interval [-x_c, x_c] with
x_c <- qnorm(1 - alpha / 2)
c_alpha <- dnorm(x_c)
c_alpha
## [1] 0.2058535
# This corresponds to the c_alpha

# Theoretical level set
x <- seq(-4, 4, by = 0.01)
plot(x, dnorm(x), type = "l", ylim = c(0, 0.5), ylab = "Density")
rug(samp)
polygon(x = c(-x_c, -x_c, x_c, x_c), y = c(0, c_alpha, c_alpha, 0),
        col = rgb(0, 0, 0, alpha = 0.5), density = 10)
abline(h = c_alpha, col = 3, lty = 2) # Level

# Kde
bw <- bw.nrd(x = samp)
c_alpha_hat <- quantile(ks::kde(x = samp, h = bw, eval.points = samp)$estimate,
                        probs = alpha)
c_alpha_hat
##       25%
## 0.1838304
kde <- density(x = samp, bw = bw, n = 4096, from = -4, to = 4)
lines(kde, col = 2)
kde_level_set(kde = kde, c = c_alpha_hat, add_plot = TRUE,
              col = rgb(1, 0, 0, alpha = 0.5))
##         [,1]     [,2]
## [1,] -1.2337 1.378266
abline(h = c_alpha_hat, col = 4, lty = 2) # Level
legend("topright", legend = expression("True density", "Kde", "True level set",
                                       "Kde level set", "Level " * c[alpha],
                                       "Level " * hat(c)[alpha]),
       lwd = 2, col = c(1, 2, rgb(0:1, 0, 0, alpha = 0.5), 3:4),
       lty = c(rep(1, 4), rep(2, 4)))
```

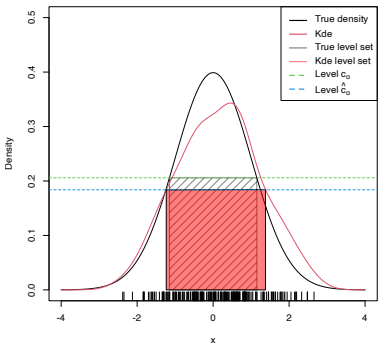


Figure 3.5: Level set $\mathcal{L}(f;c_\alpha)$ and its estimation by $\mathcal{L}(\hat{f}(\cdot;h);\hat{c}_\alpha)$ for $\alpha=0.25$ and $f=\phi$.

Observe that equation (3.19) points towards a simple way of approximating the multidimensional integral $\int_{\mathcal{L}(f;c)} f(\mathbf{x})\,\mathrm{d}\mathbf{x}$ through-

out a sample $\mathbf{X}_1, \ldots, \mathbf{X}_n$ of f:

$$\int_{\mathcal{L}(f;c)} f(\mathbf{x})\, d\mathbf{x} = \mathbb{P}[f(\mathbf{X}) \geq c] \approx \frac{1}{n} \sum_{i=1}^{n} 1_{\{f(\mathbf{X}_i) \geq c\}}. \tag{3.20}$$

This Monte Carlo method has the important advantage of avoiding an expensive numerical integration. This is precisely the same advantage behind the approximation of c_α (essentially, such that $\int_{\mathcal{L}(f;c_\alpha)} f(\mathbf{x})\, d\mathbf{x} = 1 - \alpha$ for $\alpha \in (0,1)$) by the α-quantile of $f(\mathbf{X}_1), \ldots, f(\mathbf{X}_n)$.

The application of (3.20) is illustrated below.

```
# N(0, 1) case
alpha <- 0.3
x_c <- qnorm(1 - alpha / 2)
c_alpha <- dnorm(x_c)
c_alpha
## [1] 0.2331588

# Approximates c_alpha
quantile(dnorm(samp), probs = alpha)
##       30%
## 0.2043856

# True integral: 1 - alpha (by construction)
1 - 2 * pnorm(-x_c)
## [1] 0.7

# Monte Carlo integration, approximates 1 - alpha
mean(dnorm(samp) >= c_alpha)
## [1] 0.655
```

Exercise 3.13. Consider the kde for the `faithful$eruptions` dataset with a DPI bandwidth.

- What is the estimation of the shortest set that contains the 50% of the probability? And the 75%?
- Compare the results obtained with the application of a `boxplot`. Which graphical analysis do you think is more informative?

Exercise 3.14. Repeat Exercise 3.13 with the data `airquality$Wind`.

Exercise 3.15. Section 2F in Tukey (1977) presents "the Rayleigh example" as an illustration of the weakness of box plots (referred to as "schematic plots" in Tukey's terminology) for analyzing bimodal data. The weights of nitrogen obtained by Lord Rayleigh (table in page 49 in Tukey (1977)) are:

```
x <- c(2.30143, 2.29816, 2.30182, 2.29890, 2.31017,
       2.30986, 2.31010, 2.31001, 2.29889, 2.29940,
       2.29849, 2.29889, 2.31024, 2.31030, 2.31028)
```

a. Obtain the box plot of `x`. Do you see any interesting insights?
b. Compute $\hat{h}_{\text{LSCV}}$ (beware of local minima) and draw a kde based on this bandwidth.
c. Estimate and plot the shortest sets containing 50% and 75% of the probability. What are your insights?

Exercise 3.16. Repeat Exercise 3.12 but now considering $\alpha = 0.25, 0.50$ and obtaining the corresponding c_α and $\hat{c}_\alpha$. Compute c_α by Monte Carlo from (3.20) using $M = 10,000$ replications.

Exercise 3.17. Adapt the function `kde_level_set` to:

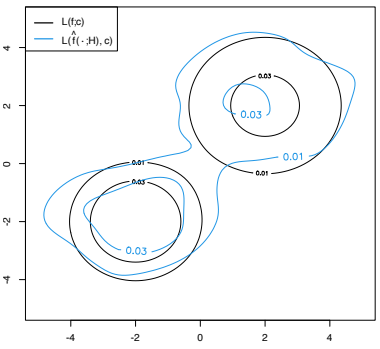
a. Receive as main arguments a sample `x` and a bandwidth `h`, instead of a `density` object.
b. Use internally `ks::kde` instead of `density`.
c. Receive an α, instead of c, and internally determine $\hat{c}_\alpha$.
d. Retain the functionality of `kde_level_set` and return also `c_alpha_hat`.

Call this new function `kde_level_set2` and validate it with the examples seen so far.

Bivariate and trivariate level sets

The computation of bivariate and trivariate level sets is conceptually similar, but their parametrization and graphical display are more challenging. In $\mathbb{R}^2$, the level sets are simply obtained from the contour levels of $\hat{f}(\cdot; \mathbf{H})$, which is done easily with the `ks` package.

```
# Simulated sample from a mixture of normals
n <- 200
set.seed(123456)
mu <- c(2, 2)
Sigma1 <- diag(c(2, 2))
Sigma2 <- diag(c(1, 1))
samp <- rbind(mvtnorm::rmvnorm(n = n / 2, mean = mu, sigma = Sigma1),
              mvtnorm::rmvnorm(n = n / 2, mean = -mu, sigma = Sigma2))

# Level set of the true density at levels c
c <- c(0.01, 0.03)
x <- seq(-5, 5, by = 0.1)
xx <- as.matrix(expand.grid(x, x))
contour(x, x, 0.5 * matrix(mvtnorm::dmvnorm(xx, mean = mu, sigma = Sigma1) +
                             mvtnorm::dmvnorm(xx, mean = -mu, sigma = Sigma2),
                           nrow = length(x), ncol = length(x)),
        levels = c)

# Plot of the contour level
H <- ks::Hpi(x = samp)
kde <- ks::kde(x = samp, H = H)
plot(kde, display = "slice", abs.cont = c, add = TRUE, col = 4) # Argument
# "abs.cont" for specifying c rather than (1 - alpha) * 100 in "cont"
legend("topleft", lwd = 2, col = c(1, 4),
       legend = expression(L * "(" * f * ";" * c * ")",
                           L * "(" * hat(f) * "(" %.% ";" * H * "), " * c *")"))
```

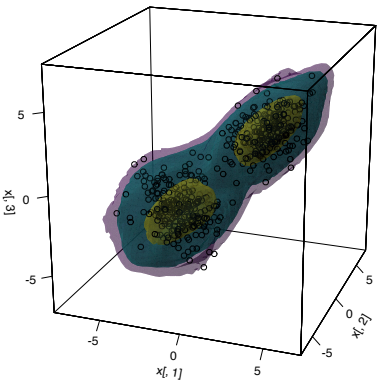


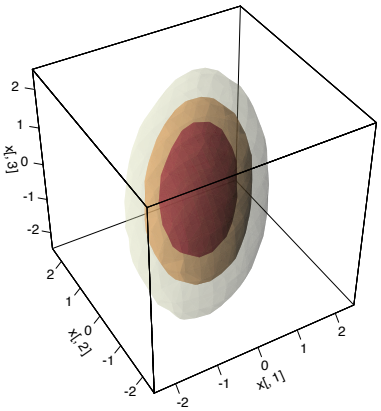

```
# Computation of the probability accumulated in the level sets by numerical
# integration
ks::contourSizes(kde, abs.cont = c)
## [1] 35.865815  7.321872
```

Exercise 3.18. Compute the level set containing the 50% of the data of the following bivariate datasets: `faithful`, `data(unicef, package = "ks")`, and `iris[, 1:2]`.

In $\mathbb{R}^3$, the level sets are more challenging to visualize, as it is required to combine three-dimensional contours and the use of transparencies.

```
# Simulate a sample from a mixture of normals
n <- 5e2
set.seed(123456)
mu <- c(2, 2, 2)
Sigma1 <- rbind(c(1, 0.5, 0.2),
                c(0.5, 1, 0.5),
                c(0.2, 0.5, 1))
Sigma2 <- rbind(c(1, 0.25, -0.5),
                c(0.25, 1, 0.5),
                c(-0.5, 0.5, 1))
samp <- rbind(mvtnorm::rmvnorm(n = n / 2, mean = mu, sigma = Sigma1),
              mvtnorm::rmvnorm(n = n / 2, mean = -mu, sigma = Sigma2))

# Plot of the contour level, changing the color palette
H <- ks::Hns(x = samp)
kde <- ks::kde(x = samp, H = H)
plot(kde, cont = 100 * c(0.99, 0.95, 0.5), col.fun = viridis::viridis,
     drawpoints = TRUE, col.pt = 1, theta = 20, phi = 20)

# Simulate a large sample from a single normal
n <- 5e4
set.seed(123456)
mu <- c(0, 0, 0)
Sigma <- rbind(c(1, 0.5, 0.2),
               c(0.5, 1, 0.5),
               c(0.2, 0.5, 1))
samp <- mvtnorm::rmvnorm(n = n, mean = mu, sigma = Sigma)

# Plot of the contour level
H <- ks::Hns(x = samp)
kde <- ks::kde(x = samp, H = H)
plot(kde, cont = 100 * c(0.75, 0.5, 0.25), xlim = c(-2.5, 2.5),
     ylim = c(-2.5, 2.5), zlim = c(-2.5, 2.5))
```





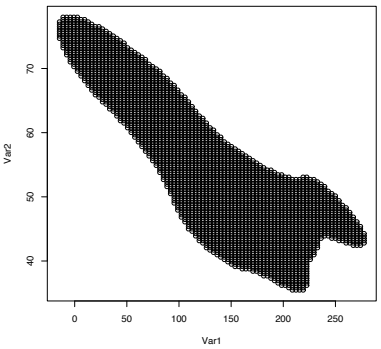
[34] Identifying outlying observations in $\mathbb{R}^p$ is much more challenging than in $\mathbb{R}$, and several statistical techniques have been proposed for that purpose. Indeed, the precise definition of what an outlier is in $\mathbb{R}^p$ admits several approaches. Here we just consider outliers as the points observed in low-density regions.

Support estimation and outlier detection

In addition to the highest density regions that accumulate $1-\alpha$ probability, for a relatively large $\alpha > 0.5$, it is also interesting to set $\alpha \approx 0$, as this gives an estimation of the *effective support* (excluding α probability) of f. The estimated effective support of $\mathbf{X}$ gives valuable graphical insight into the structure of $\mathbf{X}$ and is helpful, for example, to determine the plausible region for data points coming from f and then flag as outliers observations that fall outside the region, for *arbitrary dimension*.[34]

The following code chunk shows how to compute estimates of the support in $\mathbb{R}^2$ via `ks::ksupp`.

```
# Compute kde of unicef dataset
data(unicef, package = "ks")
kde <- ks::kde(x = unicef)

# ks::ksupp evaluates whether the points in the grid spanned by ks::kde belong
# to the level set for alpha = 0.05 and then returns the points that belong to
# the level set (when convex.hull = FALSE)
supp <- as.matrix(ks::ksupp(fhat = kde, cont = 95, convex.hull = FALSE))
plot(supp) # Effective support except for a 5% of data
```



[35] A set C is convex in $\mathbb{R}^p$ if all the paths that interpolate two points $\mathbf{x}, \mathbf{y} \in C$ are contained in C. Mathematically, C is convex if, for all $\mathbf{x}, \mathbf{y} \in C$, $t\mathbf{x} + (1-t)\mathbf{y} \in C$, for all $t \in (0,1)$.

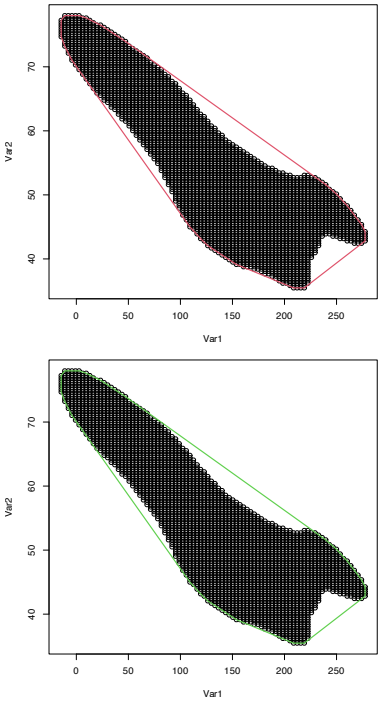
When dealing with sets defined by points on $\mathbb{R}^p$, it is useful to consider the *convex hull* of the set of points, which is defined as the

minor convex[35] set that contains all the points. The convex hull in $\mathbb{R}^2$ is computed with `chull` or by setting `convex.hull = TRUE` in `ks::ksupp`.

```
# The convex hull boundary of the level set can be computed with chull()
# It returns the indexes of the points passed that form the corners of the
# polygon of the convex hull
ch <- chull(supp)
plot(supp)
# One extra point for closing the polygon
lines(supp[c(ch, ch[1]), ], col = 2, lwd = 2)
```

```
# Alternatively, use convex.hull = TRUE (default)
plot(supp)
plot(ks::ksupp(fhat = kde, cont = 95, convex.hull = TRUE),
     border = 3, lwd = 2)
```

```
# The plotting method of ks::ksupp calls to polygon()
```



The convex hull may be seen as a conservative estimate of the effective support, since it may enlarge the level set estimation considerably. Another disadvantage is that it merges the individual components of unconnected supports. However, convex hulls pose interesting advantages: they can be computed in $\mathbb{R}^p$ via convex polyhedrons and then it is possible to check if a given point belongs to them. Thus, they give a handle to parametrize unknown-form level sets in $\mathbb{R}^p$.

These two functionalities are available in the `geometry` package[36] and have an interesting application: rather than evaluating if $\mathbf{x} \in \mathcal{L}(\hat{f}(\cdot;\mathbf{H});c)$, it is possible to *just*[37] check if $\mathbf{x} \in \text{conv}(\{\mathbf{x}_1,\ldots,\mathbf{x}_N\})$, where conv denotes the convex hull operator and $\{\mathbf{x}_1,\ldots,\mathbf{x}_N\}$ is a collection of points that roughly determines $\mathcal{L}(\hat{f}(\cdot;\mathbf{H});c)$. For example, the output of `ks::ksupp` in $\mathbb{R}^2$ or some points that belong to $\mathcal{L}(\hat{f}(\cdot;\mathbf{H});c)$ for $\mathbb{R}^p$, $p \geq 2$.

[36] Since version 0.4.0.

[37] Obviously, the accuracy of this approximation depends on how convex is $\mathcal{L}(\hat{f}(\cdot;\mathbf{H});c)$.

```
# Compute the convex hull of supp via geometry::convhulln()
C <- geometry::convhulln(p = supp)
# The output of geometry::convhulln() is different from chull()

# The geometry::inhulln() allows to check if points are inside the convex hull
geometry::inhulln(ch = C, p = rbind(c(50, 50), c(150, 50)))
## [1] FALSE  TRUE

# The convex hull works as well in R^p. An example in which the level set is
# evaluated by Monte Carlo and then the convex hull of the points in the level
# set is computed

# Sample
set.seed(2134)
samp <- mvtnorm::rmvnorm(n = 1e2, mean = rep(0, 3))

# Evaluation sample: random data in [-3, 3]^3
M <- 1e3
eval_set <- matrix(runif(n = 3 * M, -3, 3), M, 3)

# Kde of samp, evaluated at eval_set
H <- ks::Hns.diag(samp)
kde <- ks::kde(x = samp, H = H, eval.points = eval_set)
```

```
# Convex hull of points in the level set for a given c
c <- 0.01
C <- geometry::convhulln(p = eval_set[kde$estimate > c, ])

# We can test if a new point belongs to the level set by just checking if
# it belongs to the convex hull, which is much more efficient as it avoids
# re-evaluating the kde
new_points <- rbind(c(1, 1, 1), c(2, 2, 2))
geometry::inhulln(ch = C, p = new_points)
## [1]  TRUE FALSE
ks::kde(x = samp, H = H, eval.points = new_points)$estimate > c
## [1]  TRUE FALSE

# # Performance evaluation
# microbenchmark::microbenchmark(
#   geometry::inhulln(ch = C, p = new_points),
#   ks::kde(x = samp, H = H, eval.points = new_points)$estimate > c)
```

Detecting multivariate outliers in $\mathbb{R}^p$ is more challenging than in $\mathbb{R}$, and, unfortunately, the mere inspection of the marginals in the search for extreme values is not enough for successful outlier-hunting. A simple example in $\mathbb{R}^2$ is given in Figure 3.6.

The next exercise shows an application of a level set estimation to outlier detection.

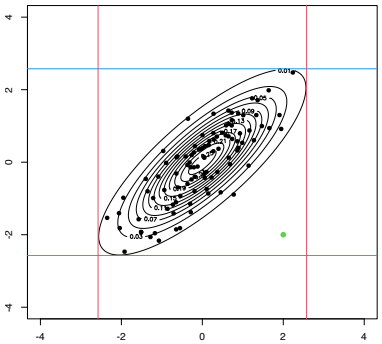


Figure 3.6: An outlier (green point) in $\mathbb{R}^2$ that is not an outlier in any of the two marginals. The contour levels represent the joint density of a $\mathcal{N}_2(\mathbf{0}, \boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma}$ is such that $\sigma_{11}^2 = \sigma_{22}^2 = 1$ and $\sigma_{12} = 0.8$. The black points are sampled from that normal. The red and blue lines represent the $1 - \alpha$ probability intervals $(-z_{\alpha/2}, z_{\alpha/2})$ for $\alpha = 0.01$.

Exercise 3.19. Megaloblastic anemia is a kind of anemia that produces abnormally large red blood cells. Among others, indicators of megaloblastic anemia are low levels of: vitamin B12 (reference values: $178 - 1,100$ pg/ml), Folic Acid (FA; reference values: > 5.4 ng), Mean Corpuscular Volume (MCV; reference values: $78 - 100$ fL), Hemoglobin (H; reference values: $13 - 17$ g/dL), and Iron (I; reference values: $60 - 160$ μg/dL). However, none can diagnose megaloblastic anemia by itself.

The dataset `healthy-patients.txt` contains realistically simulated data from $n_1 = 5,835$ blood analysis that include measurements for $\mathbf{X} = (\text{B12}, \text{FA}, \text{MCV}, \text{H}, \text{I})$, that is, a sample $\mathbf{X}_1, \ldots, \mathbf{X}_{n_1}$. On the other hand, the dataset `new-patients.txt` contains observations of $\mathbf{X}$ for $n_2 = 117$ new patients, for which the presence of megaloblastic anemia is unknown. This sample is denoted by $\tilde{\mathbf{X}}_1, \ldots, \tilde{\mathbf{X}}_{n_2}$.

Our aim is to identify outliers in this second dataset with respect to the first, in order to identify potential manifestations of megaloblastic anemia. To do so, we follow the following steps:

1. Obtain the normal scale bandwidth $\hat{\mathbf{H}}_{\text{NS}}$ for the sample $\mathbf{X}_1, \ldots, \mathbf{X}_{n_1}$.
2. Obtain the level $\hat{c}_\alpha$ such that 0.995 of the estimated probability of $\mathbf{X}$ is contained in $\hat{\mathcal{L}}(\hat{f}(\cdot; \mathbf{H}); \hat{c}_\alpha)$. *Hint:* recall how $\hat{c}_\alpha$ was defined.
3. *Evaluate* the kde $\hat{f}_i(\tilde{\mathbf{X}}_i; \hat{\mathbf{H}}_{\text{NS}})$, $i = 1, \ldots, n_2$, for the new sample $\tilde{\mathbf{X}}_1, \ldots, \tilde{\mathbf{X}}_{n_2}$, where $\hat{f}_i(\cdot; \hat{\mathbf{H}}_{\text{NS}})$ stands for the kde of $\mathbf{X}_1, \ldots, \mathbf{X}_{n_1}, \tilde{\mathbf{X}}_i$.[38]
4. Check if $\hat{f}_i(\tilde{\mathbf{X}}_i; \hat{\mathbf{H}}_{\text{NS}}) < \hat{c}_\alpha$, $i = 1, \ldots, n$, and flag $\tilde{\mathbf{X}}_i$ as an outlier if the condition holds.

Are there outliers? Do the observations flagged as outliers have the measurements inside the reference values?

Exercise 3.20. The `wines.txt` dataset contains chemical analyses of wines grown in Piedmont (Italy) coming from three different

[38] Computing $\hat{f}_i(\tilde{\mathbf{X}}_i; \hat{\mathbf{H}}_{\text{NS}})$ for evaluating the outlyingness of $\tilde{\mathbf{X}}_i$, rather than computing $\hat{f}(\tilde{\mathbf{X}}; \hat{\mathbf{H}}_{\text{NS}})$, is important. In $\hat{f}_i(\tilde{\mathbf{X}}_i; \hat{\mathbf{H}}_{\text{NS}})$, the estimated density at $\tilde{\mathbf{X}}_i$ is computed with $\mathbf{X}_1, \ldots, \mathbf{X}_{n_1}, \tilde{\mathbf{X}}_i$, so "the contribution of $\tilde{\mathbf{X}}_i$ to itself", $\frac{1}{n_1+1} K_{\hat{\mathbf{H}}_{\text{NS}}}(\mathbf{0})$, is present in the kde. Which is exactly the same situation that happens when computing $\hat{f}(\mathbf{X}_1; \hat{\mathbf{H}}_{\text{NS}}), \ldots, \hat{f}(\mathbf{X}_{n_1}; \hat{\mathbf{H}}_{\text{NS}})$ for estimating $\hat{c}_\alpha$, and so the comparison $\hat{f}_i(\tilde{\mathbf{X}}_i; \hat{\mathbf{H}}_{\text{NS}}) < \hat{c}_\alpha$ is "fair".

vintages: Nebbiolo, Barberas, and Grignolino. Perform a PCA after removing the `vintage` variable and standardizing the variables.

a. Considering the three first PCs, what is the smallest set that contains the 90% of the probability? Represent the set graphically.
b. Is the point $\mathbf{x}_1 = (0, -1.5, 0)$ inside the level set? Could it be considered an outlier (remember the previous exercise)?
c. What about $\mathbf{x}_2 = (2, -1.5, 2)$?
d. And what about $\mathbf{x}_3 = (3, -3, 3)$?
e. Visualize the locations of $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ to interpret the previous results

We conclude the section by highlighting a useful fact about the computation of areas under normal densities that is helpful to determine exactly the density levels for given probabilities. The density level c_α that contains $1-\alpha$ probability for $\mathbf{X} \sim \mathcal{N}_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ can be computed exactly by

$$1-\alpha = \mathbb{P}[\phi_{\boldsymbol{\Sigma}}(\mathbf{X}-\boldsymbol{\mu}) \geq c] = \mathbb{P}\left[\chi^2_p \leq -2\log\left(|\boldsymbol{\Sigma}|^{1/2}(2\pi)^{p/2}c\right)\right],$$

since $(\mathbf{X}-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{X}-\boldsymbol{\mu}) \sim \chi^2_p$. Therefore, for a given $\alpha \in (0,1)$

$$c_\alpha = |\boldsymbol{\Sigma}|^{-1/2}(2\pi)^{-p/2}e^{-\chi^2_{p;\alpha}/2}, \tag{3.21}$$

where $\chi^2_{p;\alpha}$ is the α-upper quantile of a χ^2_p. The following code shows the computation of (3.21).

```
alpha <- 0.4
p <- 2
c_alpha <- exp(-0.5 * qchisq(p = 1 - alpha, df = p)) /
  (sqrt(det(Sigma)) * (2 * pi)^(p / 2))
```

3.5.2 *Mean shift clustering*

Perhaps one of the best-known clustering methods is *k-means*. Given a sample $\mathbf{X}_1, \ldots, \mathbf{X}_n$ in $\mathbb{R}^p$, k-means is defined as the problem of finding the clusters $C_1, \ldots, C_k$ such that the *within-cluster variation* is as small as possible:

$$\min_{C_1,\ldots,C_k} \left\{ \sum_{j=1}^{k} \mathrm{W}(C_j) \right\}$$

where the within-cluster variation[39] of C_j is defined as

$$\mathrm{W}(C_j) := \frac{1}{|C_j|} \sum_{i,i' \in C_j} \|\mathbf{X}_i - \mathbf{X}_{i'}\|^2$$

with $|C_j| := \#\{i = 1, \ldots, n : \mathbf{X}_i \in C_j\}$ denoting the number of observations in the j-th cluster.

Behind the usefulness and simplicity of k-means hides an important drawback inherent to many clustering techniques: it is a **method mostly defined on a sample basis**[40] and for which a population analogue (in terms of the distribution of $\mathbf{X}$) is not clearly

[39] Notice that $W(C_j)$ can be seen as the sample *variance*, since $W(C_j) = \frac{1}{|C_j|}\sum_{i,i' \in C_j} \|\mathbf{X}_i - \bar{\mathbf{X}}_{C_j} + \bar{\mathbf{X}}_{C_j} - \mathbf{X}_{i'}\|^2 = \frac{2}{|C_j|}\sum_{i \in C_j} \|\mathbf{X}_i - \bar{\mathbf{X}}_{C_j}\|^2$, where $\bar{\mathbf{X}}_{C_j} := \frac{1}{|C_j|}\sum_{i \in C_j} \mathbf{X}_i$ is the sample mean of the observations in C_j. Therefore, $W(C_j) = 2S^2_{C_j}$, where $S^2_{C_j}$ is the trace of the sample covariance matrix $\frac{1}{|C_j|}\sum_{i \in C_j}(\mathbf{X}_i - \bar{\mathbf{X}}_{C_j})(\mathbf{X}_i - \bar{\mathbf{X}}_{C_j})'$.

[40] It is possible to build a population formulation of k-means. That can be seen, e.g., in Cuesta-Albertos et al. (1997), a paper that introduces a trimmed variant of k-means.

[41] The population view of k-means is conceptually more involved and less insightful than, as seen subsequently, mode clustering. The latter has an elegant simple connection with the (population) concept of mode.

[42] For example, what are the clusters of the "claw" density `nor1mix::MW.nm10`?

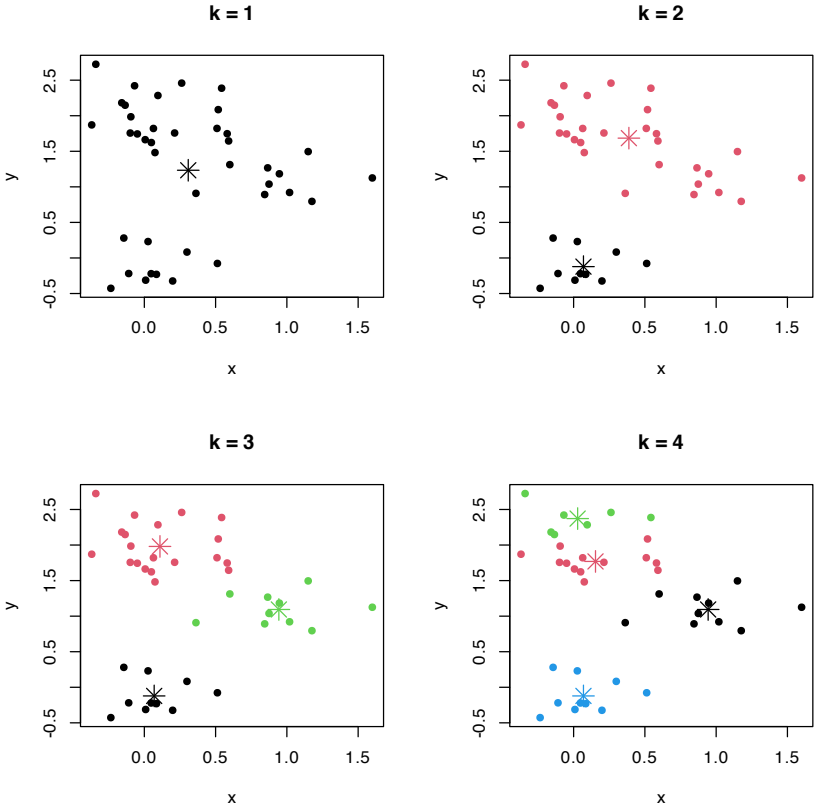


Figure 3.7: k-means partitions for a two-dimensional dataset with $k = 1, 2, 3, 4$. The center of each cluster is displayed with an asterisk.

evident.[41] This has an important consequence: the somehow subjective concept of "cluster" depends on the observed sample, and thus there is no immediate objective definition in terms of the population.[42] This implies that there is no clear population reference with which to compare the performance of a given sample clustering, that precise indications on the difficulty of performing clustering on a certain scenario are harder to make, and that there is no noticeable ground truth on what should be the *right* number of clusters for a given population.[43]

In the following, we study a principled *population approach* to clustering that tries to bypass these issues.

A population approach to clustering

The general goal of clustering, *to find clusters of data with low within-cluster variation*, can be regarded as the task of determining *data-rich regions* on the sample. From the density perspective, data-rich regions have a precise definition: *modes* and *high-density* regions. Therefore, modes are going to be crucial for defining population clusters in the sense introduced by Chacón (2015).

Given the random vector $\mathbf{X}$ in $\mathbb{R}^p$ with pdf f, we denote the modes of f by $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_m$. These are the local maxima of f, i.e., $\mathrm{D}f(\boldsymbol{\xi}_j) = \mathbf{0}$, $j = 1, \ldots, m$. Intuitively, we can think of the population clusters as the regions of $\mathbb{R}^p$ that are "associated" with each of the modes of f. This "association" can be visualized, for example, by a gravitational analogy: if $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_m$ denote *fixed* equal-mass

[43] Which is the right amount of clusters for the "claw" density?

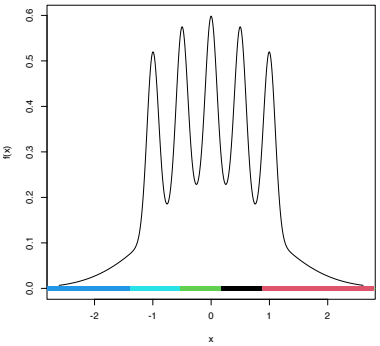


Figure 3.8: k-means partitions for 10,000 observations sampled from the claw density for $k = 5$. Notice how despite the density's having 5 clear modes, the clusters are not associated with them (the blue on the right hand side captures the data on the right tail), even for the large sample size considered.

planets distributed on the space, then the population clusters can be thought of as the regions that determine the *domains of attraction* of each planet for an object $\mathbf{x}$ in the space that has zero initial speed. If $\mathbf{x}$ is attracted by $\boldsymbol{\xi}_1$, then $\mathbf{x}$ belongs to the cluster defined by $\boldsymbol{\xi}_1$. In our setting, the role of the gravity attraction is played by the gradient of the density f, $\mathrm{D}f : \mathbb{R}^p \longrightarrow \mathbb{R}^p$, which forms a *vector field* over $\mathbb{R}^p$.

The code below illustrates the previous analogy and serves to present the function `numDeriv::grad`.

```
# Planets
th <- 2 * pi / 3
r <- 2
xi_1 <- r * c(cos(th + 0.5), sin(th + 0.5))
xi_2 <- r * c(cos(2 * th + 0.5), sin(2 * th + 0.5))
xi_3 <- r * c(cos(3 * th + 0.5), sin(3 * th + 0.5))

# Gravity force
gravity <- function(x) {

  (mvtnorm::dmvnorm(x = x, mean = xi_1, sigma = diag(rep(0.5, 2))) +
     mvtnorm::dmvnorm(x = x, mean = xi_2, sigma = diag(rep(0.5, 2))) +
     mvtnorm::dmvnorm(x = x, mean = xi_3, sigma = diag(rep(0.5, 2)))) / 3

}

# Compute numerically the gradient of an arbitrary function
attraction <- function(x) numDeriv::grad(func = gravity, x = x)

# Evaluate the vector field
x <- seq(-4, 4, l = 20)
xy <- expand.grid(x = x, y = x)
dir <- apply(xy, 1, attraction)

# Scale arrows to unit length for better visualization
len <- sqrt(colSums(dir^2))
dir <- 0.25 * scale(dir, center = FALSE, scale = len)

# Colors of the arrows according to their original magnitude
brk <- quantile(len, probs = seq(0, 1, length.out = 21))
cuts <- cut(x = len, breaks = brk)
cols <- viridis::viridis(20)[cuts]

# Vector field plot
plot(0, 0, type = "n", xlim = c(-4, 4), ylim = c(-4, 4),
     xlab = "x", ylab = "y")
arrows(x0 = xy$x, y0 = xy$y,
       x1 = xy$x + dir[1, ], y1 = xy$y + dir[2, ],
       angle = 10, length = 0.1, col = cols, lwd = 2)
points(rbind(xi_1, xi_2, xi_3), pch = 19, cex = 1.5)
```

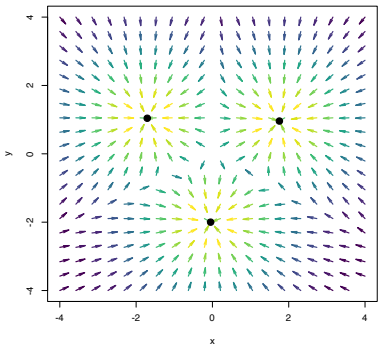


Figure 3.9: Sketch of the gravity vector field associated with three equal-mass planets (black points). The vector field is computed as the gradient of a mixture of three bivariate normals centered at the black points and having covariance matrices $\frac{1}{2}\mathbf{I}_2$. The direction of the arrows denotes the direction of the gravity field, and their color, the strength of the gravity force.

The previous idea can be mathematically formalized as follows. We seek to partition[44] $\mathbb{R}^p$ in a collection of disjoint subsets $W_+^s(\boldsymbol{\xi}_1), \ldots, W_+^s(\boldsymbol{\xi}_m)$ [45] defined as

$$W_+^s(\boldsymbol{\xi}) := \left\{ \mathbf{x} \in \mathbb{R}^p : \lim_{t \to \infty} \boldsymbol{\phi}_{\mathbf{x}}(t) = \boldsymbol{\xi} \right\}$$

where $\boldsymbol{\phi}_{\mathbf{x}} : \mathbb{R} \longrightarrow \mathbb{R}^p$ is a curve in $\mathbb{R}^p$ parametrized by $t \in \mathbb{R}$ that satisfies the following Ordinal Differential Equation (ODE):

$$\frac{\mathrm{d}}{\mathrm{d}t} \boldsymbol{\phi}_{\mathbf{x}}(t) = \mathrm{D}f(\boldsymbol{\phi}_{\mathbf{x}}(t)), \quad \boldsymbol{\phi}_{\mathbf{x}}(0) = \mathbf{x}. \tag{3.22}$$

[44] Technically, excluding sets of zero Lebesgue measure.

[45] The superscript s stands for *stable* manifold and the subscript $+$ emphasizes that the *positive* gradient is considered.

This ODE admits a clear interpretation: the *flow curve* $\boldsymbol{\phi}_{\mathbf{x}}$ is the path that, originated at $\mathbf{x}$, describes $\mathbf{x}$ when reaching $\boldsymbol{\xi}_j$ through the *direction of maximum ascent*.[46] The ODE can be solved through different numerical schemes. For example, observing that $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\phi}_{\mathbf{x}}(t) = \lim_{h\to 0}\frac{\boldsymbol{\phi}_{\mathbf{x}}(t+h)-\boldsymbol{\phi}_{\mathbf{x}}(t)}{h}$, the *Euler method* considers the approximation

$$\boldsymbol{\phi}_{\mathbf{x}}(t+h) \approx \boldsymbol{\phi}_{\mathbf{x}}(t) + h\mathrm{D}f(\boldsymbol{\phi}_{\mathbf{x}}(t)) \quad \text{if } h \approx 0, \tag{3.23}$$

which motivates the iterative scheme[47]

$$\begin{cases} \mathbf{x}_{t+1} = \mathbf{x}_t + h\mathrm{D}f(\mathbf{x}_t), \quad t = 0, \ldots, N, \\ \mathbf{x}_0 = \mathbf{x}, \end{cases} \tag{3.24}$$

for a *step* $h > 0$ and a number of maximum iterations N.

The following chunk of code illustrates the implementation of the Euler method (3.22) and gives insight into the paths $\boldsymbol{\phi}_{\mathbf{x}}$. The code implements the exact gradient of a multivariate normal given in (3.9).

[46] Notice that (3.22) forces the derivative of $\boldsymbol{\phi}_{\mathbf{x}}$ at t to coincide with the gradient of f at $\boldsymbol{\phi}_{\mathbf{x}}(t)$.

[47] Observe that the iterative scheme will converge to a stationary point when $\mathrm{D}f(\mathbf{x}_t) \approx 0$, that is, when $\mathbf{x}_t$ approaches $\boldsymbol{\xi}$ such that $\mathrm{D}f(\boldsymbol{\xi}) = 0$.

```
# Mixture parameters
mu_1 <- rep(1, 2)
mu_2 <- rep(-1.5, 2)
Sigma_1 <- matrix(c(1, -0.75, -0.75, 3), nrow = 2, ncol = 2)
Sigma_2 <- matrix(c(2, 0.75, 0.75, 3), nrow = 2, ncol = 2)
Sigma_1_inv <- solve(Sigma_1)
Sigma_2_inv <- solve(Sigma_2)
w <- 0.45

# Density
f <- function(x) {
  w * mvtnorm::dmvnorm(x = x, mean = mu_1, sigma = Sigma_1) +
    (1 - w) * mvtnorm::dmvnorm(x = x, mean = mu_2, sigma = Sigma_2)
}

# Gradient (caution: only works adequately for x a vector, it is not
# vectorized; observe that in the Sigma_inv %*% (x - mu) part the subtraction
# of mu and premultiplication by Sigma_inv are specific to a *single* point x)
Df <- function(x) {
  -(w * mvtnorm::dmvnorm(x = x, mean = mu_1, sigma = Sigma_1) *
      Sigma_1_inv %*% (x - mu_1) +
    (1 - w) * mvtnorm::dmvnorm(x = x, mean = mu_2, sigma = Sigma_2) *
      Sigma_2_inv %*% (x - mu_2))
}

# Plot density
ks::plotmixt(mus = rbind(mu_1, mu_2), Sigmas = rbind(Sigma_1, Sigma_2),
             props = c(w, 1 - w), display = "filled.contour2",
             gridsize = rep(251, 2), xlim = c(-5, 5), ylim = c(-5, 5),
             cont = seq(0, 90, by = 10), col.fun = viridis::viridis)

# Euler solution
x <- c(-2, 2)
# x <- c(-4, 0)
# x <- c(-4, 4)
N <- 1e3
h <- 0.5
phi <- matrix(nrow = N + 1, ncol = 2)
phi[1, ] <- x
for (t in 1:N) {

  phi[t + 1, ] <- phi[t, ] + h * Df(phi[t, ])# / f(phi[t, ])

}
```

```
lines(phi, type = "l")
points(rbind(x), pch = 19)
text(rbind(x), labels = "x", pos = 3)

# Mean of the components
points(rbind(mu_1, mu_2), pch = 16, col = 4)
text(rbind(mu_1, mu_2), labels = expression(mu[1], mu[2]), pos = 4, col = 4)

# The modes are different from the mean of the components! -- see the gradients
cbind(Df(mu_1), Df(mu_2))
##              [,1]         [,2]
## [1,] -0.005195479 1.528460e-04
## [2,] -0.002886377 7.132814e-05

# Modes
xi_1 <- optim(par = mu_1, fn = function(x) sum(Df(x)^2))$par
xi_2 <- optim(par = mu_2, fn = function(x) sum(Df(x)^2))$par
points(rbind(xi_1, xi_2), pch = 16, col = 2)
text(rbind(xi_1, xi_2), labels = expression(xi[1], xi[2]), col = 2, pos = 2)
```

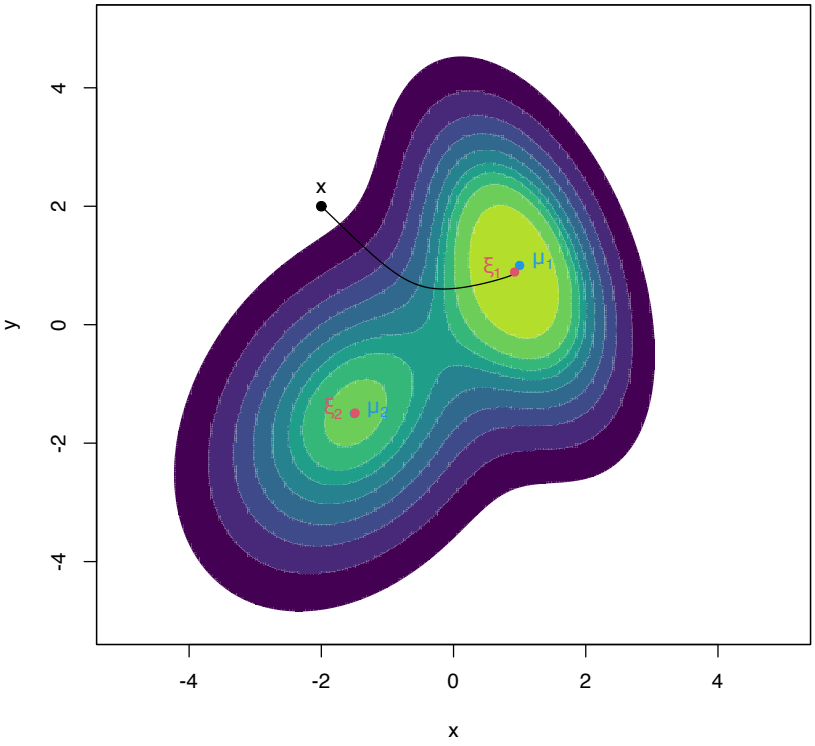


Figure 3.10: The curve $\boldsymbol{\phi}_{\mathbf{x}}$ computed by the Euler method, in black. The population density is the mixture of bivariate normals $w\phi_{\boldsymbol{\Sigma}_1}(\cdot - \boldsymbol{\mu}_1) + (1-w)\phi_{\boldsymbol{\Sigma}_2}(\cdot - \boldsymbol{\mu}_2)$, where $\boldsymbol{\mu}_1 = (1,1)'$, $\boldsymbol{\mu}_2 = (-1.5,-1.5)'$, $\boldsymbol{\Sigma}_1 = (1,-0.75;-0.75,3)$, $\boldsymbol{\Sigma}_2 = (2,0.75;0.75,3)$, and $w = 0.45$. The component means $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ are shown in blue, whereas the two modes $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$ of the density are represented in red. Note that the modes and the component means may be different.

Exercise 3.21. Inspect the code generating Figure 3.10. Then:

1. Experiment with different values for the initial point $\mathbf{x}$ and inspect the resulting flow curves. What happens if $\mathbf{x}$ is in a low-density region?
2. Change the step h to larger and smaller values. What happens? Has h an influence on the convergence to a mode?
3. Does the "adequate" step h depend on $\mathbf{x}$?

Clustering a sample: kernel mean shift clustering

From the previous example, it is clear how to assign a point $\mathbf{x}$ to a population cluster: compute its associated flow curve and assign

the j-th cluster label if $\mathbf{x} \in W_+^s(\boldsymbol{\xi}_j)$. More importantly, once the population view of clustering is clear, performing clustering for a sample is conceptually trivial: just replace f in (3.24) with its kde $\hat{f}(\cdot;\mathbf{H})$![48] By doing so, it results that

$$\begin{cases} \mathbf{x}_{t+1} = \mathbf{x}_t + h\mathrm{D}\hat{f}(\mathbf{x}_t;\mathbf{H}), & t = 0,\dots,N, \\ \mathbf{x}_0 = \mathbf{x}. \end{cases} \tag{3.25}$$

A couple[49] more tweaks are required to turn (3.25) into a more practical relation. The first tweak *boosts* the travel through low-density regions and *decreases* the ascent at high-density regions (see Figure 3.11) by adapting the step size taken at $\mathbf{x}_{t+1}$ by the density at $\mathbf{x}_t$. In version (3.24), this amounts to considering the *normalized gradient*[50]

$$\boldsymbol{\eta}(\mathbf{x}) := \frac{\mathrm{D}f(\mathbf{x})}{f(\mathbf{x})}$$

instead of the unnormalized gradient. Considering the normalized gradient can be seen as a replacement in (3.25) of the constant step h by the *variable* step $h(\mathbf{x}) := a/f(\mathbf{x})$, $a > 0$.

The second tweak replaces a in the variable step with a positive definite matrix $\mathbf{A}$ to premultiply $\boldsymbol{\eta}(\mathbf{x})$ and allow for more generality. This apparent innocuous change gives a convenient[51] choice for $\mathbf{A}$ and hence for the step in Euler's method: $\mathbf{A} = \mathbf{H}$.

Joining the previous two tweaks, it results the recurrence relation of kernel mean shift clustering:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{H}\hat{\boldsymbol{\eta}}(\mathbf{x}_t;\mathbf{H}), \quad \hat{\boldsymbol{\eta}}(\mathbf{x};\mathbf{H}) := \frac{\mathrm{D}\hat{f}(\mathbf{x};\mathbf{H})}{\hat{f}(\mathbf{x};\mathbf{H})}. \tag{3.26}$$

The recipe for clustering a sample $\mathbf{X}_1,\dots,\mathbf{X}_n$ is now simple:

1. Select a "suitable" bandwidth $\hat{\mathbf{H}}$.
2. For each element $\mathbf{X}_i$, iterate the recurrence relation (3.26) "until convergence" to a given $\mathbf{y}_i$, $i = 1,\dots,n$.
3. Find the set of "unique" end points $\{\boldsymbol{\xi}_1,\dots,\boldsymbol{\xi}_m\}$ (the modes) among $\{\mathbf{y}_1,\dots,\mathbf{y}_n\}$.
4. Label $\mathbf{X}_i$ as j if it is associated with the j-th mode $\boldsymbol{\xi}_j$.

Some practical details are important when implementing the previous algorithm. First and more importantly, a "suitable" bandwidth $\hat{\mathbf{H}}$ has to be considered in order to carry out (3.26). Observe the crucial role of the bandwidth:

- A large bandwidth $\hat{\mathbf{H}}$ will oversmooth the data and underestimate the number of modes. In the most extreme case, it will merge all the possible clusters into a single one positioned close to the sample mean $\bar{\mathbf{X}}$.
- A small bandwidth $\hat{\mathbf{H}}$ will undersmooth the data, thus overestimating the number of modes. In the most extreme case, it will detect as many modes as data points.

[48] Recall: once the population setting is well-defined, the sample version becomes immediately clear.

[49] Both inspired by the original mean shift recurrence relation proposed by Fukunaga and Hostetler (1975).

[50] Notice that, on the one hand, the normalized gradient is automatically magnified if the density at $\mathbf{x}$ is low, avoiding very slow movements in low-density regions. On the other hand, the normalizing gradient is decreased automatically at high-density regions, avoiding the "overshooting" of the local maximum.

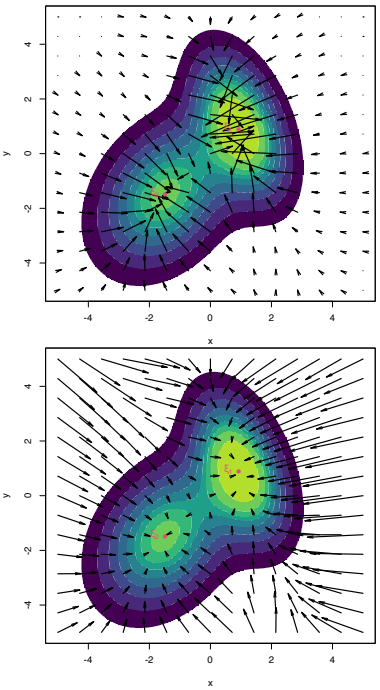


Figure 3.11: Comparison between unnormalized and normalized gradient vector fields. Observe how the unnormalized gradient almost vanishes at regions with low density and "overshoots" close to the modes. The normalized gradient gently adapts to the density region, boosting and slowing ascent as required. Both gradients have been standardized so that the norm of the maximum gradient is 2, therefore enabling their comparison.

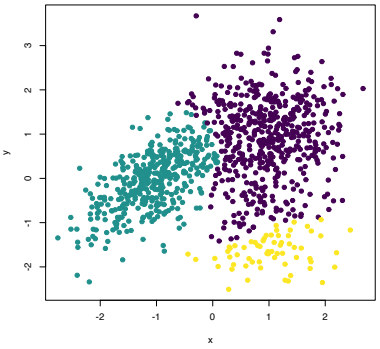
[51] Because it can be seen that a factor $\mathbf{H}^{-1}$ appears in the expansion of $\hat{\boldsymbol{\eta}}(\mathbf{x}_t;\mathbf{H})$ after taking derivatives, a factor that is canceled when premultiplied by $\mathbf{H}$ if $\mathbf{A} = \mathbf{H}$.

The kind of **bandwidth selectors** recommended are the ones designed for ***gradient* density estimation** (see Sections 3.1 and 3.4), since (3.26) critically depends on estimating adequately the gradient (see discussion in Section 6.2.2 in Chacón and Duong (2018)). These bandwidth selectors yield **larger bandwidths** than the ones designed for *density* estimation (compare (3.18) with (3.17)).

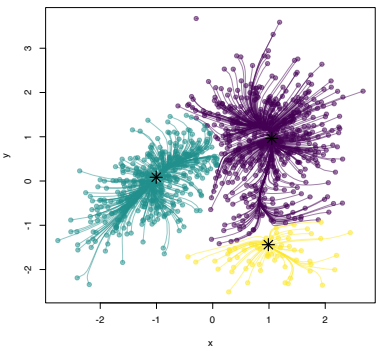
In addition, both the iteration "until convergence" and the "uniqueness" of the end points[52] have to be evaluated in practice with adequate numerical tolerances. The function `ks::kms` implements kernel mean shift clustering and automatically employs tested choices for numerical tolerances and convergence criteria.

[52] Observe that this is key to determine how many different modes, therefore clusters, the algorithm has found.

```
# A simulated example for which the population clusters are known
# Extracted from ?ks::dmvnorm.mixt
mus <- rbind(c(-1, 0), c(1, 2 / sqrt(3)), c(1, -2 / sqrt(3)))
Sigmas <- 1/25 * rbind(ks::invvech(c(9, 63/10, 49/4)),
                       ks::invvech(c(9, 0, 49/4)),
                       ks::invvech(c(9, 0, 49/4)))
props <- c(3, 3, 1) / 7

# Sample the mixture
set.seed(123456)
x <- ks::rmvnorm.mixt(n = 1000, mus = mus, Sigmas = Sigmas, props = props)

# Kernel mean shift clustering. If H is not specified, then
# H = ks::Hpi(x, deriv.order = 1) is employed. Its computation may take some
# time, so it is advisable to compute it separately for later reuse
H <- ks::Hpi(x = x, deriv.order = 1)
kms <- ks::kms(x = x, H = H)

# Plot clusters
plot(kms, col = viridis::viridis(kms$nclust), pch = 19, xlab = "x", ylab = "y")
```



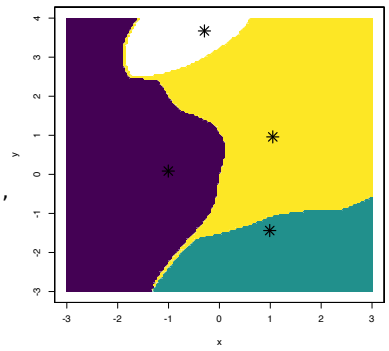
```
# Summary
summary(kms)
## Number of clusters = 3
## Cluster label table = 521 416 63
## Cluster modes =
##           V1          V2
## 1  1.0486466  0.96316436
## 2 -1.0049258  0.08419048
## 3  0.9888924 -1.43852908

# Objects in the kms object
kms$nclust # Number of clusters found
## [1] 3
kms$nclust.table # Sizes of clusters
##
##   1   2   3
## 521 416  63
kms$mode # Estimated modes
##            [,1]        [,2]
## [1,]  1.0486466  0.96316436
## [2,] -1.0049258  0.08419048
## [3,]  0.9888924 -1.43852908

# With keep.path = TRUE the ascending paths are returned
kms <- ks::kms(x = x, H = H, keep.path = TRUE)
cols <- viridis::viridis(kms$nclust, alpha = 0.5)[kms$label]
plot(x, col = cols, pch = 19, xlab = "x", ylab = "y")
for (i in 1:nrow(x)) lines(kms$path[[i]], col = cols[i])
points(kms$mode, pch = 8, cex = 2, lwd = 2)
```



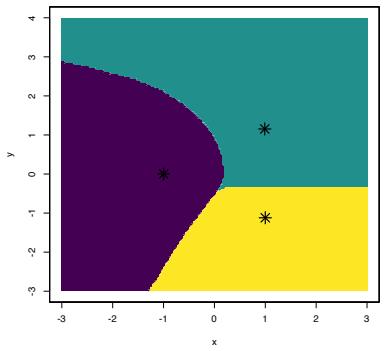
[53] This is a very appealing benefit of kernel mean shift clustering that is not possible in a principled form in other clustering methods.

The **partition of the *whole* space**[53] (not just the sample) can be done with `ks::kms.part`, whereas `ks::mvnorm.mixt.part` allows to partition the population if this is a mixture of normals.

```
# Partition of the whole sample space
kms_part <- ks::kms.part(x = x, H = H, xmin = c(-3, -3), xmax = c(3, 4),
                         gridsize = c(150, 150))
plot(kms_part, display = "filled.contour2", col = viridis::viridis(kms$nclust),
     xlab = "x", ylab = "y")
points(kms_part$mode, pch = 8, cex = 2, lwd = 2)
```



```
# Partition of the population
mixt_part <- ks::mvnorm.mixt.part(mus = mus, Sigmas = Sigmas, props = props,
                                  xmin = c(-3, -3), xmax = c(3, 4),
                                  gridsize = c(150, 150))
plot(mixt_part, display = "filled.contour2", col = viridis::viridis(kms$nclust),
     xlab = "x", ylab = "y")

# Obtain the modes of a mixture of normals automatically
modes <- ks::mvnorm.mixt.mode(mus = mus, Sigmas = Sigmas, props = props)
points(modes, pch = 8, cex = 2, lwd = 2)
```

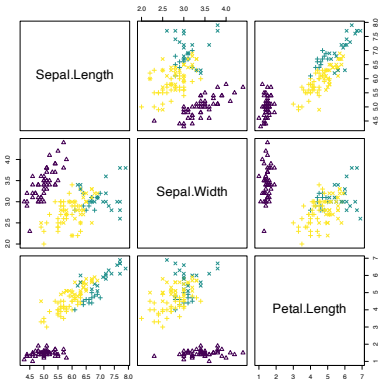


```
modes
##            [,1]         [,2]
## [1,] -0.9975330  0.002125069
## [2,]  0.9898321  1.153091638
## [3,]  0.9999969 -1.119886815
mus
##      [,1]     [,2]
## [1,]   -1  0.000000
## [2,]    1  1.154701
## [3,]    1 -1.154701
```

Finally, the `ks::kms` function is applied to the `iris[, 1:3]` dataset to illustrate a three-dimensional example.

```
# Obtain PI bandwidth
H <- ks::Hpi(x = iris[, 1:3], deriv.order = 1)

# Many (8) clusters: probably due to the repetitions in the data
kms_iris <- ks::kms(x = iris[, 1:3], H = H)
summary(kms_iris)
## Number of clusters = 8
## Cluster label table = 47 3 25 11 55 3 3 3
## Cluster modes =
##   Sepal.Length Sepal.Width Petal.Length
## 1     5.065099    3.442888     1.470614
## 2     5.783786    3.975575     1.255177
## 3     6.726385    3.026522     4.801402
## 4     5.576415    2.478507     3.861941
## 5     6.081276    2.884925     4.710959
## 6     6.168988    2.232741     4.310609
## 7     6.251387    3.375452     5.573491
## 8     7.208475    3.596510     6.118948

# Force to only find clusters that contain at least 10% of the data
# kms merges internally the small clusters with the closest ones
kms_iris <- ks::kms(x = iris[, 1:3], H = H, min.clust.size = 15)
summary(kms_iris)
## Number of clusters = 3
## Cluster label table = 50 31 69
## Cluster modes =
##   Sepal.Length Sepal.Width Petal.Length
## 1     5.065099    3.442888     1.470614
```

```
## 2     6.726385    3.026522     4.801402
## 3     5.576415    2.478507     3.861941

# Pairs plot -- good match of clustering with Species
plot(kms_iris, pch = as.numeric(iris$Species) + 1,
     col = viridis::viridis(kms_iris$nclust))
```




```
# See ascending paths
kms_iris <- ks::kms(x = iris[, 1:3], H = H, min.clust.size = 15,
                    keep.path = TRUE)
cols <- viridis::viridis(kms_iris$nclust)[kms_iris$label]
rgl::plot3d(kms_iris$x, col = cols)
for (i in 1:nrow(iris)) rgl::lines3d(kms_iris$path[[i]], col = cols[i])
rgl::points3d(kms_iris$mode, size = 5)
rgl::rglwidget()
```

Exercise 3.22. The dataset `la-liga-2015-2016.xlsx` contains team statistics for La Liga season 2015/2016.

a. How many clusters are detected for the variables `Yellow.cards` and `Red.cards`? And for `Yellow.cards`, `Red.cards`, and `Fouls.made`? Interpret the results.
b. Standardize the previous variables (divide them by their standard deviation) and recompute a. Are the results the same? Why? Does `kmeans` have a similar behavior?
c. Run a PCA on the dataset after removing `Points` and `Matches` and standardizing the variables. Then perform a clustering on the scores of as many PCs as to explain the 85% of the variance. How many clusters are detected? What teams are associated with each of them? Are the clusters interpretable? Do you see something strange?
d. Run `kmeans` on the data used in c with $k = 3, 4, 5$ and compare the results graphically.

Exercise 3.23. Load the `wines.txt` dataset and perform a PCA removing the `vintage` variable and after standardizing the variables.

a. How many clusters are detected using three PCs? What happens if `min.clust.size` is set such that the minimum cluster contains more than 5% of the sample? Does it seem like a sensible choice to set such value of `min.clust.size`?
b. Using the same `min.clust.size`, consider six PCs. How many clusters are detected? Perform a pairs scatterplot and identify the PCs that are useful to identify clusters and the ones that seem to be adding noise.
c. Using the same `min.clust.size`, do the clustering with two PCs. How many clusters are now detected? Compare the clustering with two and three PCs. Why do you think there are fewer clusters in the latter? *Hint*: observe that the incremental information is on the scores of PC1 vs. PC3 and PC2 vs. PC3.
d. Compare the cluster made in part a with the variable `vintages`.
e. Run `kmeans` on the data used in part a with $k = 3, 4$ and compare the results graphically.

Exercise 3.24. Consider the `MNIST` dataset described in Exercise 2.25. Investigate the different ways of writing the digit "3" using kernel mean shift clustering. Follow the next steps:

a. Consider the t-SNE (van der Maaten and Hinton, 2008) scores `y_tsne` of the class "3". Use only the first 2,000 t-SNE scores for the class "3" for the sake of computational expediency.
b. Compute the normal scale bandwidth matrix that is appropriate for performing kernel mean shift clustering with the first 2,000 t-SNE scores.
c. Do kernel mean shift clustering on that subset using the previously obtained bandwidth and obtain the modes $\boldsymbol{\xi}_j$, $j = 1, \ldots, M$, that cluster the t-SNE scores.
d. Determine the M images that have the closest t-SNE scores, using the Euclidean distance, to the modes $\boldsymbol{\xi}_j$.
e. Show the closest images associated with the modes. Do they represent different forms of drawing the digit "3"?

3.5.3 *Classification*

We assume in this section that $\mathbf{X}$ has a categorical random variable Y as a companion, the *labels* of Y indicating one out of m possible classes within the population $\mathbf{X}$, and that we have a sample $(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)$.

Let's denote f_j to the *conditional* pdf of $\mathbf{X}|Y = j$ and π_j to the value of the probability mass function of Y at j, i.e., $\mathbb{P}[Y = j] = \pi_j$, for $j = 1, \ldots, m$. In this framework, the *unconditional* pdf f of $\mathbf{X}$ is a mixture of the conditional pdfs. This is just a direct application of the *law of the total probability*:

$$f(\mathbf{x}) = \sum_{j=1}^{m} f_j(\mathbf{x})\pi_j. \tag{3.27}$$

The problem of assigning an observation $\mathbf{x}$ of $\mathbf{X}$ to each of the classes of Y can be tackled by maximizing the conditional probability

$$\begin{aligned} \mathbb{P}[Y = j|\mathbf{X} = \mathbf{x}] &= \frac{f_j(\mathbf{x})\mathbb{P}[Y = j]}{f(\mathbf{x})} \\ &= \frac{\pi_j f_j(\mathbf{x})}{\sum_{j=1}^{m} \pi_j f_j(\mathbf{x})}, \end{aligned} \tag{3.28}$$

where a simple application of the Bayes' *rule* involving pdfs and probabilities has been done in the first equality and (3.27) has been called in the second. The **Bayes classifier** exploits this idea of "assigning $\mathbf{x}$ to the most likely class j":[54]

$$\gamma_{\text{Bayes}}(\mathbf{x}) := \arg \max_{j=1,\ldots,m} \pi_j f_j(\mathbf{x}). \tag{3.29}$$

The Bayes classifier enjoys the **minimum possible error rate** among all possible classifiers. But, since it depends on the unknown $f_1, \ldots, f_m, \pi_1, \ldots, \pi_m$, it can not be readily computed.

[54] Observe that, since we are interested in the argument j that maximizes $\mathbb{P}[Y = j|\mathbf{X} = \mathbf{x}]$, we do not care about the standardization by $\sum_{j=1}^{m} \pi_j f_j(\mathbf{x})$ in (3.28), as this is a constant *common* factor for all $j = 1, \ldots, m$.

The solution now is evident: plug-in estimates for the unknown terms in (3.29) from the observed sample $(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)$. Estimating the class probabilities π_j is simple: just consider their relative frequencies $\hat{\pi}_j := \frac{n_j}{n}$, where $n_j := \#\{i = 1, \ldots, n : Y_i = j\}$. Estimating f_j can be done by a kde for the subsample $\{\mathbf{X}_i : Y_i = j, i = 1, \ldots, n\}$, which we denote $\hat{f}_j(\cdot; \mathbf{H}_j)$. Plugging these estimates into (3.29) gives the **kernel classifier**

$$\hat{\gamma}(\mathbf{x}; \mathbf{H}_1, \ldots, \mathbf{H}_m) := \arg \max_{j=1,\ldots,m} \hat{\pi}_j \hat{f}_j(\mathbf{x}; \mathbf{H}_j). \tag{3.30}$$

Application of (3.30) is often referred to as **kernel discriminant analysis** (kda).

The kernel classifier needs to be fed by the bandwidths $\mathbf{H}_1, \ldots, \mathbf{H}_m$, each of them to be selected from each of the m subsamples. However, these can be chosen by any of the procedures described in Section 3.4, as an adequate estimation of the conditional densities f_j will yield sensible classification errors.[55] The main advantage of (3.30) with respect to classical LDA or QDA is the high flexibility induced by the natural nonlinearity of the classification frontier, which can capture any arbitrarily complicated form (yet retaining a conceptually simple formulation).

[55] Notice that one could also choose $\mathbf{H}_j$, $j = 1, \ldots, m$, to maximize classification accuracy directly, rather than to estimate accurately f_j, $j = 1, \ldots, m$. This would be a more complicated procedure, though, as it will involve optimizing a classification criterion over $(\text{vech}(\mathbf{H}_1)', \ldots, \text{vech}(\mathbf{H}_m)')' \in \mathbb{R}^{mp(p-1)/2}$, as opposed to select m bandwidth matrices separately. As vec, the vech operator stacks the columns of any matrix into a long vector, but vech first removes the lower diagonal matrix.

The `ks::kda` function implements kernel discriminant analysis. It essentially calls m times to `ks::kde` in order to compute (3.30). In what follows the usage of `ks::kda` is illustrated, for $p = 1, 2, 3$, in the `iris` dataset.

```
# Univariate example
x <- iris$Sepal.Length
groups <- iris$Species

# By default, the ks::hpi bandwidths are computed
kda_1 <- ks::kda(x = x, x.group = groups)

# Manual specification of bandwidths via ks::hkda (we have univariate data)
hs <- ks::hkda(x = x, x.group = groups, bw = "plugin")
kda_1 <- ks::kda(x = x, x.group = groups, hs = hs)

# Estimated class probabilities
kda_1$prior.prob
## [1] 0.3333333 0.3333333 0.3333333

# Classification
head(kda_1$x.group.estimate)
## [1] setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica

# (Training) classification error
ks::compare(x.group = kda_1$x.group, est.group = kda_1$x.group.estimate)
## $cross
##                   setosa (est.) versicolor (est.) virginica (est.) Total
## setosa (true)               45                5               0    50
## versicolor (true)            6               28              16    50
## virginica (true)             1               10              39    50
## Total                       52               43              55   150
##
## $error
## [1] 0.2533333

# Classification of new observations
```

```
ind_1 <- c(5, 55, 105)
newx <- x[ind_1]
predict(kda_1, x = newx)
## [1] setosa    virginica virginica
## Levels: setosa versicolor virginica
groups[ind_1] # Reality
## [1] setosa     versicolor virginica
## Levels: setosa versicolor virginica

# Classification regions (points on the bottom)
plot(kda_1, xlab = "Sepal length", drawpoints = TRUE, col = rainbow(3))
legend("topright", legend = c("Setosa", "Versicolor", "Virginica"),
       lwd = 2, col = rainbow(3))
```

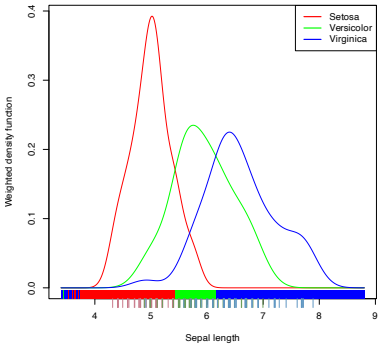


Figure 3.12: Discriminant regions classifying `iris$Species` from `iris$Sepal.Length`.

Exercise 3.25. Experiment with the code above by changing the variable `Sepal.Length` and the bandwidth selectors to see how the classification regions and errors change.

```
# Bivariate example
x <- iris[, 1:2]
groups <- iris$Species

# By default, the ks::Hpi bandwidths are computed
kda_2 <- ks::kda(x = x, x.group = groups)

# Manual specification of bandwidths via ks::Hkda
Hs <- ks::Hkda(x = x, x.group = groups, bw = "plugin")
kda_2 <- ks::kda(x = x, x.group = groups, Hs = Hs)

# Classification of new observations
ind_2 <- c(5, 55, 105)
newx <- x[ind_2, ]
predict(kda_2, x = newx)
## [1] setosa    virginica virginica
## Levels: setosa versicolor virginica
groups[ind_2] # Reality
## [1] setosa     versicolor virginica
## Levels: setosa versicolor virginica

# Classification error
ks::compare(x.group = kda_2$x.group, est.group = kda_2$x.group.estimate)
## $cross
##                  setosa (est.) versicolor (est.) virginica (est.) Total
## setosa (true)               50                0               0    50
## versicolor (true)            0               37              13    50
## virginica (true)             0               11              39    50
## Total                       50               48              52   150
##
## $error
## [1] 0.16

# Plot of classification regions
plot(kda_2, col = rainbow(3), lwd = 2, col.pt = 1, cont = seq(5, 85, by = 20),
     col.part = rainbow(3, alpha = 0.25), drawpoints = TRUE)
# The artifacts on the corners (low-density regions) are caused by
# numerically-unstable divisions close to 0/0

# The artifacts can be avoided by enlarging the effective support of the normal
# kernel that ks considers with supp (by default it is 3.7). Setting supp to
# a larger value (~10) will avoid the normal kernel to reach the value 0
# exactly (but it may be required that the default gridsize has to be enlarged
# to display the surface adequately if supp is quite large). This is a useful
# practical tweak!
kda_2 <- ks::kda(x = x, x.group = groups, Hs = Hs, supp = 10)
plot(kda_2, col = rainbow(3), lwd = 2, col.pt = 1, cont = seq(5, 85, by = 20),
     col.part = rainbow(3, alpha = 0.25), drawpoints = TRUE)
```

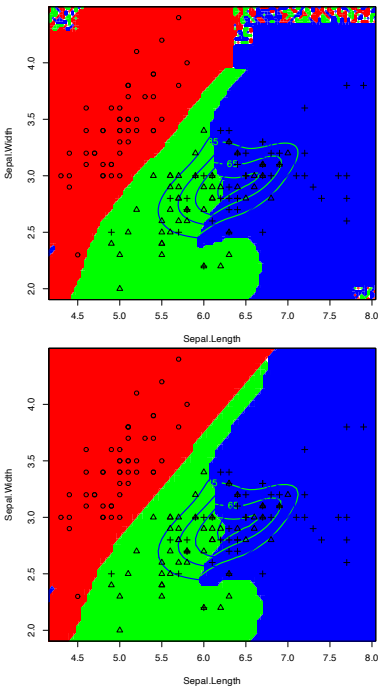
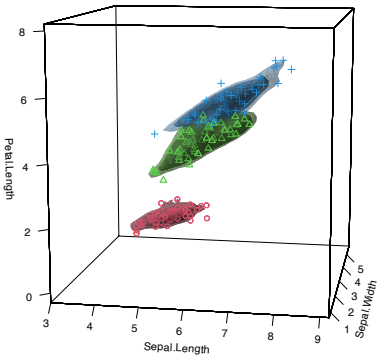


Figure 3.13: Discriminant regions classifying `iris$Species` from `iris$Sepal.Length` and `iris$Sepal.Width`. Observe that the second plot removes the artifacts of the former due to numerically-unstable divisions in low-density regions.

Exercise 3.26. Experiment with the previous code by changing the variables to see how the classification regions and errors change.

```
# Trivariate example
x <- iris[, 1:3]
groups <- iris$Species

# Normal scale bandwidths to avoid undersmoothing
Hs <- rbind(ks::Hns(x = x[groups == "setosa", ]),
            ks::Hns(x = x[groups == "versicolor", ]),
            ks::Hns(x = x[groups == "virginica", ]))
kda_3 <- ks::kda(x = x, x.group = groups, Hs = Hs)

# Classification of new observations
ind_3 <- c(5, 55, 105)
newx <- x[ind_3, ]
predict(kda_3, x = newx)
## [1] setosa     versicolor virginica
## Levels: setosa versicolor virginica
groups[ind_3] # Reality
## [1] setosa     versicolor virginica
## Levels: setosa versicolor virginica

# Classification regions
plot(kda_3, drawpoints = TRUE, col.pt = c(2, 3, 4), cont = seq(5, 85, by = 20),
     phi = 10, theta = 10)
```

There was a bug in `ks` *prior* to 1.11.5, making `ks::kda` to work with equal-size classes only. Although the bug has been fixed, if you are experiencing problems applying `ks::kda` you can always apply several times `ks::kde` and compute (3.30) directly.



Exercise 3.27. Section 4.6.1 in James et al. (2013) considers the problem of classifying `Direction` from `Lag1` and `Lag2` (bivariate example) in `data(Smarket, package = "ISLR")` by logistic regression, LDA, and QDA.

a. Perform a kda and represent the classification regions. Do you think the classes can be separated effectively?
b. Split the dataset into train (`Year < 2005`) and test subsets. Obtain the global classification error rates on the test sample given by LDA and QDA (use `MASS::lda` and `MASS::qda`).
c. Obtain the global classification error rate on the test sample given by kda. Use directly the Bayes rule for performing classification.
d. Summarize the conclusions.

Use `supp = 10` to avoid numerical artifacts.

Exercise 3.28. Load the `wines.txt` dataset and perform a PCA after removing the `vintage` variable and standardizing the variables. Then:

a. Perform a kda on the three classes of wines. First, use just the PC1 as covariate. Then, consider (PC1, PC2). Do you think the classes be separated effectively? With which information?
b. Compare the kda global classification error rates with those of LDA and QDA (use `MASS::lda` and `MASS::qda`).

c. Split the dataset into two thirds for training and one third for testing. Then compare the global classification error rates on the test sample for kda, LDA, and QDA. Again, first use just the PC1 as covariate, then consider (PC1, PC2).
d. Summarize the conclusions.

Exercise 3.29. Consider the `MNIST` dataset described in Exercise 2.25. Classify the digit images, via the t-SNE (van der Maaten and Hinton, 2008) scores `y_tsne`, into the digit labels:

a. Split the dataset into the training sample, comprised of the first $50,000$ t-SNE scores and their associated labels, and the test sample (rest of the sample).
b. Using the training sample, compute the plug-in bandwidth matrices for all the classes.
c. Use these plug-in bandwidths to perform kernel discriminant analysis.
d. Plot the contours of the kernel density estimator of each class and overlay the t-SNE scores as points. Use coherent colors between contours and points, and add a legend.
e. Obtain the successful classification rate of the kernel discriminant analysis.

Exercise 3.30. Load the `ovals.RData` file.

a. Split the dataset into the training sample, comprised of the first $2,000$ observations, and the test sample (rest of the sample). Plot the dataset with colors for its classes. What can you say about the classification problem?
b. Using the training sample, compute the plug-in bandwidth matrices for all the classes.
c. Use these plug-in bandwidths to perform kernel discriminant analysis.
d. Plot the contours of the kernel density estimator of each class and the classes partitions. Use coherent colors between contours and points.
e. Predict the class for the test sample and compare with the true classes. Then report the successful classification rate.
f. Compare the successful classification rate with the one given by LDA. Is it better than kernel discriminant analysis?
g. Repeat f with QDA.

3.5.4 Density ridge estimation

PCA is a very well-known linear dimension-reduction technique that describes the main features of the data. From a population perspective, PCA is just an eigendecomposition of the covariance matrix $\boldsymbol{\Sigma}$ of a random vector $\mathbf{X}$:

$$\boldsymbol{\Sigma} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}', \tag{3.31}$$

where $\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_p)$, $\lambda_1 \geq \ldots \geq \lambda_p \geq 0$, and $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_p)$ is an orthogonal matrix containing the *principal components* as column vectors. When $\mathbf{X}$ is *centered*,[56] the principal components give the *directions of maximal projected variance* of $\mathbf{X}$. Therefore, PCA can *always* be computed for any random vector $\mathbf{X}$ with a proper covariance matrix $\mathbf{\Sigma}$.

It is however when $\mathbf{X} \sim \mathcal{N}_p(\mathbf{0}, \mathbf{\Sigma})$ that PCA is especially meaningful. In that case, the eigendecomposition in (3.31) is related to the geometry of $\phi_{\mathbf{\Sigma}}$ by the search of the major and minor axes of the elliptical density contours. The relation between PCA and the normal distribution is not only geometrical, but also probabilistic: the principal components $\mathbf{U}$ can be reliably estimated by maximum likelihood, which amounts to performing the eigendecomposition (3.31) with $\mathbf{\Sigma}$ replaced with the sample covariance matrix[57] $\mathbf{S}$. Therefore, for an *arbitrary sample*, performing PCA can be regarded as a three-step process: (*i*) fit a normal distribution with estimated variance $\mathbf{S}$; (*ii*) perform the eigendecomposition of $\mathbf{S}$; and (*iii*) attempt to reduce the dimensionality of the sample by discarding the principal components with smallest eigenvalues or projected variances.

[56] Meaning that its expectation $\boldsymbol{\mu}$ is zero. Otherwise, the principal directions given in $\mathbf{U}$ and corresponding to $\{c\mathbf{u}_i : c \in \mathbb{R}\}$, $i = 1, \ldots, p$, need to be shifted by $\boldsymbol{\mu}$, i.e., $\{\boldsymbol{\mu} + c\mathbf{u}_i : c \in \mathbb{R}\}$, $i = 1, \ldots, p$, which is inconvenient. This requirement is not restrictive, as it is trivial to center a random vector and a sample.

[57] Because the moment and maximum likelihood estimators for $(\boldsymbol{\mu}, \mathbf{\Sigma})$ in $\mathcal{N}_p(\boldsymbol{\mu}, \mathbf{\Sigma})$ coincide.

The takeaway of the previous discussion is that a popular dimension-reduction technique like **PCA is closely related to the normal density**; indeed, PCA can be regarded as a technique that rests on an assumption of normality for its accuracy. We next aim to remove this dependence of PCA on normality and to come up with a concept that extends the first principal direction shown in Figure 3.14 in a flexible way, giving a *principal curve* describing the main characteristics of any pdf f.

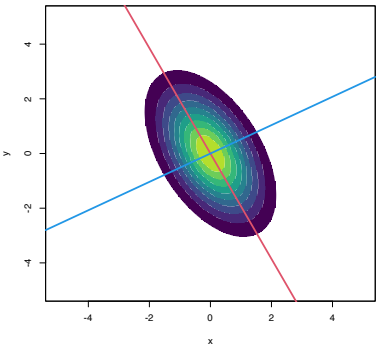


Figure 3.14: Principal directions of $\mathcal{N}_2(\mathbf{0}, \mathbf{\Sigma})$ with $\mathbf{\Sigma} = (1, -0.71; -0.71, 1)$. Red stands for PC1 and blue for PC2.

Density ridges

Approaching the example in Figure 3.14 from the three-dimensional views given in Figure 3.15 we can obtain a highly valuable insight for the PC1: it coincides with an "ascent path" to the "summit" (the mode) of the $\mathcal{N}_2(\mathbf{0}, \mathbf{\Sigma})$'s pdf that is *direct* (no unnecessary detours), yet it is neither the *steepest* (the PC2 ascent is steeper) nor the one with fastest changes in the path's slope.[58] That route follows the **ridge**[59] of the "mountain" given by $\phi_{\mathbf{\Sigma}}$. **Density ridges** are a generalization of the concept of modes (indeed, any mode belongs to a density ridge) that may be relevant for summarizing the main features of a density.

[58] Compare it, e.g., with a zig-zagged ascending path: the rate of change of its slope varies significantly at its bends.

[59] A long, narrow raised part of a surface, especially a high edge along a mountain.

We define next what a density ridge is mathematically. We begin by considering the eigendecomposition of the Hessian matrix of a density function f on $\mathbb{R}^p$, $p \geq 2$, that is evaluated at $\mathbf{x} \in \mathbb{R}^p$:

$$\text{H}f(\mathbf{x}) = \mathbf{U}(\mathbf{x})\mathbf{\Lambda}(\mathbf{x})\mathbf{U}(\mathbf{x})',$$

where $\mathbf{U}(\mathbf{x}) = (\mathbf{u}_1(\mathbf{x}), \ldots, \mathbf{u}_p(\mathbf{x}))$ and $\mathbf{\Lambda}(\mathbf{x}) = \text{diag}(\lambda_1(\mathbf{x}), \ldots, \lambda_p(\mathbf{x}))$, $\lambda_1(\mathbf{x}) \geq \ldots \geq \lambda_p(\mathbf{x})$. We denote $\mathbf{U}_{(p-1)}(\mathbf{x}) := (\mathbf{u}_2(\mathbf{x}), \ldots, \mathbf{u}_p(\mathbf{x}))$ and define the *projected gradient onto* $\{\mathbf{u}_2(\mathbf{x}), \ldots, \mathbf{u}_p(\mathbf{x})\}$ as

$$\text{D}_{(p-1)}f(\mathbf{x}) := \mathbf{U}_{(p-1)}(\mathbf{x})\mathbf{U}_{(p-1)}(\mathbf{x})'\text{D}f(\mathbf{x}),$$

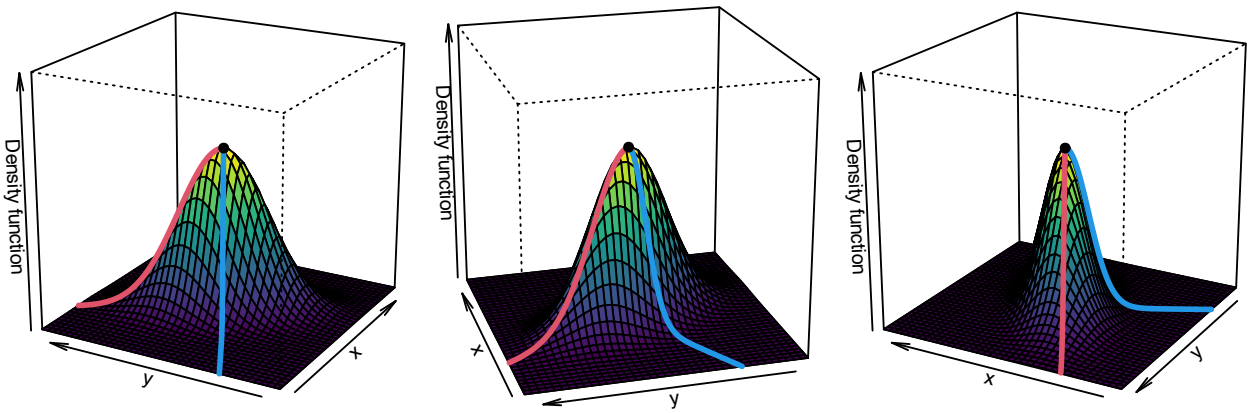


Figure 3.15: PC1 "ascent path" (red) to the "summit" of a $\mathcal{N}_2(\mathbf{0}, \mathbf{\Sigma})$'s pdf. The path precisely follows the ridge of $\phi_{\mathbf{\Sigma}}$. Blue stands for the PC2 "ascent path".

which is a mapping $\mathrm{D}_{(p-1)}f : \mathbb{R}^p \longrightarrow \mathbb{R}^p$. The density ridge of f (or 1-*ridge* of f), as defined by Genovese et al. (2014) (see also Section 6.3 in Chacón and Duong (2018)), consists of the set

$$\mathcal{R}_1(f) := \left\{\mathbf{x} \in \mathbb{R}^p : \|\mathrm{D}_{(p-1)}f(\mathbf{x})\| = 0,\, \lambda_2(\mathbf{x}) < 0\right\}. \tag{3.32}$$

This set may be comprised by the union of disjoint sets.

The intuition behind (3.32) as the precise definition for a density ridge is the following:

- Ridges happen at regions that are locally maximal in certain directions, yet they may not necessarily be formed by local maxima (recall Figure 3.15). As a consequence, $\mathrm{H}f(\mathbf{x})$ is *seminegative definite* near a ridge, and so all the eigenvalues are non-positive: $\lambda_p(\mathbf{x}) \leq \ldots \leq \lambda_1(\mathbf{x}) \leq 0$. Therefore, if $\lambda_1(\mathbf{x}) \leq 0$, the direction with largest *negative curvature*[60] is $\mathbf{u}_p$, and $\mathbf{u}_1$ is the one with the least negative curvature.

- $\mathrm{D}_{(p-1)}f(\mathbf{x})$ is the gradient projected onto the directions with *largest* negative curvatures (since we exclude $\mathbf{u}_1(\mathbf{x})$). When $\mathrm{D}_{(p-1)}f(\mathbf{x}) = \mathbf{0}$, then either (1) $\mathrm{D}f(\mathbf{x}) = \mathbf{0}$ or (2) $\mathrm{D}f(\mathbf{x})$ is orthogonal to $\mathbf{U}_{(p-1)}(\mathbf{x})\mathbf{U}_{(p-1)}(\mathbf{x})'$.

  1. If $\mathrm{D}f(\mathbf{x}) = \mathbf{0}$, then $\mathbf{x}$ is a relative extreme of f, which could be a local maximum or a saddlepoint, since $\lambda_p(\mathbf{x}) \leq \ldots \leq \lambda_2(\mathbf{x}) < 0$, depending on the sign of $\lambda_1(\mathbf{x})$. Modes and saddlepoints are actually part of ridges.
  2. If $\mathrm{D}f(\mathbf{x})$ is orthogonal to $\mathbf{U}_{(p-1)}(\mathbf{x})\mathbf{U}_{(p-1)}(\mathbf{x})'$ then, in other words, it is parallel[61] to $\mathbf{u}_1(\mathbf{x})$: $\mathrm{D}f(\mathbf{x}) \propto \mathbf{u}_1(\mathbf{x})$. Hence, **the directions of maximum ascent and "maximum curvature" at $\mathbf{x}$ coincide**.[62]

The visualization of the construction of the vector field $\mathrm{D}_{(p-1)}f$ gives very valuable insights. In the following code, we rely on the functions `grad_norm` (computes $\mathrm{D}\phi_{\mathbf{\Sigma}}(\cdot - \boldsymbol{\mu})$) and `Hess_norm` (computes $\mathrm{H}\phi_{\mathbf{\Sigma}}(\cdot - \boldsymbol{\mu})$) defined on Section 3.2 to obtain `proj_grad_norm` (computes $\mathrm{D}_{(p-1)}\phi_{\mathbf{\Sigma}}(\cdot - \boldsymbol{\mu})$).

[60] Curvature can be positive or negative!

[61] Because $\mathbf{U}(\mathbf{x})$ is an orthogonal matrix.

[62] Observe that the "maximum curvature" is understood in a *signed* form, and *not* in an absolute form. That is, 1 is a larger signed curvature than -3.

```
# Projected gradient into the Hessian s-th eigenvector subspace
proj_grad_norm <- function(x, mu, Sigma, s = 2) {

  # Gradient
  grad <- grad_norm(x = x, mu = mu, Sigma = Sigma)

  # Hessian
  Hess <- Hess_norm(x = x, mu = mu, Sigma = Sigma)

  # Eigenvectors Hessian
  eig_Hess <- t(apply(Hess, 3, function(A) {
    eigen(x = A, symmetric = TRUE)$vectors[, s]
  }))

  # Projected gradient
  proj_grad <- t(sapply(1:nrow(eig_Hess), function(i) {
    tcrossprod(eig_Hess[i, ]) %*% grad[i, ]
  }))

  # As an array
  return(proj_grad)

}
```

With the above functions, we can visualize the four vector fields $\mathbf{u}_1, \mathbf{u}_2, \mathrm{D}f, \mathrm{D}_{(p-1)}f : \mathbb{R}^p \longrightarrow \mathbb{R}^p$, in $p = 2$, that are critical to construct density ridges.

```
# Compute vector fields
mu <- c(0, 0)
Sigma <- matrix(c(1, -0.71, -0.71, 3), nrow = 2, ncol = 2)
x <- seq(-3.5, 3.5, l = 12)
xx <- as.matrix(expand.grid(x, x))
H <- Hess_norm(x = xx, Sigma = Sigma, mu = mu)
eig_val_1 <- t(apply(H, 3, function(A)
  eigen(x = A, symmetric = TRUE)$values[1]))
eig_val_2 <- t(apply(H, 3, function(A)
  eigen(x = A, symmetric = TRUE)$values[2]))
eig_vec_1 <- t(apply(H, 3, function(A)
  eigen(x = A, symmetric = TRUE)$vectors[, 1]))
eig_vec_2 <- t(apply(H, 3, function(A)
  eigen(x = A, symmetric = TRUE)$vectors[, 2]))
grad <- grad_norm(x = xx, mu = mu, Sigma = Sigma)
grad_proj <- proj_grad_norm(x = xx, mu = mu, Sigma = Sigma)

# Standardize directions for nicer plots
r <- 0.5
eig_vec_1 <- r * eig_vec_1
eig_vec_2 <- r * eig_vec_2
grad <- r * grad / sqrt(rowSums(grad^2))
grad_proj <- r * grad_proj / sqrt(rowSums(grad_proj^2))

par(mfrow = c(2, 2))
ks::plotmixt(mus = mu, Sigmas = Sigma, props = 1, display = "filled.contour2",
             gridsize = rep(251, 2), xlim = c(-4, 4), ylim = c(-4, 4),
             cont = seq(0, 90, by = 10), col.fun = viridis::viridis,
             main = expression(bold(u)[1]))
arrows(x0 = xx[, 1], y0 = xx[, 2],
       x1 = xx[, 1] + eig_vec_1[, 1], y1 = xx[, 2] + eig_vec_1[, 2],
       length = 0.1, angle = 10, col = 1)
points(xx, pch = ifelse(eig_val_1 >= 0, "+", "-"),
       col = ifelse(eig_val_1 >= 0, 2, 4))

ks::plotmixt(mus = mu, Sigmas = Sigma, props = 1, display = "filled.contour2",
             gridsize = rep(251, 2), xlim = c(-4, 4), ylim = c(-4, 4),
             cont = seq(0, 90, by = 10), col.fun = viridis::viridis,
             main = expression(bold(u)[2]))
arrows(x0 = xx[, 1], y0 = xx[, 2],
```

```
       x1 = xx[, 1] + eig_vec_2[, 1], y1 = xx[, 2] + eig_vec_2[, 2],
       length = 0.1, angle = 10, col = 1)
points(xx, pch = ifelse(eig_val_2 >= 0, "+", "-"),
       col = ifelse(eig_val_2 >= 0, 2, 4))

ks::plotmixt(mus = mu, Sigmas = Sigma, props = 1, display = "filled.contour2",
             gridsize = rep(251, 2), xlim = c(-4, 4), ylim = c(-4, 4),
             cont = seq(0, 90, by = 10), col.fun = viridis::viridis,
             main = expression(plain(D) * f))
arrows(x0 = xx[, 1], y0 = xx[, 2],
       x1 = xx[, 1] + grad[, 1], y1 = xx[, 2] + grad[, 2],
       length = 0.1, angle = 10, col = 1)

ks::plotmixt(mus = mu, Sigmas = Sigma, props = 1, display = "filled.contour2",
             gridsize = rep(251, 2), xlim = c(-4, 4), ylim = c(-4, 4),
             cont = seq(0, 90, by = 10), col.fun = viridis::viridis,
             main = expression(plain(D)[(p - 1)] * f))
arrows(x0 = xx[, 1], y0 = xx[, 2],
       x1 = xx[, 1] + grad_proj[, 1], y1 = xx[, 2] + grad_proj[, 2],
       length = 0.1, angle = 10, col = 1)
```

The construction of $\mathcal{R}_1(f)$ can be generalized to higher dimensions. The *density s-ridge*, $1 \leq s \leq p-1$, is defined as

$$\mathcal{R}_s(f) = \{\mathbf{x} \in \mathbb{R}^p : \|\mathrm{D}_{(p-s)} f(\mathbf{x})\| = 0,\, \lambda_{s+1}(\mathbf{x}) < 0\},$$

where $\mathrm{D}_{(p-s)} f(\mathbf{x}) := \mathbf{U}_{(p-s)}(\mathbf{x})\mathbf{U}_{(p-s)}(\mathbf{x})'\mathrm{D}f(\mathbf{x})$ and $\mathbf{U}_{(p-s)} := \left(\mathbf{u}_{s+1}(\mathbf{x}), \ldots, \mathbf{u}_p(\mathbf{x})\right)$. The density s-ridge allows to summarize the main features of f by surfaces of dimension s, similarly to how the plane spanned by PC1 and PC2 describes most of the variability of a three-dimensional random variable $\mathbf{X}$.

We conclude by proving that, indeed, the density 1-ridge of a $\mathcal{N}_p(\mathbf{0}, \boldsymbol{\Sigma})$ is very related to the first principal direction PC1.

Proposition 3.1. *Let* $\boldsymbol{\Sigma}$ *as in* (3.31). *Then* $\{c\mathbf{u}_1 : c \in \mathbb{R}\} \subset \mathcal{R}_1\left(\phi_{\boldsymbol{\Sigma}}\right)$.

Proof. Consider (3.9) and (3.10) with $\boldsymbol{\mu} = \mathbf{0}$:

$$\begin{aligned}
\mathrm{D}\phi_{\boldsymbol{\Sigma}}(\mathbf{x}) &= -\phi_{\boldsymbol{\Sigma}}(\mathbf{x})\boldsymbol{\Sigma}^{-1}\mathbf{x}, \\
\mathrm{H}\phi_{\boldsymbol{\Sigma}}(\mathbf{x}) &= \phi_{\boldsymbol{\Sigma}}(\mathbf{x})\left((\boldsymbol{\Sigma}^{-1}\mathbf{x})(\boldsymbol{\Sigma}^{-1}\mathbf{x})' - \boldsymbol{\Sigma}^{-1}\right).
\end{aligned}$$

Let's take $\mathbf{x} = c\mathbf{u}_1$, $c \in \mathbb{R}$, and show that $\mathrm{D}_{(p-1)} f(c\mathbf{u}_1) = \mathbf{0}$ and that the second eigenvalue of $\mathrm{H}\phi_{\boldsymbol{\Sigma}}(c\mathbf{u}_1)$ is negative.

Since $\boldsymbol{\Sigma} = \sum_{i=1}^p \lambda_i \mathbf{u}_i \mathbf{u}_i'$ and $\boldsymbol{\Sigma}^{-1} = \sum_{i=1}^p \lambda_i^{-1} \mathbf{u}_i \mathbf{u}_i'$, then

$$\mathrm{D}\phi_{\boldsymbol{\Sigma}}(c\mathbf{u}_1) = -\phi_{\boldsymbol{\Sigma}}(c\mathbf{u}_1) c\lambda_1^{-1}\mathbf{u}_1, \tag{3.33}$$

$$\begin{aligned}
\mathrm{H}\phi_{\boldsymbol{\Sigma}}(c\mathbf{u}_1) &= \phi_{\boldsymbol{\Sigma}}(c\mathbf{u}_1)\left(c^2\lambda_1^{-2}\mathbf{u}_1\mathbf{u}_1' - \boldsymbol{\Sigma}^{-1}\right) \\
&= \left(\phi_{\boldsymbol{\Sigma}}(c\mathbf{u}_1)\lambda_1^{-1}(c^2\lambda_1^{-1} - 1)\right)\mathbf{u}_1\mathbf{u}_1' \\
&\quad - \sum_{i=2}^p \left(\phi_{\boldsymbol{\Sigma}}(c\mathbf{u}_1)\lambda_i^{-1}\right)\mathbf{u}_i\mathbf{u}_i'.
\end{aligned} \tag{3.34}$$

Therefore, $\mathrm{H}\phi_{\boldsymbol{\Sigma}}$ evaluated at $\mathbf{x} = c\mathbf{u}_1$ admits the eigendecomposition

$$\mathrm{H}\phi_{\boldsymbol{\Sigma}}(c\mathbf{u}_1) = \mathbf{U}\boldsymbol{\Lambda}(c\mathbf{u}_1)\mathbf{U}'$$

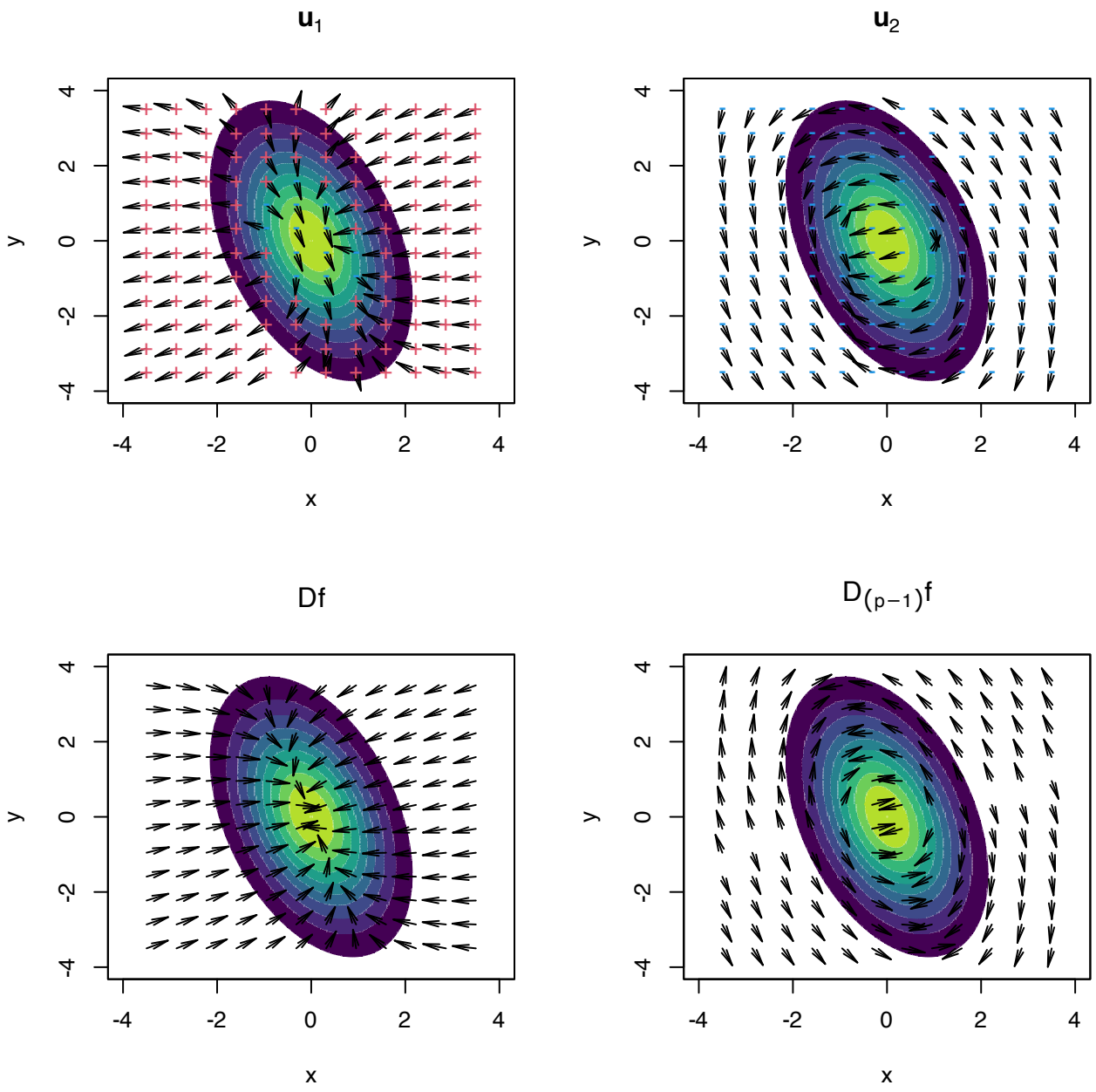


Figure 3.16: Representation of the vector fields $\mathbf{u}_1, \mathbf{u}_2, \mathrm{D}f, \mathrm{D}_{(p-1)}f : \mathbb{R}^p \longrightarrow \mathbb{R}^p$. Observe how the field $\mathrm{D}f$ pushes towards the mode but $\mathrm{D}_{(p-1)}f$ pushes towards the ridge, following the direction given by $\mathbf{u}_2$. The field of $\mathbf{u}_1$ goes roughly right-to-left and parallel to the ridge, whereas the field of $\mathbf{u}_2$ goes roughly up-to-down and perpendicular to the ridge. The red plus (respectively, blue minus) indicate positive (negative) eigenvalue $\lambda_i(\mathbf{x})$ associated with $\mathbf{u}_i$, $i = 1, 2$. The vector fields are normalized so that the norm of each vector is always the same.

with

$$\boldsymbol{\Lambda}(c\mathbf{u}_1) = \phi_{\boldsymbol{\Sigma}}(c\mathbf{u}_1)\text{diag}\left(\lambda_1^{-1}(c^2\lambda_1^{-1}-1), -\lambda_2^{-1}, \ldots, -\lambda_p^{-1}\right).$$

And then, as a consequence, all but the first eigenvalue of $\text{H}\phi_{\boldsymbol{\Sigma}}(c\mathbf{u}_1)$ are negative. Finally, joining (3.33) and (3.34),

$$\begin{aligned}\text{D}_{(p-1)}(c\mathbf{u}_1) &= \mathbf{U}_{(p-1)}\mathbf{U}'_{(p-1)}\text{D}f(c\mathbf{u}_1)\\ &= \mathbf{U}_{(p-1)}\mathbf{U}'_{(p-1)}(-\phi_{\boldsymbol{\Sigma}}(c\mathbf{u}_1)c\lambda_1^{-1})\mathbf{u}_1\\ &= \mathbf{0},\end{aligned}$$

which concludes the proof. □

Density ridges from mean shift clustering

The definition (3.32) paves the way to determine the density ridges in an exactly analogous way to how modes were located in mean shift clustering: by *flowing with the gradient*. Figure 3.16 illustrates that now this flowing has to be done using the *projected* gradient flow $\text{D}_{(p-1)}f$ until attaining $\mathbf{x} \in \mathbb{R}^p$ such that $\text{D}_{(p-1)}f(\mathbf{x}) = \mathbf{0}$.

More precisely, (3.32) can be parametrized as

$$\mathcal{R}_1(f) = \left\{\mathbf{x} \in \mathbb{R}^p : \lim_{t\to\infty} \boldsymbol{\phi}_{\mathbf{x}_0}(t) = \mathbf{x},\, \mathbf{x}_0 \in \mathbb{R}^p\right\},$$

where $\boldsymbol{\phi}_{\mathbf{x}_0} : \mathbb{R} \longrightarrow \mathbb{R}^p$ is a curve in $\mathbb{R}^p$ that satisfies the ODE

$$\frac{\text{d}}{\text{d}t}\boldsymbol{\phi}_{\mathbf{x}_0}(t) = \text{D}_{(p-1)}f(\boldsymbol{\phi}_{\mathbf{x}_0}(t)), \quad \boldsymbol{\phi}_{\mathbf{x}_0}(0) = \mathbf{x}_0, \tag{3.35}$$

where $\mathbf{x}_0$ is any point in $\mathbb{R}^p$. As done in (3.22), (3.35) can be numerically approached through the Euler method:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + h\text{D}_{(p-1)}f(\mathbf{x}_t), \quad t = 0, \ldots, N, \tag{3.36}$$

for a step $h > 0$ and a number of maximum iterations N. Considering the normalized gradient in (3.36)

$$\boldsymbol{\eta}_{(p-1)}(\mathbf{x}) := \frac{\text{D}_{(p-1)}f(\mathbf{x})}{f(\mathbf{x})},$$

instead of the unnormalized gradient, helps adapting the step size h conveniently.[63]

[63] Additionally, this removes the necessity of computing complicated normalizing constants for f, as they will cancel out in the normalized gradient.

The following chunk of code illustrates the implementation of the Euler method (3.35), showing the paths $\boldsymbol{\phi}_{\mathbf{x}_0}$ for different $\mathbf{x}_0$.

```
mu <- c(0, 0)
Sigma <- matrix(c(1, -0.71, -0.71, 2), nrow = 2, ncol = 2)
ks::plotmixt(mus = mu, Sigmas = Sigma, props = 1, display = "filled.contour2",
             gridsize = rep(251, 2), xlim = c(-5, 5), ylim = c(-5, 5),
             cont = seq(0, 90, by = 10), col.fun = viridis::viridis)

# Euler solution
x0 <- as.matrix(expand.grid(seq(-3, 3, l = 12), seq(-3, 3, l = 12)))
x <- matrix(NA, nrow = nrow(x0), ncol = 2)
N <- 500
h <- 0.5
phi <- matrix(nrow = N + 1, ncol = 2)
```

```
eps <- 1e-4
for (i in 1:nrow(x0)) {

  # Move along the flow curve
  phi[1, ] <- x0[i, ]
  for (t in 1:N) {

    # Euler update
    phi[t + 1, ] <- phi[t, ] +
      h * proj_grad_norm(phi[t, ], mu = mu, Sigma = Sigma) /
      mvtnorm::dmvnorm(x = phi[t, ], mean = mu, sigma = Sigma)

    # Stopping criterion (to save computing time!)
    abs_tol <- max(abs(phi[t + 1, ] - phi[t, ]))
    rel_tol <- abs_tol / max(abs(phi[t, ]))
    if (abs_tol < eps | rel_tol < eps) break

  }

  # Save final point
  x[i, ] <- phi[t + 1, , drop = FALSE]

  # Plot lines and x0
  lines(phi[1:(t + 1), ], type = "l")
  points(x0[i, , drop = FALSE], pch = 19)

}

# Plot final points
points(x, pch = 19, col = 2)

# Join the ridge points with lines in an automatic and sensible way:
# an Euclidean Minimum Spanning Tree (EMST) problem!
emst <- emstreeR::ComputeMST(x = x, verbose = FALSE)
segments(x0 = x[emst$from, 1], y0 = x[emst$from, 2],
         x1 = x[emst$to, 1], y1 = x[emst$to, 2], col = 2, lwd = 2)
```

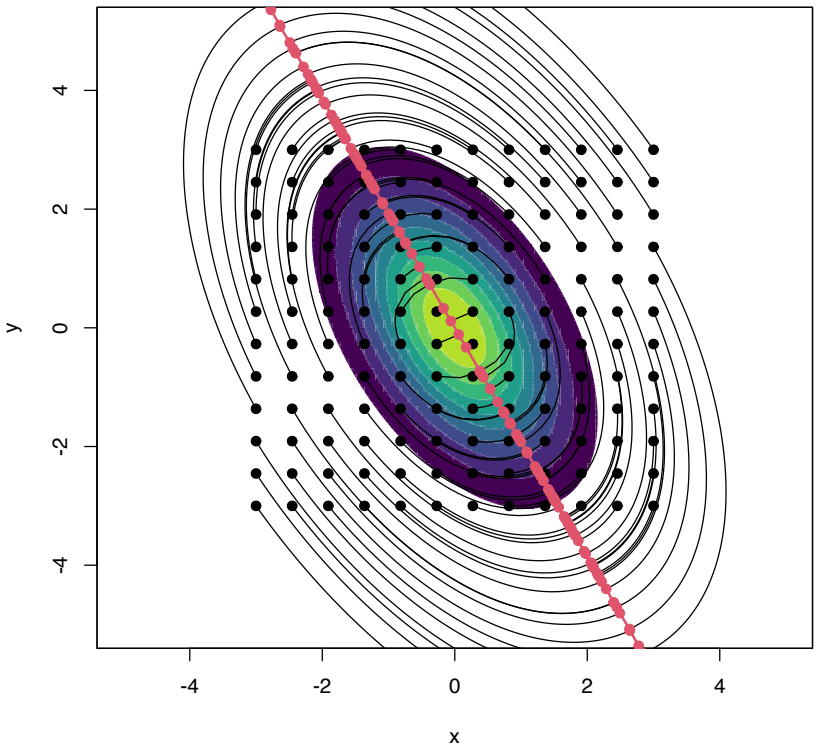


Figure 3.17: Curves $\boldsymbol{\phi}_{\mathbf{x}_0}$ computed by the Euler method, in black. The population density is $\phi_{\boldsymbol{\Sigma}}(\cdot - \boldsymbol{\mu})$, where $\boldsymbol{\mu} = (0, 0)'$ and $\boldsymbol{\Sigma} = (1, -0.71; -0.71, 2)$. The initial points $\mathbf{x}_0$ are shown in black, whereas ridge points $\mathbf{x}$ are represented in red and are joined by the Euclidean minimum spanning tree (red line).

Of course, we can compute the ridges of arbitrary density functions using (3.36). Figure 3.18 shows the density ridges of several heavily non-normal densities.[64] The figure also illustrates some practical problems that may appear when computing density ridges, even at the population level.

[64] Can you imagine what the PC1 description of such densities would be?

```
# "Oval" density
f_oval <- function(x, mu = 2, sigma = 0.35,
                   Sigma = rbind(c(1, -0.71), c(-0.71, 2))) {

  # x always as a matrix
  x <- rbind(x)

  # Rotate x with distortion
  Sigma_inv_sqrt <- solve(chol(Sigma))
  x <- x %*% Sigma_inv_sqrt

  # Polar coordinates
  r <- sqrt(rowSums(x^2))

  # Density as conditional * marginal
  f_theta <- 1 / (2 * pi)
  f_r_theta <- dnorm(x = r, mean = mu, sd = sigma)
  jacobian <-  det(Sigma_inv_sqrt) / r
  f <- f_r_theta * f_theta * jacobian
  return(f)

}

# "Croissant" density
f_crois <- function(x, mu = 2, sigma = 0.5, mu_theta = pi / 2, kappa = 1) {

  # x always as a matrix
  x <- rbind(x)

  # Polar coordinates
  theta <- atan2(x[, 2], x[, 1])
  r <- sqrt(rowSums(x^2))

  # Density as conditional * marginal
  f_theta <- exp(kappa * cos(theta - mu_theta)) /
    (2 * pi * besselI(kappa, nu = 0))
  f_r_theta <- dnorm(x = r, mean = mu, sd = sigma)
  jacobian <- 1 / r
  f <- f_r_theta * f_theta * jacobian
  return(f)

}

# "Sin" density
f_sin <- function(x, a = 0.5, b = 1.75, sigma_x = 2, sigma_y = 0.5) {

  # x always as a matrix
  x <- rbind(x)

  # Density as conditional * marginal
  f_y <- dnorm(x = x[, 1], mean = 0, sd = sigma_x)
  f_x_y <- dnorm(x = x[, 2], mean = a * (1 + x[, 1]) * sin(b * x[, 1]),
                 sd = sigma_y)
  f <- f_x_y * f_y
  return(f)

}
```

Exercise 3.31. Implement the Euler method for f being `f_oval`, `f_crois`, and `f_sin`, in order to reproduce Figure 3.18. You will

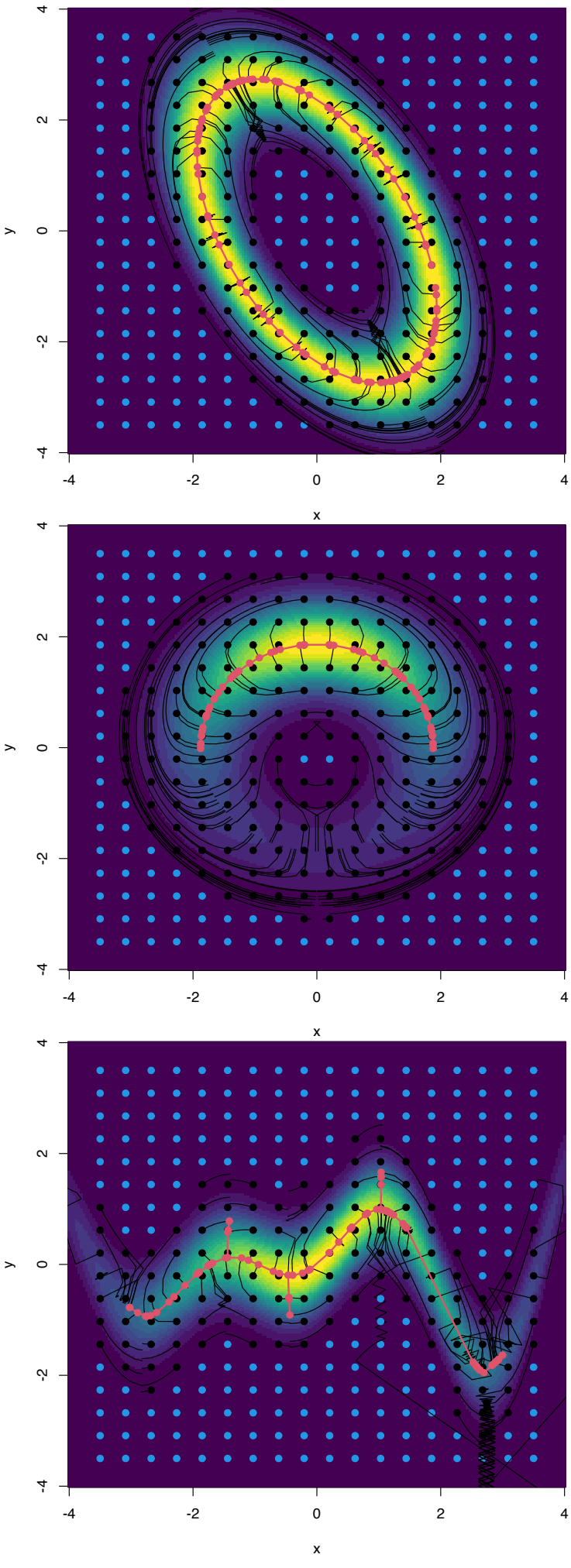


Figure 3.18: Highly non-normal densities and the construction of their density ridges. In blue, the points $\mathbf{x}_0$ that lie on low-density regions and are skipped from the Euler algorithm (longer running times and not useful output). In red, the final points $\mathbf{x}$ that lie on ridges on high-density regions (thus excluding ill-defined ridges). The red line is the Euclidean minimum spanning tree of the final points.

have to:

- Implement the gradient and Hessian of f. To avoid computing the analytical form, you can rely on `numDeriv::grad` and `numDeriv::hessian` to evaluate numerically $\mathrm{D}f$ and $\mathrm{H}f$ (although this will run slower than the analytical form).
- Skip initial values $\mathbf{x}_0$ such that $f(\mathbf{x}_0) < \delta$, so that the Euler method is not run for them. This saves computational time.
- Stop the iteration in the Euler method if absolute or relative convergence is attained. Precisely, stop if $\|\mathbf{x}_{t+1} - \mathbf{x}_t\|_\infty < \varepsilon$[65] or $\|\mathbf{x}_{t+1} - \mathbf{x}_t\|_\infty / \|\mathbf{x}_t\|_\infty < \varepsilon$. This saves computational time.
- Disregard final values $\mathbf{x}$ from the density ridge if $f(\mathbf{x}) < 50\delta$. This avoids spurious solutions.
- Join points automatically using the Euclidean minimum spanning tree algorithm.
- Tune the selection of h, N, ε, and δ. You can start with $h = 0.25$, $N = 100$, $\varepsilon = 10^{-4}$, and $\delta = 10^{-3}$.

[65] $\|\mathbf{x}\|_\infty := \max_{1 \leq j \leq p} |x_j|$.

Estimating density ridges

Once the population view of density ridges is clear, estimating density ridges from a sample is conceptually trivial: just replace f in (3.36) with its kde $\hat{f}(\cdot; \mathbf{H})$ and then apply the two tweaks employed in kernel mean shift clustering (3.26): replace the gradient with the normalized gradient and the step h by $\mathbf{H}$. By doing so, it results

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{H}\hat{\boldsymbol{\eta}}_{(p-1)}(\mathbf{x}_t; \mathbf{H}), \quad \hat{\boldsymbol{\eta}}_{(p-1)}(\mathbf{x}; \mathbf{H}) := \frac{\mathrm{D}_{(p-1)}\hat{f}(\mathbf{x}; \mathbf{H})}{\hat{f}(\mathbf{x}; \mathbf{H})}, \tag{3.37}$$

where

$$\mathrm{D}_{(p-1)}\hat{f}(\mathbf{x}; \mathbf{H}) := \hat{\mathbf{U}}_{(p-1)}(\mathbf{x}; \mathbf{H})\hat{\mathbf{U}}_{(p-1)}(\mathbf{x}; \mathbf{H})'\mathrm{D}\hat{f}(\mathbf{x}; \mathbf{H}) \tag{3.38}$$

and $\hat{\mathbf{U}}_{(p-1)}(\mathbf{x}; \mathbf{H})$ stems from the eigendecomposition of the estimated Hessian: $\mathrm{H}\hat{f}(\mathbf{x}; \mathbf{H}) = \hat{\mathbf{U}}(\mathbf{x}; \mathbf{H})\hat{\boldsymbol{\Lambda}}(\mathbf{x}; \mathbf{H})\hat{\mathbf{U}}(\mathbf{x}; \mathbf{H})'$.

The recipe for estimating density ridges from a sample $\mathbf{X}_1, \ldots, \mathbf{X}_n$ is now simple:

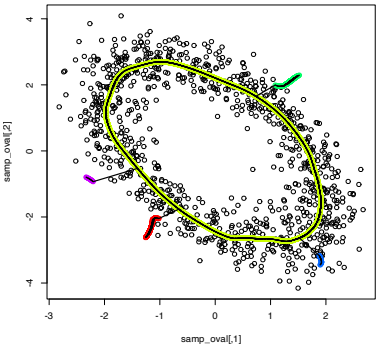
1. Select a "suitable" bandwidth $\hat{\mathbf{H}}$.
2. For each grid element $\{\mathbf{x}_{0,1}, \ldots, \mathbf{x}_{0,m}\}$, iterate the recurrence relation (3.37) "until convergence" to a given $\mathbf{x}_j$, $j = 1, \ldots, m$. Skip this step for $\mathbf{x}_{0,j}$ such that $\hat{f}(\mathbf{x}_{0,j}; \mathbf{H})$ is not larger than a "certain threshold".[66]
3. Set $\hat{\mathcal{R}}_1(f) := \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$.

[66] For example, run the step only if $\mathbf{x}_{0,j} \in \mathcal{L}(\hat{f}(\cdot; \mathbf{H}); \hat{c}_\alpha)$, for α close to zero.

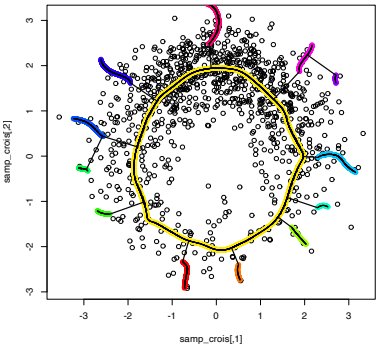
The kind of **bandwidth selectors** recommended are the ones designed for ***Hessian* density estimation** (see Sections 3.1 and 3.4), since (3.37) critically depends on adequately estimating the Hessian for computing the projection into its second eigenvector (see discussion in Section 6.3 in Chacón and Duong (2018)). These bandwidth selectors yield **larger bandwidths** than the ones designed for *density* estimation (and also density *gradient* estimation).

In addition, both the iteration "until convergence" and the "certain threshold" have to be evaluated in practice with adequate numerical tolerances. The function `ks::kdr` implements kernel density ridge estimation and automatically employs tested choices for numerical tolerances, convergence criteria, and other tuning parameters. The following chunk of code shows its basic use in the examples of the "oval", "croissant", and "sin" densities.

```
# Simulation from the "oval" density
r_oval <- function(n, mu = 2, sigma = 0.35,
                   Sigma = rbind(c(1, -0.71), c(-0.71, 2))) {

  # Sampling in polar coordinates
  r <- rnorm(n = n, mean = mu, sd = sigma)
  theta <- runif(n, 0, 2 * pi)
  x <- r * cbind(cos(theta), sin(theta))

  # Data rotation
  Sigma_sqrt <- chol(Sigma)
  return(x %*% Sigma_sqrt)

}

# Simulation from the "croissant" density
r_crois <- function(n, mu = 2, sigma = 0.5, mu_theta = pi / 2, kappa = 1) {

  # Sampling in polar coordinates as conditional * marginal
  theta <- circular:::RvonmisesRad(n = n, mu = mu_theta, kappa = kappa)
  r <- rnorm(n = n, mean = mu, sd = sigma)
  x <- r * cbind(cos(theta), sin(theta))
  return(x)

}

# Simulation from the "sin" density
r_sin <- function(n, a = 0.5, b = 1.75, sigma_x = 2, sigma_y = 0.5) {

  # Sampling as conditional * marginal
  x <- rnorm(n = n, mean = 0, sd = sigma_x)
  y <- rnorm(n = n, mean = a * (1 + x) * sin(b * x), sd = sigma_y)
  return(cbind(x, y))

}

# Oval
set.seed(123456)
samp_oval <- r_oval(n = 1e3)
kdr_oval <- ks::kdr(x = samp_oval)
plot(samp_oval)
col <- rainbow(max(kdr_oval$end.points$segment))[kdr_oval$end.points$segment]
points(kdr_oval$end.points[, 1:2], col = col)
emst <- emstreeR::ComputeMST(x = kdr_oval$end.points[, 1:2], verbose = FALSE)
segments(x0 = kdr_oval$end.points[emst$from, 1],
         y0 = kdr_oval$end.points[emst$from, 2],
         x1 = kdr_oval$end.points[emst$to, 1],
         y1 = kdr_oval$end.points[emst$to, 2], lwd = 2)


# The $end.points$segment output of ks::kdr is very useful, as it allows
# handling the components of the ridges easily

# Croissant
set.seed(526123)
samp_crois <- r_crois(n = 1e3)
kdr_crois <- ks::kdr(x = samp_crois)
plot(samp_crois)
```

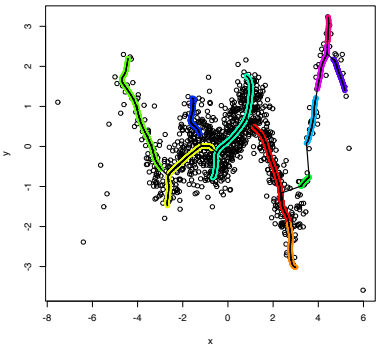


```
col <- rainbow(max(kdr_crois$end.points$segment))[kdr_crois$end.points$segment]
points(kdr_crois$end.points[, 1:2], col = col)
emst <- emstreeR::ComputeMST(x = kdr_crois$end.points[, 1:2], verbose = FALSE)
segments(x0 = kdr_crois$end.points[emst$from, 1],
         y0 = kdr_crois$end.points[emst$from, 2],
         x1 = kdr_crois$end.points[emst$to, 1],
         y1 = kdr_crois$end.points[emst$to, 2], lwd = 2)
```

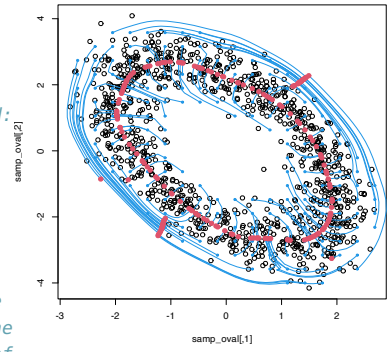



```
# Sin
set.seed(123456)
samp_sin <- r_sin(n = 1e3)
kdr_sin <- ks::kdr(x = samp_sin)
plot(samp_sin)
col <- rainbow(max(kdr_sin$end.points$segment))[kdr_sin$end.points$segment]
points(kdr_sin$end.points[, 1:2], col = col)
emst <- emstreeR::ComputeMST(x = kdr_sin$end.points[, 1:2], verbose = FALSE)
segments(x0 = kdr_sin$end.points[emst$from, 1],
         y0 = kdr_sin$end.points[emst$from, 2],
         x1 = kdr_sin$end.points[emst$to, 1],
         y1 = kdr_sin$end.points[emst$to, 2], lwd = 2)
```

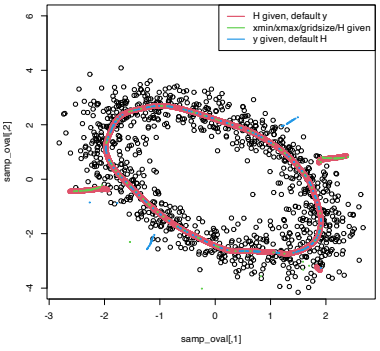
Further options of `ks::kdr` are described below.



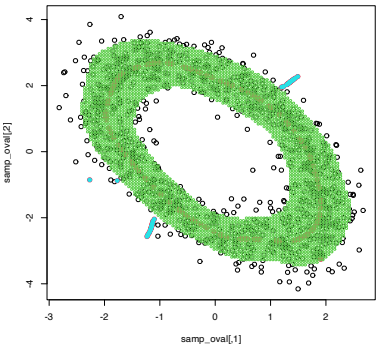

```
# The initial values are chosen automatically, but they can be specified,
# gives faster computations
y <- expand.grid(seq(-3, 3, l = 20), seq(-4, 4, l = 20))

# Use y and save paths
kdr_oval_1 <- ks::kdr(x = samp_oval, y = y, keep.path = TRUE)
plot(samp_oval)
paths <- kdr_oval_1$path
points(kdr_oval_1$y, col = 4, pch = 19, cex = 0.5)
for (i in seq_along(paths)) {

  lines(paths[[i]], col = 4, cex = 0.5)

}
points(kdr_oval_1$end.points, col = 2, pch = 19)
```



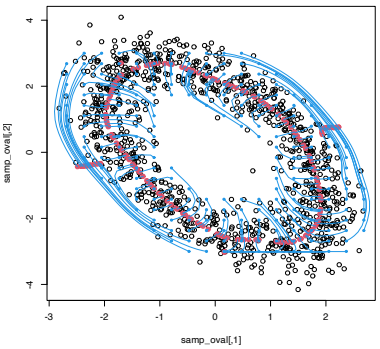

```
length(paths) # Ascent done only for 235 out of the 400 y's
## [1] 201

# By default, ks::kdr employs H = ks::Hpi(..., deriv.order = 2). It can be
# precomputed to reduce the computational cost of ks::kdr(). But care is needed:
# if H is provided, ks::kdr() needs to be called with pre = FALSE to avoid an
# internal scaling of the sample that will result in H being not adequate for
# the scaled sample
H <- ks::Hpi(x = samp_oval, deriv.order = 2)
kdr_oval_1a <- ks::kdr(x = samp_oval, H = H, pre = FALSE, keep.path = TRUE)

# There is a bug in ks 1.13.3 that prevents using pre = FALSE and y at the same
# time. A partial fix is to specify xmin/xmax and gridsize, which will determine
# y, but keeping in mind that these parameters will also affect the precision of
# the Hessian estimation (so if they are too small to save computing time, the
# accuracy of the ridge estimation will decrease)
kdr_oval_1b <- ks::kdr(x = samp_oval, H = H, xmin = c(-3, -4), xmax = c(3, 4),
                       gridsize = c(20, 20), pre = FALSE, keep.path = TRUE)

# Compare different approaches -- same main structure, different end points
# and spurious ridges depending on the size of the initial grid
plot(samp_oval, ylim = c(-4, 6))
points(kdr_oval_1a$end.points[, 1:2], col = 2, pch = 19, cex = 1)
points(kdr_oval_1b$end.points[, 1:2], col = 3, pch = 19, cex = 0.25)
points(kdr_oval_1$end.points[, 1:2], col = 4, pch = 19, cex = 0.25)
legend("topright", legend = c("H given, default y",
                              "xmin/xmax/gridsize/H given",
                              "y given, default H"), col = 2:4, lwd = 2)
```



```
length(kdr_oval_1$path) # y given, default H
## [1] 201
length(kdr_oval_1a$path) # H given, default y
## [1] 9419
length(kdr_oval_1b$path) # xmin/xmax/gridsize/H given
## [1] 201

# If we want to get rid of the points outside the oval, we can identify
# them using the density level set for alpha = 0.15
plot(samp_oval)
points(kdr_oval_1$end.points, col = 2, pch = 19)
alpha <- 0.15
supp <- ks::ksupp(fhat = ks::kde(x = samp_oval, H = H),
                  cont = (1 - alpha) * 100, convex.hull = FALSE)
points(supp, col = 3, cex = 0.5)

# Two ways of excluding the "spurious" ridges: via convex hull and via
# fhat < c_alpha
C <- geometry::convhulln(p = supp)
out_chull <- !geometry::inhulln(ch = C, p =
                                  as.matrix(kdr_oval_1$end.points)[, 1:2])
c_alpha <- quantile(ks::kde(x = samp_oval, H = H,
                            eval.points = samp_oval)$estimate, probs = alpha)
out_kde <- ks::kde(x = samp_oval, H = H, eval.points =
                     kdr_oval_1$end.points[, 1:2])$estimate < c_alpha
points(kdr_oval_1$end.points[out_chull, 1:2], col = 4, cex = 0.75, pch = 19)
points(kdr_oval_1$end.points[out_kde, 1:2], col = 5, cex = 0.75, pch = 19)
```

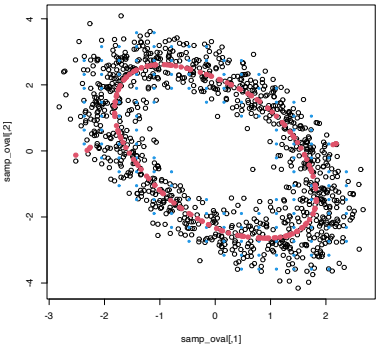



```
# The initial grid can also be specified with xmax, xmin, and gridsize
# (pre = FALSE because H is precomputed)
kdr_oval_2 <- ks::kdr(x = samp_oval, H = H, xmin = c(-3, -3), xmax = c(3, 3),
                      gridsize = c(20, 20), keep.path = TRUE, pre = FALSE)
plot(samp_oval)
points(kdr_oval_2$end.points[, 1:2], col = 2, pch = 19)
paths <- kdr_oval_2$path
points(kdr_oval_2$y, col = 4, pch = 19, cex = 0.5)
for (i in seq_along(paths)) {

  lines(paths[[i]], col = 4, cex = 0.5)

}
```




```
# Save also computing time by increasing density.cutoff
alpha <- 0.5
c_alpha <- quantile(ks::kde(x = samp_oval, H = H,
                            eval.points = samp_oval)$estimate, probs = alpha)
kdr_oval_3 <- ks::kdr(x = samp_oval, y = y, H = H, density.cutoff = c_alpha)
plot(samp_oval)
points(kdr_oval_3$y, col = 4, pch = 19, cex = 0.5)
points(kdr_oval_3$end.points[, 1:2], col = 2, pch = 19)
```

We conclude by showing a real data application of density ridges in the context of earthquakes. The snippet of code adapts the example in the documentation of `ks::kdr`.




```
# Load data
data(quake, package = "ks") # Earthquakes locations
data(plate, package = "ks") # Tectonic plate boundaries

# Select the Pacific Ring of Fire and disregard other variables
# except location of craters
quake <- quake[quake$prof == 1, c("long", "lat")]

# Fix negative longitude
```

```
quake$long[quake$long < 0] <- quake$long[quake$long < 0] + 360

# Select relevant plates
plate <- plate[plate$long < -20 | plate$long > 20, ]
plate$long[plate$long < 0 & !is.na(plate$long)] <-
  plate$long[plate$long < 0 & !is.na(plate$long)] + 360

# Display raw data
maps::map("world2", xlim = c(85, 305), ylim = c(-70, 70),
          mar = c(0, 0, 0, 0), interior = FALSE, lty = 2)
lines(plate[, 1:2], col = 3, lwd = 2)
points(quake, cex = 0.5, pch = 16, col = 2)

# Density ridges
kdr_quake <- ks::kdr(x = quake, xmin = c(70, -70), xmax = c(310, 80))
points(kdr_quake$end.points[, 1:2], cex = 0.5, pch = 16, col = 4)
```

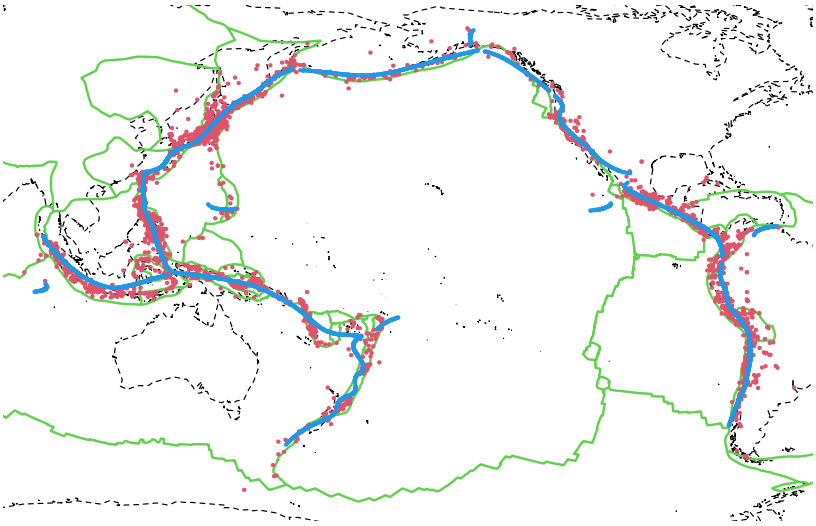


Figure 3.19: Estimated density ridges for the quake dataset. Observe how the density ridges (blue) of the earthquakes locations (red) reveal the relation between earthquakes and tectonic plate boundaries (green).

Exercise 3.32. Apply `ks::kdr` to the `msds.RData` dataset. Investigate the effect of different bandwidths selectors on the resulting estimated ridges trying $\hat{\mathbf{H}}_{\mathrm{NS},r}$ and $\hat{\mathbf{H}}_{\mathrm{PI},r}$, for $r = 0, 1, 2$. Beware! The dataset is large. You may want to precompute bandwidths (use `pre = FALSE` in `ks::kdr()` if `H` is supplied) and be careful on the selection of the initial grid. Are you able to reproduce the main features of the data? Which ones are easier to catch?

4
Kernel regression estimation I

The relation between two random variables X and Y can be completely characterized by their joint cdf F or, equivalently, by their joint pdf f if (X, Y) is continuous. In the regression setting, we are interested in predicting/explaining the *response* Y by means of the *predictor* X from a sample $(X_1, Y_1), \ldots, (X_n, Y_n)$. The role of the variables is not symmetric: X is *used* to predict/explain Y.

We first consider the simplest situation[1]: a single *continuous* predictor X to predict a response Y.[2] In this case, recall that the complete knowledge about Y when $X = x$ is given by the conditional pdf $f_{Y|X=x}(y) = \frac{f(x,y)}{f_X(x)}$. While this pdf provides full knowledge about $Y|X = x$, estimating it is also challenging: for each x we have to estimate a *different curve*! A simpler approach, yet still challenging, is to estimate the conditional mean (a scalar) for each x through the so-called *regression function*

$$m(x) := \mathbb{E}[Y|X = x] = \int y \, \mathrm{d}F_{Y|X=x}(y) = \int y f_{Y|X=x}(y) \, \mathrm{d}y. \quad (4.1)$$

As we will see, this density-based view of the regression function is very useful to motivate estimators.

[1] For the sake of introducing the main concepts in kernel regression estimation. On Chapter 5 we will see more general situations with several predictors, possibly non-continuous.

[2] Formally, the response Y does not need to be continuous. We implicitly assume Y is continuous to use (4.2) as a motivation for (4.5), but the subsequent derivations in the chapter are also valid for non-continuous responses.

4.1 Kernel regression estimation

4.1.1 Nadaraya–Watson estimator

Our objective is to estimate the regression function $m : \mathbb{R} \longrightarrow \mathbb{R}$ nonparametrically. Due to its definition, we can rewrite m in (4.1) as[3]

$$\begin{aligned} m(x) &= \mathbb{E}[Y|X = x] \\ &= \int y f_{Y|X=x}(y) \, \mathrm{d}y \\ &= \frac{\int y f(x, y) \, \mathrm{d}y}{f_X(x)}. \end{aligned} \quad (4.2)$$

[3] Observe that in the third equality we use that (X, Y) is continuous to motivate the construction of the estimator.

Exercise 4.1. Prove that $\mathbb{E}[m(X)] = \mathbb{E}[Y]$ using Proposition 1.1.

Expression (4.2) shows an interesting point: the regression function can be computed from the joint density f and the marginal f_X. Therefore, given a sample $(X_1, Y_1), \ldots, (X_n, Y_n)$, a nonparametric

estimate of m may follow by replacing the previous densities with their kernel density estimators. From the previous section, we know how to do this using the multivariate and univariate kde's given in (2.7) and (3.1), respectively.

For the multivariate kde, we can consider the kde (3.2) based on *product kernels* for the two-dimensional case and bandwidths $\mathbf{h} = (h_1, h_2)'$, that is, the estimate

$$\hat{f}(x, y; \mathbf{h}) = \frac{1}{n} \sum_{i=1}^{n} K_{h_1}(x - X_i) K_{h_2}(y - Y_i) \tag{4.3}$$

of the joint pdf of (X, Y). Besides, considering the same bandwidth h_1 for the kde of f_X, we have

$$\hat{f}_X(x; h_1) = \frac{1}{n} \sum_{i=1}^{n} K_{h_1}(x - X_i). \tag{4.4}$$

We can therefore define the estimator of m that results from replacing f and f_X in (4.2) with (4.3) and (4.4), respectively:

$$\begin{aligned}
\frac{\int y \hat{f}(x, y; \mathbf{h}) \, \mathrm{d}y}{\hat{f}_X(x; h_1)} &= \frac{\int y \frac{1}{n} \sum_{i=1}^{n} K_{h_1}(x - X_i) K_{h_2}(y - Y_i) \, \mathrm{d}y}{\frac{1}{n} \sum_{i=1}^{n} K_{h_1}(x - X_i)} \\
&= \frac{\frac{1}{n} \sum_{i=1}^{n} K_{h_1}(x - X_i) \int y K_{h_2}(y - Y_i) \, \mathrm{d}y}{\frac{1}{n} \sum_{i=1}^{n} K_{h_1}(x - X_i)} \\
&= \frac{\frac{1}{n} \sum_{i=1}^{n} K_{h_1}(x - X_i) Y_i}{\frac{1}{n} \sum_{i=1}^{n} K_{h_1}(x - X_i)} \\
&= \sum_{i=1}^{n} \frac{K_{h_1}(x - X_i)}{\sum_{i=1}^{n} K_{h_1}(x - X_i)} Y_i.
\end{aligned}$$

The resulting estimator[4] is the so-called **Nadaraya–Watson**[5] estimator of the regression function:

$$\hat{m}(x; 0, h) := \sum_{i=1}^{n} \frac{K_h(x - X_i)}{\sum_{j=1}^{n} K_h(x - X_j)} Y_i = \sum_{i=1}^{n} W_i^0(x) Y_i, \tag{4.5}$$

where[6]

$$W_i^0(x) := \frac{K_h(x - X_i)}{\sum_{j=1}^{n} K_h(x - X_j)}.$$

The Nadaraya–Watson estimator can be seen as a **weighted average** of $Y_1, \ldots, Y_n$ by means of the set of weights $\{W_i(x)\}_{i=1}^n$ (they always add to one). The set of *varying* weights depends on the evaluation point x. That means that the Nadaraya–Watson estimator is a **local mean of** $Y_1, \ldots, Y_n$ **about** $X = x$ (see Figure 4.2).

Exercise 4.2. Is it true that the *unconditional* expectation of $\hat{m}(x; 0, h)$, for any x, is $\mathbb{E}[Y]$? That is, is it true that $\mathbb{E}[\hat{m}(x; 0, h)] = \mathbb{E}[Y]$? Prove or disprove the equality.

Let's implement the Nadaraya–Watson estimator from scratch to get a feeling of how it works in practice.

[4] Notice that the estimator does *not* depend on h_2; rather, it depends on h_1, the bandwidth employed for smoothing X.

[5] Termed due to the coetaneous proposals by Nadaraya (1964) and Watson (1964). Also known as the local-constant estimators for reasons explained next.

[6] The change of the sum in the denominator from $\sum_{i=1}^n$ to $\sum_{j=1}^n$ is aimed to avoid confusions with the numerator.

```r
# A naive implementation of the Nadaraya-Watson estimator
nw <- function(x, X, Y, h, K = dnorm) {

  # Arguments
  # x: evaluation points
  # X: vector (size n) with the predictor
  # Y: vector (size n) with the response variable
  # h: bandwidth
  # K: kernel

  # Matrix of size n x length(x) (rbind() is called for ensuring a matrix
  # output if x is a scalar)
  Kx <- rbind(sapply(X, function(Xi) K((x - Xi) / h) / h))

  # Weights
  W <- Kx / rowSums(Kx) # Column recycling!

  # Means at x ("drop" to drop the matrix attributes)
  drop(W %*% Y)

}

# Generate some data to test the implementation
set.seed(12345)
n <- 100
eps <- rnorm(n, sd = 2)
m <- function(x) x^2 * cos(x)
# m <- function(x) x - x^2 # Works equally well for other regression function
X <- rnorm(n, sd = 2)
Y <- m(X) + eps
x_grid <- seq(-10, 10, l = 500)

# Bandwidth
h <- 0.5

# Plot data
plot(X, Y)
rug(X, side = 1); rug(Y, side = 2)
lines(x_grid, m(x_grid), col = 1)
lines(x_grid, nw(x = x_grid, X = X, Y = Y, h = h), col = 2)
legend("top", legend = c("True regression", "Nadaraya-Watson"),
       lwd = 2, col = 1:2)
```

Exercise 4.3. Implement in R your own version of the Nadaraya–Watson estimator and compare it with the `nw` function. Focus only on the normal kernel. You may reduce the accuracy of the final computation up to `1e-7` to achieve better efficiency. How much are you able to improve the speed of `nw`? Use the `microbenchmark::microbenchmark` function to measure the running times for a sample of size $n = 10,000$.

Similarly to kernel density estimation, in the Nadaraya–Watson estimator the bandwidth has a prominent effect on the shape of the estimator, whereas the kernel is clearly less important. The code below illustrates the effect of varying h using the `manipulate::manipulate` function.

```r
# Simple plot of N-W for varying h's
manipulate::manipulate({

  # Plot data
  plot(X, Y)
  rug(X, side = 1); rug(Y, side = 2)
```

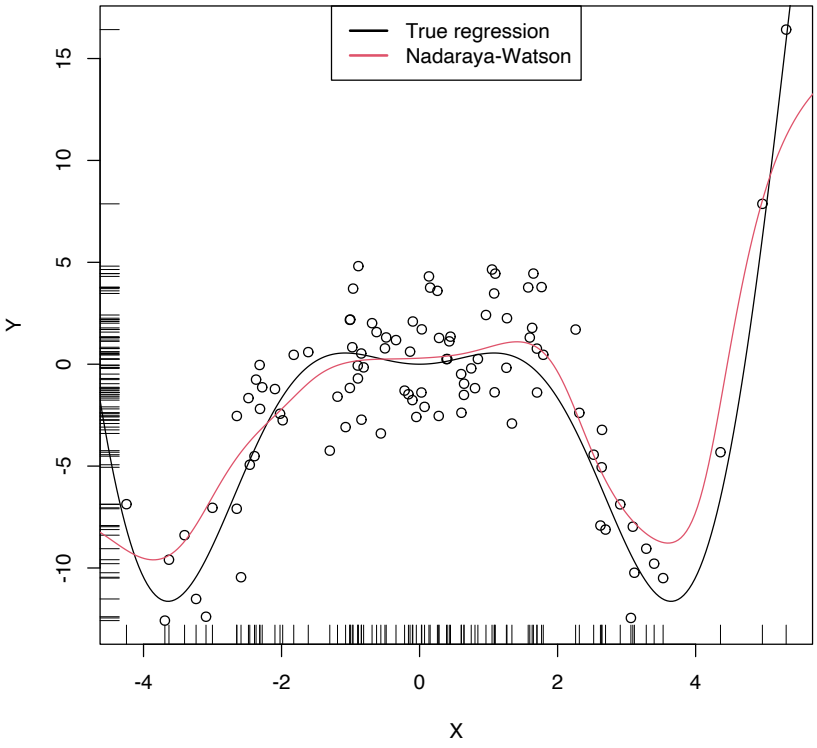


Figure 4.1: The Nadaraya–Watson estimator of an arbitrary regression function m. Observe how the Nadaraya–Watson estimator is able to estimate the nonlinear form of the regression function without prior knowledge about its shape.

```
  lines(x_grid, m(x_grid), col = 1)
  lines(x_grid, nw(x = x_grid, X = X, Y = Y, h = h), col = 2)
  legend("topright", legend = c("True regression", "Nadaraya-Watson"),
         lwd = 2, col = 1:2)

}, h = manipulate::slider(min = 0.01, max = 10, initial = 0.5, step = 0.01))
```

4.1.2 *Local polynomial estimator*

The Nadaraya–Watson estimator can be seen as a particular case of a wider class of nonparametric estimators, the so-called *local polynomial estimators*. Specifically, Nadaraya–Watson is the one that corresponds to performing a *local constant fit*. Let's see this wider class of nonparametric estimators and their advantages with respect to the Nadaraya–Watson estimator.

A motivation for the local polynomial fit comes from attempting to find an estimator $\hat{m}$ of m that "minimizes"[7] the RSS

$$\sum_{i=1}^{n}(Y_i - \hat{m}(X_i))^2 \tag{4.6}$$

without assuming any particular form for the underlying m. This is not achievable directly, since no knowledge about m is available. Recall that what it was done in parametric models, such as linear regression (see Section B.1), was to *assume a parametrization* for m (e.g., $m_\beta(\mathbf{x}) = \beta_0 + \beta_1 x$ for the simple linear model) which allowed tackling the minimization of (4.6) by means of solving

$$m_{\hat{\beta}}(\mathbf{x}) := \arg\min_{\beta} \sum_{i=1}^{n}(Y_i - m_\beta(X_i))^2.$$

[7] Obviously, avoiding the spurious perfect fit attained with $\hat{m}(X_i) := Y_i$, $i = 1, \ldots, n$.

When m has no parametrization available and can adopt any mathematical form, an alternative approach is required. The first step is to induce a *local parametrization* on m. By a p-th[8] order Taylor expression it is possible to obtain that, for x close to X_i,

$$m(X_i) \approx m(x) + m'(x)(X_i - x) + \frac{m''(x)}{2}(X_i - x)^2 + \cdots + \frac{m^{(p)}(x)}{p!}(X_i - x)^p. \tag{4.7}$$

[8] Here we employ p to denote the order of the Taylor expansion (Theorem 1.11) and, correspondingly, the order of the associated polynomial fit. Do not confuse p with the number of *original* predictors for explaining Y – there is one predictor only, X. However, with a local polynomial fit we expand this predictor to p predictors based on $(X^1, X^2, \ldots, X^p)$.

Then, plugging (4.7) in the population version of (4.6) that replaces $\hat{m}$ with m, we have that

$$\sum_{i=1}^{n}\left(Y_i - \sum_{j=0}^{p}\frac{m^{(j)}(x)}{j!}(X_i - x)^j\right)^2. \tag{4.8}$$

This expression is still not workable: it depends on $m^{(j)}(x)$, $j = 0, \ldots, p$, which of course are unknown, as m is unknown. The **key idea** now is:

> Set $\beta_j := \frac{m^{(j)}(x)}{j!}$ and turn (4.8) into a linear regression problem where the unknown parameters are precisely $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)'$.

Simply rewriting (4.8) using this idea gives

$$\sum_{i=1}^{n}\left(Y_i - \sum_{j=0}^{p}\beta_j(X_i - x)^j\right)^2. \tag{4.9}$$

Now, estimates for $\boldsymbol{\beta}$ automatically produce estimates for $m^{(j)}(x)$, $j = 0, \ldots, p$. In addition, we know how to obtain an estimate $\hat{\boldsymbol{\beta}}$ that minimizes (4.9), since this is precisely the least squares problem studied in linear models.[9] The final touch is to **weight the contributions of each datum** (X_i, Y_i) to the estimation of $m(x)$ according to the proximity of X_i to x.[10] We can achieve this precisely with kernels:

$$\hat{\boldsymbol{\beta}}_h := \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^{p+1}}\sum_{i=1}^{n}\left(Y_i - \sum_{j=0}^{p}\beta_j(X_i - x)^j\right)^2 K_h(x - X_i). \tag{4.10}$$

[9] See (B.4) in Section B.1.1.

[10] The rationale is simple: (X_i, Y_i) should be more informative about $m(x)$ than (X_j, Y_j) if x and X_i are closer than x and X_j.

Solving (4.10) is easy once the proper notation is introduced. To that end, denote

$$\mathbf{X} := \begin{pmatrix} 1 & X_1 - x & \cdots & (X_1 - x)^p \\ \vdots & \vdots & \ddots & \vdots \\ 1 & X_n - x & \cdots & (X_n - x)^p \end{pmatrix}_{n\times(p+1)}$$

and

$$\mathbf{W} := \operatorname{diag}(K_h(X_1 - x), \ldots, K_h(X_n - x)), \quad \mathbf{Y} := \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}_{n\times 1}.$$

Then we can re-express (4.10) into a *weighted least squares problem*, which has an exact solution:

$$\hat{\boldsymbol{\beta}}_h = \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^{p+1}} (\mathbf{Y}-\mathbf{X}\boldsymbol{\beta})'\mathbf{W}(\mathbf{Y}-\mathbf{X}\boldsymbol{\beta}) \tag{4.11}$$

$$= (\mathbf{X}'\mathbf{W}\mathbf{X})^{-1}\mathbf{X}'\mathbf{W}\mathbf{Y}. \tag{4.12}$$

The estimate[11] of $m(x)$ is therefore computed as[12]

$$\begin{aligned}\hat{m}(x;p,h) &:= \hat{\beta}_{h,0}\\ &= \mathbf{e}_1'(\mathbf{X}'\mathbf{W}\mathbf{X})^{-1}\mathbf{X}'\mathbf{W}\mathbf{Y}\\ &= \sum_{i=1}^n W_i^p(x)Y_i,\end{aligned} \tag{4.13}$$

where

$$W_i^p(x) := \mathbf{e}_1'(\mathbf{X}'\mathbf{W}\mathbf{X})^{-1}\mathbf{X}'\mathbf{W}\mathbf{e}_i$$

and $\mathbf{e}_i$ is the i-th canonical vector.[13] Just as the Nadaraya–Watson was, the local polynomial estimator is a **weighted linear combination of the responses**.[14]

Two cases on (4.13) deserve special attention:

- $p = 0$ is the **local constant estimator**, previously referred to as the Nadaraya–Watson estimator. In this situation, the estimator has explicit weights, as we saw before:

$$W_i^0(x) = \frac{K_h(x-X_i)}{\sum_{j=1}^n K_h(x-X_j)}. \tag{4.14}$$

- $p = 1$ is the **local linear estimator**, which has weights equal to

$$W_i^1(x) = \frac{1}{n}\frac{\hat{s}_2(x;h) - \hat{s}_1(x;h)(X_i-x)}{\hat{s}_2(x;h)\hat{s}_0(x;h) - \hat{s}_1(x;h)^2}K_h(x-X_i), \tag{4.15}$$

  where $\hat{s}_r(x;h) := \frac{1}{n}\sum_{i=1}^n (X_i-x)^r K_h(x-X_i)$.

Exercise 4.4. Show that (4.12) is the solution of (4.11), $\hat{\boldsymbol{\beta}}_h$. To do so, first prove Exercise B.1.

Exercise 4.5. Show that the local polynomial estimator yields the Nadaraya–Watson estimator when $p = 0$. Use (4.12) to obtain (4.5).

Exercise 4.6. Prove that:

1. $\sum_{i=1}^n W_i^0(x) = 1$ and $W_i^0(x) \geq 0$ for all $i = 1,\ldots,n$ (weighted mean).
2. $\sum_{i=1}^n W_i^1(x) = 1$ and $W_i^1(x) \not\geq 0$ for all $i = 1,\ldots,n$ (not a weighted mean). Give an example of positive and negative weights.

Exercise 4.7. Obtain the weight expressions (4.15) of the local linear estimator using the matrix inversion formula for 2×2 matrices:

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

[11] Recall that the entries of $\hat{\boldsymbol{\beta}}_h$ are estimating $\boldsymbol{\beta} = \left(m(x), m'(x), \frac{m''(x)}{2}, \ldots, \frac{m^{(p)}(x)}{p!}\right)'$, so we are indeed estimating $m(x)$ (first entry) *and*, in addition, its derivatives up to order p.

[12] An alternative and useful view is that, by minimizing (4.10), we are fitting the linear model $\hat{m}_x(t) := \hat{\beta}_{h,0} + \hat{\beta}_{h,1}(t-x) + \cdots + \hat{\beta}_{h,p}(t-x)^p$ that is *centered* about x. Then, we employ this model to predict Y for $X = t = x$, resulting $\hat{\beta}_{h,0}$.

[13] That is, the entries of $\mathbf{e}_i$ are all zero except for the i-th one, which is 1.

[14] Which is *not* a weighted mean in general, only if $p = 0$; see Exercise 4.6.

Remark. The local polynomial fit is computationally **more expensive** than the local constant fit: $\hat{m}(x;p,h)$ is obtained as the solution of a weighted linear problem, whereas $\hat{m}(x;0,h)$ can directly be computed as a weighted mean of the responses. Even if the explicit form (4.15) is available when $p=1$, (4.15) is roughly three times more costly than (4.14) (recall the $\hat{s}_r(x;h)$, $r=0,1,2$). The computational demand escalates with p, since computing $\hat{m}(x;p,h)$ requires inverting the $p\times p$ matrix $\mathbf{X}'\mathbf{W}\mathbf{X}$.

The local polynomial estimator $\hat{m}(\cdot;p,h)$ of m performs a **series of weighted polynomial fits**, as many as points x on which $\hat{m}(\cdot;p,h)$ is to be evaluated.

Figure 4.2 illustrates the construction of the local polynomial estimator (up to cubic degree) and shows how $\hat{\beta}_0=\hat{m}(x;p,h)$, the intercept of the local fit, estimates m at x by constructing a local polynomial fit in the neighborhood of x.

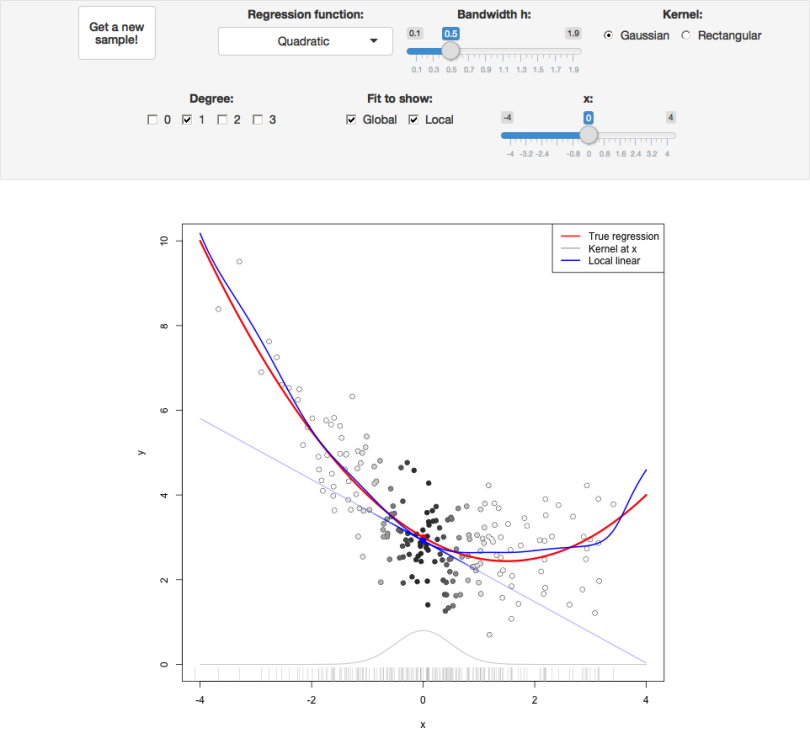


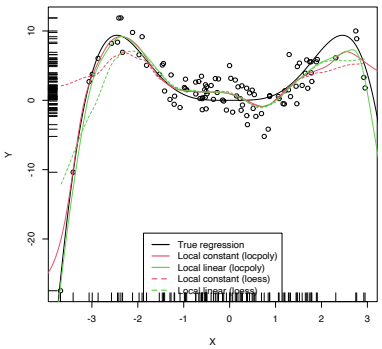
Figure 4.2: Construction of the local polynomial estimator. The animation shows how local polynomial fits in a neighborhood of x are combined to provide an estimate of the regression function, which depends on the polynomial degree, bandwidth, and kernel (gray density at the bottom). The data points are shaded according to their weights for the local fit at x. Application available here.

An inefficient implementation of the local polynomial estimator can be done relatively straightforwardly from the previous insights and from expression (4.12). However, several R packages provide implementations, such as `KernSmooth::locpoly` and R's `loess`[15] (but this one has a different control of the bandwidth plus a set of other modifications). Below are some examples of their use.

```
# Generate some data
set.seed(123456)
n <- 100
eps <- rnorm(n, sd = 2)
m <- function(x) x^3 * sin(x)
X <- rnorm(n, sd = 1.5)
Y <- m(X) + eps
```

[15] The `lowess` estimator, related to `loess`, is the one employed in R's `panel.smooth`, which is the function in charge of displaying the smooth fits in `lm` and `glm` regression diagnostics (employing a *prefixed* and not data-driven smoothing span of 2/3 – which makes it inevitably a bad choice for certain data patterns).

```
x_grid <- seq(-10, 10, l = 500)

# KernSmooth::locpoly fits
h <- 0.25
lp0 <- KernSmooth::locpoly(x = X, y = Y, bandwidth = h, degree = 0,
                           range.x = c(-10, 10), gridsize = 500)
lp1 <- KernSmooth::locpoly(x = X, y = Y, bandwidth = h, degree = 1,
                           range.x = c(-10, 10), gridsize = 500)
# Provide the evaluation points through range.x and gridsize

# loess fits
span <- 0.25 # The default span is 0.75, which works very bad in this scenario
lo0 <- loess(Y ~ X, degree = 0, span = span)
lo1 <- loess(Y ~ X, degree = 1, span = span)
# loess employs a "span" argument that plays the role of a variable bandwidth
# "span" gives the proportion of points of the sample that are taken into
# account for performing the local fit about x and then uses a triweight kernel
# (not a normal kernel) for weighting the contributions. Therefore, the final
# estimate differs from the definition of local polynomial estimator, although
# the principles in which are based are the same

# Prediction at x = 2
x <- 2
lp1$y[which.min(abs(lp1$x - x))] # Prediction by KernSmooth::locpoly
## [1] 5.445975
predict(lo1, newdata = data.frame(X = x)) # Prediction by loess
##        1
## 5.379652
m(x) # True regression
## [1] 7.274379

# Plot data
plot(X, Y)
rug(X, side = 1); rug(Y, side = 2)
lines(x_grid, m(x_grid), col = 1)
lines(lp0$x, lp0$y, col = 2)
lines(lp1$x, lp1$y, col = 3)
lines(x_grid, predict(lo0, newdata = data.frame(X = x_grid)), col = 2, lty = 2)
lines(x_grid, predict(lo1, newdata = data.frame(X = x_grid)), col = 3, lty = 2)
legend("bottom", legend = c("True regression", "Local constant (locpoly)",
                            "Local linear (locpoly)", "Local constant (loess)",
                            "Local linear (loess)"),
       lwd = 2, col = c(1:3, 2:3), lty = c(rep(1, 3), rep(2, 2)))
```



Exercise 4.8. Perform the following tasks:

a. Implement your own version of the local linear estimator. The function must take a sample `X`, a sample `Y`, the points `x` at which the estimate is to be obtained, the bandwidth `h`, and the kernel `K`.
b. Test its correct behavior by estimating an artificial dataset (e.g., the one considered in Exercise 4.16) that follows a linear model.

As with the Nadaraya–Watson, the local polynomial estimator heavily depends on h.

```
# Simple plot of local polynomials for varying h's
manipulate::manipulate({

  # Plot data
  lpp <- KernSmooth::locpoly(x = X, y = Y, bandwidth = h, degree = p,
                             range.x = c(-10, 10), gridsize = 500)
  plot(X, Y)
  rug(X, side = 1); rug(Y, side = 2)
  lines(x_grid, m(x_grid), col = 1)
  lines(lpp$x, lpp$y, col = p + 2)
```

```
  legend("bottom", legend = c("True regression", "Local polynomial fit"),
         lwd = 2, col = c(1, p + 2))

}, p = manipulate::slider(min = 0, max = 4, initial = 0, step = 1),
h = manipulate::slider(min = 0.01, max = 10, initial = 0.5, step = 0.01))
```

A more sophisticated framework for performing nonparametric estimation of the regression function is the `np` package, which is introduced in detail in Section 4.5. This package will be the approach chosen for the more challenging situation in which several predictors are present, since the former implementations (`KernSmooth::locpoly`, `loess`, and our own implementations) do not scale well for more than one predictor.

Exercise 4.9. Perform the following tasks:

a. Code your own implementation of the local cubic estimator. The function must take as input the vector of evaluation points `x`, the sample `data`, and the bandwidth `h`. Use the normal kernel. The result must be a vector of the same length as `x` containing the estimator evaluated at `x`.
b. Test the implementation by estimating the regression function in the location model $Y = m(X) + \varepsilon$, where $m(x) = (x-1)^2$, $X \sim \mathcal{N}(1,1)$, and $\varepsilon \sim \mathcal{N}(0, 0.5)$. Do it for a sample of size $n = 500$.

4.2 *Asymptotic properties*

The asymptotic properties of the local polynomial estimator give us valuable insights into its performance. In particular, they allow answering, precisely, the following questions:

> What affects the performance of the local polynomial estimator? Is local linear estimation better than local constant estimation? What is the effect of h on the estimates?

The asymptotic analysis of the local linear and local constant estimators[16] is achieved, as done in Sections 2.3 and 3.3, by examining the asymptotic bias and variance.

In order to establish a framework for the analysis, we consider the so-called *location-scale model* for Y and its predictor X:

$$Y = m(X) + \sigma(X)\varepsilon,$$

where

$$\sigma^2(x) := \mathbb{V}\mathrm{ar}[Y|X = x]$$

is the *conditional variance* of Y given X, and ε is such that $\mathbb{E}[\varepsilon] = 0$ and $\mathbb{V}\mathrm{ar}[\varepsilon] = 1$. Recall that, since the conditional variance is not forced to be constant, we are implicitly allowing for *heteroskedasticity*.[17]

Note that for the derivation of the Nadaraya–Watson estimator and the local polynomial estimator we did not assume any particular assumption, beyond the (implicit) differentiability of m up

[16] We do not address the analysis of the general case in which $p \geq 1$. The reader is referred to, for example, Theorem 3.1 in Fan and Gijbels (1996) for the full analysis.

[17] In linear models, homoscedasticity is one of the key assumptions for performing inference (Section B.1.2).

to order p for the local polynomial estimator. The following assumptions[18] are the only requirements to perform the asymptotic analysis of the estimator:

- **A1**.[19] m is twice continuously differentiable.
- **A2**.[20] σ^2 is continuous and positive.
- **A3**.[21] f, the marginal pdf of X, is continuously differentiable and *bounded away from zero*.[22]
- **A4**.[23] The kernel K is a symmetric and bounded pdf with finite second moment and is square integrable.
- **A5**.[24] $h = h_n$ is a deterministic sequence of bandwidths such that, when $n \to \infty$, $h \to 0$ and $nh \to \infty$.

The bias and variance are studied in their *conditional* versions on the predictor's sample $X_1, \ldots, X_n$. The reason for analyzing the conditional instead of the *unconditional* versions is to avoid technical difficulties that integration with respect to the unknown predictor's density may pose. This is in the spirit of what was done in parametric inference (observe Sections B.1.2 and B.2.2).

The main result follows. It provides useful insights into the effect of p, m, f (standing from now on for the marginal pdf of X), and σ^2 in the performance of $\hat{m}(\cdot; p, h)$ for $p = 0, 1$.

Theorem 4.1. *Under **A1**–**A5**, the conditional bias and variance of the local constant ($p = 0$) and local linear ($p = 1$) estimators are*

$$\mathrm{Bias}[\hat{m}(x; p, h)|X_1, \ldots, X_n] = B_p(x)h^2 + o_{\mathbb{P}}(h^2), \tag{4.16}$$

$$\mathbb{V}\mathrm{ar}[\hat{m}(x; p, h)|X_1, \ldots, X_n] = \frac{R(K)}{nhf(x)}\sigma^2(x) + o_{\mathbb{P}}((nh)^{-1}), \tag{4.17}$$

where

$$B_p(x) := \begin{cases} \frac{\mu_2(K)}{2}\left\{m''(x) + 2\frac{m'(x)f'(x)}{f(x)}\right\}, & \text{if } p = 0, \\ \frac{\mu_2(K)}{2}m''(x), & \text{if } p = 1. \end{cases}$$

Remark. The little-$o_{\mathbb{P}}$s in (4.16) and (4.17) appear (instead of little-os as in Theorem 2.1) because $\mathrm{Bias}[\hat{m}(x; p, h)|X_1, \ldots, X_n]$ and $\mathbb{V}\mathrm{ar}[\hat{m}(x; p, h)|X_1, \ldots, X_n]$ are *random variables*. Then, the asymptotic expansions of these random variables have stochastic remainders that converge to zero *in probability* at specific rates.

The bias and variance expressions (4.16) and (4.17) yield very interesting insights:

1. Bias:
   - The **bias decreases with** h ***quadratically*** for both $p = 0, 1$. That means that small bandwidths h give estimators with low bias, whereas large bandwidths provide largely biased estimators.
   - For $p = 1$, the bias at x is directly proportional to $m''(x)$. Therefore:

[18] Recall that these are the only assumptions done in the model so far. Compared with the ones linear models or generalized linear models make, they are extremely mild. Recall Y is not assumed to be continuous.

[19] This assumption requires certain smoothness of the regression function, allowing thus for Taylor expansions to be performed. This assumption is important in practice: $\hat{m}(\cdot; p, h)$ is infinitely differentiable if the considered kernels K are so too.

[20] It avoids the situation in which Y is a degenerated random variable.

[21] It avoids the degenerate situation in which m is estimated at regions without observations of the predictors (such as *holes* in the support of X).

[22] Meaning that there exists a positive lower bound for f.

[23] Mild assumption inherited from the kde.

[24] Key assumption for reducing the bias and variance of $\hat{m}(\cdot; p, h)$ *simultaneously*.

  - The *bias is negative* in regions where m is concave, i.e., $\{x \in \mathbb{R} : m''(x) < 0\}$. These regions correspond to *peaks and local maxima of* m.
  - Conversely, the *bias is positive* in regions where m is convex, i.e., $\{x \in \mathbb{R} : m''(x) > 0\}$. These regions correspond to *valleys and local minima of* m.
  - All in all, **the "wilder" the curvature** of m, the larger the bias and **the harder to estimate** m.
- For $p = 0$, the bias at x is more convoluted and is affected by $m''(x)$, $m'(x)$, $f'(x)$, and $f(x)$:
  - The quantities $m'(x)$, $f'(x)$, and $f(x)$ are not present in the bias when $p = 1$. Precisely, for the local constant estimator, the lower the density $f(x)$, the larger the bias (in absolute value). Also, the faster m and f change at x (derivatives), the larger the bias. Thus **the bias of the local constant estimator is much more sensitive to** $m(x)$ **and** $f(x)$ than the local linear (which is sensitive to $m''(x)$ only). Particularly, the fact that it depends on $f'(x)$ and $f(x)$ is referred to as the *design bias* since it depends merely on the predictor's distribution.
  - As for $p = 1$, $m''(x)$ contributes to the bias when $p = 0$, this contribution being *negative* in regions corresponding to peaks and local maxima of m, and *positive* in the valleys and local minima of m. In general, the "wilder" the curvature of m, the larger its contribution to the bias and the harder to estimate m.

2. Variance:

- The main term of the **variance is the same for** $p = 0, 1$. In addition, it depends directly on $\frac{\sigma^2(x)}{f(x)}$. As a consequence, the lower the density, the more variable $\hat{m}(x; p, h)$ is.[25] Also, the larger the conditional variance at x, $\sigma^2(x)$, the more variable $\hat{m}(x; p, h)$ is.[26]
- The **variance decreases as a factor of** $(nh)^{-1}$. This is related to the so-called *effective sample size* nh, which can be thought of as the amount of data in the neighborhood of x that is employed for performing the regression.[27]

[25] Recall that this makes perfect sense: low-density regions of X imply less information available about m.

[26] The same happened in the linear model with the error variance σ^2.

[27] The variance of an unweighted mean is reduced by a factor n^{-1} when n observations are employed. To compute $\hat{m}(x; p, h)$, n observations are used but in a *weighted* fashion that roughly amounts to considering nh *unweighted* observations.

All in all, the main takeaway of the analysis of $p = 0$ vs. $p = 1$ is:

> $p = 1$ **has, in general, smaller bias than that of** $p = 0$ (but of the same order) while **keeping the same variance as** $p = 0$.

An extended version of Theorem 4.1, given in Theorem 3.1 in Fan and Gijbels (1996), shows that this phenomenon extends to higher orders: **odd order** ($p = 2\nu + 1, \nu \in \mathbb{N}$) polynomial fits introduce an extra coefficient for the polynomial fit that allows them to **reduce the bias**, while maintaining the **same variance** of the *precedent*[28] even order ($p = 2\nu$). So, for example, local cubic fits

[28] Since the variance increases as ν does, not as p does.

are preferred to local quadratic fits. This motivates the claim that *local polynomial fitting is an odd world* (Fan and Gijbels (1996)).

Finally, we have the asymptotic pointwise normality of the estimator, an analogous result to Theorem 2.2 which is helpful to obtain pointwise confidence intervals for m afterwards.

Theorem 4.2. *Assume that* $\mathbb{E}[(Y-m(x))^{2+\delta}|X=x]<\infty$ *for some* $\delta>0$. *Then, under* ***A1*****–*****A5***,

$$\sqrt{nh}(\hat{m}(x;p,h)-\mathbb{E}[\hat{m}(x;p,h)|X_1,\ldots,X_n]) \xrightarrow{d} \mathcal{N}\left(0,\frac{R(K)\sigma^2(x)}{f(x)}\right), \tag{4.18}$$

$$\sqrt{nh}\left(\hat{m}(x;p,h)-m(x)-B_p(x)h^2\right) \xrightarrow{d} \mathcal{N}\left(0,\frac{R(K)\sigma^2(x)}{f(x)}\right). \tag{4.19}$$

Exercise 4.10. Theorem 4.1 gives some additional insights with respect to $B_p(x)$, the dominating term of the bias:

1. If m is constant[29], then $B_0(x)=0$.
2. If m is linear[30], then $B_1(x)=0$.

That is, for each of these two cases, $\text{Bias}[\hat{m}(x;p,h)|X_1,\ldots,X_n]=o_{\mathbb{P}}(h^2)$. The local constant and local linear estimators are actually *exactly* unbiased when estimating constant and linear regression functions, respectively. That is, $\mathbb{E}_c[\hat{m}(x;0,h)|X_1,\ldots,X_n]=c$ and $\mathbb{E}_{a,b}[\hat{m}(x;1,h)|X_1,\ldots,X_n]=ax+b$, where $\mathbb{E}_c[\cdot|X_1,\ldots,X_n]$ and $\mathbb{E}_{a,b}[\cdot|X_1,\ldots,X_n]$ represent the conditional expectations under the constant and linear models, respectively. Prove these two results.

[29] $m(x)=c$ for all $x\in\mathbb{R}$ and given $c\in\mathbb{R}$.
[30] $m(x)=ax+b$ for all $x\in\mathbb{R}$ and given $a,b\in\mathbb{R}$.

4.3 *Bandwidth selection*

Bandwidth selection, as for kernel density estimation, is of key practical importance for kernel regression estimation. Several bandwidth selectors have been proposed for kernel regression by following plug-in and cross-validatory ideas that are similar to the ones seen in Section 2.4. For the sake of simplicity, we first briefly overview the plug-in analogues for local linear regression for a single continuous predictor. Then, the main focus will be placed on **Least Squares Cross-Validation** (LSCV; simply referred to as CV), as the cross-validation methodology readily generalizes to the more complex settings to be seen in Section 5.1.

As in the kde case, the first step for performing bandwidth selection is to define an error criterion suitable for the estimator $\hat{m}(\cdot;p,h)$. The density-weighted ISE of $\hat{m}(\cdot;p,h)$,

$$\text{ISE}[\hat{m}(\cdot;p,h)]:=\int(\hat{m}(x;p,h)-m(x))^2f(x)\,\mathrm{d}x,$$

is often considered. Observe that this definition is very similar to the kde's ISE, except for the fact that f appears weighting the quadratic difference: what matters is to minimize the estimation error on the regions where the density of X is higher. As a consequence, this definition does not give any importance to the errors

made by $\hat{m}(\cdot; p, h)$ when estimating m at regions with virtually no data.[31]

The ISE of $\hat{m}(\cdot; p, h)$ is a random quantity that directly depends on the sample $(X_1, Y_1), \ldots, (X_n, Y_n)$. In order to avoid this nuisance, the *conditional*[32] MISE of $\hat{m}(\cdot; p, h)$ is often employed:

$$\begin{aligned}\operatorname{MISE}[\hat{m}(\cdot; p, h)|X_1, \ldots, X_n] &:= \mathbb{E}\left[\operatorname{ISE}[\hat{m}(\cdot; p, h)]|X_1, \ldots, X_n\right] \\ &= \int \mathbb{E}\left[(\hat{m}(x; p, h) - m(x))^2|X_1, \ldots, X_n\right] f(x)\,\mathrm{d}x \\ &= \int \operatorname{MSE}\left[\hat{m}(x; p, h)|X_1, \ldots, X_n\right] f(x)\,\mathrm{d}x.\end{aligned}$$

Clearly, the goal is to find the bandwidth that minimizes this error,

$$h_{\operatorname{MISE}} := \arg\min_{h>0} \operatorname{MISE}[\hat{m}(\cdot; p, h)|X_1, \ldots, X_n],$$

but this is an untractable problem because of the lack of explicit expressions. However, since the MISE follows by integrating the conditional MSE, explicit asymptotic expressions can be found.

In the case of local linear regression, the conditional MSE follows by the conditional squared bias (4.16) and the variance (4.17) given in Theorem 4.1. This produces the conditional AMISE for $\hat{m}(\cdot; p, h)$, $p = 0, 1$:

$$\begin{aligned}\operatorname{AMISE}[\hat{m}(\cdot; p, h)|X_1, \ldots, X_n] = h^4 &\int B_p(x)^2 f(x)\,\mathrm{d}x \\ &+ \frac{R(K)}{nh}\int \sigma^2(x)\,\mathrm{d}x. \qquad (4.20)\end{aligned}$$

If $p = 1$, the resulting optimal AMISE bandwidth is

$$h_{\operatorname{AMISE}} = \left[\frac{R(K)\int \sigma^2(x)\,\mathrm{d}x}{\mu_2^2(K)\theta_{22}(m)n}\right]^{1/5}, \qquad (4.21)$$

where

$$\theta_{22}(m) := \int (m''(x))^2 f(x)\,\mathrm{d}x$$

acts as the "density-weighted curvature of m" and completely resembles the curvature term $R(f'')$ that appeared in Section 2.4.

Exercise 4.11. Obtain $\operatorname{AMISE}[\hat{m}(\cdot; p, h)|X_1, \ldots, X_n]$ in (4.20) from Theorem 4.1.

Exercise 4.12. Obtain the expression for h_{AMISE} in (4.21) from $\operatorname{AMISE}[\hat{m}(\cdot; 1, h)|X_1, \ldots, X_n]$. Then, obtain the corresponding AMISE optimal bandwidth for $\hat{m}(\cdot; 0, h)$.

As happened in the density setting, the AMISE-optimal bandwidth can not be readily employed, as knowledge about the curvature of m, $\theta_{22}(m)$, is required. To make things worse, (4.21) also depends on the integrated conditional variance $\int \sigma^2(x)\,\mathrm{d}x$, which is unknown too.

[31] Estimating m at regions with no data is an *extrapolation* problem. This kind of estimation makes better sense if a specific structure for m is assumed (which is *not* done with a nonparametric estimate of m), so that this structure can determine the estimate of m at the regions with no data.

[32] The conditioning is on the sample of the predictor, as it is usually done in a regression setting. The reason why this conditioning is done is to avoid the analysis of the randomness of X.

4.3.1 *Plug-in rules*

A possibility to estimate $\theta_{22}(m)$ and $\int \sigma^2(x)\,\mathrm{d}x$, in the spirit of the **normal scale bandwidth selector** (or zero-stage plug-in selector) seen in Section 2.4.1, is presented in Section 4.2 in Fan and Gijbels (1996). There, a *global* parametric fit based on a fourth-order[33] polynomial is employed:

$$\hat{m}_Q(x) = \hat{\alpha}_0 + \sum_{j=1}^{4} \hat{\alpha}_j x^j,$$

where $\hat{\boldsymbol{\alpha}}$ is obtained from $(X_1, Y_1), \ldots, (X_n, Y_n)$. The second derivative of this quartic fit is

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

Besides, $\theta_{22}(m)$ can be written in terms of an expectation, which motivates its estimation by Monte Carlo:

$$\theta_{22}(m) = \mathbb{E}[(m''(X))^2] \approx \frac{1}{n}\sum_{i=1}^{n}(m''(X_i))^2 =: \hat{\theta}_{22}(m).$$

Therefore, combining both estimations we can substitute $\theta_{22}(m)$ with $\hat{\theta}_{22}(\hat{m}_Q)$ in (4.21).

It remains to estimate $\int \sigma^2(x)\,\mathrm{d}x$, which we can do by assuming homoscedasticity and then estimating the common variance by[34]

$$\hat{\sigma}_Q^2 := \frac{1}{n-5}\sum_{i=1}^{n}(Y_i - \hat{m}_Q(X_i))^2.$$

In addition, if we assume that the variance is zero outside the support of X (where there is no data), we can replace $\int \sigma^2(x)\,\mathrm{d}x$ with the estimate $(X_{(n)} - X_{(1)})\hat{\sigma}_Q^2$ [35] and obtain the **Rule-of-Thumb bandwidth selector** (RT)

$$\hat{h}_{\mathrm{RT}} = \left[\frac{R(K)(X_{(n)} - X_{(1)})\hat{\sigma}_Q^2}{\mu_2^2(K)\hat{\theta}_{22}(\hat{m}_Q)n}\right]^{1/5}.$$

The following code illustrates how to implement this simple selector.

```
# Evaluation grid
x_grid <- seq(0, 5, l = 500)

# Quartic-like regression function
m <- function(x) 5 * dnorm(x, mean = 1.5, sd = 0.25) - x

# # Highly nonlinear regression function
# m <- function(x) x * sin(2 * pi * x)

# # Quartic regression function (but expressed in a orthogonal polynomial)
# coefs <- attr(poly(x_grid / 5 * 3, degree = 4), "coefs")
# m <- function(x) 20 * poly(x, degree = 4, coefs = coefs)[, 4]

# # Seventh orthogonal polynomial
# coefs <- attr(poly(x_grid / 5 * 3, degree = 7), "coefs")
# m <- function(x) 20 * poly(x, degree = 7, coefs = coefs)[, 7]
```

[33] In general, a polynomial fit of order $p+3$ can be considered, but we are restricting to $p = 1$. The motivation for this order is that it has to be able to estimate the $(p+1)$-th derivative of m at x, $m^{(p+1)}(x)$, with certain flexibility, and a polynomial fit of order $p+3$ would be able to do so by a quadratic polynomial.

[34] Notice that the $n-5$ in the denominator appears because the degrees of freedom of the residuals in a polynomial fit of order $p+3$ is $n-p-4$, and we are considering $p = 1$. Hence, $\hat{\sigma}_Q^2$ is an unbiased estimator of $\sigma_Q^2 = \mathbb{V}\mathrm{ar}_Q[Y|X = x] = \mathbb{E}[(Y - m_Q(X))^2|X = x]$.

[35] This is a slightly different adaptation from Fan and Gijbels (1996), where $\int \sigma^2(x)w_0(x)\,\mathrm{d}x$ is considered and therefore the estimate $\hat{\sigma}_Q^2 \int w_0(x)\,\mathrm{d}x$ naturally follows. This weight w_0, however, can be an indicator of the support of X, which gives in that case the selector described here.

```r
# Generate some data
set.seed(123456)
n <- 250
eps <- rnorm(n)
X <- abs(rnorm(n))
Y <- m(X) + eps

# Rule-of-thumb selector
h_RT <- function(X, Y) {

  # Quartic fit
  mod_Q <- lm(Y ~ poly(X, raw = TRUE, degree = 4))

  # Estimates of unknown quantities
  int_sigma2_hat <- diff(range(X)) * sum(mod_Q$residuals^2) / mod_Q$df.residual
  theta_22_hat <- mean((2 * mod_Q$coefficients[3] +
                        6 * mod_Q$coefficients[4] * X +
                        12 * mod_Q$coefficients[5] * X^2)^2)

  # h_RT
  R_K <- 0.5 / sqrt(pi)
  ((R_K * int_sigma2_hat) / (theta_22_hat * length(X)))^(1 / 5)

}

# Selected bandwidth
(h_ROT <- h_RT(X = X, Y = Y))
## [1] 0.1111711

# Local linear fit
lp1_RT <- KernSmooth::locpoly(x = X, y = Y, bandwidth = h_ROT, degree = 1,
                              range.x = c(0, 10), gridsize = 500)

# Quartic fit employed
mod_Q <- lm(Y ~ poly(X, raw = TRUE, degree = 4))

# Plot data
plot(X, Y, ylim = c(-6, 8))
rug(X, side = 1); rug(Y, side = 2)
lines(x_grid, m(x_grid), col = 1)
lines(lp1_RT$x, lp1_RT$y, col = 2)
lines(x_grid, mod_Q$coefficients[1] +
        poly(x_grid, raw = TRUE, degree = 4) %*% mod_Q$coefficients[-1],
      col = 3)
legend("topright", legend = c("True regression", "Local linear (RT)",
                              "Quartic fit"),
       lwd = 2, col = 1:3)
```

The $\hat{h}_{\text{RT}}$ selector strongly depends on how well the curvature of m is approximated by the quartic fit. In addition, it assumes that the conditional variance is constant, which may be unrealistic. The exercise below intends to illustrate these limitations.

Exercise 4.13. Explore in the code above the effect of the regression function, and visualize how poor quartic fits tend to give poor nonparametric fits. Then, adapt the sampling mechanism to introduce heteroskedasticity on Y and investigate the behavior of the nonparametric estimator based on $\hat{h}_{\text{RT}}$.

The rule-of-thumb selector is a zero-stage plug-in selector. An ℓ-stage plug-in selector was proposed by Ruppert et al. (1995)[36] and shares the same spirit of the DPI bandwidth selector seen in Section 2.4.1. As a consequence, the DPI selector for kre turns (4.21) into an usable selector by following a known agenda: it employs a

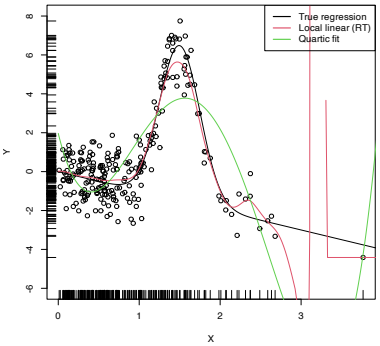


Figure 4.3: Local linear estimator with $\hat{h}_{\text{RT}}$ bandwidth and the quartic global fit. Observe how the local linear estimator behaves erratically at regions with no data – a fact due to the strong dependence of the locally weighted linear regression on few observations.

[36] See Section 5.8 in Wand and Jones (1995) for a quick introduction.

series of optimal nonparametric estimations of $\theta_{22}(m)$ that depend on high-order curvature terms $\theta_{rs}(m) := \int m^{(r)}(x)m^{(s)}(x)f(x)\,\mathrm{d}x$, ending this iterative procedure with a parametric estimation of a higher-order curvature.

The kind of parametric fit employed is a "block quartic fit". This is a series of N different quartic fits $\hat{m}_{Q,1},\ldots,\hat{m}_{Q,N}$ of the regression function m that are done in N blocks of the data. These blocks are defined by sorting the sample $X_1,\ldots,X_n$ and then partitioning it into approximately N blocks of equal size n/N. The purpose of the block quartic fit is to achieve more flexibility on the estimation of the curvature[37] terms $\theta_{22}(m)$ and $\theta_{24}(m)$. The estimation of $\int \sigma^2(x)\,\mathrm{d}x$ is carried out by pooling the individual estimates of the conditional variance in the N blocks, $\hat{\sigma}^2_{Q,1},\ldots,\hat{\sigma}^2_{Q,N}$, and by assuming a compactly supported density f.[38] The selection of N can be done in a data-driven way by relying on a model selection criterion (see Ruppert et al. (1995)), such as the AIC or BIC.

The resulting bandwidth selector, $\hat{h}_{\mathrm{DPI}}$, has a much faster convergence rate to h_{MISE} than cross-validatory selectors. However, it is notably more convoluted, and as a consequence it is less straightforward to extend to more complex settings.[39]

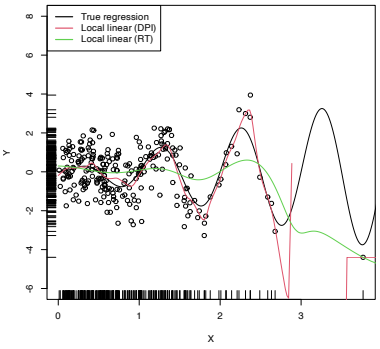
The DPI selector for the local linear estimator is implemented in `KernSmooth::dpill`.

[37] Notice that this approach does not guarantee the continuity of the final estimate, but since the goal is on estimating the curvature this is not relevant.

[38] So that, if $\sigma^2(x)$ is constant, $\int \sigma^2(x)\,\mathrm{d}x$ is finite because we integrate on a compact set.

[39] In particular, the construction of block polynomial fits may be challenging to extend to $\mathbb{R}^p$.

```
# Generate some data
set.seed(123456)
n <- 250
eps <- rnorm(n)
X <- abs(rnorm(n))
m <- function(x) x * sin(2 * pi * x)
Y <- m(X) + eps

# Selected bandwidth
(h_ROT <- h_RT(X = X, Y = Y))
## [1] 0.3489934

# DPI selector
(h_DPI <- KernSmooth::dpill(x = X, y = Y))
## [1] 0.05172781

# Fits
lp1_RT <- KernSmooth::locpoly(x = X, y = Y, bandwidth = h_ROT, degree = 1,
                              range.x = c(0, 10), gridsize = 500)
lp1_DPI <- KernSmooth::locpoly(x = X, y = Y, bandwidth = h_DPI, degree = 1,
                               range.x = c(0, 10), gridsize = 500)

# Plot data
plot(X, Y, ylim = c(-6, 8))
rug(X, side = 1); rug(Y, side = 2)
lines(x_grid, m(x_grid), col = 1)
lines(lp1_DPI$x, lp1_DPI$y, col = 2)
lines(lp1_RT$x, lp1_RT$y, col = 3)
legend("topleft", legend = c("True regression", "Local linear (DPI)",
                             "Local linear (RT)"), lwd = 2, col = 1:3)
```



4.3.2 *Cross-validation*

Following an analogy with the fit of the linear model (see Section B.1.1), an obvious possibility is to select an adequate bandwidth h

for $\hat{m}(\cdot; p, h)$ that minimizes a *Residual Sum of Squares* (RSS) of the form

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{m}(X_i; p, h))^2. \tag{4.22}$$

However, this is a bad idea. Contrarily to what happened in the linear model, where it was almost[40] impossible to perfectly fit the data due to the limited flexibility of the model, the infinite flexibility of the local polynomial estimator allows interpolating the data if $h \to 0$. As a consequence, attempting to minimize (4.22) always leads to $h \approx 0$, which results in a useless interpolation that misses the target of the estimation, m.

[40] Excluding colinear dispositions of the data and assuming that $n > p$.

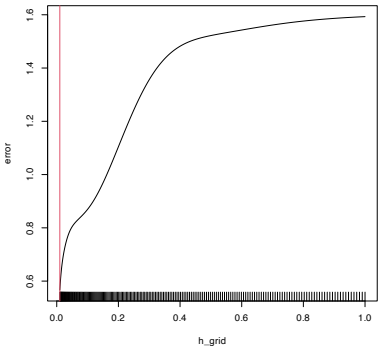
```
# Grid for representing (4.22)
h_grid <- seq(0.1, 1, l = 200)^2
error <- sapply(h_grid, function(h) {
  mean((Y - nw(x = X, X = X, Y = Y, h = h))^2)
})

# Error curve
plot(h_grid, error, type = "l")
rug(h_grid)
abline(v = h_grid[which.min(error)], col = 2)
```

The root of the problem is the comparison of Y_i with $\hat{m}(X_i; p, h)$, since there is nothing that forbids that, when $h \to 0$, $\hat{m}(X_i; p, h) \to Y_i$. We can change this behavior if we compare Y_i with $\hat{m}_{-i}(X_i; p, h)$ [41], the **leave-one-out estimate** of m computed without the i-th datum (X_i, Y_i), yielding the **least squares cross-validation** error

$$\mathrm{CV}(h) := \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{m}_{-i}(X_i; p, h))^2 \tag{4.23}$$

and then choose

$$\hat{h}_{\mathrm{CV}} := \arg\min_{h>0} \mathrm{CV}(h).$$



[41] In this case, when $h \to 0$, $\hat{m}_{-i}(X_i; p, h) \not\to Y_i$ because (X_i, Y_i) does not belong to the sample employed for computing $\hat{m}_{-i}(\cdot; p, h)$. Notice that $\hat{m}_{-i}(X_j; p, h) \to Y_j$ when $h \to 0$ if $j \neq i$, but this is not problematic because in $\sum_{i=1}^{n}(Y_i - \hat{m}_{-i}(X_i; p, h))^2$ the leave-one-out estimate is different for each $i = 1, \ldots, n$.

The optimization of (4.23) might seem to be very expensive computationally, since computing n regressions for just a *single* evaluation of the cross-validation function is required. There is, however, a simple and neat theoretical result that vastly reduces the computational complexity, at the price of increasing the memory demand. This trick allows computing, with a *single* fit, the cross-validation function evaluated at h.

Proposition 4.1. *For any $p \geq 0$, the weights of the leave-one-out estimator $\hat{m}_{-i}(x; p, h) = \sum_{\substack{j=1 \\ j\neq i}}^{n} W^p_{-i,j}(x) Y_j$ can be obtained from $\hat{m}(x; p, h) = \sum_{i=1}^{n} W^p_i(x) Y_i$:*

$$W^p_{-i,j}(x) = \frac{W^p_j(x)}{\sum_{\substack{k=1 \\ k\neq i}}^{n} W^p_k(x)} = \frac{W^p_j(x)}{1 - W^p_i(x)}. \tag{4.24}$$

This implies that

$$\mathrm{CV}(h) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i - \hat{m}(X_i; p, h)}{1 - W^p_i(X_i)}\right)^2. \tag{4.25}$$

The result can be proved by recalling that $\hat{m}(x; p, h)$ is a linear combination[42] of the responses $\{Y_i\}_{i=1}^n$.

[42] Indeed, for any other *linear smoother* of the response, the result will also hold.

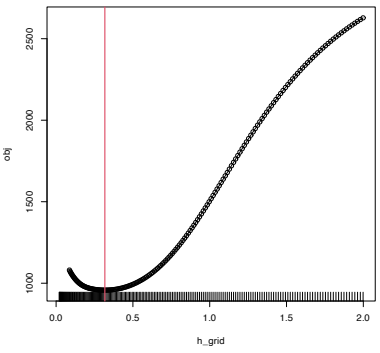
Exercise 4.14. Consider the Nadaraya–Watson estimator ($p = 0$). Then, show (4.24) using (4.14). From there, conclude the proof of Proposition 4.1.

Exercise 4.15. Prove the second equality in (4.24) for the local linear estimator ($p = 1$). From there, prove (4.25). Use (4.15).

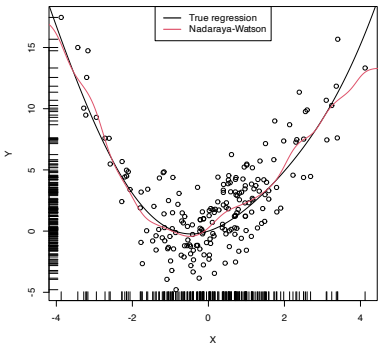
Remark. Observe that computing (4.25) requires evaluating the local polynomial estimator at the sample $\{X_i\}_{i=1}^n$ *and* obtaining $\{W_i^p(X_i)\}_{i=1}^n$ (which are needed to evaluate $\hat{m}(X_i; p, h)$). Both tasks can be achieved simultaneously from the $n \times n$ matrix $\left(W_i^p(X_j)\right)_{ij}$ and, if $p = 0$, directly from the *symmetric* $n \times n$ matrix $\left(K_h(X_i - X_j)\right)_{ij}$, for which the storage cost is $O((n^2 - n)/2)$ (the diagonal is constant).

Let's implement $\hat{h}_{\text{CV}}$ employing (4.25) for the Nadaraya–Watson estimator.

```
# Generate some data to test the implementation
set.seed(12345)
n <- 200
eps <- rnorm(n, sd = 2)
m <- function(x) x^2 + sin(x)
X <- rnorm(n, sd = 1.5)
Y <- m(X) + eps
x_grid <- seq(-10, 10, l = 500)

# Objective function
cv_nw <- function(X, Y, h, K = dnorm) {

  sum(((Y - nw(x = X, X = X, Y = Y, h = h, K = K)) /
         (1 - K(0) / colSums(K(outer(X, X, "-") / h))))^2)

}

# Find optimum CV bandwidth, with sensible grid
bw_cv_grid <- function(X, Y,
                       h_grid = diff(range(X)) * (seq(0.05, 0.5, l = 200))^2,
                       K = dnorm, plot_cv = FALSE) {

  # Minimize the CV function on a grid
  obj <- sapply(h_grid, function(h) cv_nw(X = X, Y = Y, h = h, K = K))
  h <- h_grid[which.min(obj)]

  # Add plot of the CV loss
  if (plot_cv) {

    plot(h_grid, obj, type = "o")
    rug(h_grid)
    abline(v = h, col = 2, lwd = 2)

  }

  # CV bandwidth
  return(h)

}

# Bandwidth
h <- bw_cv_grid(X = X, Y = Y, plot_cv = TRUE)
```



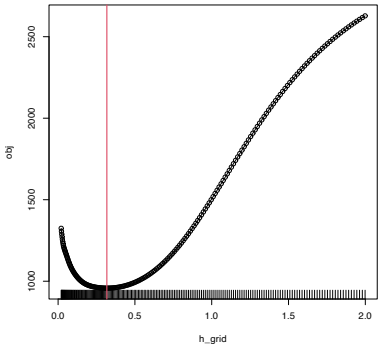
```
h
## [1] 0.3173499

# Plot result
plot(X, Y)
rug(X, side = 1); rug(Y, side = 2)
lines(x_grid, m(x_grid), col = 1)
lines(x_grid, nw(x = x_grid, X = X, Y = Y, h = h), col = 2)
legend("top", legend = c("True regression", "Nadaraya-Watson"),
       lwd = 2, col = 1:2)
```



An important warning when computing cross-validation bandwidths is that the objective function can be challenging to minimize and may present several local minima. Therefore, if possible, it is always advisable to graphically inspect the CV loss curve and to run an exhaustive search. When employing an automatic optimization procedure, it is recommended to use several starting values.

The following chunk of code presents the inefficient implementation of $\hat{h}_{\text{CV}}$ (based on the definition) and evidences its poorer performance.

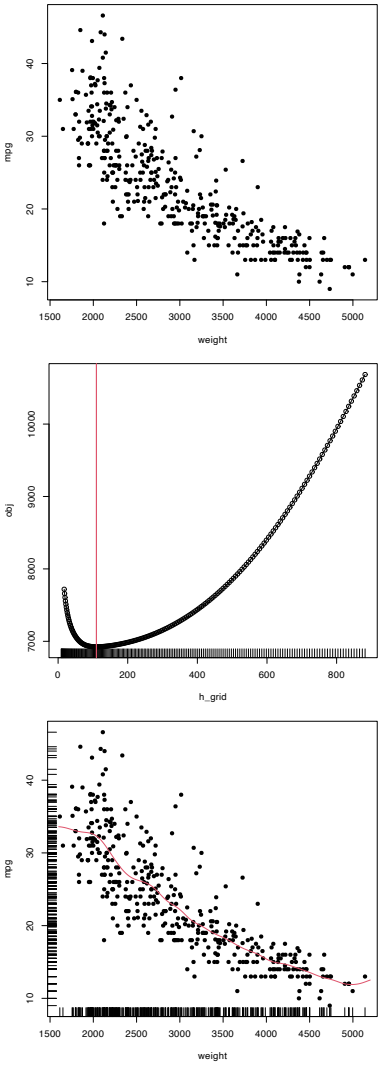
```
# Slow objective function
cv_nw_slow <- function(X, Y, h, K = dnorm) {

  sum((Y - sapply(1:length(X), function(i) {
    nw(x = X[i], X = X[-i], Y = Y[-i], h = h, K = K)
    }))^2)

}

# Optimum CV bandwidth, with sensible grid
bw_cv_grid_slow <- function(X, Y, h_grid =
                              diff(range(X)) * (seq(0.05, 0.5, l = 200))^2,
                            K = dnorm, plot_cv = FALSE) {

  # Minimize the CV function on a grid
  obj <- sapply(h_grid, function(h) cv_nw_slow(X = X, Y = Y, h = h, K = K))
  h <- h_grid[which.min(obj)]

  # Add plot of the CV loss
  if (plot_cv) {

    plot(h_grid, obj, type = "o")
    rug(h_grid)
    abline(v = h, col = 2, lwd = 2)

  }

  # CV bandwidth
  return(h)

}

# Same bandwidth
h <- bw_cv_grid_slow(X = X, Y = Y, plot_cv = TRUE)
```




```
h
## [1] 0.3173499

# # Time comparison, a factor 10 difference
# microbenchmark::microbenchmark(bw_cv_grid(X = X, Y = Y),
#                                bw_cv_grid_slow(X = X, Y = Y),
#                                times = 10)
```

Finally, the following chunk of code illustrates the estimation of the regression function of the `weight` on the `mpg` in `data(Auto, package = "ISLR")`, using a cross-validation bandwidth.

```
# Data -- nonlinear trend
data(Auto, package = "ISLR")
X <- Auto$weight
Y <- Auto$mpg
plot(X, Y, xlab = "weight", ylab = "mpg", pch = 16)



# CV bandwidth
h <- bw_cv_grid(X = X, Y = Y, plot_cv = TRUE)


h
## [1] 110.0398

# Add regression
x_grid <- seq(1600, 5200, by = 10)
plot(X, Y, xlab = "weight", ylab = "mpg", pch = 16)
rug(X, side = 1); rug(Y, side = 2)
lines(x_grid, nw(x = x_grid, X = X, Y = Y, h = h), col = 2)
```



4.4 *Regressogram*

The regressogram is the adaptation of the histogram to the regression setting. Historically, it has received attention in several applied areas.[43] This and its connection with the histogram are the reasons for its inclusion in the notes, since its performance to estimate m is definitely inferior to that of $\hat{m}(\cdot; p, h)$ (see Figure 4.4). The construction described below can be regarded as the opposite path to the one followed in Sections 2.1–2.2 for constructing the kde from the histogram: now we deconstruct the Nadaraya–Watson estimator to obtain the regressogram.

Based on (4.2), the Nadaraya–Watson estimator was constructed by plugging kdes for the joint density of (X, Y) and the marginal density of X. This resulted in

$$\hat{m}(x; 0, h) = \frac{\sum_{i=1}^n K_h(x - X_i) Y_i}{\sum_{i=1}^n K_h(x - X_i)} = \frac{\frac{1}{n}\sum_{i=1}^n K_h(x - X_i) Y_i}{\hat{f}(x; h)}, \tag{4.26}$$

which clearly emphasizes the connection between the kde $\hat{f}(x; h) = \frac{1}{n}\sum_{i=1}^n K_h(x - X_i)$ and $\hat{m}(x; 0, h)$. Evidently, this approach gives a smooth estimator for the regression function if the kernels employed in the kde are smooth.

Within (4.26), a possibility is to consider the uniform kernel $K(z) = \frac{1}{2} 1_{\{-1<x<1\}}$ in the kde[44], which results in

$$\hat{m}_{\mathrm{N}}(x; h) := \frac{\sum_{i=1}^n 1_{\{X_i - h < x < X_i + h\}} Y_i}{\sum_{i=1}^n 1_{\{X_i - h < x < X_i + h\}}} = \frac{1}{|N(x; h)|} \sum_{i \in N(x;h)} Y_i, \tag{4.27}$$

where $N(x; h) := \{i = 1, \ldots, n : |X_i - x| < h\}$ is the set of the indexes of the sample within the neighborhood of x and $|N(x; h)|$ denotes its size.[45] Estimator (4.27), a *naive regression estimator*, is

[43] For example, in astronomy, check Figure 3.2.17 in Vol. 1 in ESA (1997).

[44] Equivalently, the naive density estimator $\hat{f}_{\mathrm{N}}(x; h) = \frac{1}{2nh}\sum_{i=1}^n 1_{\{x-h<X_i<x+h\}}$ instead of the kde.

[45] Observe that (4.27) is defined only for x such that $|N(x; h)| > 0$. It is perfectly possible to have $|N(x; h)| = 0$ in practice. This does not happen for non-compactly supported kernels, for which $N(x; h) = \mathbb{R}$ for any x, h, and sample arrangement.

precisely the regression analogue of the moving histogram or naive density estimator. Its second expression in (4.27) reveals that it is just a sample mean in different blocks (or neighborhoods) of the data. As a consequence, it is discontinuous.

Another alternative to the kde in (4.26) is to employ the histogram $\hat{f}_{\mathrm{H}}(x;t_0,h)=\frac{1}{nh}\sum_{i=1}^n 1_{\{X_i\in B_k:x\in B_k\}}$, where $\{B_k=[t_k,t_{k+1}):t_k=t_0+hk,k\in\mathbb{Z}\}$ (recall Section 2.1.1). This yields the *regressogram* of m:

$$\hat{m}_{\mathrm{R}}(x;h):=\frac{\sum_{i=1}^n 1_{\{X_i\in B_k\}}Y_i}{\sum_{i=1}^n 1_{\{X_i\in B_k\}}}=\frac{1}{|B(k;h)|}\sum_{i\in B(k;h)}Y_i,\quad \text{if } x\in B_k, \tag{4.28}$$

where $B(k;h):=\{i=1,\ldots,n:X_i\in B_k\}$ and $|B(k;h)|$ stands for its size (denoted by v_k in Section 2.1.1). The difference of the regressogram with respect to the naive regression estimator is that the former pre-defines fixed bins in which to compute the bin means, producing a final estimator that, for the same bandwidth h, is notably more rigid (it is constant on each bin B_k; see Figure 4.4).

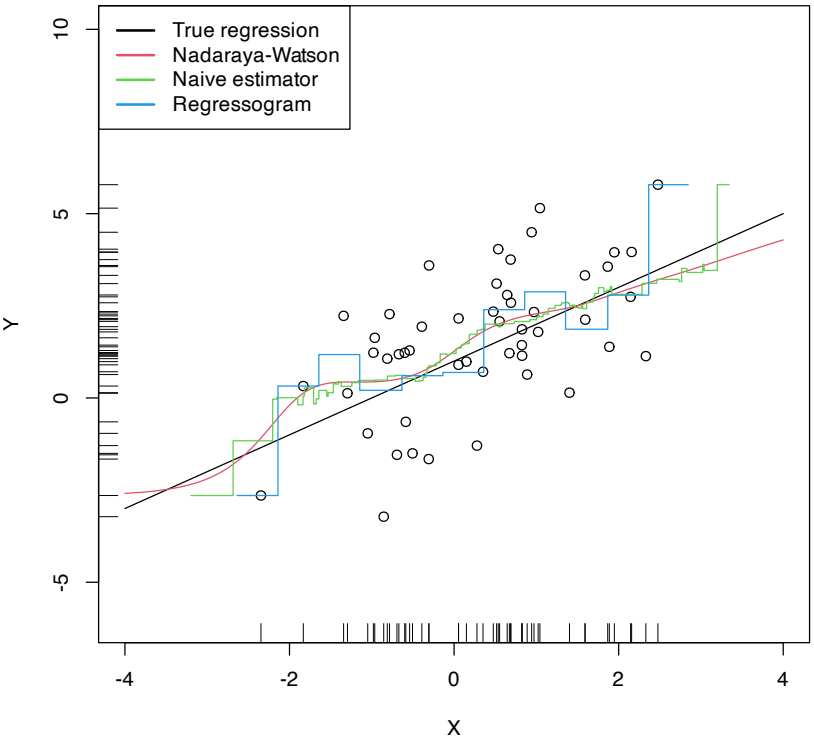


Figure 4.4: The Nadaraya–Watson estimator, the naive regression estimator, and the regressogram. Notice that the naive regression estimator and the regressogram are not defined everywhere – only for those regions in which there are nearby observations of X. All the estimators share the bandwidth $h=0.5$, a fair comparison since the *scaled* uniform kernel, $\tilde{K}(z)=\frac{1}{2\sqrt{3}}1_{\{-\sqrt{3}<z<\sqrt{3}\}}$, is employed for the naive estimator. The regressogram employs $t_0=0$. The regression setting is explained in Exercise 4.16.

Exercise 4.16. Implement in R your own version of the naive regression estimator (4.27). It must be a function that takes as inputs:

- a vector with the evaluation points x,
- a sample $(X_1,Y_1),\ldots,(X_n,Y_n)$,
- a bandwidth h,

and that returns (4.27) evaluated for *each* x. Test the implementation by estimating the regression function $m(x)=1+x$ for the regression model $Y=m(X)+\varepsilon$, where $X\sim\mathcal{N}(0,1)$ and $\varepsilon\sim\mathcal{N}(0,2)$ using $n=50$ observations. This is the setting used in Figure 4.4.

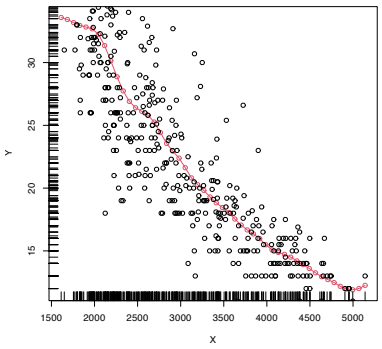
Exercise 4.17. Perform Exercise 4.16 by implementing in R the regressogram (4.28) instead of the naive regression estimator. The R function must now have an *additional* argument t_0.

4.5 *Kernel regression estimation with* np

The `np` package (Hayfield and Racine, 2008) provides a complete framework for performing a more sophisticated nonparametric regression estimation for local constant and linear estimators, and for computing cross-validation bandwidths. The two workhorse functions for these tasks are `np::npreg` and `np::npregbw`, which they illustrate the philosophy behind the `np` package: first, a suitable bandwidth for the nonparametric method is found (via `np::npregbw`) and stored in an `rbandwidth` object; then, a fitting function (`np::npreg`) is called on that `rbandwidth` object, which contains all the needed information.

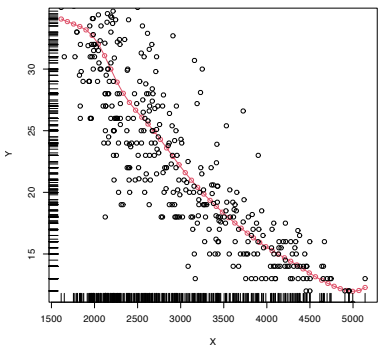
```
# Data -- nonlinear trend
data(Auto, package = "ISLR")
X <- Auto$weight
Y <- Auto$mpg

# np::npregbw computes by default the least squares CV bandwidth associated
# with a local *constant* fit and admits a formula interface (to be exploited
# more in multivariate regression)
bw0 <- np::npregbw(formula = Y ~ X)
## Multistart 1 of 1 |Multistart 1 of 1 |Multistart 1 of 1 |Multistart 1 of 1 /Multistart 1 of 1 |Multistart 1 of 1 |

# The spinner can be omitted with
options(np.messages = FALSE)

# Multiple initial points can be employed for minimizing the CV function (for
# one predictor, defaults to 1) and avoiding local minima
bw0 <- np::npregbw(formula = Y ~ X, nmulti = 2)

# The "rbandwidth" object contains useful information, see ?np::npregbw for
# all the returned objects
bw0
##
## Regression Data (392 observations, 1 variable(s)):
##
##                       X
## Bandwidth(s): 110.1476
##
## Regression Type: Local-Constant
## Bandwidth Selection Method: Least Squares Cross-Validation
## Formula: Y ~ X
## Bandwidth Type: Fixed
## Objective Function Value: 17.66388 (achieved on multistart 2)
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
head(bw0)
## $bw
## [1] 110.1476
##
## $regtype
## [1] "lc"
##
## $pregtype
## [1] "Local-Constant"
##
```

```
## $method
## [1] "cv.ls"
##
## $pmethod
## [1] "Least Squares Cross-Validation"
##
## $fval
## [1] 17.66388
# Recall that the fit is very similar to h_CV

# Once the bandwidth is estimated, np::npreg can be directly called with the
# "rbandwidth" object (it encodes the regression to be made, the data, the kind
# of estimator considered, etc). The hard work goes on np::npregbw, not on
# np::npreg
kre0 <- np::npreg(bws = bw0)
kre0
##
## Regression Data: 392 training points, in 1 variable(s)
##                       X
## Bandwidth(s): 110.1476
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1

# Plot directly the fit via plot() -- it employs as evaluation points the
# (unsorted!) sample
plot(kre0, col = 2, type = "o")
points(X, Y)
rug(X, side = 1); rug(Y, side = 2)


# lines(kre0$eval$X, kre0$mean, col = 3)
```



The type of local regression (local constant *or* local linear – no further local polynomial fits are implemented) is controlled by the argument `regtype` in `np::npregbw`: `regtype = "ll"` stands for "local linear" and `regtype = "lc"` for "local constant".

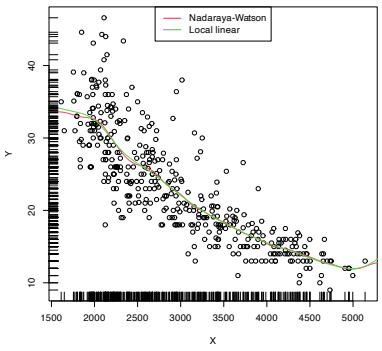
```
# Local linear fit -- find first the CV bandwidth
bw1 <- np::npregbw(formula = Y ~ X, regtype = "ll")

# Fit
kre1 <- np::npreg(bws = bw1)

# Plot
plot(kre1, col = 2, type = "o")
points(X, Y)
rug(X, side = 1); rug(Y, side = 2)
```



The following code shows in more detail the output object associated with `np::npreg` and how to change the evaluation points.

```
# Summary of the npregression object
summary(kre0)
##
## Regression Data: 392 training points, in 1 variable(s)
##                       X
## Bandwidth(s): 110.1476
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
## Residual standard error: 4.10312
## R-squared: 0.7230723
```

```
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1

# Evaluation points (a data.frame) -- by default the sample (unsorted!)
head(kre0$eval)
##      X
## 1 3504
## 2 3693
## 3 3436
## 4 3433
## 5 3449
## 6 4341

# The estimation of the regression function at the evaluation points
head(kre0$mean)
## [1] 18.35364 17.10311 18.70083 18.71652 18.63411 14.23660

# The evaluation points can be changed using "exdat"
x_grid <- seq(1000, 5500, by = 5)
kre0 <- np::npreg(bws = bw0, exdat = x_grid)
kre1 <- np::npreg(bws = bw1, exdat = x_grid)

# Notice how $eval is a data.frame containing x_grid
head(kre0$eval)
##   x_grid
## 1   1000
## 2   1005
## 3   1010
## 4   1015
## 5   1020
## 6   1025

# This allows to compare estimators in a more transparent form
plot(X, Y)
lines(kre0$eval$x_grid, kre0$mean, col = 2)
lines(kre1$eval$x_grid, kre1$mean, col = 3)
rug(X, side = 1); rug(Y, side = 2)
legend("top", legend = c("Nadaraya-Watson", "Local linear"),
       lwd = 2, col = 2:3)
```



Now that we know how to effectively compute local constant and linear estimators, two insights with practical relevance become apparent:

- The *adequate* **bandwidths for the local linear estimator** are usually **larger** than the adequate bandwidths for the local constant estimator. The reason is the extra flexibility that $\hat{m}(\cdot; 1, h)$ has, which allows faster adaptation to variations in m, whereas $\hat{m}(\cdot; 0, h)$ can achieve this adaptability only by shrinking the neighborhood about x by means of a smaller h.
- Both estimators **behave erratically at regions with low predictor's density**. This includes possible "holes" in the support of X. In the absence of close data, the **local constant estimator interpolates to a constant** function that takes the value Y_i of the closest X_i to x. However, the local linear estimator $\hat{m}(x; 1, h)$ takes the value of the local linear regression $\hat{\beta}_{h,0} + \hat{\beta}_{h,1}(\cdot - x)$ that is determined by the two observations (X_i, Y_i) with the closest X_i's to x. Hence, the **local linear estimator extrapolates a line** at regions with low X-density. This may have the undesirable consequence of yielding large spikes at these regions, which sharply

contrasts with the "conservative behavior" of the local constant estimator.

The next two exercises are meant to evidence these two insights.

Exercise 4.18. Perform the following simulation study:

1. Simulate $n = 100$ observations from the regression model $Y = m(X) + \varepsilon$, where X is distributed according to a `nor1mix::MW.nm2`, $m(x) = x\cos(x)$, and $\varepsilon + 1 \sim \text{Exp}(1)$.
2. Compute the CV bandwidths for the local constant and linear estimators.
3. Repeat Steps 1–2 $M = 1,000$ times. *Trick*: use `txtProgressBar()` and `setTxtProgressBar` to have an indication of the progress of the simulation.

Compare graphically the two samples of CV bandwidths. To do so, show in a plot the log-transformed kdes for the two samples and imitate the display given in 4.5.

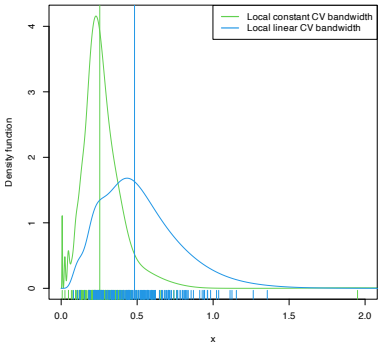


Figure 4.5: Estimated density for the local constant and local linear CV bandwidths, for the setting described in Exercise 4.18. The vertical bars denote the median CV bandwidths. Observe how local linear bandwidths tend to be larger than local constant ones.

Exercise 4.19. Perform the following simulation study for each of the local constant and local linear estimators.

1. Plot the regression function $m(x) = x\cos(x)$.
2. Simulate $n = 100$ observations from the regression model $Y = m(X) + \varepsilon$, where $\frac{1}{2}X$ is distributed according to a `nor1mix::MW.nm7` and $\varepsilon \sim \mathcal{N}(0,1)$.
3. Compute the CV bandwidths and the fit associated with this bandwidth.
4. Plot the fit as a line. *Trick*: adjust the transparency of each line for better visualization.
5. Repeat Steps 2–4 $M = 75$ times.

The two outputs should be similar to Figure 4.6.

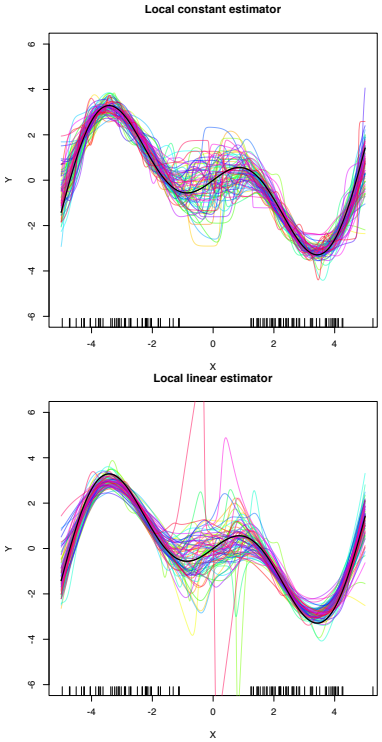


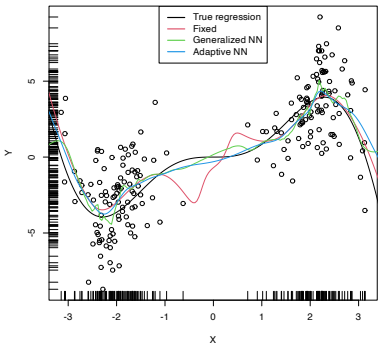
Figure 4.6: Local constant and linear estimators of the regression function in the setting described in Exercise 4.19. The support of the predictor X has a "hole", which makes both estimators to behave erratically at that region. The local linear estimator, driven by a local linear fit, may deviate stronger from m than the local constant.

There are more sophisticated options for bandwidth selection available in `np::npregbw`. For example, the argument `bwtype` allows estimating data-driven ***variable* bandwidths** $\hat{h}(x)$ that depend on the evaluation point x, rather than *fixed* bandwidths $\hat{h}$, as we have considered so far. Roughly speaking, these variable bandwidths are related to the variable bandwidth $\hat{h}_k(x)$ that is necessary to contain the k *nearest neighbors* $X_1, \ldots, X_k$ of x in the neighborhood $(x - \hat{h}_k(x), x + \hat{h}_k(x))$. There is an interesting potential gain in employing variable bandwidths, as the local polynomial estimator can adapt the amount of smoothing according to the density of the predictor, therefore avoiding the aforementioned problems related to "holes" in the support of X (see the code below). The extra flexibility of variable bandwidths change the asymptotic properties of the local polynomial estimator. We do not investigate this approach in detail[46] but rather point out to its implementation via `np`.

[46] The interested reader is referred to Chapter 14 in Li and Racine (2007) and references therein.

```
# Generate some data with bimodal density
set.seed(12345)
```

```
n <- 100
eps <- rnorm(2 * n, sd = 2)
m <- function(x) x^2 * sin(x)
X <- c(rnorm(n, mean = -2, sd = 0.5), rnorm(n, mean = 2, sd = 0.5))
Y <- m(X) + eps
x_grid <- seq(-10, 10, l = 500)

# Constant bandwidth
bwc <- np::npregbw(formula = Y ~ X, bwtype = "fixed",
                   regtype = "ll")
krec <- np::npreg(bwc, exdat = x_grid)

# Variable bandwidths
bwg <- np::npregbw(formula = Y ~ X, bwtype = "generalized_nn",
                   regtype = "ll")
kreg <- np::npreg(bwg, exdat = x_grid)
bwa <- np::npregbw(formula = Y ~ X, bwtype = "adaptive_nn",
                   regtype = "ll")
krea <- np::npreg(bwa, exdat = x_grid)

# Comparison
plot(X, Y)
rug(X, side = 1); rug(Y, side = 2)
lines(x_grid, m(x_grid), col = 1)
lines(krec$eval$x_grid, krec$mean, col = 2)
lines(kreg$eval$x_grid, kreg$mean, col = 3)
lines(krea$eval$x_grid, krea$mean, col = 4)
legend("top", legend = c("True regression", "Fixed", "Generalized NN",
                         "Adaptive NN"),
       lwd = 2, col = 1:4)


# Observe how the fixed bandwidth may yield a fit that produces serious
# artifacts in the low-density region. At that region the NN-based bandwidths
# expand to borrow strength from the points in the high-density regions,
# whereas in the high-density regions they shrink to adapt faster to the
# changes of the regression function
```



Exercise 4.20. Repeat Exercise 4.19 replacing Step 3 with:

3. Compute the variable bandwidths `bwtype = "generalized_nn"` and `bwtype = "adaptive_nn"` and the fits associated with them.

Describe the obtained results. Are they qualitatively different from those based on fixed bandwidths, given in Figure 4.6? Do the fits improve with variable bandwidths?

Exercise 4.21. Consider the `data(sunspots_births, package = "rotasym")` dataset. Then:

a. Filter the dataset to account only for the 23rd cycle.
b. Inspect the graphical relation between `dist_sun_disc` (response) and `log10(total_area + 1)` (predictor).
c. Compute the CV bandwidth for the above regression, for the local constant estimator.
d. Compute and plot the local constant estimator using CV bandwidth. Comment on the fit.
e. Repeat c and d for the local linear estimator.
f. Are your conclusions in part e the same for the 21st and 22nd cycles?

5
Kernel regression estimation II

In Chapter 4 we studied the simplest situation for performing nonparametric estimation of the regression function: a *single*, *continuous* predictor X is available for explaining Y, a numerical response that is implicitly assumed to be *continuous*.[1] This served to introduce the main concepts without the additional technicalities associated with more complex predictors.

The purpose of this chapter is to extend nonparametric regression when

1. there are **multiple predictors** $X_1, \ldots, X_p$,
2. some predictors possibly are non-continuous, i.e., they are **categorical** or discrete, and
3. the **response** Y is **not continuous**.

We concentrate first on the first two points, as the third presents a change of paradigm from the kernel regression estimator studied in Chapter 4.

[1] Local polynomial estimators also "work" with discrete responses, with a varying degree of adequacy. For example, if Y is binary, then $\hat{m}_h(x;0,h) \in [0,1]$, for any $x \in \mathbb{R}$ and $h > 0$, since $\hat{m}_h(x;0,h)$ is an x-weighted *mean* (or a convex linear combination) of $Y_1, \ldots, Y_n \in [0,1]$. Then, the regression function $m : \mathbb{R} \longrightarrow [0,1]$ is always properly estimated. However, a local linear estimator may yield improper estimators of m, since $\hat{m}_h(x;1,h)$ is an x-weighted *linear combination* of $Y_1, \ldots, Y_n \in [0,1]$ and consequently $\hat{m}_h(x;1,h)$ may "spike" outside $[0,1]$.

5.1 *Kernel regression with mixed multivariate data*

5.1.1 *Multivariate kernel regression*

We start by addressing the first generalization:

> How to extend the local polynomial estimator $\hat{m}(\cdot;q,h)$ [2] to deal with p *continuous* predictors?

[2] Here q denotes the order of the polynomial fit, since p stands for the number of predictors.

Although this can be done for $q \geq 0$, we focus on the local constant and linear estimators ($q = 0, 1$) to avoid excessive technical complications.[3]

The initial step is to state what the population object to be estimated is, which is now the function $m : \mathbb{R}^p \longrightarrow \mathbb{R}$ given by[4]

$$m(x) := \mathbb{E}[Y|\mathbf{X} = \mathbf{x}] = \int y f_{Y|\mathbf{X}=\mathbf{x}}(y)\, \mathrm{d}y, \tag{5.1}$$

where $\mathbf{X} = (X_1, \ldots, X_p)'$ denotes the random vector of the predictors. Hence, despite having a form slightly different from (4.1), (4.1) and (5.1) are conceptually equal. As a consequence, the density-based view[5] of m holds:

[3] In particular, the consideration of Taylor expansions of $m : \mathbb{R}^p \longrightarrow \mathbb{R}$ for more than two orders that involve the vector of partial derivatives $\mathrm{D}^{\otimes s} m(\mathbf{x})$, formed by $\frac{\partial^s m(\mathbf{x})}{\partial x_1^{s_1} \cdots \partial x_p^{s_p}}$, where $s = s_1 + \cdots + s_p$; see Section 3.1.

[4] Observe that in the second equality we assume that $Y|\mathbf{X} = \mathbf{x}$ is continuous.

[5] Assuming that $(\mathbf{X}, Y)$ is continuous to motivate the construction of the estimator.

$$m(x) = \frac{\int y f(\mathbf{x}, y)\, \mathrm{d}y}{f_{\mathbf{X}}(\mathbf{x})}, \tag{5.2}$$

where f is the joint density of $(\mathbf{X}, Y)$ and $f_{\mathbf{X}}$ is the marginal pdf of $\mathbf{X}$.

Therefore, given a sample $(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)$, we can estimate f and $f_{\mathbf{X}}$, analogously to how we did in Section 4.1.1, by the kde's[6]

$$\hat{f}(\mathbf{x}, y; \mathbf{H}, h) = \frac{1}{n} \sum_{i=1}^{n} K_{\mathbf{H}}(\mathbf{x} - \mathbf{X}_i) K_h(y - Y_i), \tag{5.3}$$

$$\hat{f}_{\mathbf{X}}(\mathbf{x}; \mathbf{H}) = \frac{1}{n} \sum_{i=1}^{n} K_{\mathbf{H}}(\mathbf{x} - \mathbf{X}_i). \tag{5.4}$$

Plugging these estimators in (5.2), we readily obtain the Nadaraya–Watson estimator for multivariate predictors:

$$\hat{m}(\mathbf{x}; 0, \mathbf{H}) := \sum_{i=1}^{n} \frac{K_{\mathbf{H}}(\mathbf{x} - \mathbf{X}_i)}{\sum_{j=1}^{n} K_{\mathbf{H}}(\mathbf{x} - \mathbf{X}_j)} Y_i = \sum_{i=1}^{n} W_i^0(\mathbf{x}) Y_i, \tag{5.5}$$

where

$$W_i^0(\mathbf{x}) := \frac{K_{\mathbf{H}}(\mathbf{x} - \mathbf{X}_i)}{\sum_{i=1}^{n} K_{\mathbf{H}}(\mathbf{x} - \mathbf{X}_i)}.$$

Exercise 5.1. Derive (5.5) by plugging (5.3) and (5.4) into (5.2).

Usually, to avoid a quick escalation of the number of smoothing bandwidths[7], it is customary to consider **product kernels** for smoothing $\mathbf{X}$, that is, to consider a diagonal bandwidth $\mathbf{H} = \operatorname{diag}(h_1^2, \ldots, h_p^2) = \operatorname{diag}(\mathbf{h}^2)$, which gives the estimator implemented by the `np` package:

$$\hat{m}(\mathbf{x}; 0, \mathbf{h}) := \sum_{i=1}^{n} W_i^0(\mathbf{x}) Y_i$$

where

$$W_i^0(\mathbf{x}) = \frac{K_{\mathbf{h}}(\mathbf{x} - \mathbf{X}_i)}{\sum_{j=1}^{n} K_{\mathbf{h}}(\mathbf{x} - \mathbf{X}_j)},$$

$$K_{\mathbf{h}}(\mathbf{x} - \mathbf{X}_i) = K_{h_1}(x_1 - X_{i1}) \times \overset{p}{\cdots} \times K_{h_p}(x_p - X_{ip}).$$

As in the univariate case, the Nadaraya–Watson estimate can be seen as a weighted average of $Y_1, \ldots, Y_n$ by means of the weights $\{W_i^0(\mathbf{x})\}_{i=1}^n$. Therefore, the Nadaraya–Watson estimator is a **local mean** of $Y_1, \ldots, Y_n$ about $\mathbf{X} = \mathbf{x}$.

The derivation of the **local linear estimator** involves slightly more complex arguments, but analogous to the extension of the linear model from univariate to multivariate predictors. Considering the first-order Taylor expansion

$$m(\mathbf{X}_i) \approx m(\mathbf{x}) + \mathrm{D}m(\mathbf{x})'(\mathbf{X}_i - \mathbf{x}),$$

instead of (4.7) it is possible to arrive to the analogue[8] of (4.10),

$$\hat{\boldsymbol{\beta}}_{\mathbf{h}} := \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^{p+1}} \sum_{i=1}^{n} \left(Y_i - \boldsymbol{\beta}'(1, (\mathbf{X}_i - \mathbf{x})')'\right)^2 K_{\mathbf{h}}(\mathbf{x} - \mathbf{X}_i),$$

[6] The fact that the kde for $(\mathbf{X}, Y)$ has a product kernel between the variables $\mathbf{X}$ and Y is not fundamental for constructing the Nadaraya–Watson estimator, but it simplifies the computation of the integral of the denominator of (5.2). The fact that the bandwidth matrices for estimating the $\mathbf{X}$-part of f and the marginal $f_{\mathbf{X}}$ are equal, *is* fundamental to ensure an estimator for m that is a weighted average (*with weights adding one*) of the responses $Y_1, \ldots, Y_n$.

[7] Which increases at the order $O(p^2)$.

[8] Observe that $(X_i - x)^j$ are replaced with $(X_{ij} - x_j)$, since we only consider a linear fit and now the predictors have p components.

[9] Observe that, in an abuse of notation, we denote both the random vector $(X_1, \ldots, X_p)$ and the design matrix by $\mathbf{X}$. However, the specific meaning of $\mathbf{X}$ should be clear from the context.

and then solve the problem in the exact same way but now considering the design matrix[9]

$$\mathbf{X} := \begin{pmatrix} 1 & (\mathbf{X}_1 - \mathbf{x})' \\ \vdots & \vdots \\ 1 & (\mathbf{X}_n - \mathbf{x})' \end{pmatrix}_{n \times (p+1)}$$

and

$$\mathbf{W} := \operatorname{diag}(K_{\mathbf{h}}(\mathbf{X}_1 - \mathbf{x}), \ldots, K_{\mathbf{h}}(\mathbf{X}_n - \mathbf{x})).$$

The estimate[10] for $m(\mathbf{x})$ is therefore obtained from the solution of the weighted least squares problem

$$\begin{aligned} \hat{\boldsymbol{\beta}}_{\mathbf{h}} &= \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^{p+1}} (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})' \mathbf{W} (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) \\ &= (\mathbf{X}'\mathbf{W}\mathbf{X})^{-1} \mathbf{X}'\mathbf{W}\mathbf{Y} \end{aligned}$$

as

$$\begin{aligned} \hat{m}(\mathbf{x}; 1, h) &:= \hat{\beta}_{\mathbf{h},0} \\ &= \mathbf{e}_1' (\mathbf{X}'\mathbf{W}\mathbf{X})^{-1} \mathbf{X}'\mathbf{W}\mathbf{Y} \\ &= \sum_{i=1}^{n} W_i^1(\mathbf{x}) Y_i, \end{aligned} \tag{5.6}$$

where

$$W_i^1(\mathbf{x}) := \mathbf{e}_1' (\mathbf{X}'\mathbf{W}\mathbf{X})^{-1} \mathbf{X}'\mathbf{W}\mathbf{e}_i.$$

Therefore, the local linear estimator is a **weighted linear combination of the responses**. Differently to the Nadaraya–Watson estimator, this linear combination is *not* a weighted mean in general, since the weights $W_i^1(\mathbf{x})$ can be negative despite $\sum_{i=1}^n W_i^1(\mathbf{x}) = 1$.

Exercise 5.2. Implement an R function to compute $\hat{m}(\mathbf{x}; 1, h)$ for an arbitrary dimension p. The function must receive as arguments the sample $(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)$, the bandwidth vector $\mathbf{h}$, and a collection of evaluation points $\mathbf{x}$. Implement the function by computing the design matrix $\mathbf{X}$, the weight matrix $\mathbf{W}$, the vector of responses $\mathbf{Y}$, and then applying (5.6). Test the implementation in the example below.

The following code performs the local regression estimators with two predictors and exemplifies how the use of `np::npregbw` and `np::npreg` is analogous in this bivariate case.

```
# Sample data from a bivariate regression
n <- 300
set.seed(123456)
X <- mvtnorm::rmvnorm(n = n, mean = c(0, 0),
                      sigma = matrix(c(2, 0.5, 0.5, 1.5), nrow = 2, ncol = 2))
m <- function(x) 0.5 * (x[, 1]^2 + x[, 2]^2)
epsilon <- rnorm(n)
Y <- m(x = X) + epsilon

# Plot sample and regression function
```

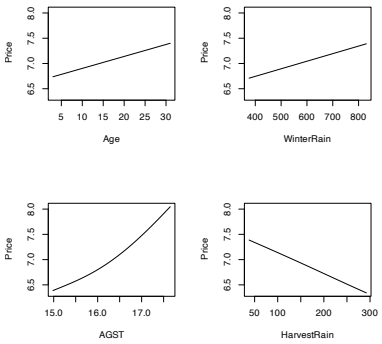
[10] Recall that now the entries of $\hat{\boldsymbol{\beta}}_{\mathbf{h}}$ are estimating $\boldsymbol{\beta} = \begin{pmatrix} m(\mathbf{x}) \\ \mathrm{D}m(\mathbf{x}) \end{pmatrix}$. So, $\hat{\boldsymbol{\beta}}_{\mathbf{h}}$ gives also an estimate of the gradient $\mathrm{D}m$ evaluated at $\mathbf{x}$.

```
rgl::plot3d(x = X[, 1], y = X[, 2], z = Y, xlim = c(-3, 3), ylim = c(-3, 3),
            zlim = c(-4, 10), xlab = "X1", ylab = "X2", zlab = "Y")
lx <- ly <- 50
x_grid <- seq(-3, 3, l = lx)
y_grid <- seq(-3, 3, l = ly)
xy_grid <- as.matrix(expand.grid(x_grid, y_grid))
rgl::surface3d(x = x_grid, y = y_grid,
               z = matrix(m(xy_grid), nrow = lx, ncol = ly),
               col = "lightblue", alpha = 1, lit = FALSE)

# Local constant fit

# An alternative for calling np::npregbw without formula
bw0 <- np::npregbw(xdat = X, ydat = Y, regtype = "lc")
kre0 <- np::npreg(bws = bw0, exdat = xy_grid) # Evaluation grid is now a matrix
rgl::surface3d(x = x_grid, y = y_grid,
               z = matrix(kre0$mean, nrow = lx, ncol = ly),
               col = "red", alpha = 0.25, lit = FALSE)

# Local linear fit
bw1 <- np::npregbw(xdat = X, ydat = Y, regtype = "ll")
kre1 <- np::npreg(bws = bw1, exdat = xy_grid)
rgl::surface3d(x = x_grid, y = y_grid,
               z = matrix(kre1$mean, nrow = lx, ncol = ly),
               col = "green", alpha = 0.25, lit = FALSE)
rgl::rglwidget()
```

Let's see an application of multivariate kernel regression for the `wine.csv` dataset. The objective in this dataset is to explain and predict the quality of a vintage, measured as its `Price`, by means of predictors associated with the vintage.

```
# Load the wine dataset
wine <- read.table(file = "datasets/wine.csv", header = TRUE, sep = ",")

# Bandwidth by CV for local linear estimator -- a product kernel with
# 4 bandwidths
# Employs 4 random starts for minimizing the CV surface
bw_wine <- np::npregbw(formula = Price ~ Age + WinterRain + AGST +
                         HarvestRain, data = wine, regtype = "ll")
bw_wine
##
## Regression Data (27 observations, 4 variable(s)):
##
##                   Age WinterRain      AGST HarvestRain
## Bandwidth(s): 3616691  290170218 0.8725243    106.5079
##
## Regression Type: Local-Linear
## Bandwidth Selection Method: Least Squares Cross-Validation
## Formula: Price ~ Age + WinterRain + AGST + HarvestRain
## Bandwidth Type: Fixed
## Objective Function Value: 0.0873955 (achieved on multistart 4)
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 4

# Regression
fit_wine <- np::npreg(bw_wine)
summary(fit_wine)
##
## Regression Data: 27 training points, in 4 variable(s)
##                   Age WinterRain      AGST HarvestRain
## Bandwidth(s): 3616691  290170218 0.8725243    106.5079
##
## Kernel Regression Estimator: Local-Linear
## Bandwidth Type: Fixed
## Residual standard error: 0.2079691
```

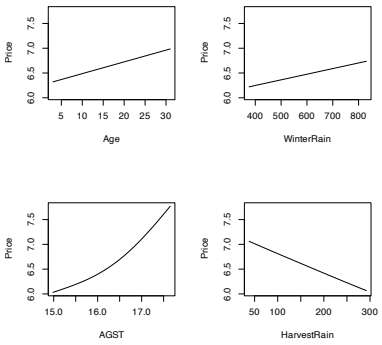
```
## R-squared: 0.8889649
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 4

# Plot the "marginal effects of each predictor" on the response
plot(fit_wine)
```



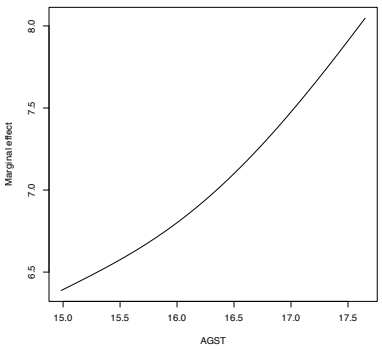

```
# These marginal effects are the p profiles of the estimated regression surface
# \hat{m}(x_1, ..., x_p) that are obtained by fixing the predictors to each of
# their median values. For example, the profile for Age is the curve
# \hat{m}(x, median_WinterRain, median_AGST, median_HarvestRain). The medians
# are:
apply(wine[c("Age", "WinterRain", "AGST", "HarvestRain")], 2, median)
##         Age  WinterRain        AGST HarvestRain
##     16.0000    600.0000     16.4167    123.0000

# Therefore, conditionally on the median values of the predictors:
# - Age is positively related to Price (almost linearly)
# - WinterRain is positively related to Price (with a subtle nonlinearity)
# - AGST is positively related to Price, but now we see what it looks like a
#   quadratic pattern
# - HarvestRain is negatively related to Price (almost linearly)
```

The many options for the `plot` method for `np::npreg` can be seen at `?np::npplot`. We illustrate as follows some of them with a view to enhancing the interpretability of the marginal effects.

```
# The argument "xq" controls the conditioning quantile of the predictors, by
# default the median (xq = 0.5). But xq can be a vector of p quantiles, for
# example (0.25, 0.5, 0.25, 0.75) for (Age, WinterRain, AGST, HarvestRain)
plot(fit_wine, xq = c(0.25, 0.5, 0.25, 0.75))
```

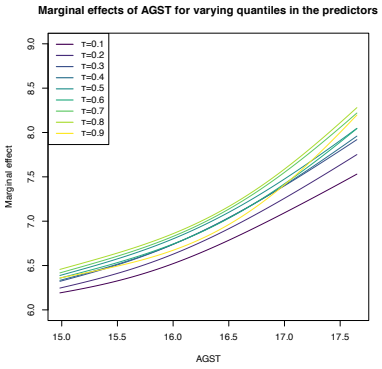



```
# With "plot.behavior = data" the plot() function returns a list with the data
# for performing the plots
res <- plot(fit_wine, xq = 0.5, plot.behavior = "data")
str(res, 1)
## List of 4
##  $ r1:List of 35
##   ..- attr(*, "class")= chr "npregression"
##  $ r2:List of 35
##   ..- attr(*, "class")= chr "npregression"
##  $ r3:List of 35
##   ..- attr(*, "class")= chr "npregression"
##  $ r4:List of 35
##   ..- attr(*, "class")= chr "npregression"

# Plot the marginal effect of AGST ($r3) alone
head(res$r3$eval) # All the predictors are constant (medians, except AGST)
##   V1  V2       V3  V4
## 1 16 600 14.98330 123
## 2 16 600 15.03772 123
## 3 16 600 15.09214 123
## 4 16 600 15.14657 123
## 5 16 600 15.20099 123
## 6 16 600 15.25541 123
plot(res$r3$eval$V3, res$r3$mean, type = "l", xlab = "AGST",
     ylab = "Marginal effect")
```




```
# Plot the marginal effects of AGST for varying quantiles in the rest of
# predictors (all with the same quantile)
tau <- seq(0.1, 0.9, by = 0.1)
```

```
res <- plot(fit_wine, xq = tau[1], plot.behavior = "data")
col <- viridis::viridis(length(tau))
plot(res$r3$eval$V3, res$r3$mean, type = "l", xlab = "AGST",
     ylab = "Marginal effect", col = col[1], ylim = c(6, 9),
     main = "Marginal effects of AGST for varying quantiles in the predictors")
for (i in 2:length(tau)) {
  res <- plot(fit_wine, xq = tau[i], plot.behavior = "data")
  lines(res$r3$eval$V3, res$r3$mean, col = col[i])
}
legend("topleft", legend = latex2exp::TeX(paste0("$\\tau =", tau, "$")),
       col = col, lwd = 2)
```




```
# These quantiles are
apply(wine[c("Age", "WinterRain", "HarvestRain")], 2, quantile, prob = tau)
##      Age WinterRain HarvestRain
## 10%  5.6      419.2        65.2
## 20%  8.2      508.8        86.2
## 30% 10.8      567.8        94.6
## 40% 13.4      576.2       114.4
## 50% 16.0      600.0       123.0
## 60% 18.6      609.2       156.8
## 70% 21.2      691.4       173.6
## 80% 23.8      716.4       187.0
## 90% 26.8      785.4       255.0
```

The `summary` of `np::npreg` returns an R^2. This statistic is defined as

$$R^2 := \frac{(\sum_{i=1}^n (Y_i - \bar{Y})(\hat{Y}_i - \bar{Y}))^2}{(\sum_{i=1}^n (Y_i - \bar{Y})^2)(\sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2)}$$

and is *neither* the squared correlation coefficient between $Y_1, \ldots, Y_n$ and $\hat{Y}_1, \ldots, \hat{Y}_n$ [11,12] *nor* "the percentage of variance explained" by the model – this interpretation makes sense within the linear model context only. It is however a quantity in $[0, 1]$ that attains $R^2 = 1$ whenever the fit is perfect (zero variability about the curve), so it can give an idea of how explicative the estimated regression function is.

[11] $\hat{Y}_i := \hat{m}(\mathbf{X}_i; q, \mathbf{h})$, $i = 1, \ldots, n$; see Section 5.3.

[12] Because it is not guaranteed that $\bar{Y} = \bar{\hat{Y}}$, what it is required to turn R^2 into that squared correlation coefficient.

5.1.2 *Kernel regression with mixed data*

Non-continuous predictors can also be taken into account in non-parametric regression. The key to do so is an adequate definition of a **suitable kernel function** for ***any* random variable** X, not just continuous. Therefore, we need to find

> a positive function that is a pdf[13] on the support of X and that allows to assign more weight to observations of the random variable that are close to a given point.

[13] Although in practice it is not required for the kernel to integrate one for performing regression, as the constants of the kernels cancel out in the quotients in $\hat{m}(\cdot; q, \mathbf{h})$.

If such kernel is adequately defined, then it can be readily employed in the weights of the linear combinations that form $\hat{m}(\cdot; q, \mathbf{h})$.

We analyze next the two main possibilities for non-continuous variables:

- *Categorical or unordered discrete variables*. For example, `iris$species` and `Auto$origin` are categorical variables in which ordering

[14] Recall $K_h(x - X_i)$.

does not make any sense. Categorical variables are specified in base R by `factor`. Due to the lack of ordering, the basic mathematical operation behind a kernel, a distance computation[14], is delicate. This motivates the Aitchison and Aitken (1976) kernel.

Assume that the categorical random variable X_d has u_d different levels. These levels can be represented as $U_d := \{0, 1, \ldots, u_d - 1\}$. For $x_d, X_d \in U_d$, the Aitchison and Aitken (1976) **unordered discrete kernel** is

$$l_u(x_d, X_d; \lambda) := \begin{cases} 1 - \lambda, & \text{if } x_d = X_d, \\ \frac{\lambda}{u_d - 1}, & \text{if } x_d \neq X_d, \end{cases}$$

where $\lambda \in [0, (u_d - 1)/u_d]$ is the bandwidth. Observe that this kernel is constant if $x_d \neq X_d$: since the levels of the variable are unordered, there is no sense of proximity between them. In other words, the kernel only distinguishes if two levels are equal or not to assign weight. If $\lambda = 0$, then no weight is assigned if $x_d \neq X_d$ and no information is borrowed from different levels; hence, nonparametrically regressing Y onto X_d is equivalent to doing u_d separate nonparametric regressions. If $\lambda = (u_d - 1)/u_d$, then all levels are assigned the same weight, regardless of x_d; hence, X_d is irrelevant for explaining Y.

- *Ordinal or ordered discrete variables*. For example, `wine$Year` and `Auto$cylinders` are discrete variables with clear orders, but they are not continuous. These variables are specified by `ordered` (an ordered `factor` in base R). Despite the existence of an ordering, the possible distances between the observations of these variables are discrete.

  Assume that the ordered discrete random variable X_d takes values in a set O_d. For $x_d, X_d \in O_d$, a possible (Li and Racine, 2007) **ordered discrete kernel** is[15]

  $$l_o(x_d, X_d; \eta) := \eta^{|x_d - X_d|},$$

  where $\eta \in [0, 1]$ is the bandwidth.[16] If $\eta = 0, 1$, then

  $$l_o(x_d, X_d; 0) = \begin{cases} 0, & x_d \neq X_d, \\ 1, & x_d = X_d, \end{cases} \quad l_o(x_d, X_d; 1) = 1.$$

  Hence, if $\eta = 0$, nonparametrically regressing Y onto X_d is equivalent to doing separate nonparametric regressions for each of the levels of X_d. If $\eta = 1$, X_d is irrelevant for explaining Y.

Exercise 5.3. Show that, for any $X_d \in U_d$ and any $\lambda \in [0, (u_d - 1)/u_d]$, the kernel $x \mapsto l_u(x, X_d; \lambda)$ "integrates" one over U_d.

Once we have defined the suitable kernels for ordered and unordered discrete variables, we can "aggregate" information from nearby observations of *mixed* data. Assume that, among the $p = p_c + p_u + p_o$ predictors, the first p_c are continuous, the next p_u

[15] Notice that this kernel does not integrate one! To normalize it we should take the specific support of X_d into account, which, e.g., could be $\{0, 1, 2, 3\}$ or $\mathbb{N}$. This is slightly cumbersome, so we avoid it because there is no difference between normalized and unnormalized kernels for *regression*: the kernel normalizing constants cancel out in the computation of the weights (5.8).

[16] Importantly, note that this kernel is assumming that the levels are equally-separated! Care is needed in applications when dealing with unequally-separated ordinal variables and using `ordered` with the default `levels` specification.

are discrete unordered (or categorical), and the last p_o are discrete ordered (or ordinal). The Nadaraya–Watson for mixed multivariate data is defined as

$$\hat{m}(\mathbf{x}; 0, (\mathbf{h}_c, \boldsymbol{\lambda}_u, \boldsymbol{\eta}_o)) := \sum_{i=1}^{n} W_i^0(\mathbf{x}) Y_i, \tag{5.7}$$

with the weights of the estimator being

$$W_i^0(\mathbf{x}) = \frac{L_\Pi(\mathbf{x}, \mathbf{X}_i)}{\sum_{j=1}^{n} L_\Pi(\mathbf{x}, \mathbf{X}_j)}. \tag{5.8}$$

The weights are based on the mixed product kernel

$$L_\Pi(\mathbf{x}, \mathbf{X}_i) := \prod_{j=1}^{p_c} K_{h_j}(x_j - X_{ij}) \prod_{k=1}^{p_u} l_u(x_k, X_{ik}; \lambda_k) \prod_{\ell=1}^{p_o} l_o(x_\ell, X_{i\ell}; \eta_\ell),$$

which features the vectors of bandwidths $\mathbf{h}_c = (h_1, \ldots, h_{p_c})'$, $\boldsymbol{\lambda}_u = (\lambda_1, \ldots, \lambda_{p_u})'$, and $\boldsymbol{\eta}_o = (\eta_1, \ldots, \eta_{p_o})'$ for each cluster of equal-type predictors.

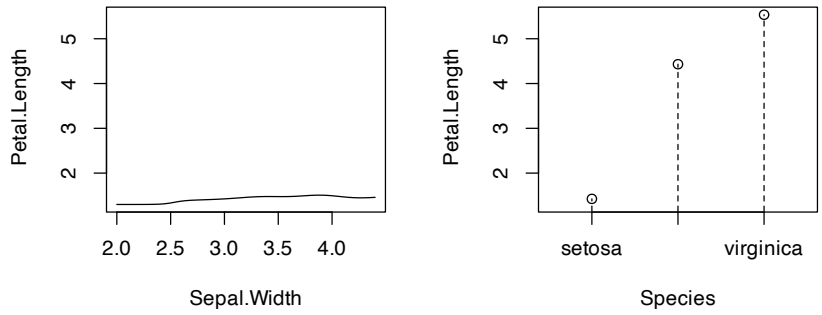
The adaptation of the local linear estimator is conceptually similar to that of the local constant, but it is more cumbersome as linear approximations make sense for quantitative variables, but not for unordered predictors. Therefore, the local linear estimator with mixed predictors is skipped in these notes.

The `np` package employs a variation of the previous kernels and implements the local constant and linear estimators for mixed multivariate data.

```
# Bandwidth by CV for local linear estimator
# Recall that Species is a factor!
bw_iris <- np::npregbw(formula = Petal.Length ~ Sepal.Width + Species,
                       data = iris)
bw_iris
##
## Regression Data (150 observations, 2 variable(s)):
##
##              Sepal.Width      Species
## Bandwidth(s):   0.1900724 9.149043e-09
##
## Regression Type: Local-Constant
## Bandwidth Selection Method: Least Squares Cross-Validation
## Formula: Petal.Length ~ Sepal.Width + Species
## Bandwidth Type: Fixed
## Objective Function Value: 0.1564197 (achieved on multistart 2)
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
##
## Unordered Categorical Kernel Type: Aitchison and Aitken
## No. Unordered Categorical Explanatory Vars.: 1
# Product kernel with 2 bandwidths

# Regression
fit_iris <- np::npreg(bw_iris)
summary(fit_iris)
##
## Regression Data: 150 training points, in 2 variable(s)
##              Sepal.Width      Species
## Bandwidth(s):   0.1900724 9.149043e-09
##
```

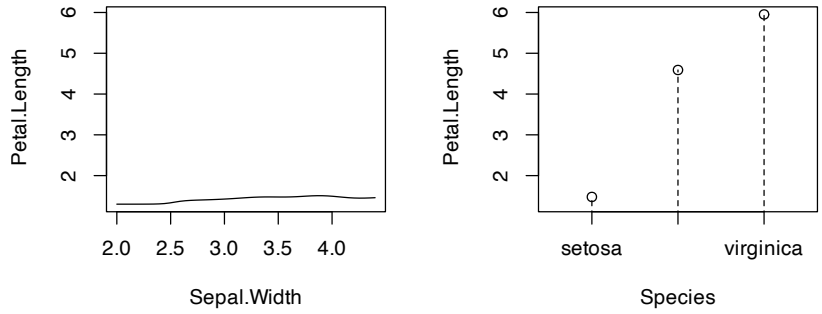
```
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
## Residual standard error: 0.3715506
## R-squared: 0.9554132
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
##
## Unordered Categorical Kernel Type: Aitchison and Aitken
## No. Unordered Categorical Explanatory Vars.: 1
```

```
# Plot marginal effects (for quantile 0.5) of each predictor on the response
par(mfrow = c(1, 2))
plot(fit_iris, plot.par.mfrow = FALSE)
```




```
# Options for the plot method for np::npreg can be seen at ?np::npplot
```

```
# Plot marginal effects (for quantile 0.9) of each predictor on the response
par(mfrow = c(1, 2))
plot(fit_iris, xq = 0.9, plot.par.mfrow = FALSE)
```



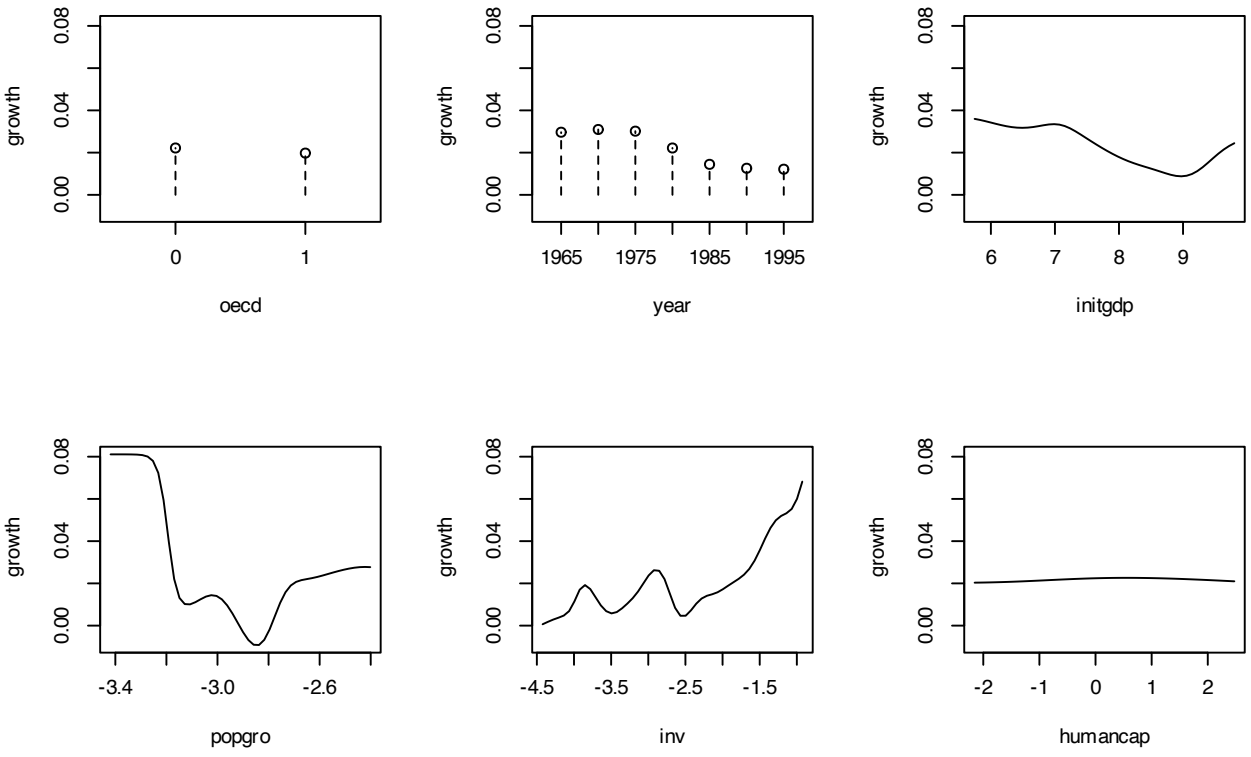
The following is an example from `?np::npreg`: the modeling of the GDP growth of a country from economic indicators. The predictors contain a mix of unordered, ordered, and continuous variables.

```
# Load data
data(oecdpanel, package = "np")
```

```
# Bandwidth by CV for local constant -- use only two starts to reduce the
# computation time
bw_OECD <- np::npregbw(formula = growth ~ oecd + ordered(year) +
                         initgdp + popgro + inv + humancap, data = oecdpanel,
                       regtype = "lc", nmulti = 2)
bw_OECD
##
## Regression Data (616 observations, 6 variable(s)):
##
##                      oecd ordered(year)   initgdp    popgro       inv  humancap
## Bandwidth(s): 0.01206227     0.8794879 0.3329921 0.0597245 0.1594767 0.9407897
##
## Regression Type: Local-Constant
## Bandwidth Selection Method: Least Squares Cross-Validation
```

```
## Formula: growth ~ oecd + ordered(year) + initgdp + popgro + inv + humancap
## Bandwidth Type: Fixed
## Objective Function Value: 0.0006358702 (achieved on multistart 2)
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 4
##
## Unordered Categorical Kernel Type: Aitchison and Aitken
## No. Unordered Categorical Explanatory Vars.: 1
##
## Ordered Categorical Kernel Type: Li and Racine
## No. Ordered Categorical Explanatory Vars.: 1

# Recall that ordered(year) is doing an in-formula transformation of year,
# which is *not* codified as an ordered factor in the oecdpanel dataset
# Therefore, if ordered() was not present, year would have been treated
# as continuous, as illustrated below
np::npregbw(formula = growth ~ oecd + year + initgdp + popgro +
              inv + humancap, data = oecdpanel, regtype = "lc", nmulti = 2)
##
## Regression Data (616 observations, 6 variable(s)):
##
##                      oecd     year   initgdp     popgro       inv  humancap
## Bandwidth(s): 0.01644512 7.481232 0.3486883 0.05860077 0.1639842 0.8371089
##
## Regression Type: Local-Constant
## Bandwidth Selection Method: Least Squares Cross-Validation
## Formula: growth ~ oecd + year + initgdp + popgro + inv + humancap
## Bandwidth Type: Fixed
## Objective Function Value: 0.0006338394 (achieved on multistart 1)
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 5
##
## Unordered Categorical Kernel Type: Aitchison and Aitken
## No. Unordered Categorical Explanatory Vars.: 1

# A cleaner approach to avoid doing the in-formula transformation, which
# may be problematic when using predict() or np_pred_CI(), is to directly
# change in the dataset the nature of the factor/ordered variables that are
# not codified as such. For example:
oecdpanel$year <- ordered(oecdpanel$year)
bw_OECD <- np::npregbw(formula = growth ~ oecd + year + initgdp + popgro +
                         inv + humancap, data = oecdpanel,
                       regtype = "lc", nmulti = 2)
bw_OECD
##
## Regression Data (616 observations, 6 variable(s)):
##
##                      oecd      year  initgdp     popgro       inv  humancap
## Bandwidth(s): 0.01206276 0.8794881 0.332989 0.05972475 0.1594737 0.9408023
##
## Regression Type: Local-Constant
## Bandwidth Selection Method: Least Squares Cross-Validation
## Formula: growth ~ oecd + year + initgdp + popgro + inv + humancap
## Bandwidth Type: Fixed
## Objective Function Value: 0.0006358702 (achieved on multistart 2)
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 4
##
## Unordered Categorical Kernel Type: Aitchison and Aitken
## No. Unordered Categorical Explanatory Vars.: 1
##
## Ordered Categorical Kernel Type: Li and Racine
## No. Ordered Categorical Explanatory Vars.: 1

# Regression
```

```
fit_OECD <- np::npreg(bw_OECD)
summary(fit_OECD)
##
## Regression Data: 616 training points, in 6 variable(s)
##                      oecd      year  initgdp     popgro       inv  humancap
## Bandwidth(s): 0.01206276 0.8794881 0.332989 0.05972475 0.1594737 0.9408023
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
## Residual standard error: 0.01737053
## R-squared: 0.7143326
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 4
##
## Unordered Categorical Kernel Type: Aitchison and Aitken
## No. Unordered Categorical Explanatory Vars.: 1
##
## Ordered Categorical Kernel Type: Li and Racine
## No. Ordered Categorical Explanatory Vars.: 1

# Plot marginal effects of each predictor on the response
par(mfrow = c(2, 3))
plot(fit_OECD, plot.par.mfrow = FALSE)
```



Exercise 5.4. Obtain the local constant estimator of the regression of `mpg` on `cylinders` (ordered discrete), `horsepower`, `weight`, and `origin` in the `data(Auto, package = "ISLR")` dataset. Summarize the model and plot the marginal effects of each predictor on the response (for the quantile 0.5).

5.2 *Bandwidth selection*

Cross-validatory bandwidth selection, as studied in Section 4.3, extends neatly to the mixed multivariate case. For the fully continuous case, the least-squares cross validation selector is defined as

$$\mathrm{CV}(\mathbf{h}) := \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{m}_{-i}(\mathbf{X}_i; q, \mathbf{h}))^2,$$
$$\hat{\mathbf{h}}_{\mathrm{CV}} := \arg\min_{h_1,\ldots,h_p>0} \mathrm{CV}(\mathbf{h}).$$

The cross-validation objective function becomes more challenging to minimize as p grows. This is the reason why employing several starting values for optimizing it (as `np` does) is advisable.

The mixed case is defined in a completely analogous manner by just replacing continuous kernels $K_h(\cdot)$ with categorical $l_u(\cdot, \cdot; \lambda)$ or ordered discrete $l_o(\cdot, \cdot; \eta)$ kernels.

Importantly, the trick described in Proposition 4.1 holds with obvious modifications. It also holds for the mixed case and the Nadaraya–Watson estimator.

Proposition 5.1. *For $q = 0, 1$, the weights of the leave-one-out estimator $\hat{m}_{-i}(\mathbf{x}; q, h) = \sum_{\substack{j=1 \\ j\neq i}}^{n} W^q_{-i,j}(\mathbf{x}) Y_j$ can be obtained from $\hat{m}(\mathbf{x}; q, h) = \sum_{i=1}^{n} W^q_i(\mathbf{x}) Y_i$:*

$$W^q_{-i,j}(\mathbf{x}) = \frac{W^q_j(\mathbf{x})}{\sum_{\substack{k=1 \\ k\neq i}}^{n} W^q_k(\mathbf{x})} = \frac{W^q_j(\mathbf{x})}{1 - W^q_i(\mathbf{x})}. \tag{5.9}$$

This implies that

$$\mathrm{CV}(\mathbf{h}) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i - \hat{m}(\mathbf{X}_i; q, h)}{1 - W^q_i(\mathbf{X}_i)}\right)^2. \tag{5.10}$$

Remark. As in the univariate case, computing (5.10) requires evaluating the local polynomial estimator at the sample $\{\mathbf{X}_i\}_{i=1}^n$ *and* obtaining $\{W^q_i(\mathbf{X}_i)\}_{i=1}^n$ (which are needed to evaluate $\hat{m}(\mathbf{X}_i; q, h)$). Both tasks can be achieved simultaneously from the $n \times n$ matrix $\left(W^q_i(\mathbf{X}_j)\right)_{ij}$. Evaluating $\hat{m}_{-i}(\mathbf{x}; q, h)$, because of (5.9), can be done with the weights $\{W^q_i(\mathbf{x})\}_{i=1}^n$.

Exercise 5.5. Implement an R function to compute (5.10) for the local constant estimator with multivariate (continuous) predictor. The function must receive as arguments the sample $(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)$ and the bandwidth vector $\mathbf{h}$. Use the normal kernel. Test your implementation by:

1. Simulating a random sample from a regression model with two predictors.
2. Computing its cross-validation bandwidths via `np::npregbw`.
3. Plotting a contour of the function $(h_1, h_2) \mapsto \mathrm{CV}(h_1, h_2)$ and checking that the minimizers and minimum of this surface coincide with the solution given by `np::npregbw`.

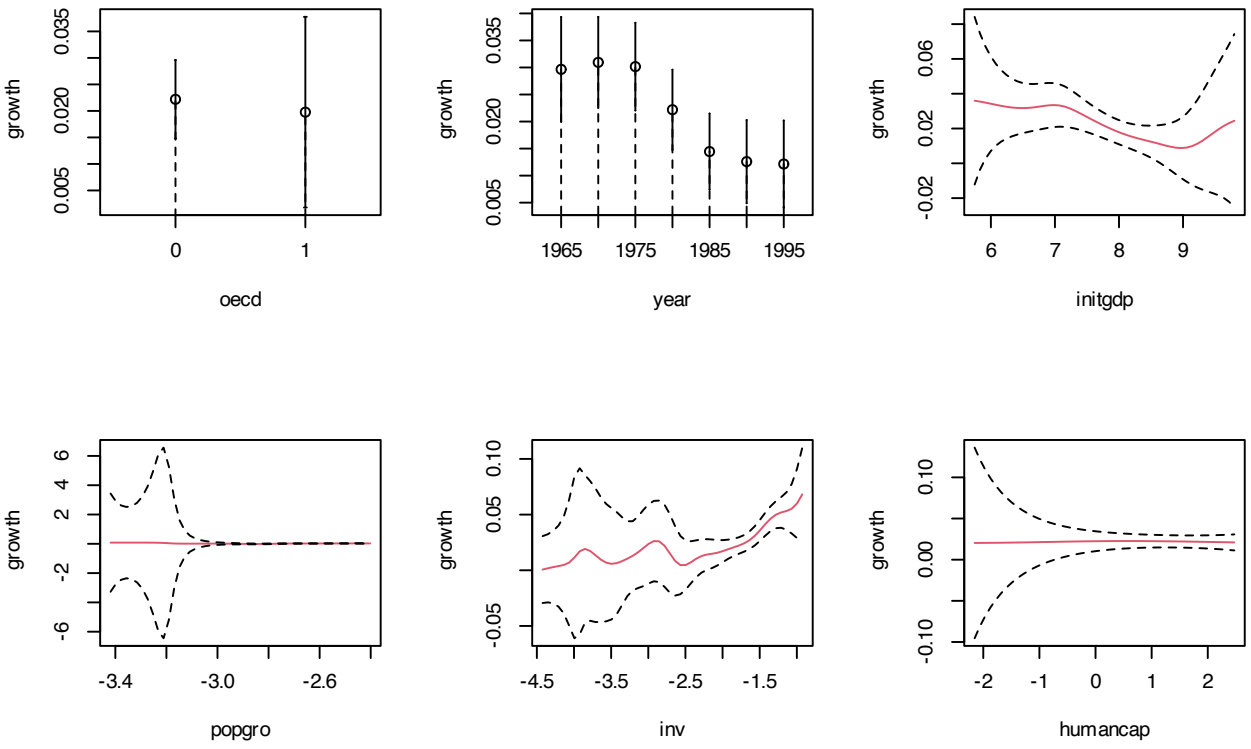
5.3 *Prediction and confidence intervals*

Prediction via the conditional expectation $m(\mathbf{x}) = \mathbb{E}[Y|\mathbf{X} = \mathbf{x}]$ reduces to evaluate $\hat{m}(\mathbf{x}; q, \mathbf{h})$. The fitted values are, therefore, $\hat{Y}_i := \hat{m}(\mathbf{X}_i; q, \mathbf{h})$, $i = 1, \ldots, n$. The `np` package has methods to perform these operations via the `predict` and `fitted` functions.

More interesting is the discussion about the *uncertainty* of $\hat{m}(\mathbf{x}; q, \mathbf{h})$ and, as a consequence, of the predictions. Differently to what happened in parametric models, in nonparametric regression there is no parametric distribution of the response that can help to carry out the inference and, consequently, to address the uncertainty of the estimation. Because of this, it is required to resort to somewhat convoluted asymptotic expressions[17] that rely on plugging-in estimates for the unknown terms on the asymptotic bias and variance. The next code chunk exemplifies how to compute asymptotic confidence intervals with `np`, both for the marginal effects and the conditional expectation $m(\mathbf{x})$. In the latter case, the confidence intervals are $(\hat{m}(\mathbf{x}; q, \mathbf{h}) \pm z_{\alpha/2} \hat{\text{se}}(\hat{m}(\mathbf{x}; q, \mathbf{h})))$, where $\hat{\text{se}}(\hat{m}(\mathbf{x}; q, \mathbf{h}))$ is the asymptotic estimation of the standard deviation of $\hat{m}(\mathbf{x}; q, \mathbf{h})$.

[17] Precisely, the ones associated with the asymptotic normality of $\hat{m}(\mathbf{x}; q, \mathbf{h})$ and that are collected in Theorem 4.2 for the case of a single continuous predictor.

```
# Asymptotic confidence bands for the marginal effects of each predictor on
# the response
par(mfrow = c(2, 3))
plot(fit_OECD, plot.errors.method = "asymptotic", common.scale = FALSE,
     plot.par.mfrow = FALSE, col = 2)
```




```
# The asymptotic standard errors associated with the regression evaluated at
# the evaluation points are in $merr
head(fit_OECD$merr)
```

```
## [1] 0.007982761 0.006815515 0.002888752 0.002432522 0.005004411 0.008483572

# Recall that in $mean we had the regression evaluated at the evaluation points,
# by default the sample of the predictors, so in this case it is the same as
# the fitted values
head(fit_OECD$mean)
## [1] 0.02213477 0.02460647 0.03216185 0.04167737 0.01068088 0.04456299

# Fitted values
head(fitted(fit_OECD))
## [1] 0.02213477 0.02460647 0.03216185 0.04167737 0.01068088 0.04456299

# Prediction for the first 3 points + asymptotic standard errors
pred <- predict(fit_OECD, newdata = oecdpanel[1:3, ], se.fit = TRUE)

# Predictions
pred$fit
## [1] 0.02213477 0.02460647 0.03216185

# Manual computation of the asymptotic 100 * (1 - alpha)% confidence intervals
# for the conditional mean of the first 3 points
alpha <- 0.05
z_alpha2 <- qnorm(1 - alpha / 2)
cbind(pred$fit - z_alpha2 * pred$se.fit, pred$fit + z_alpha2 * pred$se.fit)
##            [,1]       [,2]
## [1,] 0.00648885 0.03778070
## [2,] 0.01124831 0.03796464
## [3,] 0.02649999 0.03782370

# Recall that z_alpha2 is almost 2
z_alpha2
## [1] 1.959964
```

A non-asymptotic alternative to approximate the sampling distribution of $\hat{m}(\mathbf{x}; q, \mathbf{h})$ is a *bootstrap resampling procedure*. Indeed, bootstrap approximations of confidence intervals for $m(\mathbf{x})$, if performed adequately, are generally more trustworthy than asymptotic approximations.[18] **Naive bootstrap** resampling is `np`'s default bootstrap procedure to compute confidence intervals for $m(\mathbf{x})$ using the estimator $\hat{m}(\mathbf{x}; q, \mathbf{h})$. This procedure is summarized as follows:

1. Compute $\hat{m}(\mathbf{x}; q, \mathbf{h}) = \sum_{i=1}^n W_i^q(\mathbf{x}) Y_i$ from the *original sample* $(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)$.
2. Enter the "bootstrap world". For $b = 1, \ldots, B$:
   i. Obtain the *bootstrap sample* $(\mathbf{X}_1^{*b}, Y_1^{*b}), \ldots, (\mathbf{X}_n^{*b}, Y_n^{*b})$ by drawing, with replacement, random observations from the set $\{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$.[19]
   ii. Compute $\hat{m}^{*b}(\mathbf{x}; q, \mathbf{h}) = \sum_{i=1}^n W_i^{q,*b}(\mathbf{x}) Y_i^{*b}$ from $(\mathbf{X}_1^{*b}, Y_1^{*b}), \ldots, (\mathbf{X}_n^{*b}, Y_n^{*b})$.
3. From $\hat{m}(\mathbf{x}; q, \mathbf{h})$ and $\{\hat{m}^{*b}(\mathbf{x}; q, \mathbf{h})\}_{b=1}^B$, compute a bootstrap $100(1-\alpha)\%$-confidence interval for $m(\mathbf{x})$. The two most common approaches for doing so are:
   - *Normal approximation* (`"standard"`): $\left(\hat{m}(\mathbf{x}; q, \mathbf{h}) \pm z_{\alpha/2} \hat{\text{se}}^*(\hat{m}(\mathbf{x}; q, \mathbf{h}))\right)$, where $\hat{\text{se}}^*(\hat{m}(\mathbf{x}; q, \mathbf{h}))$ is the sample standard deviation of $\{\hat{m}^{*b}(\mathbf{x}; q, \mathbf{h})\}_{b=1}^B$.
   - *Quantile-based* (`"quantile"`): $\left(m^*_{\alpha/2}, m^*_{1-\alpha/2}\right)$, where m^*_α is the (lower) sample α-quantile of $\{\hat{m}^{*b}(\mathbf{x}; q, \mathbf{h})\}_{b=1}^B$.

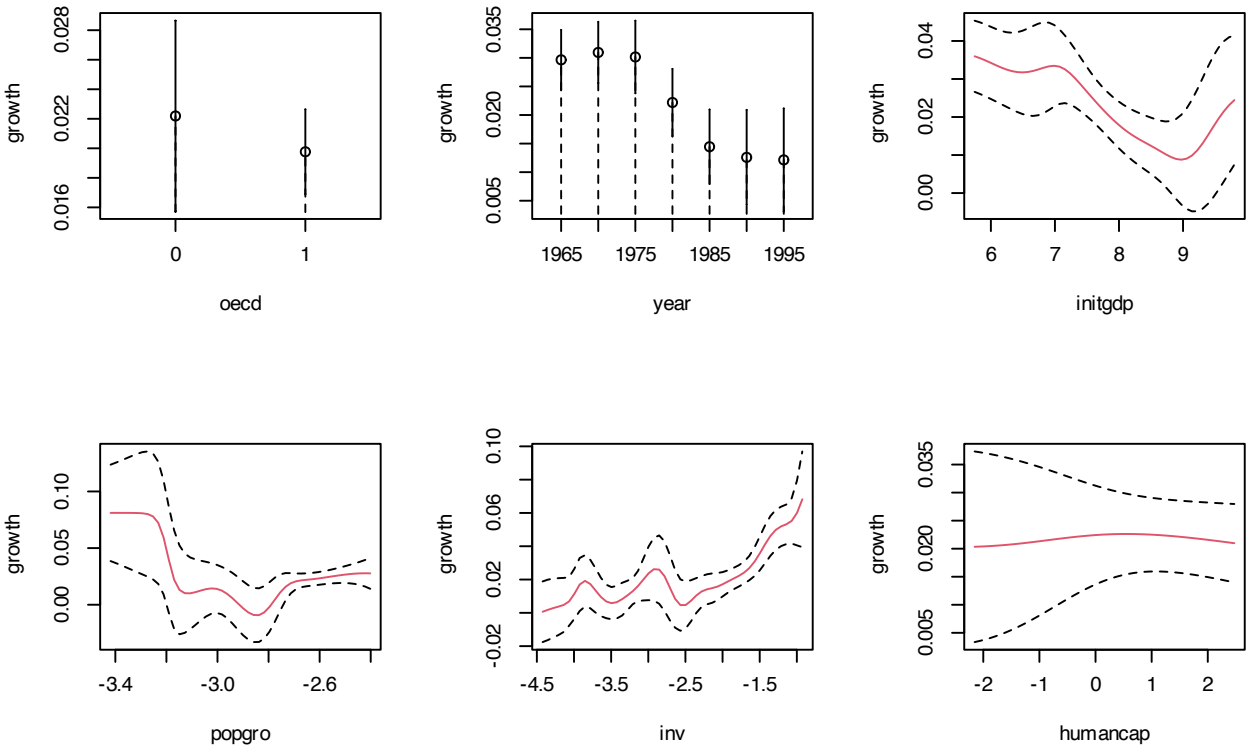
[18] As usable asymptotic approximations are hampered by two factors: (*i*) the slow convergence speed towards the asymptotic distribution (recall the effective sample size in Theorem 4.2); and (*ii*) the need to estimate the unknown terms in the asymptotic bias and variance, which leads to an endless cycle (similar to that in Section 2.4.1).

[19] This is the same as randomly accessing n times the rows of the matrix $\begin{pmatrix} X_{11} & \ldots & X_{1p} & Y_1 \\ \vdots & \ddots & \vdots & \vdots \\ X_{n1} & \ldots & X_{np} & Y_n \end{pmatrix}_{n \times (p+1)}$.

Bootstrap standard errors are entirely computed by `np::npplot` (and neither by `np::npreg` nor `predict`, as for asymptotic standard errors). The default method, `plot.errors.boot.method = "inid"`, implements the naive bootstrap to approximate the confidence intervals of the marginal effects.[20]

[20] Although this might be confusing, since Hayfield and Racine (2008) states that "Specifying the type of bootstrapping as 'inid' admits general heteroskedasticity of unknown form via the wild bootstrap (Liu, 1988), though it does not allow for dependence." but, as of version 0.60-10, the naive bootstrap is run when `plot.errors.boot.method = "inid"` (see code).

```
# Bootstrap confidence bands (using naive bootstrap, the default)
# They take more time to compute because a resampling + refitting takes place
B <- 200
par(mfrow = c(2, 3))
plot(fit_OECD, plot.errors.method = "bootstrap", common.scale = FALSE,
     plot.par.mfrow = FALSE, plot.errors.boot.num = B, random.seed = 42,
     col = 2)
```

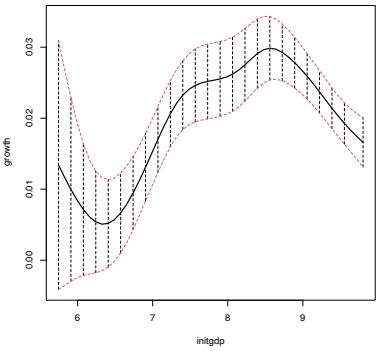



```
# plot.errors.boot.num is B and defaults to 399
# random.seed fixes the seed to always get the same bootstrap errors. It
# defaults to 42 if not specified
```

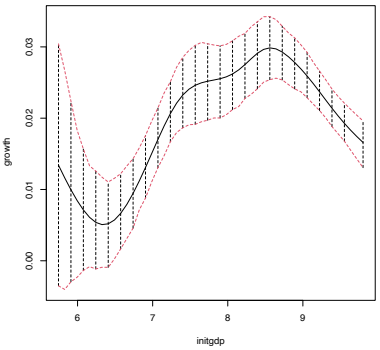
The following chunk of code helps extracting the information about the bootstrap confidence intervals from the call to `np::npplot` and illustrates further bootstrap options.

```
# Univariate local constant regression with CV bandwidth
bw1 <- np::npregbw(formula = growth ~ initgdp, data = oecdpanel, regtype = "lc")
fit1 <- np::npreg(bw1)
summary(fit1)
##
## Regression Data: 616 training points, in 1 variable(s)
##                 initgdp
## Bandwidth(s): 0.2774471
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
## Residual standard error: 0.02907224
```

```
## R-squared: 0.08275453
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1

# Asymptotic (not bootstrap) standard errors
head(fit1$merr)
## [1] 0.002229093 0.002342670 0.001712938 0.002210457 0.002390530 0.002333956

# Normal approximation confidence intervals + extraction of errors
npplot_std <- plot(fit1, plot.errors.method = "bootstrap",
                   plot.errors.type = "standard", plot.errors.boot.num = B,
                   plot.errors.style = "bar", plot.behavior = "plot-data",
                   lwd = 2)
lines(npplot_std$r1$eval[, 1], npplot_std$r1$mean + npplot_std$r1$merr[, 1],
      col = 2, lty = 2)
lines(npplot_std$r1$eval[, 1], npplot_std$r1$mean + npplot_std$r1$merr[, 2],
      col = 2, lty = 2)
```

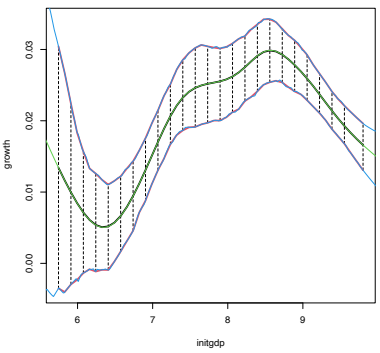



```
# These bootstrap standard errors are different from the asymptotic ones
head(npplot_std$r1$merr)
##              [,1]        [,2]
## [1,] -0.017509208 0.017509208
## [2,] -0.015174698 0.015174698
## [3,] -0.012930989 0.012930989
## [4,] -0.010925377 0.010925377
## [5,] -0.009276906 0.009276906
## [6,] -0.008031584 0.008031584

# Quantile confidence intervals + extraction of errors
npplot_qua <- plot(fit1, plot.errors.method = "bootstrap",
                   plot.errors.type = "quantiles", plot.errors.boot.num = B,
                   plot.errors.style = "bar", plot.behavior = "plot-data")
lines(npplot_qua$r1$eval[, 1], npplot_qua$r1$mean + npplot_qua$r1$merr[, 1],
      col = 2, lty = 2)
lines(npplot_qua$r1$eval[, 1], npplot_qua$r1$mean + npplot_qua$r1$merr[, 2],
      col = 2, lty = 2)
```

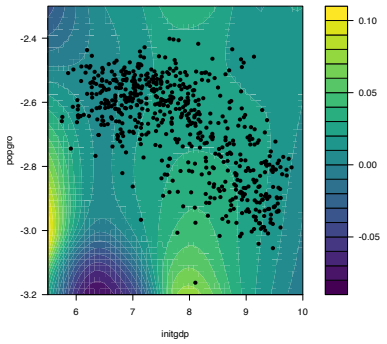



```
# These bootstrap standard errors are different from the asymptotic ones,
# and also from the previous bootstrap errors (different confidence
# interval method)
head(npplot_qua$r1$merr)
##              [,1]        [,2]
## [1,] -0.016923716 0.017015321
## [2,] -0.015649920 0.015060896
## [3,] -0.012913035 0.012441999
## [4,] -0.010752339 0.009864879
## [5,] -0.008496998 0.008547620
## [6,] -0.006912847 0.007220871

# There is no predict() method featuring bootstrap confidence intervals,
# it has to be coded manually!

# Function to predict and compute confidence intervals for m(x). Takes as main
# arguments a np::npreg object (npfit) and the values of the predictors where
# to carry out prediction (exdat). Requires that exdat is a data.frame of the
# same type than the one used for the predictors (e.g., there will be an error
# if one variable appears as a factor for computing npfit and then is passed
# as a numeric in exdat)
np_pred_CI <- function(npfit, exdat, B = 200, conf = 0.95,
                       type_CI = c("standard", "quantiles")[1]) {

  # Extract predictors
  xdat <- npfit$eval
```

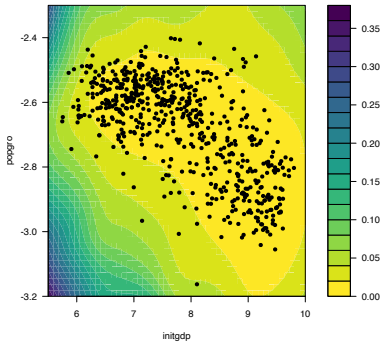
```
  # Extract response, using a trick from np::npplot.rbandwidth
  tt <- terms(npfit$bws)
  tmf <- npfit$bws$call[c(1, match(c("formula", "data"),
                                   names(npfit$bws$call)))]
  tmf[[1]] <- as.name("model.frame")
  tmf[["formula"]] <- tt
  tmf <- eval(tmf, envir = environment(tt))
  ydat <- model.response(tmf)

  # Predictions
  m_hat <- np::npreg(txdat = xdat, tydat = ydat, exdat = exdat,
                     bws = npfit$bws)$mean

  # Function for performing Step 3
  boot_function <- function(data, indices) {

    np::npreg(txdat = xdat[indices,], tydat = ydat[indices],
              exdat = exdat, bws = npfit$bws)$mean

  }

  # Carry out Step 3
  m_hat_star <- boot::boot(data = data.frame(xdat), statistic = boot_function,
                           R = B)$t

  # Confidence intervals
  alpha <- 1 - conf
  if (type_CI == "standard") {

    z <- qnorm(p = 1 - alpha / 2)
    se <- apply(m_hat_star, 2, sd)
    lwr <- m_hat - z * se
    upr <- m_hat + z * se

  } else if (type_CI == "quantiles") {

    q <- apply(m_hat_star, 2, quantile, probs = c(alpha / 2, 1 - alpha / 2))
    lwr <- q[1, ]
    upr <- q[2, ]

  } else {

    stop("Incorrect type_CI")

  }

  # Return evaluation points, estimates, and confidence intervals
  return(data.frame("exdat" = exdat, "m_hat" = m_hat, "lwr" = lwr, "upr" = upr))

}

# Obtain predictions and confidence intervals along a fine grid, using the
# same seed employed by np::npplot for proper comparison
set.seed(42)
ci1 <- np_pred_CI(npfit = fit1, B = B, exdat = seq(5, 10, by = 0.01),
                  type_CI = "quantiles")

# Reconstruction of np::npplot's figure -- the curves coincide perfectly
plot(fit1, plot.errors.method = "bootstrap", plot.errors.type = "quantiles",
     plot.errors.boot.num = B, plot.errors.style = "bar", lwd = 3)
lines(npplot_qua$r1$eval[, 1], npplot_qua$r1$mean + npplot_qua$r1$merr[, 1],
      col = 2, lwd = 3)
lines(npplot_qua$r1$eval[, 1], npplot_qua$r1$mean + npplot_qua$r1$merr[, 2],
      col = 2, lwd = 3)
lines(ci1$exdat, ci1$m_hat, col = 3)
lines(ci1$exdat, ci1$lwr, col = 4)
lines(ci1$exdat, ci1$upr, col = 4)
```



```
# But np_pred_CI is also valid for regression models with several predictors!
# An example with bivariate local linear regression with CV bandwidth
bw2 <- np::npregbw(formula = growth ~ initgdp + popgro, data = oecdpanel,
                   regtype = "ll")
fit2 <- np::npreg(bw2)

# Predictions and confidence intervals along a bivariate grid
L <- 50
x_initgdp <- seq(5.5, 10, l = L)
x_popgro <- seq(-3.2, -2.3, l = L)
exdat <- expand.grid(x_initgdp, x_popgro)
ci2 <- np_pred_CI(npfit = fit2, exdat = exdat)

# Regression surface. Observe the extrapolatory artifacts for
# low-density regions
m_hat <- matrix(ci2$m_hat, nrow = L, ncol = L)
filled.contour(x_initgdp, x_popgro, m_hat, nlevels = 20,
               color.palette = viridis::viridis,
               xlab = "initgdp", ylab = "popgro",
               plot.axes = {
                 axis(1); axis(2);
                 points(popgro ~ initgdp, data = oecdpanel, pch = 16)
                 })
```




```
# Length of the 95%-confidence intervals for the regression. Observe how
# they grow for low-density regions
ci_dif <- matrix(ci2$upr - ci2$lwr, nrow = L, ncol = L)
filled.contour(x_initgdp, x_popgro, ci_dif, nlevels = 20,
               color.palette = function(n) viridis::viridis(n, direction = -1),
               xlab = "initgdp", ylab = "popgro",
               plot.axes = {
                 axis(1); axis(2);
                 points(popgro ~ initgdp, data = oecdpanel, pch = 16)
                 })
```



Exercise 5.6. Split the `data(Auto, package = "ISLR")` dataset by taking `set.seed(12345); ind_train <- sample(nrow(Auto), size = nrow(Auto) - 12)` as the index for the training dataset and use the remaining observations for the validation dataset. Then, for the regression problem described in Exercise 5.4:

a. Fit the local constant and linear estimators with CV bandwidths by taking the nature of the variables into account. *Hint*: remember using `ordered` and `factor` if necessary.
b. Interpret the nonparametric fits via marginal effects (for the quantile 0.5) and bootstrap confidence intervals.
c. Obtain the mean squared prediction error on the validation dataset for the two fits. Which estimate gives the lowest error?
d. Compare the errors with the ones made by a linear estimation. Which approach gives lowest errors?

Exercise 5.7. The `data(ChickWeight)` dataset in R contains 578 observations of the `weight`, `Time`, and `Diet` of chicks.

a. Consider the regression `weight ~ Time + Diet`. Fit the local linear estimator with CV bandwidths by taking the nature of the variables into account.

b. Plot the marginal effects of `Time` for the four possible types of `Diet`. Overlay the four samples and use different colors. Explain your interpretations.
c. Obtain the bootstrap 90%-confidence intervals of the expected weight of a chick on the 20th day, for each of the four types of diets (take $B = 500$). With a 90% confidence, state which diet dominates in terms of weight the other diets, and which are comparable.
d. Produce a plot that shows the extrapolation of the expected weight, for each of the four diets, from the 20th to the 30th day. Add bootstrap 90%-confidence bands. Explain your interpretations.

Exercise 5.8. Investigate the accuracy of the naive bootstrap confidence intervals implemented in `np::npplot`. To do so:

1. Simulate $M = 500$ samples of size $n = 100$ from the regression model $Y = m(X) + \varepsilon$, where $m(x) = 0.25x^2 - 0.75x + 3$, $X \sim \mathcal{N}(0, 1.5^2)$, and $\varepsilon \sim \mathcal{N}(0, 0.75^2)$.
2. Compute the 95%-confidence intervals for $m(x)$ along `x <- seq(-5, 5, by = 0.1)`, for each of the M samples. Do it for the normal approximation and quantile-based confidence intervals.
3. Check if $m(x)$ belongs to each of the confidence intervals, for each x.
4. Approximate the actual coverage of the confidence intervals.

Once you have a working solution, increase n to $n = 200, 500$ and summarize your conclusions. Use $B = 500$.

Naive bootstrap, although conceptually simple, does not capture the uncertainty of $\hat{m}(\mathbf{x}; q, \mathbf{h})$ in all situations, and may severely underestimate the variability of $\hat{m}(\mathbf{x}; q, \mathbf{h})$. A more sophisticated bootstrap procedure is the so-called ***wild* bootstrap** (Wu, 1986; Liu, 1988; Hardle and Marron, 1991), as it is a resampling strategy particularly well-suited for regression problems with heteroskedasticity. The wild bootstrap approach replaces Step 2 in the previous bootstrap resampling with:

i. Simulate $V_1^{*b}, \ldots, V_n^{*b}$ to be iid copies of V such that $\mathbb{E}[V] = 0$ and $\mathbb{V}\text{ar}[V] = 1$.[21]
ii. Compute the *perturbed residuals* $\varepsilon_i^{*b} := \hat{\varepsilon}_i V_i^{*b}$, where $\hat{\varepsilon}_i := Y_i - \hat{m}(\mathbf{X}_i; q, \mathbf{h})$, $i = 1, \ldots, n$.
iii. Obtain the *bootstrap sample* $(\mathbf{X}_1, Y_1^{*b}), \ldots, (\mathbf{X}_n, Y_n^{*b})$, where $Y_i^{*b} := \hat{m}(\mathbf{X}_i; q, \mathbf{h}) + \varepsilon_i^{*b}$, $i = 1, \ldots, n$.
iv. Compute $\hat{m}^{*b}(\mathbf{x}; q, \mathbf{h}) = \sum_{i=1}^n W_i^q(\mathbf{x}) Y_i^{*b}$ from $(\mathbf{X}_1, Y_1^{*b}), \ldots, (\mathbf{X}_n, Y_n^{*b})$.

[21] For example, the *golden section* binary variable defined by $\mathbb{P}[V = 1 - \phi] = p$ and $\mathbb{P}[V = \phi] = 1 - p$, with $\phi = (1 + \sqrt{5})/2$ and $p = (\phi + 2)/5$.

Unfortunately, `np` does not seem to implement the wild bootstrap. The next exercise points to its implementation.

Exercise 5.9. Adapt the `np_pred_CI` function to include the argument `type_boot`, which can take either the value `"naive"` or `"wild"`. If `type_boot = "wild"`, then the function must perform the wild

bootstrap algorithm described above, implemented from scratch following Steps i–iv. Compare and validate the correct behavior of the confidence intervals, for the two specifications of `type_boot`, in the model considered in Exercise 5.7 (without performing the full simulation study).

5.4 *Local likelihood*

We next explore an extension of the local polynomial estimator that imitates the expansion that generalized linear models made of linear models, allowing the former to consider non-continuous responses. This extension is aimed to estimate the regression function by relying on the likelihood, rather than the least squares. The main idea behind local likelihood is, therefore, to **locally fit parametric models by maximum likelihood**.

We begin by seeing that local likelihood based on the linear model is equivalent to local polynomial modeling. Theorem B.1 shows that, under the assumptions given in Section B.1.2, the maximum likelihood estimate of $\boldsymbol{\beta}$ in the linear model

$$Y|(X_1,\ldots,X_p) \sim \mathcal{N}(\beta_0+\beta_1X_1+\cdots+\beta_pX_p,\sigma^2) \tag{5.11}$$

is equivalent to the least squares estimate, $\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$. The reason for this is the form of the conditional (on $X_1,\ldots,X_p$) log-likelihood:

$$\begin{aligned}\ell(\boldsymbol{\beta}) = &-\frac{n}{2}\log(2\pi\sigma^2)\\ &-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(Y_i-\beta_0-\beta_1X_{i1}-\ldots-\beta_pX_{ip})^2.\end{aligned}$$

If there is a single predictor X, the situation on which we focus on, a polynomial fitting of order p of the conditional mean can be achieved by the well-known trick of identifying the j-th predictor X_j in (5.11) by X^j.[22] This results in

$$Y|X \sim \mathcal{N}(\beta_0+\beta_1X+\cdots+\beta_pX^p,\sigma^2). \tag{5.12}$$

[22] Observe that, for the sake of easier presentation, we come back to the notation and setting employed in Chapter 4, in which only one predictor X was available for explaining Y, and where p denoted the order of the polynomial fit employed.

Therefore, we can define the *weighted log-likelihood* of the linear model (5.12) *about* x as

$$\begin{aligned}\ell_{x,h}(\boldsymbol{\beta}) := &-\frac{n}{2}\log(2\pi\sigma^2) \qquad (5.13)\\ &-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(Y_i-\beta_0-\beta_1(X_i-x)-\ldots-\beta_p(X_i-x)^p)^2K_h(x-X_i).\end{aligned}$$

Maximizing with respect to $\boldsymbol{\beta}$ the *local* log-likelihood (5.13) provides $\hat{\beta}_0 = \hat{m}(x;p,h)$, which is precisely the local polynomial estimator, as it was obtained in (4.10), but now obtained from a likelihood-based perspective. The key point is to realize that the very same idea can be applied to the family of **generalized linear models**, for which linear regression (in this case manifested as a polynomial regression) is just a particular case.

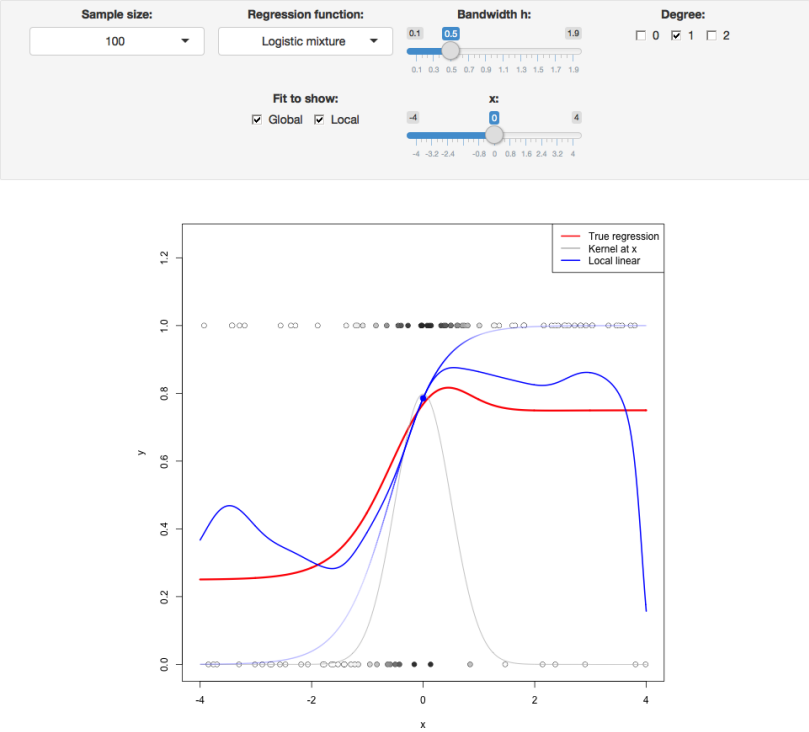


Figure 5.1: Construction of the local likelihood estimator. The animation shows how local likelihood fits in a neighborhood of x are combined to provide an estimate of the regression function for binary response, which depends on the polynomial degree, bandwidth, and kernel (gray density at the bottom). The data points are shaded according to their weights for the local fit at x. Application available here.

We illustrate the local likelihood principle for the logistic regression (see Section B.2 for a quick review). In this case, the sample is $(X_1, Y_1), \ldots, (X_n, Y_n)$ with

$$Y_i | X_i \sim \text{Ber}(\text{logistic}(\eta(X_i))), \quad i = 1, \ldots, n,$$

with the polynomial term[23]

$$\eta(x) := \beta_0 + \beta_1 x + \cdots + \beta_p x^p.$$

The log-likelihood of $\boldsymbol{\beta}$ is

$$\begin{aligned}\ell(\boldsymbol{\beta}) &= \sum_{i=1}^{n} \{Y_i \log(\text{logistic}(\eta(X_i))) + (1 - Y_i) \log(1 - \text{logistic}(\eta(X_i)))\} \\ &= \sum_{i=1}^{n} \ell(Y_i, \eta(X_i)),\end{aligned}$$

where we consider the *log-likelihood addend* $\ell(y, \eta) := y\eta - \log(1 + e^{\eta})$. For the sake of clarity in the next developments, we make explicit the dependence on $\eta(x)$ of this addend; conversely, we make its dependence on $\boldsymbol{\beta}$ implicit.

The local log-likelihood of $\boldsymbol{\beta}$ about x is then defined as

$$\ell_{x,h}(\boldsymbol{\beta}) := \sum_{i=1}^{n} \ell(Y_i, \eta(X_i - x)) K_h(x - X_i). \tag{5.14}$$

Maximizing[24] the local log-likelihood (5.14) with respect to $\boldsymbol{\beta}$ provides

$$\hat{\boldsymbol{\beta}}_{\mathbf{h}} = \arg \max_{\boldsymbol{\beta} \in \mathbb{R}^{p+1}} \ell_{x,h}(\boldsymbol{\beta}).$$

[23] If $p = 1$, then we have the usual simple logistic model. Notice that we are just doing the "polynomial trick" done in linear regression, which consisted in expanding the number of predictors with the powers X^j, $j = 1, \ldots, p$, of X to achieve more flexibility in the parametric fit.

[24] There is no analytical solution for the optimization problem due to its nonlinearity, so a numerical approach is required.

The local likelihood estimate of $\eta(x)$ is the intercept of this fit,

$$\hat{\eta}(x) := \hat{\beta}_{h,0}.$$

Note that the dependence of $\hat{\beta}_{0,h}$ on x is omitted. From $\hat{\eta}(x)$, we can obtain the **local logistic regression** evaluated at x as[25]

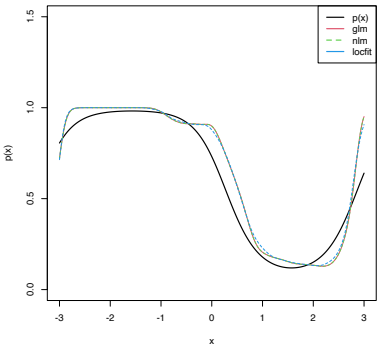
$$\hat{m}_\ell(x; p, h) := g^{-1}\left(\hat{\eta}(x)\right) = \text{logistic}(\hat{\beta}_{h,0}). \tag{5.15}$$

Each evaluation of $\hat{m}_\ell(x; p, h)$ in a different x requires, thus, a weighted fit of the underlying local logistic model.

The code below shows three different ways of implementing in R the local logistic regression (with $p = 1$).

[25] An alternative and useful view is that, by maximizing (5.14), we are fitting the logistic model $\hat{p}_x(t) := \text{logistic}(\hat{\beta}_{h,0} + \hat{\beta}_{h,1}(t - x) + \cdots + \hat{\beta}_{h,p}(t - x)^p)$ that is *centered* about x. Then, we employ this model to predict Y for $X = t = x$, resulting $\text{logistic}(\hat{\beta}_{h,0})$.

```
# Simulate some data
n <- 200
logistic <- function(x) 1 / (1 + exp(-x))
p <- function(x) logistic(1 - 3 * sin(x))
set.seed(123456)
X <- sort(runif(n = n, -3, 3))
Y <- rbinom(n = n, size = 1, prob = p(X))

# Set bandwidth and evaluation grid
h <- 0.25
x <- seq(-3, 3, l = 501)

# Approach 1: optimize the weighted log-likelihood through the workhorse
# function underneath glm, glm.fit
suppressWarnings(
  fit_glm <- sapply(x, function(x) {
    K <- dnorm(x = x, mean = X, sd = h)
    glm.fit(x = cbind(1, X - x), y = Y, weights = K,
            family = binomial())$coefficients[1]
  })
)

# Approach 2: optimize directly the weighted log-likelihood
suppressWarnings(
  fit_nlm <- sapply(x, function(x) {
    K <- dnorm(x = x, mean = X, sd = h)
    nlm(f = function(beta) {
      -sum(K * (Y * (beta[1] + beta[2] * (X - x)) -
                  log(1 + exp(beta[1] + beta[2] * (X - x)))))
      }, p = c(0, 0))$estimate[1]
  })
)

# Approach 3: employ locfit::locfit
# CAREFUL: locfit::locfit uses a different internal parametrization for h!
# As it can be see, the bandwidths in approaches 1-2 and approach 3 do NOT
# give the same results! With h = 0.75 in locfit::locfit the fit is close
# to the previous ones (done for h = 0.25), but not exactly the same...
fit_locfit <- locfit::locfit(Y ~ locfit::lp(X, deg = 1, h = 0.75),
                             family = "binomial", kern = "gauss")

# Compare fits
plot(x, p(x), ylim = c(0, 1.5), type = "l", lwd = 2)
lines(x, logistic(fit_glm), col = 2)
lines(x, logistic(fit_nlm), col = 3, lty = 2)
lines(x, predict(fit_locfit, newdata = x), col = 4, lty = 2)
legend("topright", legend = c("p(x)", "glm", "nlm", "locfit"), lwd = 2,
       col = c(1, 2, 3, 4), lty = c(1, 1, 2, 1))
```



Bandwidth selection can be done by means of *likelihood cross-validation*. The objective is to maximize the local log-likelihood fit at

(X_i, Y_i) but removing the influence by the datum itself. That is,

$$\hat{h}_{\text{LCV}} = \arg\max_{h>0} \text{LCV}(h),$$
$$\text{LCV}(h) = \sum_{i=1}^{n} \ell(Y_i, \hat{\eta}_{-i}(X_i)), \tag{5.16}$$

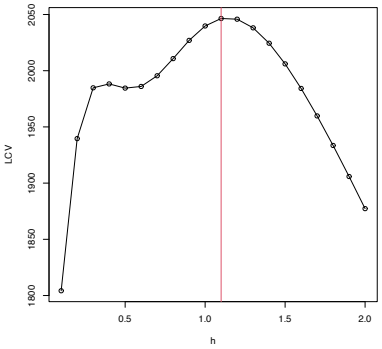
where $\hat{\eta}_{-i}(X_i)$ represents the local fit at X_i without the i-th datum (X_i, Y_i).[26] Unfortunately, the nonlinearity of (5.15) forbids a simplifying result as the one in Proposition 4.1. Thus, in principle, it is required to fit n local likelihoods for sample size $n-1$ to obtain a single evaluation of (5.16).[27]

We conclude by illustrating how to compute the LCV function and optimize it. Keep in mind that much more efficient implementations are possible!

[26] Observe that (5.16) is equivalent to (4.23) if the generalized linear model is a linear model.

[27] The interested reader is referred to Sections 4.3.3 and 4.4.3 in Loader (1999) for an *approximation* of (5.16) that only requires a local likelihood fit for a single sample.

```
# Exact LCV - recall that we *maximize* the LCV!
h <- seq(0.1, 2, by = 0.1)
suppressWarnings(
  LCV <- sapply(h, function(h) {
  sum(sapply(1:n, function(i) {
    K <- dnorm(x = X[i], mean = X[-i], sd = h)
    nlm(f = function(beta) {
      -sum(K * (Y[-i] * (beta[1] + beta[2] * (X[-i] - X[i])) -
                  log(1 + exp(beta[1] + beta[2] * (X[-i] - X[i])))))
      }, p = c(0, 0))$minimum
    }))
  })
)
plot(h, LCV, type = "o")
abline(v = h[which.max(LCV)], col = 2)
```



Exercise 5.10 (Adapted from Example 4.6 in Wasserman (2006))**.** The dataset at `http://www.stat.cmu.edu/~larry/all-of-nonpar/=data/bpd.dat` (alternative link) contains information about the presence of bronchopulmonary dysplasia (binary response) and the birth weight in grams (predictor) of 223 newborns.

a. Use the three approaches described above to compute the local logistic regression (first degree) and plot their outputs for a bandwidth that is somehow adequate.
b. Using $\hat{h}_{\text{LCV}}$, explore and comment on the resulting estimates, providing insights into the data.
c. From the obtained fit, derive several simple diagnostic rules for the probability of the presence of bronchopulmonary dysplasia from the birth weight.

Exercise 5.11. The `challenger.txt` dataset contains information regarding the state of the solid rocket boosters after launch for 23 shuttle flights prior the Challenger launch. Each row has, among others, the variables `fail.field` (indicator of whether there was an incident with the O-rings), `nfail.field` (number of incidents with the O-rings), and `temp` (temperature in the day of launch, measured in Celsius degrees). See Section 5.1 in García-Portugués (2022) for further context on the Challenger case study.

a. Fit a local logistic regression (first degree) for `fails.field ~ temp`, for three choices of bandwidths: one that oversmooths, another that is somehow adequate, and another that undersmooths. Do the effects of `temp` on `fails.field` seem to be significant?
b. Obtain $\hat{h}_{\text{LCV}}$ and plot the LCV function with a reasonable accuracy.
c. Using $\hat{h}_{\text{LCV}}$, predict the probability of an incident at temperatures -0.6 (launch temperature of the Challenger) and 11.67 (vice president of engineers' recommended launch threshold).
d. What are the local odds at -0.6 and 11.67? Show the local logistic models about these points, in spirit of Figure 5.1, and interpret the results.

Exercise 5.12. Implement your own version of the local likelihood estimator (first degree) for the Poisson regression model. To do so:

a. Derive the local log-likelihood about x for the Poisson regression (which is analogous to (5.14)). You can check Section 5.2.2 in García-Portugués (2022) for information on the Poisson regression.
b. Code from scratch an R function, `loc_pois`, that maximizes the previous local likelihood for a vector of evaluation points. `loc_pois` must take as input the samples `X` and `Y`, the vector of evaluation points `x`, the bandwidth `h`, and the kernel `K`.
c. Implement a `cv_loc_pois` function that obtains the cross-validated bandwidth for the local Poisson regression.
d. Validate the correct behavior of `loc_pois` and `cv_loc_pois` by sampling from $Y|X = x \sim \text{Poisson}(\lambda(x))$, where $\lambda(x) = e^{\sin(x)}$ and X is distributed according to a `nor1mix::MW.nm7`.
e. Compare your results with the `locfit::locfit` function using `family = "poisson"`.

6
Nonparametric tests

This chapter overviews *some* well-known nonparametric hypothesis tests.[1] The reviewed tests are intended for different purposes, mostly related to: (*i*) the evaluation of the *goodness-of-fit* of a distribution model to a dataset; and (*ii*) the assessment of the relation between two random variables.

A **nonparametric test** evaluates a null hypothesis H_0 against an alternative H_1 **without assuming any parametric model**, on neither H_0 nor H_1. Consequently, a nonparametric test is free from the overhead of evaluating a parametric assumption that one needs to conduct before applying a parametric test.[2] More importantly, it is quite likely that the inspection of these parametric assumptions has a negative outcome that forbids the subsequent application of a parametric test. The **direct applicability** and **generality** of nonparametric tests are the reasons for their usefulness in real-data applications.

Nonparametric tests have lower efficiency with respect to optimal parametric tests for specific parametric problems.[3] Statistical inference is full of instances of such parametric tests, especially within the context of normal populations.[4] For example, given two iid samples $X_{11}, \ldots, X_{1n_1}$ and $X_{21}, \ldots, X_{2n_2}$ from two normal populations $X_1 \sim \mathcal{N}(\mu_1, \sigma^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma^2)$, the test for the equality of the means,

$$H_0 : \mu_1 = \mu_2 \quad \text{vs.} \quad H_1 : \mu_1 \neq \mu_2,$$

is optimally carried out using the test statistic $T_n := \frac{\bar{X}_1 - \bar{X}_2}{S\sqrt{1/n_1 + 1/n_2}}$, where $S^2 := \frac{1}{n_1+n_2-2}\left(\sum_{i=1}^{n_1}(X_{1i} - \bar{X}_1)^2 + \sum_{i=1}^{n_2}(X_{2i} - \bar{X}_2)^2\right)$ is the pooled sample variance. The distribution of T_n under H_0 is $t_{n_1+n_2-2}$, which is compactly denoted by $T_n \overset{H_0}{\sim} t_{n_1+n_2-2}$. For this result to hold, it is key that the two populations are indeed normally distributed, an assumption that may be unrealistic in practice. Recall that, under H_0, this test states the equality of distributions of X_1 and X_2. A nonparametric alternative therefore is the Kolmogorov–Smirnov test for two samples, to be seen in Section 6.2. It evaluates if the distributions of $X_1 \sim F_1$ and $X_2 \sim F_2$ are equal:

$$H_0 : F_1 = F_2 \quad \text{vs.} \quad H_1 : F_1 \neq F_2.$$

[1] If necessary, see Section C for an informal review on the main concepts involved in hypothesis testing.

[2] This prior assessment is of key importance to ensure coherency between the *real* and the *assumed* data distributions, as the parametric test bases its decision on the latter. An example to dramatize this point follows. Let $X_1 \sim \mathcal{N}(\mu, \sigma^2)$ and $X_2 \sim \Gamma(\mu/\sigma^2, \mu^2/\sigma^2)$, for $\mu, \sigma^2 > 0$. The cdfs of X_1 and X_2, F_1 and F_2, are different for all $\mu, \sigma^2 > 0$. Yet $\mathbb{E}[X_1] = \mathbb{E}[X_2]$ and $\mathbb{V}\mathrm{ar}[X_1] = \mathbb{V}\mathrm{ar}[X_2]$. When testing $H_0 : F_1 = F_2$, if one assumes that X_1 and X_2 are normally distributed (which is partially true), then one can use a t-test with unknown variances. The t-test will believe H_0 is true, since $\mathbb{E}[X_1] = \mathbb{E}[X_2]$ and $\mathbb{V}\mathrm{ar}[X_1] = \mathbb{V}\mathrm{ar}[X_2]$, thus having a rejection rate equal to the significance level α. However, by construction, H_0 is false. The t-test fails to reject H_0 because its parametric assumption does not match the reality.

[3] These optimal parametric tests are often obtained by maximum likelihood theory.

[4] See, e.g., Section 6.2 in Molina-Peralta and García-Portugués (2022).

Finally, the term **goodness-of-fit** refers to the statistical tests that check the **adequacy of a model for explaining a sample**. For example, a goodness-of-fit test allows answering if a normal model is "acceptable" to describe a given sample $X_1, \ldots, X_n$. Initially, the concept of goodness-of-fit test was proposed for distribution models, but it was later extended to regression[5] and other statistical models,[6] although such extensions are not addressed in these notes.

6.1 Goodness-of-fit tests for distribution models

Assume that an iid sample $X_1, \ldots, X_n$ from an arbitrary distribution F is given.[7] We next address the ***one-sample problem*** of testing a statement about the *unknown* distribution F.

6.1.1 Simple hypothesis tests

We first address tests for the ***simple null hypothesis***

$$H_0 : F = F_0 \tag{6.1}$$

against the most general alternative[8]

$$H_1 : F \neq F_0,$$

where here and henceforth "$F \neq F_0$" means that there exists *at least one* $x \in \mathbb{R}$ such that $F(x) \neq F_0(x)$, and F_0 is a **pre-specified**, **not-data-dependent** distribution model. This latter aspect is very important:

> If some parameters of F_0 are estimated from the sample, the presented tests for (6.1) will not respect the significance level α for which they are constructed, and as a consequence they will be highly *conservative*.[9]

Recall that the ecdf (1.1) of $X_1, \ldots, X_n$, $F_n(x) = \frac{1}{n}\sum_{i=1}^n 1_{\{X_i \leq x\}}$, is a nonparametric estimator of F, as seen in Section 1.6. Therefore, a measure of proximity of F_n (driven by the sample) and F_0 (specified by H_0) will be indicative of the veracity of (6.1): a "large" distance between F_n and F_0 evidences that H_0 is likely to be false.[10,11]

The following three well-known goodness-of-fit tests arise from the same principle: considering as the test statistic a particular type of distance between the *functions* F_n and F_0.

Kolmogorov–Smirnov test

- *Test purpose*. Given $X_1, \ldots, X_n \sim F$, it tests $H_0 : F = F_0$ vs. $H_1 : F \neq F_0$ consistently against all the alternatives in H_1.
- *Statistic definition*. The test statistic uses the **supremum distance**[12,13] between F_n and F_0:

$$D_n := \sqrt{n} \sup_{x \in \mathbb{R}} |F_n(x) - F_0(x)| .$$

[5] See González-Manteiga and Crujeiras (2013) for an exhaustive review of the topic.

[6] For example, there are goodness-of-fit tests for time series models, such as ARMA(p, q) models (see, e.g., Velilla (1994) and references therein).

[7] F does not need to be continuous.

[8] The fact that $F = F_0$ is tested in full generality against *any* distribution that is different from F_0 raises the *omnibus test* (all-purpose test) terminology for the kind of tests that are able to detect all the alternatives to H_0 that are within H_1.

[9] That is, the tests will have a rejection probability lower than α when H_0 is true.

[10] If F and F_0 were continuous, the density approach to this problem would compare the kde $\hat{f}(\cdot; h)$ seen in Chapter 2 with the density f_0. While using a kde for testing $H_0 : F = F_0$ via $H_0 : f = f_0$ is a perfectly valid approach and has certain advantages (e.g., better detection of local violations of H_0), it is also more demanding (it requires selecting the bandwidth h) and limited (applicable to continuous random variables only). The cdf optic, although not appropriate for visualization or for the applications described in Section 3.5, suffices for conducting hypothesis testing in a nonparametric way.

[11] A popular approach to goodness-of-fit tests, not covered in these notes, is the chi-squared test (see `?chisq.test`). This test is based on aggregating the values of X in k classes I_i, $i = 1, \ldots, k$, where $\cup_{i=1}^k I_i$ cover the support of X. Worryingly, one can modify the outcome of the test by altering the form of these classes and the choice of k – a sort of tuning parameter. The choice of k also affects the quality of the asymptotic null distribution. Therefore, for continuous and discrete random variables, the chi-squared test has significant drawbacks when compared to the ecdf-based tests. Nevertheless, the "chi-squared philosophy" of testing is remarkably general: as opposed to ecdf-based tests, it can be readily applied to the analysis of categorical variables and contingency tables.

[12] Or the ∞-distance $\|F_n - F_0\|_\infty = \sup_{x \in \mathbb{R}} |F_n(x) - F_0(x)|$.

[13] With the addition of the $\sqrt{n}$ factor in order to standardize D_n in terms of its asymptotic distribution under H_0. Many authors do not consider this factor as a part of the test statistic itself.

If $H_0 : F = F_0$ holds, then D_n tends to be small. Conversely, when $F \neq F_0$, larger values of D_n are expected, and the test rejects H_0 when D_n is large.

- *Statistic computation*. The computation of D_n can be efficiently achieved by realizing that the maximum difference between F_n and F_0 happens at $x = X_i$, for a certain X_i (observe Figure 6.1). From here, sorting the sample and applying the probability transformation F_0 gives[14]

$$\begin{aligned} D_n &= \max(D_n^+, D_n^-), \\ D_n^+ &:= \sqrt{n} \max_{1 \leq i \leq n} \left\{ \frac{i}{n} - U_{(i)} \right\}, \\ D_n^- &:= \sqrt{n} \max_{1 \leq i \leq n} \left\{ U_{(i)} - \frac{i-1}{n} \right\}, \end{aligned} \tag{6.2}$$

  where $U_{(j)}$ stands for the j-th *sorted* $U_i := F_0(X_i)$, $i = 1, \ldots, n$.

- *Distribution under H_0*. If H_0 holds and F_0 is **continuous**, then D_n has an asymptotic[15] cdf given by the *Kolmogorov–Smirnov's K*[16] *function*:

$$\lim_{n \to \infty} \mathbb{P}[D_n \leq x] = K(x) := 1 - 2 \sum_{j=1}^{\infty} (-1)^{j-1} e^{-2j^2 x^2}. \tag{6.3}$$

- *Highlights and caveats*. The Kolmogorov–Smirnov test is a ***distribution-free*** test because its distribution under H_0 *does not depend* on F_0. However, this is the case only **if F_0 is continuous** and the sample $X_1, \ldots, X_n$ is also continuous, i.e., if the **sample has no ties**.[17] If these assumptions are met, then the iid sample $X_1, \ldots, X_n \overset{H_0}{\sim} F_0$ generates the iid sample $U_1, \ldots, U_n \overset{H_0}{\sim} \mathcal{U}(0,1)$. As a consequence, the distribution of (6.2) **does not depend on** F_0.[18,19] If F_0 is not continuous or there are ties on the sample, the K function is *not* the true asymptotic cdf. An alternative for discrete F_0 is given below. In case there are ties on the sample, a possibility is to slightly perturb the sample in order to remove such ties.[20]

- *Implementation in R*. For continuous data and continuous F_0, the test statistic D_n and the asymptotic p-value[21] are readily available through the `ks.test` function. The asymptotic cdf K is internally coded as the `pkolmogorov1x` function within the source code of `ks.test`. For discrete F_0, see `dgof::ks.test`.

The construction of the Kolmogorov–Smirnov test statistic is illustrated in the following chunk of code.

```
# Sample data
n <- 10
mu0 <- 2
sd0 <- 1
set.seed(54321)
samp <- rnorm(n = n, mean = mu0, sd = sd0)

# Fn vs. F0
```

[14] Notice that $F_n(X_{(i)}) = \frac{i}{n}$ and $F_n(X_{(i)}^-) = \frac{i-1}{n}$.

[15] When the sample size n is large: $n \to \infty$.

[16] The infinite series converges quite fast and 50 terms are usually enough to evaluate (6.3) with high accuracy for $x \in \mathbb{R}$.

[17] A tie occurs when two elements of the sample are numerically equal, an event with probability zero if the sample comes from a truly continuous random variable.

[18] More precisely, the pdf of D_n would follow from the joint pdf of the sorted sample $(U_{(1)}, \ldots, U_{(n)})$ that is generated from a $\mathcal{U}(0,1)$ and for which the transformation (6.2) is employed. The exact analytical formula, for a given n, is very cumbersome, hence the need for an asymptotic approximation.

[19] That is, if F_0 is the cdf of a $\mathcal{N}(0,1)$ or the cdf of a $\text{Exp}(\lambda)$, the distribution of D_n under H_0 is exactly the same.

[20] Ties may appear as a consequence of a measuring process of a continuous quantity having low precision.

[21] Which is $\lim_{n \to \infty} \mathbb{P}[d_n > D_n] = 1 - K(d_n)$, where d_n is the observed statistic and D_n is the random variable (6.2).

```
plot(ecdf(samp), main = "", ylab = "Probability", xlim = c(-1, 6),
     ylim = c(0, 1.3))
curve(pnorm(x, mean = mu0, sd = sd0), add = TRUE, col = 4)

# Add Dn+ and Dn-
samp_sorted <- sort(samp)
Ui <- pnorm(samp_sorted, mean = mu0, sd = sd0)
Dn_plus <- (1:n) / n - Ui
Dn_minus <- Ui - (1:n - 1) / n
i_plus <- which.max(Dn_plus)
i_minus <- which.max(Dn_minus)
lines(rep(samp_sorted[i_plus], 2),
      c(i_plus / n, pnorm(samp_sorted[i_plus], mean = mu0, sd = sd0)),
      col = 3, lwd = 2, pch = 16, type = "o", cex = 0.75)
lines(rep(samp_sorted[i_minus], 2),
      c((i_minus - 1) / n, pnorm(samp_sorted[i_minus], mean = mu0, sd = sd0)),
      col = 2, lwd = 2, pch = 16, type = "o", cex = 0.75)
rug(samp)
legend("topleft", lwd = 2, col = c(1, 4, 3, 2),
       legend = latex2exp::TeX(c("$F_n$", "$F_0$", "$D_n^+$", "$D_n^-$")))
```

Exercise 6.1. Modify if the parameters `mu0` and `sd0` in the previous code in order to have $F \neq F_0$. What happens with $\sup_{x \in \mathbb{R}} |F_n(x) - F_0(x)|$?

Let's see an example of the use of `ks.test` (also available as `stats::ks.test`), the function in base R.

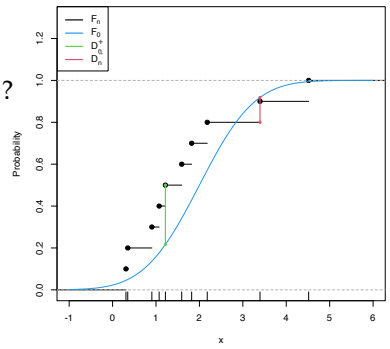


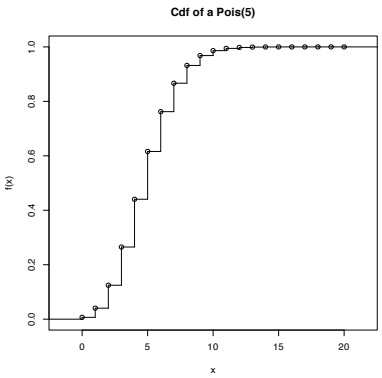
Figure 6.1: Computation of the Kolmogorov–Smirnov statistic $D_n = \max(D_n^+, D_n^-)$ for a sample of size $n = 10$ coming from $F_0(\cdot) = \Phi\left(\frac{\cdot - \mu_0}{\sigma_0}\right)$, where $\mu_0 = 2$ and $\sigma_0 = 1$. In the example shown, $D_n = D_n^+$.

```
# Sample data from a N(0, 1)
n <- 50
set.seed(3245678)
x <- rnorm(n = n)

# Kolmogorov-Smirnov test for H_0: F = N(0, 1). Does not reject
(ks <- ks.test(x = x, y = "pnorm")) # In "y" we specify F0 as a function
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 0.050298, p-value = 0.9989
## alternative hypothesis: two-sided

# Structure of "htest" class
str(ks)
## List of 5
##  $ statistic  : Named num 0.0503
##   ..- attr(*, "names")= chr "D"
##  $ p.value    : num 0.999
##  $ alternative: chr "two-sided"
##  $ method     : chr "One-sample Kolmogorov-Smirnov test"
##  $ data.name  : chr "x"
##  - attr(*, "class")= chr "htest"

# Kolmogorov-Smirnov test for H_0: F = N(0.5, 1). Rejects
ks.test(x = x, y = "pnorm", mean = 0.5)
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 0.24708, p-value = 0.003565
## alternative hypothesis: two-sided

# Kolmogorov-Smirnov test for H_0: F = Exp(2). Strongly rejects
ks.test(x = x, y = "pexp", rate = 1/2)
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  x
```

```
## D = 0.53495, p-value = 6.85e-14
## alternative hypothesis: two-sided
```

The following chunk of code shows how to call `dgof::ks.test` to perform a correct Kolmogorov–Smirnov when F_0 is discrete.

```
# Sample data from a Pois(5)
n <- 100
set.seed(3245678)
x <- rpois(n = n, lambda = 5)

# Kolmogorov-Smirnov test for H_0: F = Pois(5) without specifying that the
# distribution is discrete. Rejects (!?) giving a warning message
ks.test(x = x, y = "ppois", lambda = 5)
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 0.20596, p-value = 0.0004135
## alternative hypothesis: two-sided

# We rely on dgof::ks.test, which works as stats::ks.test if the "y" argument
# is not marked as a "stepfun" object, the way the dgof package codifies
# discrete distribution functions

# Step function containing the cdf of the Pois(5). The "x" stands for the
# location of the steps and "y" for the value of the steps. "y" needs to have
# an extra point for the initial value of the function before the first step
x_eval <- 0:20
ppois_stepfun <- stepfun(x = x_eval, y = c(0, ppois(q = x_eval, lambda = 5)))
plot(ppois_stepfun, main = "Cdf of a Pois(5)")
```




```
# Kolmogorov-Smirnov test for H_0: F = Pois(5) adapted for discrete data,
# with data coming from a Pois(5)
dgof::ks.test(x = x, y = ppois_stepfun)
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 0.032183, p-value = 0.9999
## alternative hypothesis: two-sided

# If data is normally distributed, the test rejects H_0
dgof::ks.test(x = rnorm(n = n, mean = 5), y = ppois_stepfun)
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  rnorm(n = n, mean = 5)
## D = 0.38049, p-value = 5.321e-13
## alternative hypothesis: two-sided
```

Exercise 6.2. Implement the Kolmogorov–Smirnov test by:

1. Coding a function to compute the test statistic (6.2) from a sample $X_1, \ldots, X_n$ and a cdf F_0.
2. Implementing the K function (6.3).
3. Calling the previous functions from a routine that returns the asymptotic p-value of the test.

Check that the implementations coincide with the ones of the `ks.test` function when `exact = FALSE` for data simulated from a $\mathcal{U}(0,1)$ and any n. *Note*: `ks.test` computes $D_n/\sqrt{n}$ instead of D_n.

[22] Recall that the p-value of any statistical test is a random variable, since it depends on the sample.

The following code chunk exemplifies with a small simulation study how the distribution of the p-values[22] is uniform if H_0 holds, and how it becomes more concentrated about 0 when H_1 holds.

```
# Simulation of p-values when H_0 is true
set.seed(131231)
n <- 100
M <- 1e4
pvalues_H0 <- sapply(1:M, function(i) {
  x <- rnorm(n) # N(0, 1)
  ks.test(x, "pnorm")$p.value
})

# Simulation of p-values when H_0 is false -- the data does not
# come from a N(0, 1) but from a N(0, 2)
pvalues_H1 <- sapply(1:M, function(i) {
  x <- rnorm(n, mean = 0, sd = sqrt(2)) # N(0, 2)
  ks.test(x, "pnorm")$p.value
})

# Comparison of p-values
par(mfrow = 1:2)
hist(pvalues_H0, breaks = seq(0, 1, l = 20), probability = TRUE,
     main = latex2exp::TeX("$H_0$"), ylim = c(0, 4))
abline(h = 1, col = 2)
hist(pvalues_H1, breaks = seq(0, 1, l = 20), probability = TRUE,
     main = latex2exp::TeX("$H_1$"), ylim = c(0, 4))
abline(h = 1, col = 2)
```

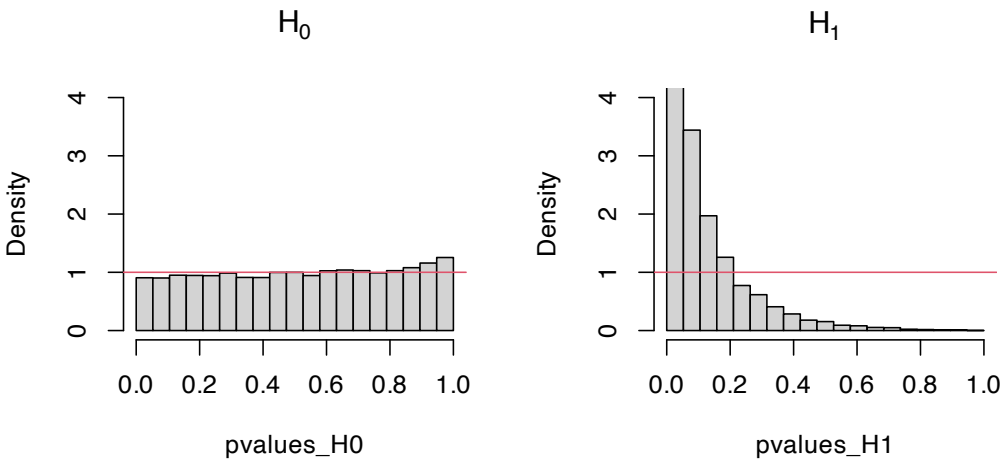


Figure 6.2: Comparison of the distribution of p-values under H_0 and H_1 for the Kolmogorov–Smirnov test. Observe that the frequency of low p-values, associated with the rejection of H_0, increases when H_0 does not hold. Under H_0, the distribution of the p-values is uniform.

Exercise 6.3. Modify the parameters of the normal distribution used to sample under H_1 in order to increase and decrease the deviation from H_0. What do you observe in the resulting distributions of the p-values?

Cramér–von Mises test

- *Test purpose*. Given $X_1, \ldots, X_n \sim F$, it tests $H_0 : F = F_0$ vs. $H_1 : F \neq F_0$ consistently against all the alternatives in H_1.
- *Statistic definition*. The test statistic uses a **quadratic distance**[23] between F_n and F_0:

$$W_n^2 := n \int (F_n(x) - F_0(x))^2 \, \mathrm{d}F_0(x). \tag{6.4}$$

[23] Recall that $\mathrm{d}F_0(x) = f_0(x)\,\mathrm{d}x$ if F_0 is continuous.

If $H_0 : F = F_0$ holds, then W_n^2 tends to be small. Hence, rejection happens for large values of W_n^2.

- *Statistic computation*. The computation of W_n^2 can be significantly simplified by expanding the square in the integrand of (6.4) and then applying the change of variables $u = F_0(x)$:

$$W_n^2 = \sum_{i=1}^{n} \left\{ U_{(i)} - \frac{2i-1}{2n} \right\}^2 + \frac{1}{12n}, \tag{6.5}$$

  where again $U_{(j)}$ stands for the j-th sorted $U_i = F_0(X_i)$, $i = 1, \ldots, n$.

- *Distribution under H_0*. If H_0 holds and F_0 is **continuous**, then W_n^2 has an asymptotic cdf given by[24]

$$\lim_{n\to\infty} \mathbb{P}[W_n^2 \leq x] = 1 - \frac{1}{\pi} \sum_{j=1}^{\infty} (-1)^{j-1} W_j(x), \tag{6.6}$$

$$W_j(x) := \int_{(2j-1)^2\pi^2}^{4j^2\pi^2} \sqrt{\frac{-\sqrt{y}}{\sin\sqrt{y}}} \frac{e^{-\frac{xy}{2}}}{y} \, \mathrm{d}y.$$

[24] See page 10 in del Barrio (2007).

- *Highlights and caveats*. By a reasoning analogous to the one done in the Kolmogorov–Smirnov test, the Cramér–von Mises test can be seen to be **distribution-free if F_0 is continuous** and the **sample has no ties**. Otherwise, (6.6) is *not* the true asymptotic distribution. Although the Kolmogorov–Smirnov test is the most popular ecdf-based test, empirical evidence suggests that the Cramér–von Mises test is often **more powerful than the Kolmogorov–Smirnov test** for a broad class of alternative hypotheses.[25]

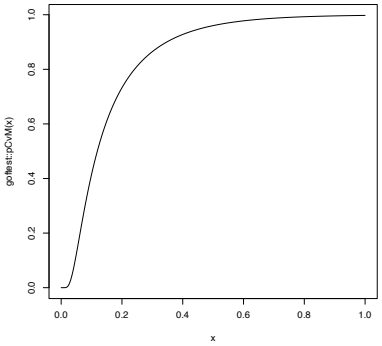
[25] See, e.g., Section 5 in Stephens (1974) or page 110 in D'Agostino and Stephens (1986). Observe this is only empirical evidence for certain scenarios.

- *Implementation in R*. For continuous data, the test statistic W_n^2 and the asymptotic p-value are implemented in the `goftest::cvm.test` function. The asymptotic cdf (6.6) is given in `goftest::pCvM` (`goftest::qCvM` computes its inverse). For discrete F_0, see `dgof::cvm.test`.

The following chunk of code points to the implementation of the Cramér–von Mises test.

```
# Sample data from a N(0, 1)
set.seed(3245678)
n <- 50
x <- rnorm(n = n)

# Cramér-von Mises test for H_0: F = N(0, 1). Does not reject
goftest::cvm.test(x = x, null = "pnorm")
##
##  Cramer-von Mises test of goodness-of-fit
##  Null hypothesis: Normal distribution
##  Parameters assumed to be fixed
##
## data:  x
## omega2 = 0.022294, p-value = 0.9948

# Comparison with Kolmogorov-Smirnov
ks.test(x = x, y = "pnorm")
##
```

```
##  One-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 0.050298, p-value = 0.9989
## alternative hypothesis: two-sided

# Sample data from a Pois(5)
set.seed(3245678)
n <- 100
x <- rpois(n = n, lambda = 5)

# Cramér-von Mises test for H_0: F = Pois(5) without specifying that the
# distribution is discrete. Rejects (!?) without giving a warning message
goftest::cvm.test(x = x, null = "ppois", lambda = 5)
##
##  Cramer-von Mises test of goodness-of-fit
##  Null hypothesis: Poisson distribution
##  with parameter lambda = 5
##  Parameters assumed to be fixed
##
## data:  x
## omega2 = 0.74735, p-value = 0.009631

# We rely on dgof::cvm.test and run a Cramér-von Mises test for H_0: F = Pois(5)
# adapted for discrete data, with data coming from a Pois(5)
x_eval <- 0:20
ppois_stepfun <- stepfun(x = x_eval, y = c(0, ppois(q = x_eval, lambda = 5)))
dgof::cvm.test(x = x, y = ppois_stepfun)
##
##  Cramer-von Mises - W2
##
## data:  x
## W2 = 0.038256, p-value = 0.9082
## alternative hypothesis: Two.sided

# Plot the asymptotic null distribution function
curve(goftest::pCvM(x), from = 0, to = 1, n = 300)
```



Exercise 6.4. Implement the Cramér–von Mises test by:

1. Coding the Cramér–von Mises statistic (6.5) from a sample $X_1, \ldots, X_n$ and a cdf F_0. Check that the implementation coincides with the one of the `goftest::cvm.test` function.
2. Computing the *asymptotic* p-value using `goftest::pCvM`. Compare this asymptotic p-value with the *exact* p-value given by `goftest::cvm.test`, observing that for a large n the difference is inappreciable.

Exercise 6.5. Verify the correctness of the asymptotic null distribution of W_n^2 that was given in (6.6). To do so, numerically implement $W_j(x)$ and compute (6.6). Validate your implementation by:

1. Simulating $M = 1,000$ samples of size $n = 200$ under H_0, obtaining M statistics $W_{n;1}^2, \ldots, W_{n;M}^2$, and plotting their ecdf.
2. Overlaying your asymptotic cdf and the one provided by `goftest::pCvM`.
3. Checking that the three curves approximately coincide.

Anderson–Darling test

- *Test purpose.* Given $X_1, \ldots, X_n \sim F$, it tests $H_0 : F = F_0$ vs. $H_1 : F \neq F_0$ consistently against all the alternatives in H_1.

- *Statistic definition*. The test statistic uses a **quadratic distance** between F_n and F_0 **weighted** by $w(x) = (F_0(x)(1 - F_0(x)))^{-1}$:

$$A_n^2 := n \int \frac{(F_n(x) - F_0(x))^2}{F_0(x)(1 - F_0(x))} \mathrm{d}F_0(x).$$

  If H_0 holds, then A_n^2 tends to be small (because of the denominator). Hence, rejection happens for large values of A_n^2. Note that, compared with W_n^2, A_n^2 places more weight on the deviations between $F_n(x)$ and $F_0(x)$ that happen on the tails, that is, when $F_0(x) \approx 0$ or $F_0(x) \approx 1$.[26,27]

- *Statistic computation*. The computation of A_n^2 can be significantly simplified:

$$A_n^2 = -n - \frac{1}{n} \sum_{i=1}^{n} \Big\{ (2i - 1) \log(U_{(i)}) + (2n + 1 - 2i) \log(1 - U_{(i)}) \Big\}. \tag{6.7}$$

- *Distribution under H_0*. If H_0 holds and F_0 is **continuous**, then the asymptotic cdf of A_n^2 is the cdf of the random variable

$$\sum_{j=1}^{\infty} \frac{Y_j}{j(j+1)}, \quad \text{where } Y_j \sim \chi_1^2,\ j \geq 1, \text{ are iid.} \tag{6.8}$$

  Unfortunately, the cdf of (6.8) does not admit a simple analytical expression. It can, however, be approximated by Monte Carlo by sampling from the random variable (6.8).

- *Highlights and caveats*. As with the previous tests, the Anderson–Darling test is also **distribution-free if F_0 is continuous** and there are **no ties in the sample**. Otherwise, the null asymptotic distribution is different from the one of (6.8). As for the Cramér–von Mises test, there is also empirical evidence indicating that the Anderson–Darling test is **more powerful than the Kolmogorov–Smirnov test** for a broad class of alternative hypotheses. In addition, due to its construction, the Anderson–Darling test is able to detect better the situations in which F_0 and F differ on the tails (that is, for extreme data).

- *Implementation in R*. For continuous data, the test statistic A_n^2 and the asymptotic p-value are implemented in the `goftest::ad.test` function. The asymptotic cdf of (6.8) is given in `goftest::pAD` (`goftest::qAD` computes its inverse). For discrete F_0, see `dgof::cvm.test` with `type = "A2"`.

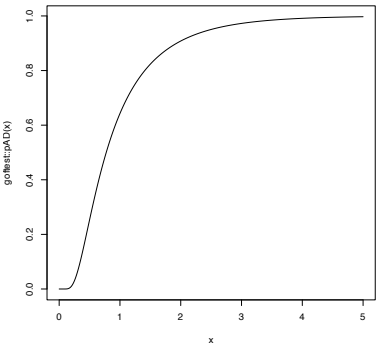
The following code chunk illustrates the implementation of the Anderson–Darling test.

```
# Sample data from a N(0, 1)
set.seed(3245678)
n <- 50
x <- rnorm(n = n)
```

[26] The motivation for considering $w(x) = (F_0(x)(1 - F_0(x)))^{-1}$ stems from recalling that $\mathbb{E}[(F_n(x) - F_0(x))^2] \overset{H_0}{=} F_0(x)(1 - F_0(x))/n$ (see Example 1.7). That is, estimating F_0 on the tails is less variable than at its center. A_n^2 weights $(F_n(x) - F_0(x))^2$ with its variability by incorporating w.

[27] Actually, due to the allocation of greater weight to the tails, the statistic A_n^2 is somehow on the edge of existence. If F_0 is continuous, the weight function $w(x) = (F_0(x)(1 - F_0(x)))^{-1}$ is *not integrable* on $\mathbb{R}$. Indeed, $\int w(x) \mathrm{d}F_0(x) = \int_0^1 (x(1-x))^{-1} \mathrm{d}x = \not\exists$. It is only thanks to the factor $(F_n(x) - F_0(x))^2$ in the integrand that A_n^2 is well defined.

```
# Anderson-Darling test for H_0: F = N(0, 1). Does not reject
goftest::ad.test(x = x, null = "pnorm")
##
##  Anderson-Darling test of goodness-of-fit
##  Null hypothesis: Normal distribution
##  Parameters assumed to be fixed
##
## data:  x
## An = 0.18502, p-value = 0.994

# Sample data from a Pois(5)
set.seed(3245678)
n <- 100
x <- rpois(n = n, lambda = 5)

# Anderson-Darling test for H_0: F = Pois(5) without specifying that the
# distribution is discrete. Rejects (!?) without giving a warning message
goftest::ad.test(x = x, null = "ppois", lambda = 5)
##
##  Anderson-Darling test of goodness-of-fit
##  Null hypothesis: Poisson distribution
##  with parameter lambda = 5
##  Parameters assumed to be fixed
##
## data:  x
## An = 3.7279, p-value = 0.01191

# We rely on dgof::cvm.test and run an Anderson-Darling test for H_0: F = Pois(5)
# adapted for discrete data, with data coming from a Pois(5)
x_eval <- 0:20
ppois_stepfun <- stepfun(x = x_eval, y = c(0, ppois(q = x_eval, lambda = 5)))
dgof::cvm.test(x = x, y = ppois_stepfun, type = "A2")
##
##  Cramer-von Mises - A2
##
## data:  x
## A2 = 0.3128, p-value = 0.9057
## alternative hypothesis: Two.sided

# Plot the asymptotic null distribution function
curve(goftest::pAD(x), from = 0, to = 5, n = 300)
```



Exercise 6.6. Implement the Anderson–Darling test by:

1. Coding the Anderson–Darling statistic (6.7) from a sample $X_1, \ldots, X_n$ and a cdf F_0. Check that the implementation coincides with the one of the `goftest::ad.test` function.
2. Computing the *asymptotic* p-value using `goftest::pAD`. Compare this asymptotic p-value with the *exact* p-value given by `goftest::ad.test`, observing that for large n the difference is inappreciable.

Exercise 6.7. Verify the correctness of the asymptotic representation of A_n^2 that was given in (6.8). To do so:

1. Simulate $M = 1,000$ samples of size $n = 200$ under H_0, obtain M statistics $A_{n;1}^2, \ldots, A_{n;M}^2$, and draw its histogram.
2. Simulate M samples from the random variable (6.8) and draw its histogram.
3. Check that the two histograms approximately coincide.

Exercise 6.8. Let's investigate empirically the performance of the Kolmogorov–Smirnov, Cramér–von Mises, and Anderson–Darling tests for a specific scenario. We consider $H_0 : X \sim \mathcal{N}(0,1)$ and the following simulation study:

- Generate $M = 1,000$ samples of size n from $\mathcal{N}(\mu, 1)$.
- For each sample, test H_0 with the three tests.
- Obtain the relative frequency of rejections at level α for each test.
- Draw three histograms for the p-values in the spirit of Figure 6.3.

In order to cleanly perform the previous steps for several choices of (n, μ, α), code a function that performs the simulation study from those arguments and gives something similar to Figure 6.3 as output. Then, use the following settings and accurately comment on the outcome of each of them:

1. H_0 holds.
   - Take $n = 25$, $\mu = 0$, and $\alpha = 0.05$. Check that the relative frequency of rejections is about α.
   - Take $n = 25$, $\mu = 0$, and $\alpha = 0.10$.
   - Take $n = 100$, $\mu = 0$, and $\alpha = 0.10$.
2. H_0 does not hold.
   - $n = 25$, $\mu = 0.25$, $\alpha = 0.05$. Check that the relative frequency of rejections is above α.
   - $n = 50$, $\mu = 0.25$, $\alpha = 0.05$.
   - $n = 25$, $\mu = 0.50$, $\alpha = 0.05$.
   - Replace $\mathcal{N}(\mu, 1)$ with t_{10}. Take $n = 50$ and $\alpha = 0.05$. Which test is clearly better? Why?

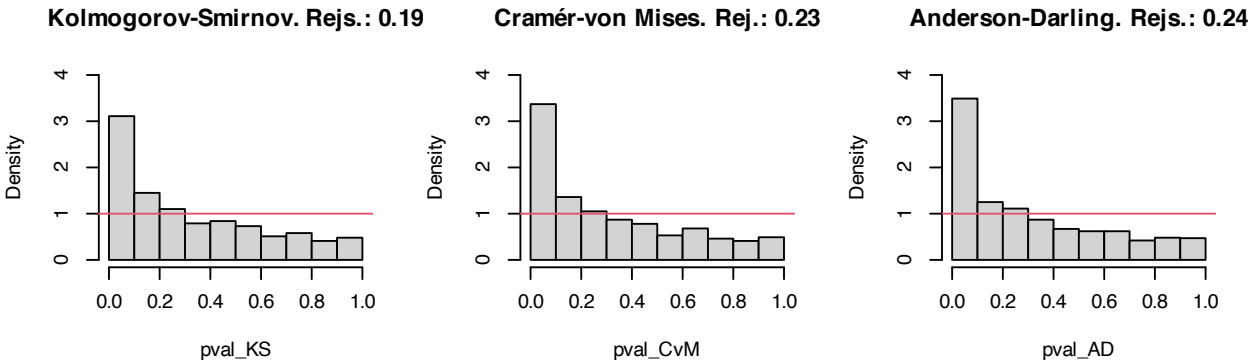


Figure 6.3: The histograms of the p-values and the relative rejection frequencies for the Kolmogorov–Smirnov, Cramér–von Mises, and Anderson–Darling tests. The null hypothesis is $H_0 : X \sim \mathcal{N}(0,1)$ and the sample of size $n = 25$ is generated from a $\mathcal{N}(0.25, 1)$. The significance level is $\alpha = 0.05$.

6.1.2 *Normality tests*

The tests seen in the previous section have a very notable practical limitation: they require complete knowledge of F_0, the hypothesized distribution for X. In practice, such a precise knowledge about X is unrealistic. Practitioners are more interested in answering more general questions, one of them being:

Is the data "normally distributed"?

With the statement "X is normally distributed" we refer to the fact that $X \sim \mathcal{N}(\mu, \sigma^2)$ for *certain* parameters $\mu \in \mathbb{R}$ and $\sigma \in \mathbb{R}^+$, possibly unknown. This statement is notably more general than "X is distributed as a $\mathcal{N}(0,1)$" or "X is distributed as a $\mathcal{N}(0.1, 1.3)$".

The test for normality is formalized as follows. Given an iid sample $X_1, \ldots, X_n$ from the distribution F, we test the null hypothesis

$$H_0 : F \in \mathcal{F}_{\mathcal{N}} := \left\{ \Phi\left(\frac{\cdot - \mu}{\sigma}\right) : \mu \in \mathbb{R}, \sigma \in \mathbb{R}^+ \right\}, \tag{6.9}$$

where $\mathcal{F}_{\mathcal{N}}$ is the *class of normal distributions*, against the most general alternative

$$H_1 : F \notin \mathcal{F}_{\mathcal{N}}.$$

Comparing (6.9) with (6.1), it is evident that the former is more general, as a *range* of possible values for F is included in H_0. This is the reason why H_0 is called a ***composite null hypothesis***, rather than a simple null hypothesis, as (6.1) was.

The testing of (6.9) requires the estimation of the unknown parameters (μ, σ). Their maximum likelihood estimators are $(\bar{X}, S)$, where $S := \sqrt{\frac{1}{n}\sum_{i=1}^n (X_i - \bar{X})^2}$ is the sample standard deviation Therefore, a tempting possibility is to apply the tests seen in Section 6.1.1 to

$$H_0 : F = \Phi\left(\frac{\cdot - \bar{X}}{S}\right).$$

However, as warned at the beginning of Section 6.1, this naive approach would result in **seriously conservative tests** that would fail to reject H_0 adequately. The intuitive explanation is simple: when estimating (μ, σ) from the sample we are obtaining the *closest* normal cdf to F, that is, $\Phi\left(\frac{\cdot - \bar{X}}{S}\right)$. Therefore, setting $F_0 = \Phi\left(\frac{\cdot - \bar{X}}{S}\right)$ and then running one of the ecdf-based tests on $H_0 : F = F_0$ will ignore this **estimation step**, which will **bias the test decision towards** H_0.

Adequately accounting for the estimation of (μ, σ) amounts to study the asymptotic distributions under H_0 of the test statistics that are computed with the *data-dependent* cdf $\Phi\left(\frac{\cdot - \bar{X}}{S}\right)$ in the place of F_0. This is what is precisely done by the tests that adapt the ecdf-based tests seen in Section 6.1.1 by

1. replacing F_0 with $\Phi\left(\frac{\cdot - \bar{X}}{S}\right)$ in the formulation of the test statistics;
2. replacing U_i with $P_i := \Phi\left(\frac{X_i - \bar{X}}{\hat{S}}\right)$, $i = 1, \ldots, n$, in the computational forms of the statistics;[28]
3. obtaining a *different* null asymptotic distribution that is usually more convoluted and is approximated by tabulation or simulation.

[28] Importantly, recall that $P_1, \ldots, P_n$ are *not* independent, since $\bar{X}$ and $\hat{S}$ depend on the whole sample $X_1, \ldots, X_n$. To dramatize this point, imagine that X_1 is very large. Then, it would drive $\frac{X_i - \bar{X}}{\hat{S}}$ towards large negative values, revealing the dependence of $P_2, \ldots, P_n$ on P_1. This is a very remarkable qualitative difference with respect to $U_1, \ldots, U_n$, which was an iid sample. $P_1, \ldots, P_n$ is an id sample, but not an iid sample.

Lilliefors test (Kolmogorov–Smirnov test for normality)

- *Test purpose.* Given $X_1, \ldots, X_n$, it tests $H_0 : F \in \mathcal{F}_{\mathcal{N}}$ vs. $H_1 : F \notin \mathcal{F}_{\mathcal{N}}$ consistently against all the alternatives in H_1.
- *Statistic definition and computation.* The test statistic uses the supremum distance between F_n and $\Phi\left(\frac{\cdot - \bar{X}}{S}\right)$:

$$\begin{aligned} D_n &= \sqrt{n} \sup_{x \in \mathbb{R}} \left| F_n(x) - \Phi\left(\frac{x - \bar{X}}{S}\right) \right| = \max(D_n^+, D_n^-), \\ D_n^+ &= \sqrt{n} \max_{1 \leq i \leq n} \left\{ \frac{i}{n} - P_{(i)} \right\}, \\ D_n^- &= \sqrt{n} \max_{1 \leq i \leq n} \left\{ P_{(i)} - \frac{i-1}{n} \right\}, \end{aligned}$$

  where $P_{(j)}$ stands for the j-th sorted $P_i = \Phi\left(\frac{X_i - \bar{X}}{S}\right)$, $i = 1, \ldots, n$. Clearly, if H_0 holds, then D_n tends to be small. Hence, rejection happens when D_n is large.
- *Distribution under H_0.* If H_0 holds, the critical values at level of significance α can be obtained from Lilliefors (1967).[29] Dallal and Wilkinson (1986) provides an analytical approximation to the null distribution.
- *Highlights and caveats.* The Lilliefors test is *not* distribution-free, in the sense that the null distribution of the test statistic clearly depends on the *normality* assumption. However, it is *parameter-free*, since the distribution does not depend on the actual parameters (μ, σ) for which $F = \Phi\left(\frac{\cdot - \mu}{\sigma}\right)$ is satisfied. This result comes from the iid sample $X_1, \ldots, X_n \overset{H_0}{\sim} \mathcal{N}(\mu, \sigma^2)$ generating the id sample $\frac{X_1 - \bar{X}}{S}, \ldots, \frac{X_n - \bar{X}}{S} \overset{H_0}{\sim} \tilde{f}_0$, where $\tilde{f}_0$ is a certain pdf that does not depend on (μ, σ).[30] Therefore, $P_1, \ldots, P_n$ is an id sample that does not depend on (μ, σ).
- *Implementation in R.* The `nortest::lillie.test` function implements the test.

[29] H_0 is rejected with significance level α if the statistic is above the critical value for α.

[30] The precise form of $\tilde{f}_0$ (if S stands for the *quasi*-variance) can be seen in equation (3.1) in Shao (1999).

Cramér–von Mises normality test

- *Test purpose.* Given $X_1, \ldots, X_n$, it tests $H_0 : F \in \mathcal{F}_{\mathcal{N}}$ vs. $H_1 : F \notin \mathcal{F}_{\mathcal{N}}$ consistently against all the alternatives in H_1.
- *Statistic definition and computation.* The test statistic uses the quadratic distance between F_n and $\Phi\left(\frac{\cdot - \bar{X}}{S}\right)$:

$$\begin{aligned} W_n^2 &= n \int \left(F_n(x) - \Phi\left(\frac{x - \bar{X}}{S}\right) \right)^2 \mathrm{d}\Phi\left(\frac{x - \bar{X}}{S}\right) \\ &= \sum_{i=1}^{n} \left\{ P_{(i)} - \frac{2i-1}{2n} \right\}^2 + \frac{1}{12n}, \end{aligned}$$

  where $P_{(j)}$ stands for the j-th sorted $P_i = \Phi\left(\frac{X_i - \bar{X}}{S}\right)$, $i = 1, \ldots, n$. Rejection happens when W_n^2 is large.

- *Distribution under H_0.* If H_0 holds, the p-values of the test can be approximated using Table 4.9 in D'Agostino and Stephens (1986).
- *Highlights and caveats.* Like the Lilliefors test, the Cramér–von Mises test is also *parameter-free*. Its usual power superiority over the Kolmogorov–Smirnov also extends to the testing of normality.
- *Implementation in R.* The `nortest::cvm.test` function implements the test.

Anderson–Darling normality test

- *Test purpose.* Given $X_1, \ldots, X_n$, it tests $H_0 : F \in \mathcal{F}_{\mathcal{N}}$ vs. $H_1 : F \notin \mathcal{F}_{\mathcal{N}}$ consistently against all the alternatives in H_1.
- *Statistic definition and computation.* The test statistic uses a weighted quadratic distance between F_n and $\Phi\left(\frac{\cdot-\bar{X}}{S}\right)$:

$$\begin{aligned} A_n^2 &= n \int \frac{\left(F_n(x) - \Phi\left(\frac{x-\bar{X}}{S}\right)\right)^2}{\Phi\left(\frac{x-\bar{X}}{S}\right)\left(1-\Phi\left(\frac{x-\bar{X}}{S}\right)\right)} \mathrm{d}\Phi\left(\frac{x-\bar{X}}{S}\right) \\ &= -n - \frac{1}{n}\sum_{i=1}^{n}\left\{(2i-1)\log(P_{(i)}) + (2n+1-2i)\log(1-P_{(i)})\right\}. \end{aligned}$$

  Rejection happens when A_n^2 is large.
- *Distribution under H_0.* If H_0 holds, the p-values of the test can be approximated using Table 4.9 in D'Agostino and Stephens (1986).
- *Highlights and caveats.* The test is also parameter-free. Since the test statistic places more weight on the tails than the Cramér–von Mises, it is better suited for detecting heavy-tailed deviations from normality.
- *Implementation in R.* The `nortest::ad.test` function implements the test.

The following code chunk gives the implementation of these normality tests.

```
# Sample data from a N(10, 1)
set.seed(123456)
n <- 100
x <- rnorm(n = n, mean = 10)

# Normality tests -- do not reject H0
nortest::lillie.test(x = x)
##
##  Lilliefors (Kolmogorov-Smirnov) normality test
##
## data:  x
## D = 0.054535, p-value = 0.6575
nortest::cvm.test(x = x)
##
##  Cramer-von Mises normality test
##
```

```
## data:  x
## W = 0.032898, p-value = 0.8027
nortest::ad.test(x = x)
##
##  Anderson-Darling normality test
##
## data:  x
## A = 0.22888, p-value = 0.8052

# Sample data from a Student's t with 3 degrees of freedom
x <- rt(n = n, df = 3)

# Normality tests -- reject H0
nortest::lillie.test(x = x)
##
##  Lilliefors (Kolmogorov-Smirnov) normality test
##
## data:  x
## D = 0.08453, p-value = 0.07499
nortest::cvm.test(x = x)
##
##  Cramer-von Mises normality test
##
## data:  x
## W = 0.19551, p-value = 0.005965
nortest::ad.test(x = x)
##
##  Anderson-Darling normality test
##
## data:  x
## A = 1.2842, p-value = 0.002326

# Flawed normality tests -- do not reject because the null distribution
# that is employed is wrong!
ks.test(x = x, y = "pnorm", mean = mean(x), sd = sd(x))
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 0.08453, p-value = 0.4725
## alternative hypothesis: two-sided
goftest::cvm.test(x = x, null = "pnorm", mean = mean(x), sd = sd(x))
##
##  Cramer-von Mises test of goodness-of-fit
##  Null hypothesis: Normal distribution
##  with parameters mean = 0.0151268546198777, sd = 1.33709561396292
##  Parameters assumed to be fixed
##
## data:  x
## omega2 = 0.19551, p-value = 0.2766
goftest::ad.test(x = x, null = "pnorm", mean = mean(x), sd = sd(x))
##
##  Anderson-Darling test of goodness-of-fit
##  Null hypothesis: Normal distribution
##  with parameters mean = 0.0151268546198777, sd = 1.33709561396292
##  Parameters assumed to be fixed
##
## data:  x
## An = 1.2842, p-value = 0.2375
```

Shapiro–Francia normality test

We now consider a different normality test that is not based on the ecdf, the Shapiro–Francia normality test. This test is a modification of the wildly popular Shapiro–Wilk test, but the Shapiro–Francia test is easier to interpret and explain.[31]

[31] The Shapiro–Wilk test is implemented in the `shapiro.test` R function. The test has the same highlights and caveats as `nortest::sf.test`.

The test is based on the **QQ-plot** of the sample $X_1, \ldots, X_n$. The plot is comprised of the pairs $\left(\Phi^{-1}\left(\frac{i}{n+1}\right), X_{(i)}\right)$, for $i = 1, \ldots, n$.[32] That is, the QQ-plot plots $\mathcal{N}(0,1)$**-quantiles against sample quantiles**.

Since the α-*lower* quantile of a $\mathcal{N}(\mu, \sigma^2)$, $z_{\alpha;\mu,\sigma}$, verifies that

$$z_{\alpha;\mu,\sigma} = \mu + \sigma z_{\alpha;0,1}, \tag{6.10}$$

then, if the sample is distributed as a $\mathcal{N}(\mu, \sigma^2)$, the points of the QQ-plot are expected to align about the straight line (6.10). This is illustrated in the code below.

```
n <- 100
mu <- 10
sigma <- 2
set.seed(12345678)
x <- rnorm(n, mean = mu, sd = sigma)
qqnorm(x)
abline(a = mu, b = sigma, col = 2)
```

The QQ-plot can therefore be used as a graphical check for normality: if the data is distributed about *some* straight line, then it likely is normally distributed. However, this graphical check is subjective and, to complicate things further, it is usual for departures from the diagonal to be larger in the extremes than in the center, even under normality, although these departures are more evident if the data is non-normal.[33] Figure 6.5, generated with the code below, depicts the uncertainty behind the QQ-plot.

```
M <- 1e3
n <- 100
plot(0, 0, xlim = c(-3.5, 3.5), ylim = c(-3.5, 3.5), type = "n",
     xlab = "Theoretical Quantiles", ylab = "Sample Quantiles",
     main = "Confidence bands for QQ-plot")
x <- matrix(rnorm(M * n), nrow = n, ncol = M)
matpoints(qnorm(ppoints(n)), apply(x, 2, sort), pch = 19, cex = 0.5,
          col = gray(0, alpha = 0.01))
abline(a = 0, b = 1)
p <- seq(0, 1, l = 1e4)
xi <- qnorm(p)
lines(xi, xi - qnorm(0.975) / sqrt(n) * sqrt(p * (1 - p)) / dnorm(xi),
      col = 2, lwd = 2)
lines(xi, xi + qnorm(0.975) / sqrt(n) * sqrt(p * (1 - p)) / dnorm(xi),
      col = 2, lwd = 2)
```

The Shapiro–Francia test evaluates how tightly distributed the QQ plot is about the linear trend that must arise if the data is normally distributed.

- *Test purpose*. Given $X_1, \ldots, X_n$, it tests $H_0 : F \in \mathcal{F}_{\mathcal{N}}$ vs. $H_1 : F \notin \mathcal{F}_{\mathcal{N}}$ consistently against all the alternatives in H_1.
- *Statistic definition and computation*. The test statistic, W_n', is simply the **squared correlation coefficient** for the sample[34]

$$\left(\Phi^{-1}\left(\frac{i-3/8}{n-1/4}\right), X_{(i)}\right), \quad i = 1, \ldots, n.$$

- *Distribution under* H_0. Royston (1993) derived a simple analytical formula that serves to approximate the null distribution of W_n' for $5 \leq n \leq 5{,}000$.

[32] Minor modifications of the $\frac{i}{n+1}$-*lower* quantile $\Phi^{-1}\left(\frac{i}{n+1}\right)$ may be considered.

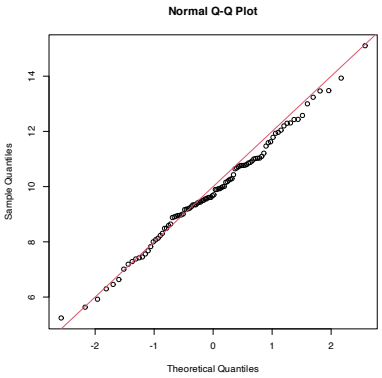


Figure 6.4: QQ-plot (`qqnorm`) for data simulated from a $\mathcal{N}(\mu, \sigma^2)$ and theoretical line $y = \mu + \sigma x$ (red).

[33] For $X \sim F$, the p-th quantile $x_p = F^{-1}(p)$ of X is estimated through the sample quantile $\hat{x}_p := F_n^{-1}(p)$. If $X \sim f$ is continuous, then $\sqrt{n}(\hat{x}_p - x_p)$ is asymptotically $\mathcal{N}\left(0, \frac{p(1-p)}{f(x_p)^2}\right)$. Therefore, the variance of $\hat{x}_p$ grows if $p \to 0, 1$ and more variability is expected in the extremes of the QQ-plot.

[34] Observe that `qnorm(ppoints(x, a = 3 / 8))` is used as an approximation to the expected ordered quantiles from a $\mathcal{N}(0,1)$, whereas `qnorm(ppoints(x, a = 1 / 2))` is employed internally in `qqplot`.

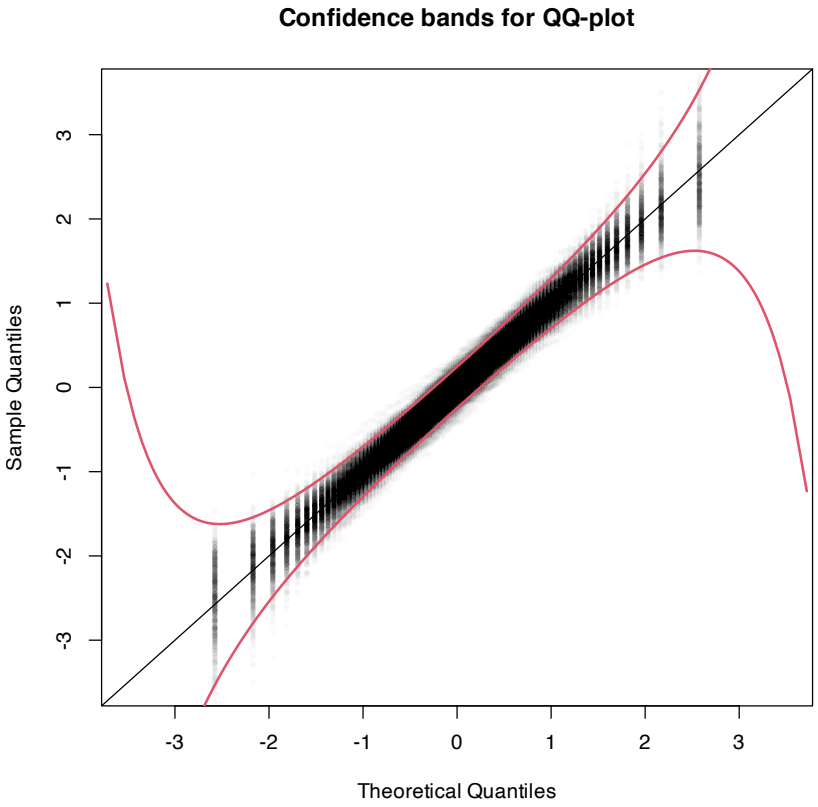


Figure 6.5: The uncertainty behind the QQ-plot. The figure aggregates $M = 1,000$ different QQ-plots of $\mathcal{N}(0,1)$ data with $n = 100$, displaying for each of them the pairs $(x_p, \hat{x}_p)$ evaluated at $p = \frac{i-1/2}{n}$, $i = 1, \ldots, n$ (as they result from `ppoints(n)`). The uncertainty is measured by the asymptotic $100(1-\alpha)\%$ CIs for $\hat{x}_p$, given by $\left(x_p \pm \frac{z_{\alpha/2}}{\sqrt{n}} \frac{\sqrt{p(1-p)}}{\phi(x_p)}\right)$. These curves are displayed in red for $\alpha = 0.05$. Observe that the vertical strips arise because the x_p coordinate is deterministic.

- *Highlights and caveats.* The test is also *parameter-free*, since its distribution does not depend on the actual parameters (μ, σ) for which the equality H_0 holds.
- *Implementation in R.* The `nortest::sf.test` function implements the test. The condition $5 \leq n \leq 5,000$ is enforced by the function.[35]

The use of the function is described below.

[35] However, the R code of the function allows computing the statistic and carrying out a bootstrap-based test as described in Section 6.1.3 for $n \geq 5,000$. The `shapiro.test` function, on the contrary, does not allow this simple extraction, as the core of the function is written in C (function `C_SWilk`) and internally enforces the condition $5 \leq n \leq 5,000$.

```
# Does not reject H0
set.seed(123456)
n <- 100
x <- rnorm(n = n, mean = 10)
nortest::sf.test(x)
##
##  Shapiro-Francia normality test
##
## data:  x
## W = 0.99336, p-value = 0.8401

# Rejects H0
x <- rt(n = n, df = 3)
nortest::sf.test(x)
##
##  Shapiro-Francia normality test
##
## data:  x
## W = 0.91545, p-value = 2.754e-05

# Test statistic
cor(x = sort(x), y = qnorm(ppoints(n, a = 3/8)))^2
## [1] 0.9154466
```

6.1.3 *Bootstrap-based approach to goodness-of-fit testing*

Each normality test discussed in the previous section is an instance of a **goodness-of-fit test for a parametric distribution model**. Precisely, the normality tests are goodness-of-fit tests for the normal distribution model. Therefore, they address a specific instance of the test of the null *composite* hypothesis

$$H_0 : F \in \mathcal{F}_\Theta := \{F_\theta : \theta \in \Theta \subset \mathbb{R}^q\},$$

where $\mathcal{F}_\Theta$ denotes a certain *parametric* family of distributions indexed by the (possibly multidimensional) parameter $\theta \in \Theta$, versus the most general alternative

$$H_1 : F \notin \mathcal{F}_\Theta.$$

The normality case demonstrated that the goodness-of-fit tests for the *simple* hypothesis were not readily applicable, and that the null distributions were affected by the estimation of the unknown parameters $\theta = (\mu, \sigma)$. In addition, these null distributions tend to be cumbersome, requiring **analytical approximations** that have to be done on a **test-by-test basis** in order to obtain computable p-values.

To make things worse, the **derivations** that were done to obtain the asymptotic distributions of the normality test are **not reusable if $\mathcal{F}_\Theta$ is different**. For example, if we wanted to test *exponentiality*, that is,

$$H_0 : F \in \mathcal{F}_{\mathbb{R}^+} = \left\{F_\theta(x) = (1 - e^{-\theta x})1_{\{x>0\}} : \theta \in \mathbb{R}^+\right\}, \tag{6.11}$$

then new asymptotic distributions with their corresponding new analytical approximations would need to be derived. Clearly, this is not a very practical approach if we are to evaluate the goodness-of-fit of several parametric models. A practical computational-based solution is to rely on **bootstrap**, specifically, on *parametric bootstrap*.

Since the main problem is to establish the distribution of the test statistic under H_0, then a possibility is to approximate this distribution by the distribution of the *bootstrapped statistic*. Precisely, let T_n be the statistic computed from the sample

$$X_1, \ldots, X_n \overset{H_0}{\sim} F_\theta$$

and let T_n^* be its bootstrapped version, that is, the statistic computed from the simulated sample[36]

$$X_1^*, \ldots, X_n^* \sim F_{\hat{\theta}}. \tag{6.12}$$

Then, **if H_0 holds, the distribution of T_n is approximated**[37] **by the one of T_n^***. In addition, since the sampling in (6.12) is completely under our control, we can **approximate arbitrarily well the distribution of T_n^* by Monte Carlo**. For example, the computation of the upper tail probability $\mathbb{P}^*[T_n^* > x]$, required to obtain p-values

[36] Notice that X_i^* is *always* distributed as $F_{\hat{\theta}}$, no matter whether H_0 holds or not.

[37] With increasing precision as $n \to \infty$.

in all the tests we have seen, can be done by

$$\mathbb{P}^*[T_n^* > x] \approx \frac{1}{B}\sum_{b=1}^{B} 1_{\{T_n^{*b}>x\}},$$

where $T_n^{*1}, \ldots, T_n^{*B}$ is a sample from T_n^* obtained by simulating B bootstrap samples from (6.12).

The whole approach can be summarized in the following **bootstrap-based procedure** for performing a goodness-of-fit test for a parametric distribution model:

1. Estimate θ from the sample $X_1, \ldots, X_n$, obtaining $\hat{\theta}$ (for example, use maximum likelihood).
2. Compute the statistic T_n from $X_1, \ldots, X_n$ and $\hat{\theta}$.
3. Enter the "bootstrap world". For $b = 1, \ldots, B$:
   i. Simulate a boostrap sample $X_1^{*b}, \ldots, X_n^{*b}$ from $F_{\hat{\theta}}$.
   ii. Compute $\hat{\theta}^{*b}$ from $X_1^{*b}, \ldots, X_n^{*b}$ exactly in the same form that $\hat{\theta}$ was computed from $X_1, \ldots, X_n$.
   iii. Compute T_n^{*b} from $X_1^{*b}, \ldots, X_n^{*b}$ and $\hat{\theta}^{*b}$.
4. Obtain the p-value approximation

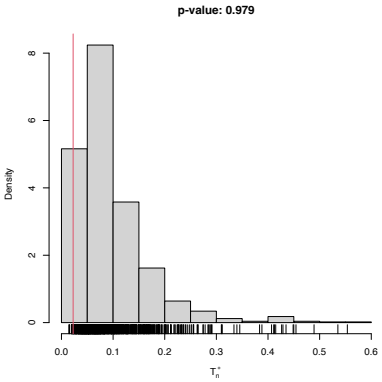
$$p\text{-value} \approx \frac{1}{B}\sum_{b=1}^{B} 1_{\{T_n^{*b}>T_n\}}$$

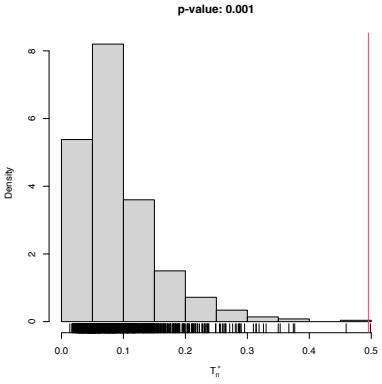
   and emit a test decision from it. Modify it accordingly if rejection of H_0 does not happen for large values of T_n.

The following chunk of code provides a template function for implementing the previous algorithm. The template is initialized with the specifics for testing (6.11), for which $\hat{\theta}_{\text{ML}} = 1/\bar{X}$. The function uses the `boot::boot` function for carrying out the parametric bootstrap.

```
# A goodness-of-fit test of the exponential distribution using the
# Cramér-von Mises statistic
cvm_exp_gof <- function(x, B = 5e3, plot_boot = TRUE) {

  # Test statistic function (depends on the data only)
  Tn <- function(data) {

    # Maximum likelihood estimator -- MODIFY DEPENDING ON THE PROBLEM
    theta_hat <- 1 / mean(data)

    # Test statistic -- MODIFY DEPENDING ON THE PROBLEM
    goftest::cvm.test(x = data, null = "pexp", rate = theta_hat)$statistic

  }

  # Function to simulate bootstrap samples X_1^*, ..., X_n^*. Requires TWO
  # arguments, one being the data X_1, ..., X_n and the other containing
  # the parameter theta
  r_mod <- function(data, theta) {

    # Simulate from an exponential. In this case, the function only uses
    # the sample size from the data argument to estimate theta -- MODIFY
```

```
    # DEPENDING ON THE PROBLEM
    rexp(n = length(data), rate = 1 / theta)

  }

  # Estimate of theta -- MODIFY DEPENDING ON THE PROBLEM
  theta_hat <- 1 / mean(x)

  # Perform bootstrap resampling with the aid of boot::boot
  Tn_star <- boot::boot(data = x, statistic = Tn, sim = "parametric",
                        ran.gen = r_mod, mle = theta_hat, R = B)

  # Test information -- MODIFY DEPENDING ON THE PROBLEM
  method <- "Bootstrap-based Cramér-von Mises test for exponentiality"
  alternative <- "any alternative to exponentiality"

  # p-value: modify if rejection does not happen for large values of the
  # test statistic
  pvalue <- mean(Tn_star$t > Tn_star$t0)

  # Construct an "htest" result
  result <- list(statistic = c("stat" = Tn_star$t0), p.value = pvalue,
                 theta_hat = theta_hat, statistic_boot = drop(Tn_star$t),
                 B = B, alternative = alternative, method = method,
                 data.name = deparse(substitute(x)))
  class(result) <- "htest"

  # Plot the position of the original statistic with respect to the
  # bootstrap replicates?
  if (plot_boot) {

    hist(result$statistic_boot, probability = TRUE,
         main = paste("p-value:", result$p.value),
         xlab = latex2exp::TeX("$T_n^*$"))
    rug(result$statistic_boot)
    abline(v = result$statistic, col = 2)

  }

  # Return "htest"
  return(result)

}

# Check the test for H0 true
set.seed(123456)
x <- rgamma(n = 100, shape = 1, scale = 1)
gof0 <- cvm_exp_gof(x = x, B = 1e3)

gof0
##
##  Bootstrap-based Cramér-von Mises test for exponentiality
##
## data:  x
## stat.omega2 = 0.022601, p-value = 0.979
## alternative hypothesis: any alternative to exponentiality

# Check the test for H0 false
x <- rgamma(n = 100, shape = 2, scale = 1)
gof1 <- cvm_exp_gof(x = x, B = 1e3)

gof1
##
##  Bootstrap-based Cramér-von Mises test for exponentiality
##
## data:  x
## stat.omega2 = 0.49536, p-value = 0.001
## alternative hypothesis: any alternative to exponentiality
```

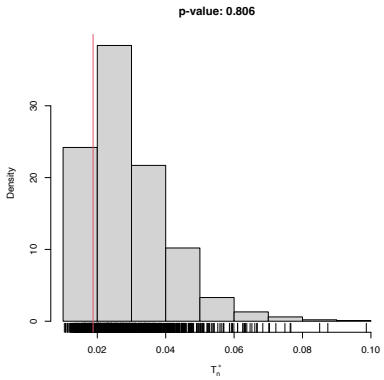




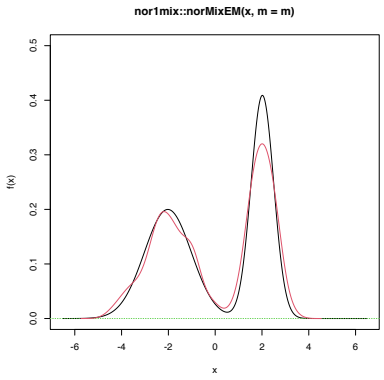
Another example is given below. It adapts the previous template for the flexible class of mixtures of normal distributions.

```
# A goodness-of-fit test of a mixture of m normals using the
# Cramér-von Mises statistic
cvm_nm_gof <- function(x, m, B = 1e3, plot_boot = TRUE) {

  # Test statistic function (depends on the data only)
  Tn <- function(data) {

    # EM algorithm for fitting normal mixtures. With trace = 0 we disable the
    # default convergence messages or otherwise they will saturate the screen
    # with the bootstrap loop. Be aware that this is a potentially dangerous
    # practice, as we may lose important information about the convergence of
    # the EM algorithm
    theta_hat <- nor1mix::norMixEM(x = data, m = m, trace = 0)

    # Test statistic
    goftest::cvm.test(x = data, null = nor1mix::pnorMix,
                      obj = theta_hat)$statistic

  }

  # Function to simulate bootstrap samples X_1^*, ..., X_n^*. Requires TWO
  # arguments, one being the data X_1, ..., X_n and the other containing
  # the parameter theta
  r_mod <- function(data, theta) {

    nor1mix::rnorMix(n = length(data), obj = theta)

  }

  # Estimate of theta
  theta_hat <- nor1mix::norMixEM(x = x, m = m, trace = 0)

  # Perform bootstrap resampling with the aid of boot::boot
  Tn_star <- boot::boot(data = x, statistic = Tn, sim = "parametric",
                        ran.gen = r_mod, mle = theta_hat, R = B)

  # Test information
  method <- "Bootstrap-based Cramér-von Mises test for normal mixtures"
  alternative <- paste("any alternative to a", m, "normal mixture")

  # p-value: modify if rejection does not happen for large values of the
  # test statistic
  pvalue <- mean(Tn_star$t > Tn_star$t0)

  # Construct an "htest" result
  result <- list(statistic = c("stat" = Tn_star$t0), p.value = pvalue,
                 theta_hat = theta_hat, statistic_boot = drop(Tn_star$t),
                 B = B, alternative = alternative, method = method,
                 data.name = deparse(substitute(x)))
  class(result) <- "htest"

  # Plot the position of the original statistic with respect to the
  # bootstrap replicates?
  if (plot_boot) {

    hist(result$statistic_boot, probability = TRUE,
         main = paste("p-value:", result$p.value),
         xlab = latex2exp::TeX("$T_n^*$"))
    rug(result$statistic_boot)
    abline(v = result$statistic, col = 2)

  }

  # Return "htest"
  return(result)
```

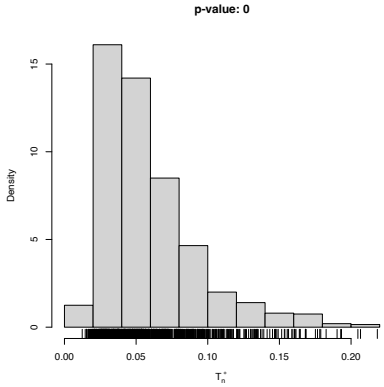
```
}

# Check the test for H0 true
set.seed(123456)
x <- c(rnorm(n = 100, mean = 2, sd = 0.5), rnorm(n = 100, mean = -2))
gof0 <- cvm_nm_gof(x = x, m = 2, B = 1e3)
```



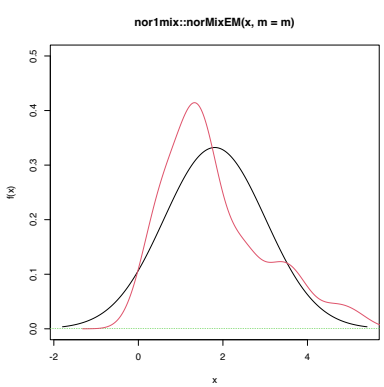

```
gof0
##
##  Bootstrap-based Cramér-von Mises test for normal mixtures
##
## data:  x
## stat.omega2 = 0.018769, p-value = 0.806
## alternative hypothesis: any alternative to a 2 normal mixture

# Graphical assessment that H0 (parametric fit) and data (kde) match
plot(gof0$theta_hat, p.norm = FALSE, ylim = c(0, 0.5))
plot(ks::kde(x), col = 2, add = TRUE)
```




```
# Check the test for H0 false
x <- rgamma(n = 100, shape = 2, scale = 1)
gof1 <- cvm_nm_gof(x = x, m = 1, B = 1e3)
```




```
gof1
##
##  Bootstrap-based Cramér-von Mises test for normal mixtures
##
## data:  x
## stat.omega2 = 0.44954, p-value < 2.2e-16
## alternative hypothesis: any alternative to a 1 normal mixture

# Graphical assessment that H0 (parametric fit) and data (kde) do not match
plot(gof1$theta_hat, p.norm = FALSE, ylim = c(0, 0.5))
plot(ks::kde(x), col = 2, add = TRUE)
```



Exercise 6.9. Adapt the previous template to perform the following tasks:

a. Test the goodness-of-fit of the Weibull distribution using the Anderson–Darling statistic. Check with a small simulation study that the test is correctly implemented (such that the significance level α is respected if H_0 holds).
b. Test the goodness-of-fit of the lognormal distribution using the Anderson–Darling statistic. Check with a small simulation study that the test is correctly implemented.
c. Apply the previous two tests to inspect if `av_gray_one` from Exercise 2.25 and if temps-other.txt are distributed according to a Weibull or lognormal. Explain the results.

6.2 *Comparison of two distributions*

Assume that two iid samples $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$ arbitrary distributions F and G are given. We next address the ***two-sample problem*** of comparing the *unknown* distributions F and G.

6.2.1 *Homogeneity tests*

The comparison of F and G can be done by testing their equality, which is known as the testing for homogeneity of the samples $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$. We can therefore address the *two-sided* hypothesis test

$$H_0 : F = G \quad \text{vs.} \quad H_1 : F \neq G. \tag{6.13}$$

Recall that, differently from the simple and composite hypothesis that appears in goodness-of-fit problems, in the homogeneity problem the null hypothesis $H_0 : F = G$ does not specify any form for the distributions F and G, only their equality.

The *one-sided* hypothesis in which $H_1 : F < G$ (or $H_1 : F > G$) is also very relevant. Here and henceforth "$F < G$" has a special meaning:

"$F < G$": there exists *at least one* $x \in \mathbb{R}$ such that $F(x) < G(x)$.[38]

Obviously, the alternative $H_1 : F < G$ is implied if, *for all* $x \in \mathbb{R}$,

$$\begin{aligned} F(x) < G(x) &\iff \mathbb{P}[X \leq x] < \mathbb{P}[Y \leq x] \\ &\iff \mathbb{P}[X > x] > \mathbb{P}[Y > x]. \end{aligned} \tag{6.14}$$

Consequently, (6.14) means that X **produces observations above *any* fixed threshold** $x \in \mathbb{R}$ **more likely than** Y. This concept is known as *(strict)*[39] *stochastic dominance* and, precisely, it is said that X **is stochastically greater than** Y if (6.14) holds.[40]

Stochastic dominance is an intuitive concept to interpret $H_1 : F < G$, although it is a stronger statement on the relation of F and G. A more accurate, yet still intuitive, way of regarding $H_1 : F < G$ is as a ***local* stochastic dominance**: X is stochastically greater than Y only for some specific thresholds $x \in \mathbb{R}$.[41] Figures 6.6 and 6.8 give two examples of presence/absence of stochastic dominance where $H_1 : F < G$ holds.

The ecdf-based goodness-of-fit tests seen in Section 6.1.1 can be adapted to the homogeneity problem, with varying difficulty and versatility. The two-sample Kolmogorov–Smirnov test offers the highest simplicity and versatility, yet its power is inferior to that of the two-sample Cramér–von Mises and Anderson–Darling tests.

Two-sample Kolmogorov–Smirnov test (two-sided)

- *Test purpose.* Given $X_1, \ldots, X_n \sim F$ and $Y_1, \ldots, Y_m \sim G$, it tests $H_0 : F = G$ vs. $H_1 : F \neq G$ consistently against all the alternatives in H_1.
- *Statistic definition.* The test statistic uses the supremum distance between F_n and G_m:[42]

$$D_{n,m} := \sqrt{\frac{nm}{n+m}} \sup_{x \in \mathbb{R}} |F_n(x) - G_m(x)|.$$

If $H_0 : F = G$ holds, then $D_{n,m}$ tends to be small. Conversely, when $F \neq G$, larger values of $D_{n,m}$ are expected, and the test rejects H_0 when $D_{n,m}$ is large.

[38] $F < G$ does not mean that $F(x) < G(x)$ *for all* $x \in \mathbb{R}$. If $F(x) > G(x)$ for *some* $x \in \mathbb{R}$ and $F(x) < G(x)$ or $F(x) = G(x)$ for other $x \in \mathbb{R}$, then $H_1 : F < G$ is true!

[39] Strict because $F(x) < G(x)$ and not $F(x) \leq G(x)$. Since we will use the strict variant of stochastic ordering, we will omit the adjective "strict" henceforth for the sake of simplicity.

[40] Note that if $F(x) < G(x)$, $\forall x \in \mathbb{R}$, then $X \sim F$ is stochastically *greater* than $Y \sim G$. The direction of stochastic dominance is *opposite* to the direction of dominance of the cdfs.

[41] With this terminology, clearly *global* stochastic dominance implies local stochastic dominance, but not the other way around.

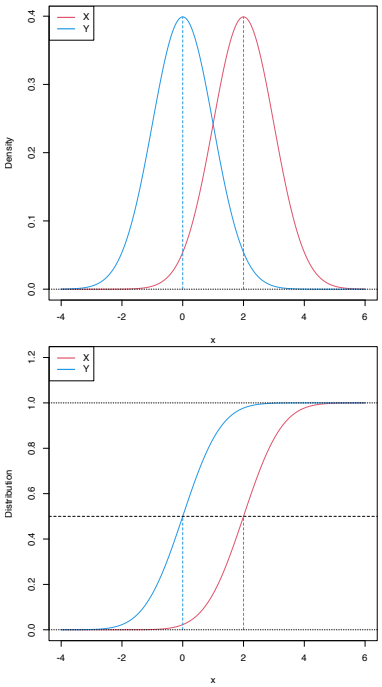


Figure 6.6: Pdfs and cdfs of $X \sim \mathcal{N}(2,1)$ and $Y \sim \mathcal{N}(0,1)$. X is stochastically greater than Y, which is visualized in terms of the pdfs (shift in the mean) and cdfs (domination of Y's cdf). The means are shown in solid vertical lines. Note that the variances of X and Y are common; compare this situation with Figure 6.8.

[42] G_m is the ecdf of $Y_1, \ldots, Y_m$.

- *Statistic computation*. The computation of $D_{n,m}$ can be efficiently achieved by realizing that the maximum difference between F_n and G_m happens at $x = X_i$ or $x = Y_j$, for a certain X_i or Y_j (observe Figure 6.7). Therefore,

$$\begin{aligned} D_{n,m} &= \max(D_{n,m,1}, D_{n,m,2}), \\ D_{n,m,1} &:= \sqrt{\frac{nm}{n+m}} \max_{1\leq i\leq n} \left| \frac{i}{n} - G_m(X_{(i)}) \right|, \\ D_{n,m,2} &:= \sqrt{\frac{nm}{n+m}} \max_{1\leq j\leq m} \left| F_n(Y_{(j)}) - \frac{j}{m} \right|. \end{aligned} \tag{6.15}$$

- *Distribution under H_0*. If H_0 holds and $F = G$ is continuous, then $D_{n,m}$ has the **same asymptotic cdf as** D_n (check (6.3)).[43] That is,

$$\lim_{n,m\to\infty} \mathbb{P}[D_{n,m} \leq x] = K(x).$$

[43] Asymptotic when the sample sizes n *and* m are large: $n, m \to \infty$.

- *Highlights and caveats*. The two-sample Kolmogorov–Smirnov test is **distribution-free if $F = G$ is continuous** and the samples $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$ have **no ties**. If these assumptions are met, then the iid sample $X_1, \ldots, X_n, Y_1, \ldots, Y_m \overset{H_0}{\sim} F = G$ generates the iid sample $U_1, \ldots, U_{n+m} \overset{H_0}{\sim} \mathcal{U}(0,1)$, where $U_i := F(X_i)$ and $U_{n+j} := G(Y_j)$, for $i = 1, \ldots, n$, $j = 1, \ldots, m$. As a consequence, the distribution of (6.2) does not depend on $F = G$. If $F = G$ is not continuous or there are ties on the sample, the K function is *not* the true asymptotic distribution.

- *Implementation in R*. For continuous data and continuous $F = G$, the test statistic $D_{n,m}$ and the asymptotic p-value are readily available through the `ks.test` function (with its default `alternative = "two-sided"`). For discrete $F = G$, a test implementation can be achieved through the permutation approach to be seen in Section 6.2.3.

The construction of the two-sample Kolmogorov–Smirnov test statistic is illustrated in the following chunk of code.

```
# Sample data
n <- 10; m <- 10
mu1 <- 0; sd1 <- 1
mu2 <- 0; sd2 <- 2
set.seed(1998)
samp1 <- rnorm(n = n, mean = mu1, sd = sd1)
samp2 <- rnorm(n = m, mean = mu2, sd = sd2)

# Fn vs. Gm
plot(ecdf(samp1), main = "", ylab = "Probability", xlim = c(-3.5, 3.5),
     ylim = c(0, 1.3))
lines(ecdf(samp2), main = "", col = 4)

# Add Dnm1
samp1_sorted <- sort(samp1)
samp2_sorted <- sort(samp2)
Dnm_1 <- abs((1:n) / n - ecdf(samp2)(samp1_sorted))
i1 <- which.max(Dnm_1)
lines(rep(samp2_sorted[i1], 2),
      c(i1 / m, ecdf(samp1_sorted)(samp2_sorted[i1])),
```

```
      col = 3, lwd = 2, type = "o", pch = 16, cex = 0.75)
rug(samp1, col = 1)

# Add Dnm2
Dnm_2 <- abs(ecdf(samp1)(samp2_sorted) - (1:m) / m)
i2 <- which.max(Dnm_2)
lines(rep(samp1_sorted[i2], 2),
      c(i2 / n, ecdf(samp2_sorted)(samp1_sorted[i2])),
      col = 2, lwd = 2, type = "o", pch = 16, cex = 0.75)
rug(samp2, col = 4)
legend("topleft", lwd = 2, col = c(1, 4, 3, 2), legend =
         latex2exp::TeX(c("$F_n$", "$G_m$", "$D_{n,m,1}$", "$D_{n,m,2}$")))
```

An instance of the use of the `ks.test` function for the two-sided homogeneity test is given below.

```
# Check the test for H0 true
set.seed(123456)
x0 <- rgamma(n = 50, shape = 1, scale = 1)
y0 <- rgamma(n = 100, shape = 1, scale = 1)
ks.test(x = x0, y = y0)
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  x0 and y0
## D = 0.14, p-value = 0.5185
## alternative hypothesis: two-sided

# Check the test for H0 false
x1 <- rgamma(n = 50, shape = 2, scale = 1)
y1 <- rgamma(n = 75, shape = 1, scale = 1)
ks.test(x = x1, y = y1)
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  x1 and y1
## D = 0.35333, p-value = 0.0008513
## alternative hypothesis: two-sided
```

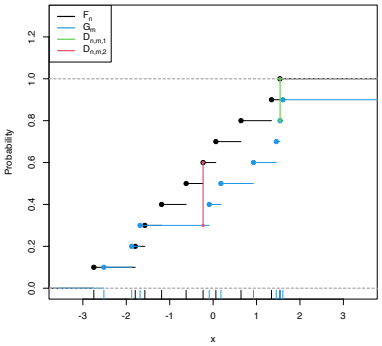


Figure 6.7: Kolmogorov–Smirnov distance $D_{n,m} = \max(D_{n,m,1}, D_{n,m,2})$ for two samples of sizes $n = m = 10$ coming from $F(\cdot) = \Phi\left(\frac{\cdot - \mu_1}{\sigma_1}\right)$ and $G(\cdot) = \Phi\left(\frac{\cdot - \mu_2}{\sigma_2}\right)$, where $\mu_1 = \mu_2 = 0$, and $\sigma_1 = 1$ and $\sigma_2 = \sqrt{2}$.

Two-sample Kolmogorov–Smirnov test (one-sided)

- *Test purpose*. Given $X_1, \ldots, X_n \sim F$ and $Y_1, \ldots, Y_m \sim G$, it tests $H_0 : F = G$ vs. $H_1 : F < G$.[44] **Rejection** of H_0 in favor of H_1 gives evidence for the ***local* stochastic dominance of** X **over** Y (which may or may not be *global*).

- *Statistic definition*. The test statistic uses the maximum *signed* distance between F_n and G_m:

$$D^-_{n,m} := \sqrt{\frac{nm}{n+m}} \sup_{x \in \mathbb{R}} (G_m(x) - F_n(x)).$$

  If $H_1 : F < G$ holds, then $D^-_{n,m}$ tends to have large *positive* values. Conversely, when $H_0 : F = G$ or $F > G$ holds, smaller values of $D^-_{n,m}$ are expected, possibly negative. The test rejects H_0 when $D^-_{n,m}$ is large.

- *Statistic computation*. The computation of $D^-_{n,m}$ is analogous to that of $D_{n,m}$ given in (6.15), but removing absolute values and changing its sign:[45]

[44] The case $H_1 : F > G$ is analogous and not required, as the roles of F and G can be interchanged.

[45] Analogously, $D^+_{n,m,j}$ changes the sign inside the maximum of $D^-_{n,m,j}$, $j = 1, 2$, and $D^+_{n,m} := \max(D^+_{n,m,1}, D^+_{n,m,2})$.

$$
\begin{aligned}
D^-_{n,m} &= \max(D^-_{n,m,1}, D^-_{n,m,2}),\\
D^-_{n,m,1} &:= \sqrt{\frac{nm}{n+m}} \max_{1\leq i\leq n} \left(G_m(X_{(i)}) - \frac{i}{n} \right),\\
D^-_{n,m,2} &:= \sqrt{\frac{nm}{n+m}} \max_{1\leq j\leq m} \left(\frac{j}{m} - F_n(Y_{(j)}) \right).
\end{aligned}
$$

- *Distribution under H_0.* If H_0 holds and $F = G$ is continuous, then $D^-_{n,m}$ has the **same asymptotic cdf as** $D_{n,m}$.
- *Highlights and caveats.* The one-sided two-sample Kolmogorov–Smirnov test is also distribution-free if $F = G$ is continuous and the samples $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$ have no ties.
- *Implementation in R.* For continuous data and continuous $F = G$, the test statistic $D^-_{n,m}$ and the asymptotic p-value are readily available through the `ks.test` function. The one-sided test is carried out if `alternative = "less"` or `alternative = "greater"` (not the defaults). The argument `alternative` means the *direction of cdf dominance*: **`"less"`** $\equiv F < G$; **`"greater"`** $\equiv F > G$.[46],[47] Permutations (see Section 6.2.3) can be used for obtaining non-asymptotic p-values and dealing with discrete samples.

[46] Confusingly, `ks.test`'s `alternative` does not refer to the direction of stochastic dominance, which is the codification used in the `alternative` arguments of the `t.test` and `wilcox.test` functions.

[47] Observe that `ks.test` also implements the one-sample Kolmogorov–Smirnov test seen in Section 6.1.1 with one-sided alternatives. That is, one can conduct the test $H_0 : F = F_0$ vs. $H_1 : F < F_0$ (or $H_1 : F > F_0$) using `alternative = "less"` (or `alternative = "greater"`).

```
# Check the test for H0 true
set.seed(123456)
x0 <- rgamma(n = 50, shape = 1, scale = 1)
y0 <- rgamma(n = 100, shape = 1, scale = 1)
ks.test(x = x0, y = y0, alternative = "less") # H1: F < G
##
##  Exact two-sample Kolmogorov-Smirnov test
##
## data:  x0 and y0
## D^- = 0.08, p-value = 0.643
## alternative hypothesis: the CDF of x lies below that of y
ks.test(x = x0, y = y0, alternative = "greater") # H1: F > G
##
##  Exact two-sample Kolmogorov-Smirnov test
##
## data:  x0 and y0
## D^+ = 0.14, p-value = 0.2638
## alternative hypothesis: the CDF of x lies above that of y

# Check the test for H0 false
x1 <- rnorm(n = 50, mean = 1, sd = 1)
y1 <- rnorm(n = 75, mean = 0, sd = 1)
ks.test(x = x1, y = y1, alternative = "less") # H1: F < G
##
##  Exact two-sample Kolmogorov-Smirnov test
##
## data:  x1 and y1
## D^- = 0.52667, p-value = 2.205e-08
## alternative hypothesis: the CDF of x lies below that of y
ks.test(x = x1, y = y1, alternative = "greater") # H1: F > G
##
##  Exact two-sample Kolmogorov-Smirnov test
##
## data:  x1 and y1
## D^+ = 1.4502e-15, p-value = 1
## alternative hypothesis: the CDF of x lies above that of y
# Interpretations:
# 1. Strong rejection of H0: F = G in favor of H1: F < G when
# alternative = "less", as in reality x1 is stochastically greater than y1.
# This outcome allows stating that "there is strong evidence supporting that
```

```
# x1 is locally stochastically greater than y1"
# 2. No rejection ("strong acceptance") of H0: F = G versus H1: F > G when
# alternative = "greater". Even if in reality x1 is stochastically greater than
# y1 (so H0 is false), the alternative to which H0 is confronted is even less
# plausible. A p-value ~ 1 indicates one is probably conducting the test in
# the uninteresting direction alternative!
```

Exercise 6.10. Consider the two populations described in Figure 6.8. $X \sim F$ is not stochastically greater than $Y \sim G$. However, Y is locally stochastically greater than X in $(-\infty, -0.75)$. Therefore, although hard to detect, the two-sample Kolmogorov–Smirnov test should eventually reject $H_0 : F = G$ in favor of $H_1 : F > G$. Conduct a simulation study to verify how fast this rejection takes place:

1. Simulate $X_1, \ldots, X_n \sim F$ and $Y_1, \ldots, Y_n \sim G$.
2. Apply `ks.test` with the corresponding `alternative` and evaluate if it rejects H_0 at significance level α.
3. Repeat Steps 1–2 $M = 100$ times and approximate the rejection probability by Monte Carlo.
4. Perform Steps 1–3 for n taken as `seq(10, 3e2, by = 10)` and plot the curve of rejection probability as a function of n. Do this for $\alpha = 0.10, 0.05, 0.01$.
5. Once you have a working solution, increase M to $M = 1,000$. What are your conclusions?
6. Take now n as `seq(50, 3e3, by = 50)`. Summarize your conclusions.

Two-sample Cramér–von Mises test

- *Test purpose*. Given $X_1, \ldots, X_n \sim F$, it tests $H_0 : F = G$ vs. $H_1 : F \neq G$ consistently against all the alternatives in H_1.

- *Statistic definition*. The test statistic proposed by Anderson (1962) uses a quadratic distance between F_n and G_m:

$$W_{n,m}^2 := \frac{nm}{n+m} \int (F_n(z) - G_m(z))^2 \, \mathrm{d}H_{n+m}(z), \tag{6.16}$$

  where H_{n+m} is the ecdf of the *pooled sample* $Z_1, \ldots, Z_{n+m}$, where $Z_i := X_i$ and $Z_{j+n} := Y_j$, $i = 1, \ldots, n$ and $j = 1, \ldots, m$.[48] Therefore, $H_{n+m}(z) = \frac{n}{n+m} F_n(z) + \frac{m}{n+m} G_m(z)$, $z \in \mathbb{R}$, and (6.16) equals[49]

$$W_{n,m}^2 = \frac{nm}{(n+m)^2} \sum_{k=1}^{n+m} (F_n(Z_k) - G_m(Z_k))^2. \tag{6.17}$$

- *Statistic computation*. The formula (6.17) is reasonably direct. Better computational efficiency may be obtained with ranks from the pooled sample.

- *Distribution under H_0*. If H_0 holds and $F = G$ is continuous, then $W_{n,m}^2$ has the **same asymptotic cdf as** W_n^2 (see (6.6)).

[48] Observe that $F = G$ is *not* employed to weight the integrand (6.16), as was the case in (6.4) with $\mathrm{d}F_0(x)$. The reason is because $F = G$ is unknown. However, under H_0, $F = G$ can be estimated better with H_{n+m}.

[49] The Lebesgue–Stieljes integral $\int h(z) \, \mathrm{d}H_{m+n}(z)$ is simply the sum $\frac{1}{n+m} \sum_{k=1}^{n+m} h(Z_k)$, since the pooled ecdf H_{m+n} is a discrete cdf assigning $1/(n+m)$ probability mass to each Z_k, $k = 1, \ldots, n+m$.

- *Highlights and caveats*. The two-sample Cramér–von Mises test is also **distribution-free if F_0 is continuous** and the sample has **no ties**. Otherwise, (6.6) is *not* the true asymptotic distribution. As in the one-sample case, empirical evidence suggests that the Cramér–von Mises test is often **more powerful than the Kolmogorov–Smirnov test**. However, the Cramér–von Mises is less versatile, since it does not admit simple modifications to test against one-sided alternatives.
- *Implementation in R*. See below for the *statistic* implementation. Asymptotic p-values can be obtained using `goftest::pCvM`. Permutations (see Section 6.2.3) can be used for obtaining non-asymptotic p-values and dealing with discrete samples.

```
# Two-sample Cramér-von Mises statistic
cvm2_stat <- function(x, y) {

  # Sample sizes
  n <- length(x)
  m <- length(y)

  # Pooled sample
  z <- c(x, y)

  # Statistic computation via ecdf()
  (n * m / (n + m)^2) * sum((ecdf(x)(z) - ecdf(y)(z))^2)

}

# Check the test for H0 true
set.seed(123456)
x0 <- rgamma(n = 50, shape = 1, scale = 1)
y0 <- rgamma(n = 100, shape = 1, scale = 1)
cvm0 <- cvm2_stat(x = x0, y = y0)
pval0 <- 1 - goftest::pCvM(q = cvm0)
c("statistic" = cvm0, "p-value"= pval0)
## statistic   p-value
## 0.1294889 0.4585971

# Check the test for H0 false
x1 <- rgamma(n = 50, shape = 2, scale = 1)
y1 <- rgamma(n = 75, shape = 1, scale = 1)
cvm1 <- cvm2_stat(x = x1, y = y1)
pval1 <- 1 - goftest::pCvM(q = cvm1)
c("statistic" = cvm1, "p-value"= pval1)
##    statistic      p-value
## 1.3283733333 0.0004264598
```

Two-sample Anderson–Darling test

- *Test purpose*. Given $X_1, \ldots, X_n \sim F$, it tests $H_0 : F = G$ vs. $H_1 : F \neq G$ consistently against all the alternatives in H_1.
- *Statistic definition*. The test statistic proposed by Pettitt (1976) uses a weighted quadratic distance between F_n and G_m:

$$A^2_{n,m} := \frac{nm}{n+m} \int \frac{(F_n(z) - G_m(z))^2}{H_{n+m}(z)(1 - H_{n+m}(z))} \, \mathrm{d}H_{n+m}(z), \tag{6.18}$$

  where H_{n+m} is the ecdf of $Z_1, \ldots, Z_{n+m}$. Therefore, (6.18) equals

$$A^2_{n,m} = \frac{nm}{(n+m)^2} \sum_{k=1}^{n+m} \frac{(F_n(Z_k) - G_m(Z_k))^2}{H_{n+m}(Z_k)(1 - H_{n+m}(Z_k))}. \tag{6.19}$$

In (6.19), it is implicitly assumed that the largest observation of the pooled sample, $Z_{(n+m)}$, is excluded from the sum, since $H_{n+m}(Z_{(n+m)}) = 1$.[50]

[50] Why is the minimum $Z_{(1)}$ not excluded?

- *Statistic computation*. The formula (6.19) is reasonably direct for computation. Better computational efficiency may be obtained with ranks from the pooled sample. The implicit exclusion of $Z_{(n+m)}$ from the sum in (6.19) can be made explicit with

$$A^2_{n,m} = \frac{nm}{(n+m)^2} \sum_{k=1}^{n+m-1} \frac{(F_n(Z_{(k)}) - G_m(Z_{(k)}))^2}{H_{n+m}(Z_{(k)})(1 - H_{n+m}(Z_{(k)}))}.$$

- *Distribution under H_0*. If H_0 holds and $F = G$ is continuous, then $A^2_{n,m}$ has the **same asymptotic cdf as** A^2_n (see (6.8)).

- *Highlights and caveats*. The two-sample Anderson–Darling test is also **distribution-free if F_0 is continuous** and the sample has **no ties**. Otherwise, (6.8) is *not* the true asymptotic distribution. As in the one-sample case, empirical evidence suggests that the Anderson–Darling test is often **more powerful than the Kolmogorov–Smirnov and Cramér–von Mises tests**. The two-sample Anderson–Darling test is also less versatile, since it does not admit simple modifications to test against one-sided alternatives.

- *Implementation in R*. See below for the *statistic* implementation. Asymptotic p-values can be obtained using `goftest::pAD`. Permutations (see Section 6.2.3) can be used for obtaining non-asymptotic p-values and dealing with discrete samples.

```
# Two-sample Anderson-Darling statistic
ad2_stat <- function(x, y) {

  # Sample sizes
  n <- length(x)
  m <- length(y)

  # Pooled sample and pooled ecdf
  z <- c(x, y)
  z <- z[-which.max(z)] # Exclude the largest point
  H <- rank(z) / (n + m)

  # Statistic computation via ecdf()
  (n * m / (n + m)^2) * sum((ecdf(x)(z) - ecdf(y)(z))^2 / ((1 - H) * H))

}

# Check the test for H0 true
set.seed(123456)
x0 <- rgamma(n = 50, shape = 1, scale = 1)
y0 <- rgamma(n = 100, shape = 1, scale = 1)
ad0 <- ad2_stat(x = x0, y = y0)
pval0 <- 1 - goftest::pAD(q = ad0)
c("statistic" = ad0, "p-value"= pval0)
## statistic   p-value
## 0.8603617 0.4394751

# Check the test for H0 false
x1 <- rgamma(n = 50, shape = 2, scale = 1)
y1 <- rgamma(n = 75, shape = 1, scale = 1)
```

```
ad1 <- ad2_stat(x = x1, y = y1)
pval1 <- 1 - goftest::pAD(q = ad1)
c("statistic" = ad1, "p-value"= pval1)
##    statistic      p-value
## 6.7978295422 0.0004109238
```

Exercise 6.11. Verify the correctness of the asymptotic null distributions of $W^2_{n,m}$ and $A^2_{n,m}$. To do so:

1. Simulate $M = 1,000$ pairs of samples of sizes $n = 200$ and $m = 150$ under H_0.
2. Obtain the statistics $W^2_{n,m;1}, \ldots, W^2_{n,m;M}$ and plot their ecdf.
3. Overlay the asymptotic cdf of $W_{n,m}$, provided by `goftest::pCvM` for the one-sample test.
4. Repeat Steps 2–3 for $A^2_{n,m}$.

Exercise 6.12. Implement in R a single function to properly conduct either the two-sample Cramér–von Mises test or the two-sample Anderson–Darling test. The function must return a tidied `"htest"` object. Is your implementation of $W^2_{n,m}$ and $A^2_{n,m}$ able to improve `cvm2_stat` and `ad2_stat` in speed? Measure the running times for $n = 300$ and $m = 200$ with the `microbenchmark::microbenchmark` function.

6.2.2 *Specific tests for distribution shifts*

Among the previous ecdf-based tests of homogeneity, only the two-sample Kolmogorov–Smirnov test was able to readily deal with one-sided alternatives. However, this test is often more conservative than the Cramér–von Mises and the Anderson–Darling, so it is apparent that there is room for improvement. We next see two nonparametric tests, the *Wilcoxon–Mann–Whitney test*[51] and the *Wilcoxon signed-rank test*,[52] which are designed to detect **one-sided alternatives *related* to X being stochastically greater than Y**. Both tests are more powerful than the one-sided Kolmogorov–Smirnov.

In a very vague and imprecise form, these tests can be interpreted as "nonparametric t-tests" for unpaired and paired data.[53] The rationale is that, very often, the aforementioned one-sided alternatives are related to differences in the distributions produced by a *shift* in their main masses of probability. However, as we will see, there are **many subtleties and sides** to the previous interpretation. Unfortunately, these nuances are usually ignored in the literature and, as a consequence, the Wilcoxon–Mann–Whitney and Wilcoxon signed-rank tests are massively misunderstood and misused.

Wilcoxon–Mann–Whitney test

The Wilcoxon–Mann–Whitney test addresses the hypothesis testing of

$$H_0 : F = G \quad \text{vs.} \quad H_1 : \mathbb{P}[X \geq Y] > 0.5.$$

The alternative hypothesis H_1 requires special care:

[51] Also known as *Wilcoxon rank-sum test*, *Mann–Whitney U test*, and *Mann–Whitney–Wilcoxon test*. The literature is not unanimous in the terminology, which can be confusing. This ambiguity is partially explained by the almost-coetaneity of the papers by Wilcoxon (1945) and Mann and Whitney (1947).

[52] Also known as the *signed-rank test*.

[53] Recall that the two-sample t-test for $H_0 : \mu_X = \mu_Y$ vs. $H_1 : \mu_X \neq \mu_Y$ (one-sided versions: $H_1 : \mu_X > \mu_Y$, $H_1 : \mu_X < \mu_Y$) is actually a *nonparametric* test for comparing means of arbitrary random variables, possibly non-normal, if employed for **large sample sizes**. Indeed, due to the CLT, $(\bar{X} - \bar{Y})/\sqrt{(\hat{S}^2_X/n + \hat{S}^2_Y/m)} \xrightarrow{d} \mathcal{N}(0,1)$ as $n, m \to \infty$, irrespective of how X and Y are distributed. Here $\hat{S}^2_X$ and $\hat{S}^2_Y$ represent the quasi-variances of $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$, respectively. Equivalently, the one-sample t-test for $H_0 : \mu_X = \mu_0$ vs. $H_1 : \mu_X \neq \mu_0$ (one-sided versions: $H_1 : \mu_X > \mu_0$, $H_1 : \mu_X < \mu_0$) is also nonparametric if employed for large sample sizes: $\bar{X}/(\hat{S}/\sqrt{n}) \xrightarrow{d} \mathcal{N}(0,1)$ as $n \to \infty$. The function `t.test` in R implements one/two-sample tests for two/one-sided alternatives.

- That $\mathbb{P}[X \geq Y] > 0.5$ intuitively means that **the main mass of probability of X is above that of Y** : it is more likely to observe $X \geq Y$ than $X < Y$.

- H_1 is *related to* X being stochastically greater than Y. Precisely, H_1 is essentially[54] implied by the latter:

$$F(x) < G(x) \text{ for all } x \in \mathbb{R} \implies \mathbb{P}[X \geq Y] \geq 0.5.$$

  The converse implication is false (see Figure 6.8). Hence, $H_1 : \mathbb{P}[X \geq Y] > 0.5$ is more specific than "X is stochastically greater than Y". H_1 neither implies nor is implied by the two-sample Kolmogorov–Smirnov's alternative $H_1' : F < G$ (which can be regarded as *local* stochastic dominance).[55]

- In general, H_1 does not directly inform on the medians/means of X and Y. Thus, **in general, the Wilcoxon–Mann–Whitney test does not compare medians/means**. The probability in H_1 is, in general, exogenous to medians/means:[56]

$$\mathbb{P}[X \geq Y] = \int F_Y(x) \, \mathrm{d}F_X(x). \tag{6.20}$$

  Only in very specific circumstances H_1 is related to medians/means. If $X \sim F$ and $Y \sim G$ are **symmetric** random variables with medians m_X and m_Y (if the means exist, they equal the medians), then

$$\mathbb{P}[X \geq Y] > 0.5 \iff m_X > m_Y. \tag{6.21}$$

  This characterization informs on the closest the Wilcoxon–Mann–Whitney test gets to a nonparametric version of the t-test.[57] However, as the two counterexamples in Figures 6.9 and 6.10 respectively illustrate, (6.21) is not true in general, neither for means nor for medians. It may happen that: (*i*) $\mathbb{P}[X \geq Y] > 0.5$ but $\mu_X < \mu_Y$; or (*ii*) $\mathbb{P}[X \geq Y] > 0.5$ but $m_X < m_Y$.

We are now ready to see the main concepts of the Wilcoxon–Mann–Whitney test:

- *Test purpose*. Given $X_1, \ldots, X_n \sim F$ and $Y_1, \ldots, Y_m \sim G$, it tests $H_0 : F = G$ vs. $H_1 : \mathbb{P}[X \geq Y] > 0.5$.

- *Statistic definition*. On the one hand, the test statistic proposed by Mann and Whitney (1947) directly targets $\mathbb{P}[X \geq Y]$[58] with the (unstandardized) estimator

$$U_{n,m;\mathrm{MW}} = \sum_{i=1}^{n} \sum_{j=1}^{m} 1_{\{X_i > Y_j\}}. \tag{6.22}$$

  Values of U_n that are significantly larger than $(nm)/2$, the expected value under H_0, indicate evidence in favor of H_1. On the other hand, the test statistic proposed by Wilcoxon (1945) is

$$U_{n,m;\mathrm{W}} = \sum_{i=1}^{n} \mathrm{rank}_{X,Y}(Y_i), \tag{6.23}$$

[54] Recall the strict inequality in H_1; see Exercise 6.14 for more details.

[55] Recall that the exact meaning of "$F < G$" was that $F(x) < G(x)$ for *some* $x \in \mathbb{R}$, not for *all* $x \in \mathbb{R}$.

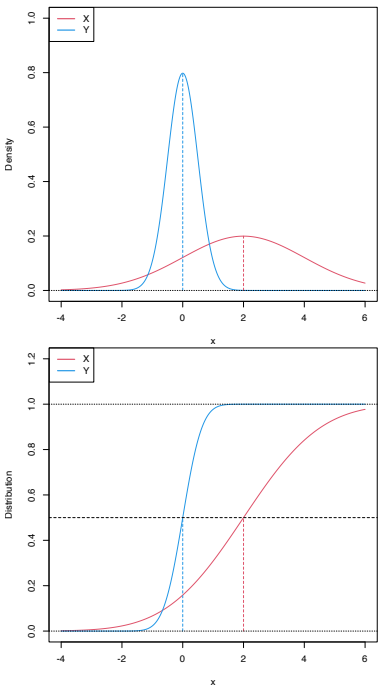


Figure 6.8: Pdfs and cdfs of $X \sim \mathcal{N}(2,4)$ and $Y \sim \mathcal{N}(0,0.25)$. X is *not* stochastically greater than Y, as the cdfs cross, but $\mathbb{P}[X \geq Y] = 0.834$. Y is *locally* stochastically greater than X in $(-\infty, -0.75)$. The means are shown in vertical lines. Note that the variances of X and Y are not common; recall the difference with respect to the situation in Figure 6.6.

[56] Observe that $\mathbb{P}[Y \geq X] = \int F_X(x) \, \mathrm{d}F_Y(x)$ and how this probability is complementary of (6.20) when X or Y are absolutely continuous.

[57] Due to the easier interpretation of the Wilcoxon–Mann–Whitney test for symmetric populations, this particular case is sometimes included as an assumption of the test. This simplified presentation of the test unnecessarily narrows its scope.

where $\text{rank}_{X,Y}(Y_i) := nF_n(Y_i) + mG_m(Y_i)$ is the rank of Y_i on the pooled sample $X_1, \ldots, X_n, Y_1, \ldots, Y_m$. It happens that the tests based on either (6.22) or (6.23) are equivalent[59] when there are no ties on the samples of X and Y, since

$$U_{n,m;\text{MW}} = nm + \frac{m(m+1)}{2} - U_{n,m;\text{W}}. \tag{6.24}$$

- *Statistic computation*. Computing (6.23) is easier than (6.22).
- *Distribution under H_0*. The test statistic (6.22) is a non-degenerate U-statistic. Therefore, since $\mathbb{E}[U_{n,m;\text{MW}}] = (nm)/2$ and $\mathbb{V}\text{ar}[U_{n,m;\text{MW}}] = nm(n+m+1)/12$, under H_0 and with $F = G$ continuous, when $n, m \to \infty$,

$$\frac{U_{n,m;\text{MW}} - (nm)/2}{\sqrt{nm(n+m+1)/12}} \xrightarrow{d} \mathcal{N}(0,1).$$

- *Highlights and caveats*. The Wilcoxon–Mann–Whitney test is often regarded as a "nonparametric t-test" or as a "nonparametric test for comparing medians". This interpretation is only correct if both X and Y are symmetric. In this case, it is a nonparametric *analogue* to the t-test. Otherwise, it is a nonparametric *extension* of the t-test that evaluates if there is a **shift in the main mass of probability** of X or Y. In any case, $H_1 : \mathbb{P}[X \geq Y] > 0.5$ is related to X being stochastically greater than Y, though it is less strict, and it is not comparable to the one-sided Kolmogorov–Smirnov's alternative $H_1' : F < G$. The test is **distribution-free**.
- *Implementation in R*. The test statistic $U_{n,m;\text{MW}}$ (implemented using (6.24)) and the exact/asymptotic p-value are readily available through the `wilcox.test` function. One must specify `alternative = "greater"` to test H_0 against $H_1 : \mathbb{P}[X \geq Y] > 0.5$.[60] The *exact* cdf of $U_{n,m;\text{MW}}$ under H_0 is available in `pwilcoxon`.

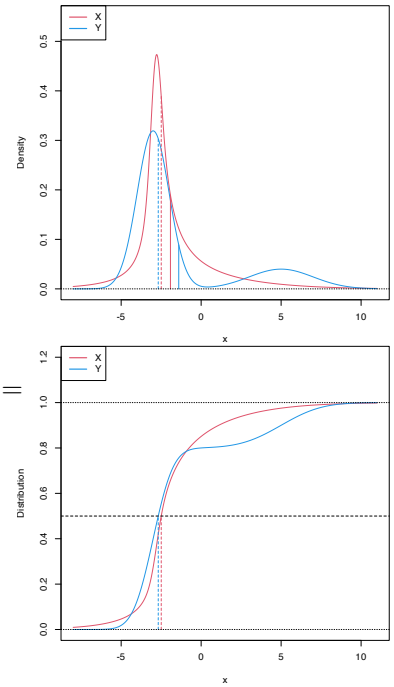


Figure 6.9: Pdfs and cdfs of X and $Y \sim 0.8\mathcal{N}(-3,1) + 0.2\mathcal{N}(5,4)$, where X is distributed as the mixture `nor1mix::MW.nm3` but with its standard deviations multiplied by 5. $\mathbb{P}[X \geq Y] = 0.5356$ but $\mu_X < \mu_Y$, since $\mu_X = -1.9189$ and $\mu_Y = -1.4$. The means are shown in solid vertical lines; the dashed vertical lines stand for the medians $m_X = -2.4944$ and $m_Y = -2.6814$.

The following code demonstrates how to carry out the Wilcoxon–Mann–Whitney test.

```
# Check the test for H0 true
set.seed(123456)
n <- 50
m <- 100
x0 <- rgamma(n = n, shape = 1, scale = 1)
y0 <- rgamma(n = m, shape = 1, scale = 1)
wilcox.test(x = x0, y = y0) # H1: P[X >= Y] != 0.5
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  x0 and y0
## W = 2403, p-value = 0.7004
## alternative hypothesis: true location shift is not equal to 0
wilcox.test(x = x0, y = y0, alternative = "greater") # H1: P[X >= Y] > 0.5
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  x0 and y0
## W = 2403, p-value = 0.6513
```

```
## alternative hypothesis: true location shift is greater than 0
wilcox.test(x = x0, y = y0, alternative = "less") # H1: P[X <= Y] > 0.5
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  x0 and y0
## W = 2403, p-value = 0.3502
## alternative hypothesis: true location shift is less than 0

# Check the test for H0 false
x1 <- rnorm(n = n, mean = 0, sd = 1)
y1 <- rnorm(n = m, mean = 1, sd = 2)
wilcox.test(x = x1, y = y1) # H1: P[X >= Y] != 0.5
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  x1 and y1
## W = 1684, p-value = 0.001149
## alternative hypothesis: true location shift is not equal to 0
wilcox.test(x = x1, y = y1, alternative = "greater") # H1: P[X >= Y] > 0.5
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  x1 and y1
## W = 1684, p-value = 0.9994
## alternative hypothesis: true location shift is greater than 0
wilcox.test(x = x1, y = y1, alternative = "less") # H1: P[X <= Y] > 0.5
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  x1 and y1
## W = 1684, p-value = 0.0005746
## alternative hypothesis: true location shift is less than 0

# Exact pmf versus asymptotic pdf
# Beware! dwilcox considers (m, n) as the sample sizes of (X, Y)
x <- seq(1500, 3500, by = 100)
plot(x, dwilcox(x = x, m = n, n = m), type = "h", ylab = "Density")
curve(dnorm(x, mean = n * m / 2, sd = sqrt((n * m * (n + m + 1)) / 12)),
      add = TRUE, col = 2)
```

Exercise 6.13. Prove (6.21) for:

a. Continuous random variables.
b. Random variables.

Exercise 6.14. Derive (6.20) and, using it, prove that

$$F_X(x) < F_Y(x) \text{ for all } x \in \mathbb{R} \implies \mathbb{P}[X \geq Y] \geq 0.5.$$

Show that $\mathbb{P}[X \geq Y] > 0.5$ is verified if, in addition, $F_X(x) < F_Y(x)$ for $x \in D$, where $\mathbb{P}[X \in D] > 0$ or $\mathbb{P}[Y \in D] > 0$. You can assume X and Y are absolutely continuous variables.

Exercise 6.15. Derive additional counterexamples, significantly different from those in Figures 6.9 and 6.10, for the *erroneous* claims

a. $\mu_X > \mu_Y \implies \mathbb{P}[X \geq Y] > 0.5$;
b. $m_X > m_Y \implies \mathbb{P}[X \geq Y] > 0.5$.

Exercise 6.16. Verify (6.24) numerically in R with X and Y being continuous random variables. Do so by implementing functions for the test statistics $U_{n,m;\mathrm{MW}}$ and $U_{n,m;\mathrm{W}}$. Check also that (6.24) is not satisfied when the samples of X and Y have ties.

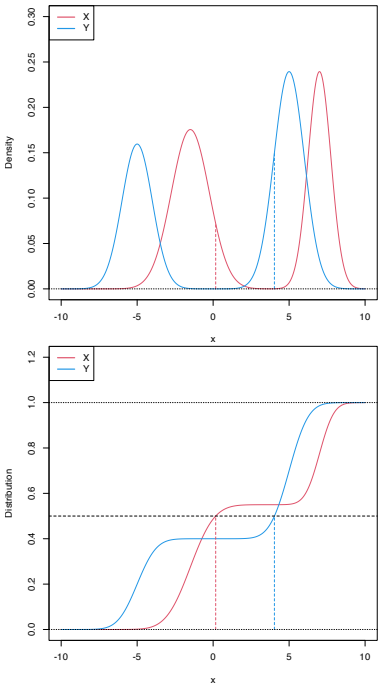
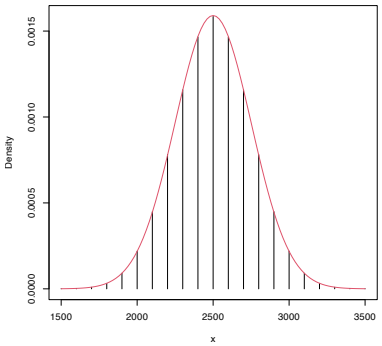


Figure 6.10: Pdfs and cdfs of $X \sim 0.55\mathcal{N}(-1.5, 1.25^2) + 0.45\mathcal{N}(7, 0.75^2)$ and $Y \sim 0.4\mathcal{N}(-5, 1) + 0.6\mathcal{N}(5, 1)$. $\mathbb{P}[X \geq Y] = 0.6520$ but $m_X < m_Y$, since $m_X = 0.169$ and $m_Y = 4.0326$. The medians are shown in dashed vertical lines; the means are $\mu_X = 2.325$ and $\mu_Y = 1$.

[58] Under continuity of X and Y.

[59] Hence the terminology Wilcoxon–Mann–Whitney. Why are the tests based on (6.22) and (6.23) equivalent?

[60] Observe that `wilcox.test` also implements the one-sided alternative $H_1 : \mathbb{P}[X \leq Y] > 0.5$ if `alternative = "less"` (rejection for small values of (6.22)) and the two equivalent two-sided alternatives $H_1 : \mathbb{P}[X \geq Y] \neq 0.5$ or $H_1 : \mathbb{P}[X \leq Y] \neq 0.5$ if `alternative = "two.sided"` (default; rejection for large and small values of (6.22)).



Wilcoxon signed-rank test

Wilcoxon signed-rank test adapts the Wilcoxon–Mann–Whitney test for *paired* measurements (X, Y) arising from the same individual.

- *Test purpose*. Given $(X_1, Y_1), \ldots, (X_n, Y_n) \sim F$, it tests $H_0 : F_X = F_Y$ vs. $H_1 : \mathbb{P}[X \geq Y] > 0.5$, where F_X and F_Y are the marginal cdfs of X and Y.
- *Statistic definition*. Wilcoxon (1945)'s test statistic directly targets $\mathbb{P}[X \geq Y]$,[61] now using the paired observations, with the (unstandardized) estimator

$$S_n := \sum_{i=1}^{n} 1_{\{X_i > Y_i\}}. \tag{6.25}$$

  Values of S_n that are significantly larger than $n/2$, the expected value under H_0, indicate evidence in favor of H_1.

- *Statistic computation*. Computing (6.25) is straightforward.
- *Distribution under H_0*. Under H_0, it follows trivially that $S_n \sim \mathrm{B}(n, 0.5)$.
- *Highlights and caveats*. The Wilcoxon signed-rank test shares the same highlights and caveats as its unpaired counterpart. It can be regarded as a "nonparametric paired t-test" if both X and Y are symmetric. In this case, it is a nonparametric *analogue* to the paired t-test. Otherwise, it is a nonparametric *extension* of the paired t-test that evaluates if there is a **shift in the main mass of probability** of X or Y when both are related.[62] The test is **distribution-free**.
- *Implementation in R*. The test statistic S_n and the exact p-value are available through the `wilcox.test` function. One must specify `alternative = "greater"` to test H_0 against $H_1 : \mathbb{P}[X \geq Y] > 0.5$.[63] The exact cdf of S_n under H_0 is available in `psignrank`.

[61] Again, under continuity.

[62] Equivalently, it evaluates if the main mass of probability of $X - Y$ is located further from 0.

[63] One- and two-sided alternatives are also possible, analogously to the unpaired case.

The following code exemplifies how to carry out the Wilcoxon signed-rank test.

```
# Check the test for H0 true
set.seed(123456)
x0 <- rgamma(n = 50, shape = 1, scale = 1)
y0 <- x0 + rnorm(n = 50)
wilcox.test(x = x0, y = y0, paired = TRUE) # H1: P[X >= Y] != 0.5
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  x0 and y0
## V = 537, p-value = 0.3344
## alternative hypothesis: true location shift is not equal to 0
wilcox.test(x = x0, y = y0, paired = TRUE, alternative = "greater")
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  x0 and y0
```

```
## V = 537, p-value = 0.8352
## alternative hypothesis: true location shift is greater than 0
# H1: P[X >= Y] > 0.5
wilcox.test(x = x0, y = y0, paired = TRUE, alternative = "less")
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  x0 and y0
## V = 537, p-value = 0.1672
## alternative hypothesis: true location shift is less than 0
# H1: P[X <= Y] > 0.5

# Check the test for H0 false
x1 <- rnorm(n = 50, mean = 0, sd = 1)
y1 <- x1 + rnorm(n = 50, mean = 1, sd = 2)
wilcox.test(x = x1, y = y1, paired = TRUE) # H1: P[X >= Y] != 0.5
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  x1 and y1
## V = 255, p-value = 0.0002264
## alternative hypothesis: true location shift is not equal to 0
wilcox.test(x = x1, y = y1, paired = TRUE, alternative = "greater")
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  x1 and y1
## V = 255, p-value = 0.9999
## alternative hypothesis: true location shift is greater than 0
# H1: P[X >= Y] > 0.5
wilcox.test(x = x1, y = y1, paired = TRUE, alternative = "less")
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  x1 and y1
## V = 255, p-value = 0.0001132
## alternative hypothesis: true location shift is less than 0
# H1: P[X <= Y] > 0.5
```

Exercise 6.17. Explore by Monte Carlo the power of the following two-sample tests: one-sided Kolmogorov–Smirnov, Wilcoxon–Mann–Whitney, and t-test. Take $\delta =$ `0:3` as the "deviation strength from H_0" for the following scenarios:

a. $X \sim \mathcal{N}(0.25\delta, 1)$ and $Y \sim \mathcal{N}(0, 1)$.
b. $X \sim t_3 + 0.25\delta$ and $Y \sim t_1$.
c. $X \sim 0.5\mathcal{N}(-\delta, (1 + 0.25\delta)^2) + 0.5\mathcal{N}(\delta, (1 + 0.25\delta)^2)$ and $Y \sim \mathcal{N}(0, 2)$.

For each scenario, plot the power curves of the three tests for $n =$ `seq(10, 200, by = 10)`. Use $\alpha = 0.05$. Consider first $M = 100$ to have a working solution, then increase it to $M = 1,000$. To visualize all the results effectively, you may want to build a `Shiny` app that, pre-caching the Monte Carlo study, displays the three power curves with respect to n, for a choice of δ, scenario, and tests to display. Summarize your conclusions in separated points (you may want to visualize first the models a–b).

6.2.3 *Permutation-based approach to comparing distributions*

Under the null hypothesis of homogeneity, the iid samples $X_1, \ldots, X_n \sim F$ and $Y_1, \ldots, Y_m \sim G$ have the same distributions. Therefore, since

both samples are assumed to be independent from each other,[64] they are **exchangeable under** $H_0 : F = G$. This means that, for example, the random vector $(X_1, \ldots, X_{\min(n,m)})'$ has the same distribution as $(Y_1, \ldots, Y_{\min(n,m)})'$ under H_0. More generally, each of the $p!$ random subvectors of length $p \leq n + m$ that can be extracted without repetitions from $(X_1, \ldots, X_n, Y_1, \ldots, Y_m)'$ have the exact same distribution under H_0.[65] This revelation motivates a resampling procedure that is similar in spirit to the bootstrap seen in Section 6.1.3, yet it is much simpler as it does not require parametric estimation: **permutations**.

We consider a N-permutation function $\sigma : \{1, \ldots, N\} \longrightarrow \{1, \ldots, N\}$. This function is such that $\sigma(i) = j$, for certain $i, j \in \{1, \ldots, N\}$. The number of N-permutations quickly becomes extremely large, as there are $N!$ of them.[66] As an example, if $N = 3$, there exist $3! = 6$ possible 3-permutations $\sigma_1, \ldots, \sigma_6$:

i	$\sigma_1(i)$	$\sigma_2(i)$	$\sigma_3(i)$	$\sigma_4(i)$	$\sigma_5(i)$	$\sigma_6(i)$
1	1	1	2	2	3	3
2	2	3	1	3	1	2
3	3	2	3	1	2	1

Using permutations, it is simple to precise that, for *any* $(n + m)$-permutation σ, under H_0,

$$(Z_{\sigma(1)}, \ldots, Z_{\sigma(n+m)}) \overset{d}{=} (Z_1, \ldots, Z_{n+m}), \tag{6.26}$$

where we use the standard notation for the pooled sample, $Z_i = X_i$ and $Z_{j+n} = Y_j$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$. A very important consequence of (6.26) is that, under H_0,

$$T_{n,m}(Z_{\sigma(1)}, \ldots, Z_{\sigma(n+m)}) \overset{d}{=} T_{n,m}(Z_1, \ldots, Z_{n+m})$$

for any proper test statistic $T_{n,m}$ for $H_0 : F = G$. Therefore, denoting $T_{n,m} \equiv T_{n,m}(Z_1, \ldots, Z_{n+m})$ and $T^{\sigma}_{n,m} \equiv T_{n,m}(Z_{\sigma(1)}, \ldots, Z_{\sigma(n+m)})$, it happens that $\mathbb{P}[T_{n,m} \leq x] = \mathbb{P}[T^{\sigma}_{n,m} \leq x]$. Hence, it readily follows that

$$\mathbb{P}[T_{n,m} \leq x] = \frac{1}{(n+m)!} \sum_{k=1}^{(n+m)!} \mathbb{P}[T^{\sigma_k}_{n,m} \leq x], \tag{6.27}$$

where σ_k is the k-th out of $(n + m)!$ permutations.

We can now exploit (6.27) to approximate the distribution of $T_{n,m}$ under H_0. The key insight is to realize that, *conditionally* on the observed sample $Z_1, \ldots, Z_{n+m}$, we know whether or not $T^{\sigma_k}_{n,m} \leq x$ is true. We can then replace $\mathbb{P}[T^{\sigma_k}_{n,m} \leq x]$ with its estimation $1_{\left\{T^{\sigma_k}_{n,m} \leq x\right\}}$ in (6.27). However, the evaluation of the sum of $(n + m)!$ terms is often impossible. Instead, an estimation of its value is obtained by considering B **randomly-chosen** $(n + m)$**-permutations of the pooled sample**. Joining these two considerations, it follows that

$$\mathbb{P}[T_{n,m} \leq x] \approx \frac{1}{B} \sum_{b=1}^{B} 1_{\left\{T^{\hat{\sigma}_b}_{n,m} \leq x\right\}}, \tag{6.28}$$

[64] We postpone until the end of the section the treatment of the paired sample case.

[65] If repetitions were allowed, the distributions would be degenerate. For example, if X_1, X_2, X_3 are iid, the distribution of (X_1, X_1, X_2) is degenerate.

[66] If $N = 60$, there are 8.32×10^{81} N-permutations. This number is already 10 times larger than the estimated number of atoms in the observable universe.

where $\hat{\sigma}_b$, $b = 1, \ldots, B$, denote the B randomly-chosen $(n+m)$-permutations.[67]

The null distribution of $T_{n,m}$ can be approximated through (6.28). We summarize next the whole **permutation-based procedure** for performing a homogeneity test that rejects H_0 for large values of the test statistic:

1. Compute $T_{n,m} \equiv T_{n,m}(Z_1, \ldots, Z_{n+m})$.
2. Enter the "permutation world". For $b = 1, \ldots, B$:
   i. Simulate a randomly-permuted sample $Z_1^{*b}, \ldots, Z_{n+m}^{*b}$ from $\{Z_1, \ldots, Z_{n+m}\}$.[68]
   ii. Compute $T_{n,m}^{*b} \equiv T_{n,m}(Z_1^{*b}, \ldots, Z_{n+m}^{*b})$.
3. Obtain the p-value approximation

$$p\text{-value} \approx \frac{1}{B} \sum_{b=1}^{B} 1_{\{T_{n,m}^{*b} > T_{n,m}\}}$$

   and emit a test decision from it. Modify it accordingly if rejection of H_0 does not happen for large values of $T_{n,m}$.

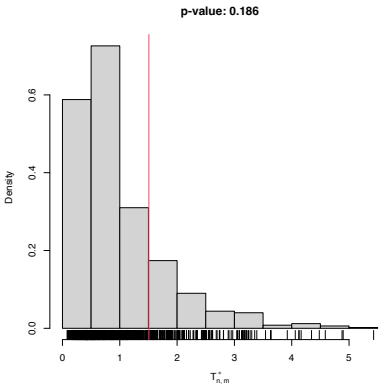
The following chunk of code provides a template function for implementing the previous algorithm. It is illustrated with an application of the two-sample Anderson–Darling test to discrete data (using the function `ad2_stat`).

```
# A homogeneity test using the Anderson-Darling statistic
perm_comp_test <- function(x, y, B = 1e3, plot_boot = TRUE) {

  # Sizes of x and y
  n <- length(x)
  m <- length(y)

  # Test statistic function. Requires TWO arguments, one being the original
  # data (X_1, ..., X_n, Y_1, ..., Y_m) and the other containing the random
  # index for permuting the sample
  Tn <- function(data, perm_index) {

    # Permute sample by perm_index
    data <- data[perm_index]

    # Split into two samples
    x <- data[1:n]
    y <- data[(n + 1):(n + m)]

    # Test statistic -- MODIFY DEPENDING ON THE PROBLEM
    ad2_stat(x = x, y = y)

  }

  # Perform permutation resampling with the aid of boot::boot
  Tn_star <- boot::boot(data = c(x, y), statistic = Tn,
                        sim = "permutation", R = B)

  # Test information -- MODIFY DEPENDING ON THE PROBLEM
  method <- "Permutation-based Anderson-Darling test of homogeneity"
  alternative <- "any alternative to homogeneity"

  # p-value: modify if rejection does not happen for large values of the
```

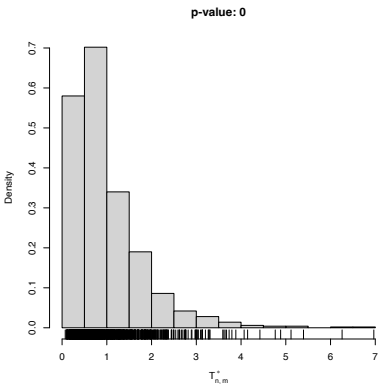
[67] Since $N! = (n+m)!$ quickly becomes extremely large, the probability of obtaining at least one duplicated permutation among B randomly-extracted permutations (with replacement) is very small. Precisely, the probability is `pbirthday(n = B, classes = factorial(N))`. If $B = 10^4$ and $N = 18!$ ($n = m = 9$), this probability is 7.81×10^{-9}.

[68] This sampling is done by extracting, *without replacement*, elements from the pooled sample `z <- c(x, y)`. This can be simply done with `sample(z)`.

```
  # test statistic
  pvalue <- mean(Tn_star$t > Tn_star$t0)

  # Construct an "htest" result
  result <- list(statistic = c("stat" = Tn_star$t0), p.value = pvalue,
                 statistic_perm = drop(Tn_star$t),
                 B = B, alternative = alternative, method = method,
                 data.name = deparse(substitute(x)))
  class(result) <- "htest"

  # Plot the position of the original statistic with respect to the
  # permutation replicates?
  if (plot_boot) {

    hist(result$statistic_perm, probability = TRUE,
         main = paste("p-value:", result$p.value),
         xlab = latex2exp::TeX("$T_{n,m}^*$"))
    rug(result$statistic_perm)
    abline(v = result$statistic, col = 2)

  }

  # Return "htest"
  return(result)

}

# Check the test for H0 true
set.seed(123456)
x0 <- rpois(n = 75, lambda = 5)
y0 <- rpois(n = 75, lambda = 5)
comp0 <- perm_comp_test(x = x0, y = y0, B = 1e3)
```




```
comp0
##
##  Permutation-based Anderson-Darling test of homogeneity
##
## data:  x0
## stat = 1.5077, p-value = 0.186
## alternative hypothesis: any alternative to homogeneity

# Check the test for H0 false
x1 <- rpois(n = 50, lambda = 3)
y1 <- rpois(n = 75, lambda = 5)
comp1 <- perm_comp_test(x = x1, y = y1, B = 1e3)
```




```
comp1
##
##  Permutation-based Anderson-Darling test of homogeneity
##
## data:  x1
## stat = 8.3997, p-value < 2.2e-16
## alternative hypothesis: any alternative to homogeneity
```

In a **paired sample** $(X_1, Y_1), \ldots, (X_n, Y_n) \sim F$, differently to the unpaired case, there is no independence between the X- and Y-components. Therefore, a variation on the previous permutation algorithm is needed, as the permutation has to be done within the individual observations, and not across individuals. This is achieved by replacing Step i in the previous algorithm with:

i. Simulate a randomly-permuted sample $(X_1^{*b}, Y_1^{*b}), \ldots, (X_n^{*b}, Y_n^{*b})$, where, for each $i = 1, \ldots, n$, (X_i^{*b}, Y_i^{*b}) is randomly extracted from $\{(X_i, Y_i), (Y_i, X_i)\}$.[69]

[69] This sampling is done by switching, randomly for each row, the columns of the sample. If `z <- cbind(x, y)`, this can be simply done with `ind_perm <- runif(n) > 0.5; z[ind_perm, ] <- z[ind_perm, 2:1]`.

Exercise 6.18. The F-statistic $F = \hat{S}_1^2/\hat{S}_2^2$ is employed to test $H_0 : \sigma_1^2 = \sigma_2^2$ vs. $H_1 : \sigma_1^2 > \sigma_2^2$ in normal populations $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$. Under H_0, F is distributed as a Snedecor's F distribution with degrees of freedom $n_1 - 1$ and $n_2 - 1$. The F-test is fully parametric. It is implemented in `var.test`.

The grades.txt dataset contains grades of two exams (`e1` and `e2`) in a certain subject. The group of 41 students contains two subgroups in `mf`.

a. Compute `avg`, the variable with the average of `e1` and `e2`. Perform a kda of `avg` for the classes in `mf`. Then, compare the kde's of `e1` and `e2`.
b. Compare the variances of `avg` for each of the subgroups in `mf`. Which group has larger sample variance? Perform a permutation-based F-test to check if the variance of that group is significantly larger. Compare your results with `var.test`. Are they coherent?
c. Compare the variances of `e1` and `e2`. Which exam has larger sample variance? Implement a permutation-based *paired* F-test to assess if there are significant differences in the variances. Do it without relying on `boot::boot`. Compare your results with the unpaired test from b and with `var.test`. Are they coherent?

6.3 Independence tests

Assume that a bivariate iid sample $(X_1, Y_1), \ldots, (X_n, Y_n)$ from an arbitrary distribution $F_{X,Y}$ is given. We next address the **independence problem** of testing the existence of possible dependence between X and Y.

Differently to the previous sections, we will approach this problem through the use of coefficients that capture, up to different extension, the degree of dependence between X and Y.[70,71]

6.3.1 Concordance-based tests

Concordance measures

The random variables X and Y are *concordant* if "large" values of one tend to be associated with "large" values of the other and, analogously, "small" values of one with "small" values of the other. More precisely, given (x_i, y_i) and (x_j, y_j), two observations of (X, Y), we say that:

- (x_i, y_i) and (x_j, y_j) are *concordant* if $x_i < x_j$ and $y_i < y_j$, or if $x_i > x_j$ and $y_i > y_j$.
- (x_i, y_i) and (x_j, y_j) are *discordant* if $x_i > x_j$ and $y_i < y_j$, or if $x_i < x_j$ and $y_i > y_j$.

Alternatively, (x_i, y_i) and (x_j, y_j) are concordant/discordant depending on whether $(x_i - x_j)(y_i - y_j)$ is *positive* or *negative*. Therefore, the concordance concept is similar to the correlation concept, but with a main foundational difference: the former does not use

[70] Approaches to independence testing through the ecdf perspective are also possible (see, e.g., Blum et al. (1961)), since the independence of X and Y happens if and only if the joint cdf $F_{X,Y}$ factorizes as $F_X F_Y$, the product of marginal cdfs. These approaches go along the lines of Sections 6.1.1 and 6.2.

[71] A popular approach to independence and homogeneity testing, not covered in these notes, is the chi-squared test for contingency tables $k \times m$ (see `?chisq.test`). This test is applied to a table that contains the observed frequencies of (X, Y) in km classes $I_i \times J_j$, where $\cup_{i=1}^k I_i$ and $\cup_{j=1}^m J_j$ cover the supports of X and Y, respectively. As with the chi-squared goodness-of-fit test, one can modify the outcome of the test by altering the form of these classes and the choice of (k, m) – a sort of tuning parameter. The choice of (k, m) also affects the quality of the asymptotic null distribution. Therefore, for continuous and discrete random variables, the chi-squared test has significant drawbacks when compared to the ecdf-based tests. However, as opposed to the latter, chi-squared tests can be readily applied to the comparison of categorical variables, where the choice of (k, m) is often more canonical.

the *value* of $(x_i - x_j)(y_i - y_j)$, only its *sign*. Consequently, whether X and Y are concordant or discordant will not be driven by its linear association, as happens with positive/negative correlation.

We say that X and Y are concordant/discordant if the "majority" of the observations of (X, Y) are concordant/discordant. This "majority" is quantified by a certain **concordance measure**. Many concordance measures exist, with different properties and interpretations. Scarsini (1984) gave an axiomatic definition of the properties that any valid concordance measure for continuous random variables must satisfy.

Definition 6.1 (Concordance measure)**.** A measure κ of dependence between two continuous random variables X and Y is a *concordance measure* if it satisfies the following axioms:

i. Domain: $\kappa(X, Y)$ is defined for any (X, Y) with continuous cdf.
ii. Symmetry: $\kappa(X, Y) = \kappa(Y, X)$.
iii. Coherence: $\kappa(X, Y)$ is monotone in $C_{X,Y}$ (the *copula* of (X, Y)).[72] That is, if $C_{X,Y} \geq C_{W,Z}$, then $\kappa(X, Y) \geq \kappa(W, Z)$.[73]
iv. Range: $-1 \leq \kappa(X, Y) \leq 1$.
v. Independence: $\kappa(X, Y) = 0$ if X and Y are independent.
vi. Change of sign: $\kappa(-X, Y) = -\kappa(X, Y)$.
vii. Continuity: if $(X, Y) \sim H$ and $(X_n, Y_n) \sim H_n$, and if H_n converges pointwise to H (H_n and H continuous), then $\lim_{n\to\infty} \kappa(X_n, Y_n) = \kappa(X, Y)$.[74]

[72] The copula $C_{X,Y}$ of a continuous random vector $(X, Y) \sim F_{X,Y}$ is the bivariate function $(u, v) \in [0, 1]^2 \mapsto C_{X,Y}(u, v) = F_{X,Y}(F_X^{-1}(u), F_Y^{-1}(v)) \in [0, 1]$, where F_X and F_Y are the marginal cdfs of X and Y, respectively.

[73] That $C_{X,Y} \geq C_{W,Z}$ means that $C_{X,Y}(u, v) \geq C_{W,Z}(u, v)$ for all $(u, v) \in [0, 1]^2$.

[74] The assumption is slightly stronger than asking that $(X_n, Y_n) \xrightarrow{d} (X, Y)$; recall Definition 1.1.

The next theorem, less technical, gives direct insight into what a concordance measure brings on top of correlation: **characterize if (X, Y) is such that one variable can be "perfectly predicted" from the other**.[75] Perfect prediction can only be achieved if each continuous variable is a *monotone* function g of the other.[76]

[75] It does not suffice if only Y is perfectly predictable from X, such as in the case $Y = X^2$.

[76] So the function can be inverted almost everywhere, as opposed to the function $g(x) = x^2$ from the previous note.

Theorem 6.1. *Let κ be a concordance measure for two continuous random variables X and Y:*

i. If $Y = g(X)$ (almost surely) and g is an increasing function, then $\kappa(X, Y) = 1$.
ii. If $Y = g(X)$ (almost surely) and g is a decreasing function, then $\kappa(X, Y) = -1$.
iii. If α and β are strictly monotonic functions almost everywhere in the supports of X and Y, then $\kappa(\alpha(X), \beta(Y)) = \kappa(X, Y)$.

Remark. Correlation is *not* a concordance measure. For example, it fails to verify the third property of Theorem 6.1: $\text{Cor}[X, Y] = \text{Cor}[\alpha(X), \beta(Y)]$ is true only if α and β are linear functions.

Important for our interests is Axiom v: **independence implies a zero concordance measure**, for any proper concordance measure. Note that **the converse is false**. However, since dependence *often* manifests itself in concordance patterns, testing independence by testing zero concordance is a useful approach with a more informative rejection of H_0.

Kendall's tau and Spearman's rho

The two most famous measures of concordance for (X, Y) are *Kendall's tau* and *Spearman's rho*. Both can be regarded as generalizations of $\text{Cor}[X, Y]$ that are not driven by linear relations; rather, they look into the concordance of (X, Y) and the monotone relation between X and Y.

The **Kendall's tau** of (X, Y), denoted by $\tau(X, Y)$, is defined as

$$\begin{aligned}\tau(X, Y) := &\mathbb{P}[(X - X')(Y - Y') > 0] \\ &- \mathbb{P}[(X - X')(Y - Y') < 0],\end{aligned} \tag{6.29}$$

where (X', Y') is an *iid copy* of (X, Y).[77,78] That is, $\tau(X, Y)$ is the **difference of the concordance and discordance probabilities** between two random observations of (X, Y).

[77] Beware that, here and henceforth, the random variable X' (and not a random vector) is not the transpose of X.

[78] (X', Y') is an iid copy of (X, Y) if $(X', Y') \overset{d}{=} (X, Y)$ and (X', Y') is independent from (X, Y).

The **Spearman's rho** of (X, Y), denoted by $\rho(X, Y)$, is defined as

$$\begin{aligned}\rho(X, Y) := 3\big\{&\mathbb{P}[(X - X')(Y - Y'') > 0] \\ &- \mathbb{P}[(X - X')(Y - Y'') < 0]\big\},\end{aligned} \tag{6.30}$$

where (X', Y') and (X'', Y'') are two iid copies from (X, Y).[79] Recall that (X', Y'') is a random vector that, by construction, has independent components with marginals equal to those of (X, Y). Hence, (X', Y'') **is the "independence version" of** (X, Y). Therefore, $\rho(X, Y)$ is proportional to the difference of the concordance and discordance probabilities between two random observations, one from (X, Y) and another from its independence version.

[79] An equivalent definition is $\rho(X, Y) = 3\big\{\mathbb{P}[(X - X'')(Y - Y') > 0] - \mathbb{P}[(X - X'')(Y - Y') < 0]\big\}$.

A neater interpretation of the Spearman's rho follows from the following other equivalent definition:

$$\rho(X, Y) = \text{Cor}[F_X(X), F_Y(Y)], \tag{6.31}$$

where F_X and F_Y are the marginal cdfs of X and Y, respectively. Expression (6.31) hides the concordance view of $\rho(X, Y)$, but it states that the **Spearman's rho between two random variables is the correlation between their probability transforms**.[80]

[80] Consequently, Spearman's rho coincides with Pearson's linear correlation coefficient for uniform variables.

Exercise 6.19. Obtain (6.31) from (6.30). *Hint*: prove first the equality for $X \sim \mathcal{U}(0, 1)$ and $Y \sim \mathcal{U}(0, 1)$.

Although different, Kendall's tau and Spearman's rho are heavily related. For example, it is impossible to have $\tau(X, Y) = -0.5$ and $\rho(X, Y) = 0.5$ for the same continuous random vector (X, Y). Their relation can be precisely visualized in Figure 6.11, which shows the feasibility region for (τ, ρ) given by Theorem 6.2. The region is the optimal possible and cannot be narrowed.

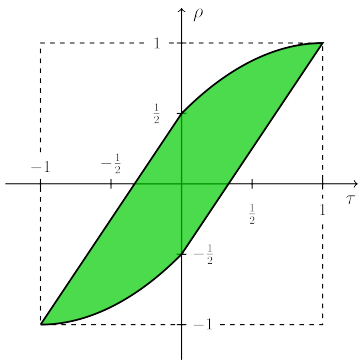


Figure 6.11: Feasible region for (τ, ρ) given by Theorem 6.2.

Theorem 6.2. *Let X and Y be two continuous random variables with $\tau \equiv \tau(X, Y)$ and $\rho \equiv \rho(X, Y)$. Then,*

$$\begin{aligned}\frac{3\tau - 1}{2} \leq \rho \leq \frac{1 + 2\tau - \tau^2}{2}, &\qquad \tau \geq 0, \\ \frac{\tau^2 + 2\tau - 1}{2} \leq \rho \leq \frac{1 + 3\tau}{2}, &\qquad \tau \leq 0.\end{aligned}$$

Kendall's tau and Spearman's rho tests of no concordance

- *Tests purpose*. Given $(X_1, Y_1), \ldots, (X_n, Y_n) \sim F_{X,Y}$, test $H_0 : \tau = 0$ vs. $H_1 : \tau \neq 0$, and $H_0 : \rho = 0$ vs. $H_1 : \rho \neq 0$.[81]

[81] One-sided versions of the tests replace the alternatives by $H_1 : \tau > 0$ and $H_1 : \rho > 0$, or by $H_1 : \tau < 0$ and $H_1 : \rho < 0$.

- *Statistics definition*. The tests are based on the sample versions of (6.31) and (6.31):

$$\hat{\tau} := \frac{c - d}{\binom{n}{2}} = \frac{2(c-d)}{n(n-1)}, \tag{6.32}$$

$$\hat{\rho} := \frac{\sum_{i=1}^n (R_i - \bar{R})(S_i - \bar{S})}{\sqrt{\sum_{i=1}^n (R_i - \bar{R})^2 \sum_{i=1}^n (S_i - \bar{S})^2}}, \tag{6.33}$$

  where $c = \sum_{\substack{i,j=1 \\ i<j}}^n 1_{\{(X_i - X_j)(Y_i - Y_j) > 0\}}$ denotes the number of concordant pairs in the sample, $d = \binom{n}{2} - c$ is the number of discordant pairs, and $R_i := \text{rank}_X(X_i) = nF_{X,n}(X_i)$ and $S_i := \text{rank}_Y(Y_i) = nF_{Y,n}(Y_i)$, $i = 1, \ldots, n$ are the ranks of the samples of X and Y.

  Large positive (negative) values of $\hat{\tau}$ and $\hat{\rho}$ indicate presence of concordance (discordance) between X and Y. The two-sided tests reject H_0 for large absolute values of $\hat{\tau}$ and $\hat{\rho}$.

- *Statistics computation*. Formulae (6.32) and (6.33) are reasonably explicit, but the latter can be improved to

$$\hat{\rho} = 1 - \frac{6}{n(n^2-1)} \sum_{i=1}^n (R_i - S_i)^2.$$

  The naive computation of $\hat{\tau}$ involves $O(n^2)$ operations, hence it escalates to large sample sizes worse than $\hat{\rho}$. A faster implementation of $O(n \log(n))$ is available through `pcaPP::cor.fk`.

- *Distributions under H_0*. The asymptotic null distributions of $\hat{\tau}$ and $\hat{\rho}$ follow from

$$\sqrt{\frac{9n(n-1)}{2(2n+5)}}\hat{\tau} \xrightarrow{d} \mathcal{N}(0,1), \quad \sqrt{n-1}\hat{\rho} \xrightarrow{d} \mathcal{N}(0,1)$$

  under H_0. Exact distributions are available for small n's.

- *Highlights and caveats*. Both $\tau = 0$ and $\rho = 0$ **do not characterize independence**. Therefore, absence of rejection does not guarantee lack of evidence in favor of dependence. Rejection of H_0 does guarantee the existence of significant dependence, in the form of concordance (positive values of $\hat{\tau}$ and $\hat{\rho}$) or discordance (negative values). Both tests are distribution-free.

- *Implementation in R*. The test statistics $\hat{\tau}$ and $\hat{\rho}$ and the exact/asymptotic p-values are available through the `cor.test` function, specifying `method = "kendall"` or `method = "spearman"`. The default `alternative = "two.sided"` carries out the two-sided tests for $H_1 : \tau \neq 0$ and $H_1 : \rho \neq 0$. The one-sided alternatives follows with `alternative = "greater"` (> 0) and `alternative = "less"` (< 0).

The following chunk of code illustrates the use of `cor.test`. It also gives some examples of the advantages of $\hat{\tau}$ and $\hat{\rho}$ over the correlation coefficient.

```
# Outliers fool correlation, but do not fool concordance
set.seed(123456)
n <- 200
x <- rnorm(n)
y <- rnorm(n)
y[n] <- x[n] <- 1e4
cor.test(x, y, method = "pearson") # Close to 1 due to outlier
##
##  Pearson's product-moment correlation
##
## data:  x and y
## t = 6731, df = 198, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.9999971 0.9999983
## sample estimates:
##       cor
## 0.9999978
cor.test(x, y, method = "kendall") # Close to 0, as it should
##
##  Kendall's rank correlation tau
##
## data:  x and y
## z = -0.89189, p-value = 0.3725
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##         tau
## -0.04241206
cor.test(x, y, method = "spearman") # Close to 0, as it should
##
##  Spearman's rank correlation rho
##
## data:  x and y
## S = 1419086, p-value = 0.3651
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##         rho
## -0.06434111
cor(rank(x), rank(y)) # Spearman's rho
## [1] -0.06434111

# Outliers fool the sign of correlation, but do not fool concordance
x <- rnorm(n)
y <- x
x[n] <- 1e3
y[n] <- -1e4
cor.test(x, y, method = "pearson", alternative = "greater") # Change of sign!
##
##  Pearson's product-moment correlation
##
## data:  x and y
## t = -958.47, df = 198, p-value = 1
## alternative hypothesis: true correlation is greater than 0
## 95 percent confidence interval:
##  -0.9999148  1.0000000
## sample estimates:
##        cor
## -0.9998923
cor.test(x, y, method = "kendall", alternative = "greater") # Fine
##
##  Kendall's rank correlation tau
##
## data:  x and y
## z = 20.608, p-value < 2.2e-16
```

```
## alternative hypothesis: true tau is greater than 0
## sample estimates:
##  tau
## 0.98
cor.test(x, y, method = "spearman", alternative = "greater") # Fine
##
##  Spearman's rank correlation rho
##
## data:  x and y
## S = 39800, p-value < 2.2e-16
## alternative hypothesis: true rho is greater than 0
## sample estimates:
##       rho
## 0.9701493
```

However, as previously said, concordance measures do not characterize independence. Below are some dependence situations that are undetected by $\hat{\tau}$ and $\hat{\rho}$.

```
# Non-monotone dependence fools concordance
set.seed(123456)
n <- 200
x <- rnorm(n)
y <- abs(x) + rnorm(n)
cor.test(x, y, method = "kendall")
##
##  Kendall's rank correlation tau
##
## data:  x and y
## z = -1.5344, p-value = 0.1249
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##         tau
## -0.07296482
cor.test(x, y, method = "spearman")
##
##  Spearman's rank correlation rho
##
## data:  x and y
## S = 1473548, p-value = 0.1381
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##        rho
## -0.1051886

# Dependence in terms of conditional variance fools concordance
x <- rnorm(n)
y <- rnorm(n, sd = abs(x))
cor.test(x, y, method = "kendall")
##
##  Kendall's rank correlation tau
##
## data:  x and y
## z = 1.0271, p-value = 0.3044
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##        tau
## 0.04884422
cor.test(x, y, method = "spearman")
##
##  Spearman's rank correlation rho
##
## data:  x and y
## S = 1251138, p-value = 0.3857
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##        rho
## 0.06162304
```

6.3.2 *Distance correlation tests*

A characterization of independence

No-concordance tests based on $\hat{\tau}$ and $\hat{\rho}$ are easy to interpret and carry out, but they are not *omnibus* for testing independence: they do not detect all kinds of dependence between X and Y, only those expressible with $\tau \neq 0$ and $\rho \neq 0$. Therefore, and as evidenced previously, Kendall's tau and Spearman's rho may not detect dependence patterns in the form of non-monotone relations or heteroskedasticity.

The handicap of concordance measures on not being able to detect all kinds of dependence has been recently solved by Székely et al. (2007) with the introduction of a new type of correlation measure that **completely characterizes independence**.[82] This new type of correlation is not a correlation between *values* of (X, Y). Rather, it is related to the correlation between *pairwise distances* of X and Y. The following definitions, adapted from Section 7.1 in Székely and Rizzo (2017), give the precise form of *distance covariance* and *distance variance* between two random variables X and Y, which are the pillars of a *distance correlation*.

[82] This new type of correlation belongs to a wide class of new statistics that are referred to as *energy statistics*; see the reviews by Rizzo and Székely (2016) (lighter introduction with R examples), Székely and Rizzo (2013), and Székely and Rizzo (2017) (the latter two being deeper reviews). Energy statistics generate tests alternative to the classical approaches to test goodness-of-fit, normality, homogeneity, independence, and other hypotheses.

Definition 6.2 (Distance covariance and variance)**.** The *distance covariance* (dCov) between two random variables X and Y with $\mathbb{E}[|X|] < \infty$ and $\mathbb{E}[|Y|] < \infty$ is the non-negative number $\mathcal{V}(X, Y)$ with square equal to

$$\begin{aligned}\mathcal{V}^2(X,Y) :=& \mathbb{E}\left[|X-X'||Y-Y'|\right] + \mathbb{E}\left[|X-X'|\right]\mathbb{E}\left[|Y-Y'|\right] \\ &- 2\mathbb{E}\left[|X-X'||Y-Y''|\right] \qquad (6.34)\\ =& \operatorname{Cov}\left[|X-X'|,|Y-Y'|\right] \\ &- 2\operatorname{Cov}\left[|X-X'|,|Y-Y''|\right], \qquad (6.35)\end{aligned}$$

where (X', Y') and (X'', Y'') are iid copies of (X, Y). The *distance variance* (dVar) of X is defined as

$$\begin{aligned}\mathcal{V}^2(X) :=& \mathcal{V}^2(X,X) \\ =& \mathbb{V}\mathrm{ar}\left[|X-X'|\right] - 2\operatorname{Cov}\left[|X-X'|,|X-X''|\right]. \qquad (6.36)\end{aligned}$$

Some interesting points implicit in Definition 6.2 are:

- **dCov is unsigned**: dCov does not inform on the "direction" of the dependence between X and Y.
- From (6.34), it is clear that $\mathcal{V}(X, Y) = 0$ if X and Y are independent, as in that case $\mathbb{E}\left[|X-X'||Y-Y'|\right] = \mathbb{E}\left[|X-X'|\right]\mathbb{E}\left[|Y-Y'|\right] = \mathbb{E}\left[|X-X'||Y-Y''|\right]$.
- Equation (6.34) is somehow reminiscent of Spearman's rho (see (6.30)) and uses (X', Y''), the independence version of (X, Y).
- As (6.35) points out, dCov is *not* the covariance between pairwise distances of X and Y, $\operatorname{Cov}\left[|X-X'|,|Y-Y'|\right]$. However, $\mathcal{V}^2(X, Y)$ is *related to* such covariances: it can be regarded as a covariance

between pairwise distances that is **"centered" by the covariances of the "independent versions" of the pairwise distances**:

$$\begin{aligned}\mathcal{V}^2(X,Y) = &\operatorname{Cov}\left[|X-X'|,|Y-Y'|\right] \\ &- \operatorname{Cov}\left[|X-X'|,|Y-Y''|\right] - \operatorname{Cov}\left[|X-X''|,|Y-Y'|\right].\end{aligned}$$

Distance correlation is defined analogously to how correlation is defined from covariance and variance.

Definition 6.3 (Distance correlation)**.** The *distance correlation* (dCor) between two random variables X and Y with $\mathbb{E}[|X|] < \infty$ and $\mathbb{E}[|Y|] < \infty$ is the non-negative number $\mathcal{R}(X,Y)$ with square equal to

$$\mathcal{R}^2(X,Y) := \frac{\mathcal{V}^2(X,Y)}{\sqrt{\mathcal{V}^2(X)\mathcal{V}^2(Y)}} \tag{6.37}$$

if $\mathcal{V}^2(X)\mathcal{V}^2(Y) > 0$. If $\mathcal{V}^2(X)\mathcal{V}^2(Y) = 0$, then $\mathcal{R}^2(X,Y) := 0$.

The following theorem collects useful properties of dCov, dVar, and dCor. Some of them are related to the properties of standard covariance (recall, e.g., (1.3)) and (unsigned) correlation. However, as anticipated, Property *iv* gives the clear edge of dCor over concordance measures on completely characterizing independence.

Theorem 6.3 (Properties of dCov, dVar, and dCor)**.** *Let X and Y be two random variables with $\mathbb{E}[|X|] < \infty$ and $\mathbb{E}[|Y|] < \infty$. Then:*

i. $\mathcal{V}(a+bX, c+dY) = \sqrt{|bd|}\mathcal{V}(X,Y)$ *for all* $a,b,c,d \in \mathbb{R}$.
ii. $\mathcal{V}(a+bX) = |b|\mathcal{V}(X)$ *for all* $a,b \in \mathbb{R}$.
iii. If $\mathcal{V}(X) = 0$, *then* $X = \mathbb{E}[X]$ *almost surely.*
iv. $\mathcal{R}(X,Y) = \mathcal{V}(X,Y) = 0$ *if and only if X and Y are independent.*
v. $0 \leq \mathcal{R}(X,Y) \leq 1$.
vi. If $\mathcal{R}(X,Y) = 1$, *then there exist* $a,b \in \mathbb{R}$, $b \neq 0$, *such that* $Y = a+bX$.
vii. If (X,Y) is normally distributed and is such that $\operatorname{Cor}[X,Y] = \rho$, $-1 \leq \rho \leq 1$, *then*

$$\mathcal{R}^2(X,Y) = \frac{\rho\sin^{-1}(\rho) + \sqrt{1-\rho^2} - \rho\sin^{-1}(\rho/2) - \sqrt{4-\rho^2} + 1}{1+\pi/3-\sqrt{3}}$$

and $\mathcal{R}(X,Y) \leq |\rho|$.

Two important simplifications have been done on the previous definitions and theorem for the sake of a simplified exposition. Precisely, dCov admits a more general definition that addresses two generalizations, the first one being especially relevant for practical purposes:

1. **Multivariate extensions**. dCov, dVar, and dCor can be straightforwardly extended to account for multivariate vectors $\mathbf{X}$ and $\mathbf{Y}$ supported on $\mathbb{R}^p$ and $\mathbb{R}^q$. It is as simple as replacing the absolute values $|\cdot|$ in Definition 6.2 with the Euclidean norms $\|\cdot\|_p$ and

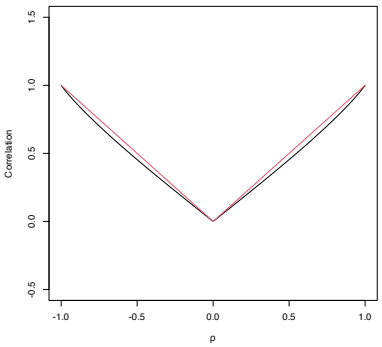


Figure 6.12: Mapping $\rho \mapsto \mathcal{R}(X(\rho),Y(\rho))$ (in black) for the normally distributed vector $(X(\rho),Y(\rho))$ such that $\operatorname{Cor}[X(\rho),Y(\rho)] = \rho$. In red, the absolute value function $\rho \mapsto |\rho|$.

$\|\cdot\|_q$ for $\mathbb{R}^p$ and $\mathbb{R}^q$, respectively. The properties stated in Theorem 6.3 hold once suitably adapted. For example, for $\mathbf{X}$ and $\mathbf{Y}$ such that $\mathbb{E}[\|\mathbf{X}\|_p] < \infty$ and $\mathbb{E}[\|\mathbf{Y}\|_q] < \infty$, dCov is defined as the square root of

$$\begin{aligned}\mathcal{V}^2(\mathbf{X},\mathbf{Y}) := &\operatorname{Cov}\left[\|\mathbf{X}-\mathbf{X}'\|_p, \|\mathbf{Y}-\mathbf{Y}'\|_q\right] \\ &- 2\operatorname{Cov}\left[\|\mathbf{X}-\mathbf{X}'\|_p, \|\mathbf{Y}-\mathbf{Y}''\|_q\right]\end{aligned}$$

and dVar as $\mathcal{V}^2(\mathbf{X}) = \mathcal{V}^2(\mathbf{X},\mathbf{X})$. Recall that both dCov and dVar are *scalars*, whereas $\mathbb{V}\mathrm{ar}[\mathbf{X}]$ is a $p \times p$ matrix. Consequently, the dCor of $\mathbf{X}$ and $\mathbf{Y}$ is also a scalar that condenses all the dependencies between $\mathbf{X}$ and $\mathbf{Y}$.[83]

[83] In comparison, there are pq (linear) correlations between the entries of $\mathbf{X}$ and $\mathbf{Y}$.

2. **Characteristic-function view of dCov**. dCov does not require $\mathbb{E}[|X|] < \infty$ and $\mathbb{E}[|Y|] < \infty$ to be defined. It can be defined, without resorting to expectations, through the *characteristic functions*[84] of X, Y, and (X,Y):

$$\hat{f}_X(s) := \mathbb{E}[e^{\mathrm{i}sX}], \;\; \hat{f}_Y(t) := \mathbb{E}[e^{\mathrm{i}tY}], \;\; \hat{f}_{X,Y}(s,t) := \mathbb{E}[e^{\mathrm{i}(sX+tY)}],$$

where $s,t \in \mathbb{R}$. Indeed,

$$\mathcal{V}^2(X,Y) = \frac{1}{c_1^2}\int_{\mathbb{R}^2} \left|\hat{f}_{X,Y}(s,t) - \hat{f}_X(s)\hat{f}_Y(t)\right|^2 \frac{\mathrm{d}t\,\mathrm{d}s}{|t|^2|s|^2}, \tag{6.38}$$

where $c_p := \pi^{(p+1)/2}/\Gamma((p+1)/2)$ and $|\cdot|$ represents the modulus of a complex number. The alternative definition (6.38) shows that $\mathcal{V}^2(X,Y)$ can be regarded as a weighted squared distance between the joint and marginal characteristic functions.[85] This alternative view also holds for the multivariate case, where (6.38) becomes[86]

$$\mathcal{V}^2(\mathbf{X},\mathbf{Y}) = \frac{1}{c_p c_q}\int_{\mathbb{R}^p}\int_{\mathbb{R}^q} \left|\hat{f}_{\mathbf{X},\mathbf{Y}}(\mathbf{s},\mathbf{t}) - \hat{f}_{\mathbf{X}}(\mathbf{s})\hat{f}_{\mathbf{Y}}(\mathbf{t})\right|^2 \frac{\mathrm{d}\mathbf{t}\,\mathrm{d}\mathbf{s}}{\|\mathbf{t}\|_q^{q+1}\|\mathbf{s}\|_p^{p+1}}.$$

[84] Recall that the characteristic function is a complex-valued function that offers an alternative to the cdf for characterizing the behavior of *any* random variable.

[85] Two random variables X and Y are independent if and only if $\hat{f}_{X,Y}(s,t) = \hat{f}_X(s)\hat{f}_Y(t)$, for all $s,t \in \mathbb{R}$.

[86] The characteristic function of a random vector $\mathbf{X}$ is the function $\mathbf{s} \in \mathbb{R}^p \mapsto \hat{f}_{\mathbf{X}}(\mathbf{s}) := \mathbb{E}[e^{\mathrm{i}\mathbf{s}'\mathbf{X}}] \in \mathbb{C}$.

Distance correlation test

Independence tests for (X,Y) can be constructed by evaluating if $\mathcal{R}(X,Y)$ or $\mathcal{V}(X,Y) = 0$ holds. To do so, we first need to consider the empirical versions of dCov and dCor. These follow from (6.34) and (6.37).

Definition 6.4 (Empirical distance covariance and variance)**.** The dCov for the random sample $(X_1,Y_1),\ldots,(X_n,Y_n)$ is the nonnegative number $\mathcal{V}_n(X,Y)$ with square equal to

$$\mathcal{V}_n^2(X,Y) := \frac{1}{n^2}\sum_{k,\ell=1}^n A_{k\ell}B_{k\ell}, \tag{6.39}$$

where

$$\begin{aligned}A_{k\ell} &:= a_{k\ell} - \bar{a}_{k\bullet} - \bar{a}_{\bullet\ell} + \bar{a}_{\bullet\bullet}, \quad a_{k\ell} := |X_k - X_\ell|, \\ \bar{a}_{k\bullet} &:= \frac{1}{n}\sum_{l=1}^n a_{k\ell}, \quad \bar{a}_{\bullet\ell} := \frac{1}{n}\sum_{k=1}^n a_{k\ell}, \quad \bar{a}_{\bullet\bullet} := \frac{1}{n^2}\sum_{k,\ell=1}^n a_{k\ell},\end{aligned}$$

and $B_{k\ell} := b_{k\ell} - \bar{b}_{k\bullet} - \bar{b}_{\bullet\ell} + \bar{b}_{\bullet\bullet}$ is defined analogously for $b_{k\ell} := |Y_k - Y_\ell|$. The dVar of $X_1,\ldots,X_n$ is defined as $\mathcal{V}_n^2(X) := \frac{1}{n^2}\sum_{k,\ell=1}^n A_{k\ell}^2$.

Definition 6.5 (Empirical distance correlation)**.** The dCor for the random sample $(X_1, Y_1), \ldots, (X_n, Y_n)$ is the non-negative number $\mathcal{R}_n(X, Y)$ with square equal to

$$\mathcal{R}_n^2(X, Y) := \frac{\mathcal{V}_n^2(X, Y)}{\sqrt{\mathcal{V}_n^2(X)\mathcal{V}_n^2(Y)}} \tag{6.40}$$

if $\mathcal{V}_n^2(X)\mathcal{V}_n^2(Y) > 0$. If $\mathcal{V}_n^2(X)\mathcal{V}_n^2(Y) = 0$, then $\mathcal{R}_n^2(X, Y) := 0$.

Remark. The previous empirical versions of dCov, dVar, and dCor also allow for multivariate versions. These are based on $a_{k\ell} := \|\mathbf{X}_k - \mathbf{X}_\ell\|_p$ and $b_{k\ell} := \|\mathbf{Y}_k - \mathbf{Y}_\ell\|_q$.

We next introduce the independence tests based on (6.39)–(6.40).

- *Test purpose*. Given $(X_1, Y_1), \ldots, (X_n, Y_n) \sim F_{X,Y}$, test $H_0 : F_{X,Y} = F_X F_Y$ vs. $H_1 : F_{X,Y} \neq F_X F_Y$ consistently against all the alternatives in H_1.
- *Statistic definition*. The dCov test rejects H_0 for large values of the test statistic $n\mathcal{V}_n^2(X, Y)$ (not the dCov!), which indicates a departure from independence. The dCor test uses as test statistic $\mathcal{R}_n^2(X, Y)$ and rejects H_0 for large values of it.
- *Statistic computation*. Formulae (6.39) and (6.40) are directly implementable, yet they involve $O(n^2)$ computations.[87]
- *Distribution under H_0*. If H_0 holds, then the asymptotic cdf of $n\mathcal{V}_n^2(X, Y)$ is the cdf of the random variable

$$\sum_{j=1}^{\infty} \lambda_j Y_j, \quad \text{where } Y_j \sim \chi_1^2,\ j \geq 1, \text{ are iid} \tag{6.41}$$

  and $\{\lambda_j\}_{j=1}^{\infty}$ are certain non-negative constants that depend on $F_{X,Y}$. As with (6.8), the cdf of (6.41) does not admit a simple analytical expression. In addition, it is mostly unusable in practice, since $\{\lambda_j\}_{j=1}^{\infty}$ are unknown. The asymptotic distribution for $\mathcal{R}_n^2(X, Y)$ is related to that of $n\mathcal{V}_n^2(X, Y)$. However, both test statistics can be calibrated by permutations (see Section 6.3.3).
- *Highlights and caveats*. On the positive side, the dCov/dCor test is **much more general** than concordance-based tests: (*i*) it is an **omnibus** test for independence; and (*ii*) it **can be applied to multivariate data**. On the negative side, the dCov/dCor test is *not* distribution-free, so fast asymptotic p-values are not available. Besides, the rejection of the dCov/dCor tests does not inform on the *kind* of dependence between X and Y. The dCov and dCor tests perform similarly and their computational burdens are roughly equivalent. The dCor test offers the edge of having an absolute scale for its test statistic, since $\mathcal{R}_n^2(X, Y) \in [0, 1]$.
- *Implementation in R*. The `energy` package implements both tests through the `energy::dcov.test` and `energy::dcor.test` functions, which compute the test statistics and give p-values approximated by permutations (see Section 6.3.3).

[87] A related bias-corrected estimator of $\mathcal{V}^2(X, Y)$ has been seen to involve $O(n \log n)$ computations (see the discussion in Section 7.2 in Székely and Rizzo (2017)).

The use of `energy::dcov.test` and `energy::dcor.test` for testing the independence between X and Y is shown in the following chunk of code.

```
# Distance correlation detects non-monotone dependence
set.seed(123456)
n <- 200
x <- rnorm(n)
y <- abs(x) + rnorm(n)

# Distance covariance and correlation tests. R is the number of permutations
# and needs to be specified (the default is R = 0 -- no test is produced)
energy::dcov.test(x, y, R = 1e3)
##
##  dCov independence test (permutation test)
##
## data:  index 1, replicates 1000
## nV^2 = 6.4595, p-value = 0.001998
## sample estimates:
##      dCov
## 0.1797145
energy::dcor.test(x, y, R = 1e3)
##
##  dCor independence test (permutation test)
##
## data:  index 1, replicates 1000
## dCor = 0.26188, p-value = 0.000999
## sample estimates:
##      dCov      dCor   dVar(X)   dVar(Y)
## 0.1797145 0.2618840 0.6396844 0.7361771

# Distance correlation detects conditional variance as dependence
x <- rnorm(n)
y <- rnorm(n, sd = abs(x))

# Distance covariance and correlation tests
energy::dcov.test(x, y, R = 1e3)
##
##  dCov independence test (permutation test)
##
## data:  index 1, replicates 1000
## nV^2 = 3.5607, p-value = 0.001998
## sample estimates:
##      dCov
## 0.1334296
energy::dcor.test(x, y, R = 1e3)
##
##  dCor independence test (permutation test)
##
## data:  index 1, replicates 1000
## dCor = 0.25031, p-value = 0.004995
## sample estimates:
##      dCov      dCor   dVar(X)   dVar(Y)
## 0.1334296 0.2503077 0.6152985 0.4618172
```

Testing the independence of two random vectors $\mathbf{X}$ and $\mathbf{Y}$ is also straightforward, as the code below illustrates.

```
# A multivariate case with independence
set.seed(123456)
n <- 200
p <- 5
x <- matrix(rnorm(n * p), nrow = n, ncol = p)
y <- matrix(rnorm(n * p), nrow = n, ncol = p)
energy::dcov.test(x, y, R = 1e3)
##
##  dCov independence test (permutation test)
```

```
##
## data:  index 1, replicates 1000
## nV^2 = 8.5627, p-value = 0.7493
## sample estimates:
##      dCov
## 0.2069139
energy::dcor.test(x, y, R = 1e3)
##
##  dCor independence test (permutation test)
##
## data:  index 1, replicates 1000
## dCor = 0.28323, p-value = 0.7732
## sample estimates:
##      dCov      dCor   dVar(X)   dVar(Y)
## 0.2069139 0.2832256 0.7163536 0.7450528

# A multivariate case with dependence
y <- matrix(0.2 * rnorm(n = n * p, mean = c(x)) +
              rnorm(n * p, sd = 1.25), nrow = n, ncol = p)
energy::dcov.test(x, y, R = 1e3)
##
##  dCov independence test (permutation test)
##
## data:  index 1, replicates 1000
## nV^2 = 14.592, p-value = 0.01099
## sample estimates:
##      dCov
## 0.2701076
energy::dcor.test(x, y, R = 1e3)
##
##  dCor independence test (permutation test)
##
## data:  index 1, replicates 1000
## dCor = 0.32299, p-value = 0.007992
## sample estimates:
##      dCov      dCor   dVar(X)   dVar(Y)
## 0.2701076 0.3229851 0.7163536 0.9762951
```

Exercise 6.20. Illustrate with a simulation study the performance of the independence tests for (X, Y) based on Spearman's rho and distance covariance. To that end, use four simulation scenarios and build a 4×3 plot such that:

- The first column contains illustrative scatterplots of the generated data.
- The second row shows a histogram of the p-values of Spearman's rho test in that scenario.
- The third row is analogous to the second, but for the distance covariance test.

For the four scenarios, consider: (1) independence; (2) a nonlinear scenario for which Spearman's rho is optimal; (3) a scenario for which both tests perform similarly; (4) an undetectable alternative by Spearman's rho. Choose the scenarios originally and in such a way the histogram of the p-values are not completely extreme. Use $M = 1,000$ Monte Carlo replicates and set n at your convenience. Explain the obtained results.

6.3.3 *Permutation-based approach to testing independence*

The calibration of a test of independence can be approached by permutations, in an analogous way as that carried out in Section 6.2.3 for tests of homogeneity. None of the permutation strategies previously seen is adequate, though, since the independence problem, expressed as

$$H_0 : F_{X,Y} = F_X F_Y \quad \text{vs.} \quad H_1 : F_{X,Y} \neq F_X F_Y,$$

is substantially different from (6.13). Here, $F_{X,Y}$ represents the joint cdf of (X, Y), F_X and F_Y stand for the corresponding marginal cdfs, and "$F_{X,Y} \neq F_X F_Y$" denotes that there exists at least one $(x, y) \in \mathbb{R}^2$ such that $F_{X,Y}(x, y) \neq F_X(x)F_Y(y)$.

In the unpaired homogeneity case, the *elements* of the samples $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$ are randomly exchanged using the pooled sample. In the paired case, the *components of each pair* $(X_1, Y_1), \ldots, (X_n, Y_n)$ are randomly exchanged. When testing for **independence**, **each of the two sample components** of $(X_1, Y_1), \ldots, (X_n, Y_n)$ are ***separately* and randomly exchanged**. That the resampling is conducted separately for each component is a direct consequence of the null hypothesis of independence.

Precisely, for *any* two n-permutations σ_1 and σ_2, under independence, it happens that

$$T_n((X_{\sigma_1(1)}, Y_{\sigma_2(1)}), \ldots, (X_{\sigma_1(n)}, Y_{\sigma_2(n)})) \overset{d}{=} T_n((X_1, Y_1), \ldots, (X_n, Y_n))$$

for any proper independence test statistic T_n. Furthermore, it is reasonably evident that

$$T_n((X_{\sigma_1(1)}, Y_{\sigma_2(1)}), \ldots, (X_{\sigma_1(n)}, Y_{\sigma_2(n)})) = T_n((X_1, Y_{\sigma_3(1)}), \ldots, (X_n, Y_{\sigma_3(n)}))$$

for $\sigma_3(i) := \sigma_2(\sigma_1^{-1}(i))$, $i \in \{1, \ldots, n\}$.[88] That is, **any "double permutation" amounts to "single permutation"** and it suffices to permute the second component of the sample (leaving the first component fixed) to attain any of the possible values of $T_n((X_{\sigma_1(1)}, Y_{\sigma_2(1)}), \ldots, (X_{\sigma_1(n)}, Y_{\sigma_2(n)}))$. From the previous arguments, it follows that

$$\mathbb{P}[T_n \leq x] = \frac{1}{n!} \sum_{k=1}^{n!} \mathbb{P}[T_n^{\sigma_k} \leq x] \approx \frac{1}{B} \sum_{b=1}^{B} 1_{\left\{T_n^{\hat{\sigma}_b} \leq x\right\}},$$

where $T_n^{\sigma} \equiv T_n((X_1, Y_{\sigma(1)}), \ldots, (X_n, Y_{\sigma(n)}))$ and $\hat{\sigma}_b$, $b = 1, \ldots, B$, denote the B randomly-chosen n-permutations.

The whole **permutation-based procedure** for performing an independence test that rejects H_0 for large values of the test statistic is summarized below:

1. Compute $T_n \equiv T_n((X_1, Y_1), \ldots, (X_n, Y_n))$.
2. Enter the "permutation world". For $b = 1, \ldots, B$:
   i. Simulate a randomly-permuted sample $Y_1^{*b}, \ldots, Y_n^{*b}$ from $\{Y_1, \ldots, Y_n\}$.[89]

[88] Intuitively: once you have the two components permuted, sort the observations in such a way that the first component matches the original sample. Sorting the sample must not affect any proper independence test statistic (we are in the iid case).

[89] This sampling is done by extracting, *without replacement*, random elements from $\{Y_1, \ldots, Y_n\}$.

ii. Compute $T_n^{*b} \equiv T_n((X_1, Y_1^{*b}), \ldots, (X_n, Y_n^{*b}))$.

3. Obtain the p-value approximation

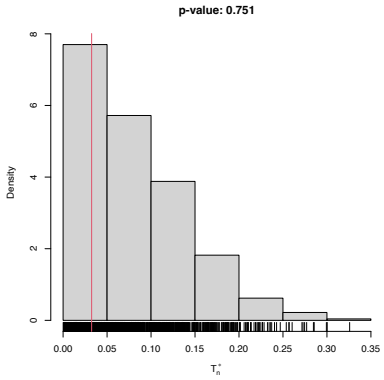
$$p\text{-value} \approx \frac{1}{B} \sum_{b=1}^{B} 1_{\{T_n^{*b} > T_n\}}$$

and emit a test decision from it. Modify it accordingly if rejection of H_0 does not happen for large values of T_n.

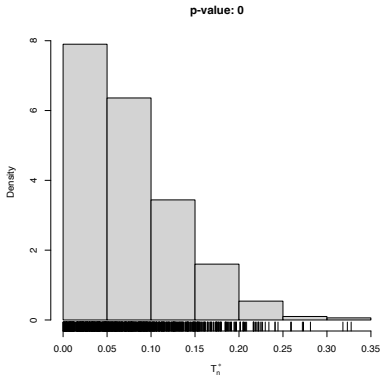
The following chunk of code provides a template for implementing the previous permutation algorithm.

```
# A no-association test using the absolute value of Spearman's rho statistic
perm_ind_test <- function(x, B = 1e3, plot_boot = TRUE) {

  # Test statistic function. Requires TWO arguments, one being the original
  # data (X_1, Y_1), ..., (X_n, Y_n) and the other containing the random
  # index for permuting the second component of sample
  Tn <- function(data, perm_index) {

    # Permute sample by perm_index -- only permute the second component
    data[, 2] <- data[, 2][perm_index]

    # Test statistic -- MODIFY DEPENDING ON THE PROBLEM
    abs(cor(x = data[, 1], y = data[, 2], method = "spearman"))

  }

  # Perform permutation resampling with the aid of boot::boot
  Tn_star <- boot::boot(data = x, statistic = Tn, sim = "permutation", R = B)

  # Test information -- MODIFY DEPENDING ON THE PROBLEM
  method <- "Permutation-based Spearman's rho test of no concordance"
  alternative <- "Spearman's rho is not zero"

  # p-value: modify if rejection does not happen for large values of the
  # test statistic
  pvalue <- mean(Tn_star$t > Tn_star$t0)

  # Construct an "htest" result
  result <- list(statistic = c("stat" = Tn_star$t0), p.value = pvalue,
                 statistic_perm = drop(Tn_star$t),
                 B = B, alternative = alternative, method = method,
                 data.name = deparse(substitute(x)))
  class(result) <- "htest"

  # Plot the position of the original statistic with respect to the
  # permutation replicates?
  if (plot_boot) {

    hist(result$statistic_perm, probability = TRUE,
         main = paste("p-value:", result$p.value),
         xlab = latex2exp::TeX("$T_n^*$"))
    rug(result$statistic_perm)
    abline(v = result$statistic, col = 2)

  }

  # Return "htest"
  return(result)

}

# Check the test for H0 true
set.seed(123456)
```

```
x0 <- mvtnorm::rmvnorm(n = 100, mean = c(0, 0), sigma = diag(c(1:2)))
ind0 <- perm_ind_test(x = x0, B = 1e3)
```




```
ind0
##
##  Permutation-based Spearman's rho test of no concordance
##
## data:  x0
## stat = 0.032439, p-value = 0.751
## alternative hypothesis: Spearman's rho is not zero

# Check the test for H0 false
x1 <- mvtnorm::rmvnorm(n = 100, mean = c(0, 0), sigma = toeplitz(c(1, 0.5)))
ind1 <- perm_ind_test(x = x1, B = 1e3)
```




```
ind1
##
##  Permutation-based Spearman's rho test of no concordance
##
## data:  x1
## stat = 0.52798, p-value < 2.2e-16
## alternative hypothesis: Spearman's rho is not zero
```

A
Confidence intervals for the density function

Obtaining a Confidence Interval (CI) for $f(x)$ is a challenging task. Thanks to the bias results[1] seen in Section 2.3, we know that the kde is biased for finite sample sizes and it is only *asymptotically* unbiased when $h \to 0$. This bias is called the *smoothing bias* and, in essence, complicates the obtention of CIs for $f(x)$, but not for $(K_h * f)(x)$. Let's see the differences between these two objects with an illustrative example.

[1] For example, $\mathbb{E}[\hat{f}(x;h)] = (K_h * f)(x)$.

Some well-known facts for normal densities (see Appendix C in Wand and Jones (1995)) are:

$$(\phi_{\sigma_1}(\cdot - \mu_1) * \phi_{\sigma_2}(\cdot - \mu_2))(x) = \phi_{(\sigma_1^2+\sigma_2^2)^{1/2}}(x - \mu_1 - \mu_2), \tag{A.1}$$

$$\int \phi_{\sigma_1}(x - \mu_1)\phi_{\sigma_2}(x - \mu_2)\,\mathrm{d}x = \phi_{(\sigma_1^2+\sigma_2^2)^{1/2}}(\mu_1 - \mu_2), \tag{A.2}$$

$$\phi_\sigma(x - \mu)^r = \frac{1}{\sigma^{r-1}}(2\pi)^{(1-r)/2}\phi_{\sigma/r^{1/2}}(x - \mu)\frac{1}{r^{1/2}}. \tag{A.3}$$

Consequently, if $K = \phi$ (i.e., $K_h = \phi_h$) and $f(\cdot) = \phi_\sigma(\cdot - \mu)$:

$$(K_h * f)(x) = \phi_{(h^2+\sigma^2)^{1/2}}(x - \mu), \tag{A.4}$$

$$\begin{aligned}(K_h^2 * f)(x) &= \left(\frac{1}{(2\pi)^{1/2}h}\phi_{h/2^{1/2}}/2^{1/2} * f\right)(x) \\ &= \frac{1}{2\pi^{1/2}h}\left(\phi_{h/2^{1/2}} * f\right)(x) \\ &= \frac{1}{2\pi^{1/2}h}\phi_{(h^2/2+\sigma^2)^{1/2}}(x - \mu). \end{aligned} \tag{A.5}$$

Thus, the exact expectation of the kde for estimating the density of a $\mathcal{N}(\mu, \sigma^2)$ is precisely the density of a $\mathcal{N}(\mu, \sigma^2 + h^2)$. Clearly, when $h \to 0$, we can see how the bias disappears. Removing this finite-sample size bias is not simple: if the bias is expanded, f'' appears. Hence, to attempt to unbias $\hat{f}(\cdot; h)$ we have to estimate f'', which is a harder task than estimating f. As seen in Section 3.2, taking second derivatives on the kde does not work out-of-the-box, since the bandwidths for estimating f and f'' scale differently.

The previous deadlock can be solved if we limit our ambitions. Rather than constructing a confidence interval for $f(x)$, we can construct it for $\mathbb{E}[\hat{f}(x;h)] = (K_h * f)(x)$. There is nothing wrong with this change of view, as long as we are explicitly report the CI as that for $(K_h * f)(x)$ and not for $f(x)$.

The building block for the CI for $\mathbb{E}[\hat{f}(x;h)] = (K_h * f)(x)$ is the first result[2] in Theorem 2.2, which concludes that

$$\sqrt{nh}(\hat{f}(x;h) - \mathbb{E}[\hat{f}(x;h)]) \xrightarrow{d} \mathcal{N}(0, R(K)f(x)).$$

Plugging $\hat{f}(x;h) = f(x) + O_{\mathbb{P}}\left(h^2 + (nh)^{-1/2}\right) = f(x)(1 + o_{\mathbb{P}}(1))$ (see Exercise 2.10) as an estimate for $f(x)$ in the variance, we have by the Slutsky's theorem that

$$\begin{aligned}\sqrt{\frac{nh}{R(K)\hat{f}(x;h)}}&(\hat{f}(x;h) - \mathbb{E}[\hat{f}(x;h)]) \\ &= \sqrt{\frac{nh}{R(K)f(x)}}(\hat{f}(x;h) - \mathbb{E}[\hat{f}(x;h)])(1 + o_{\mathbb{P}}(1)) \\ &\xrightarrow{d} \mathcal{N}(0,1).\end{aligned}$$

Therefore, an asymptotic $100(1-\alpha)\%$ confidence interval for $\mathbb{E}[\hat{f}(x;h)]$ that can be straightforwardly computed is

$$I = \left(\hat{f}(x;h) \pm z_{\alpha/2}\sqrt{\frac{R(K)\hat{f}(x;h)}{nh}}\right). \tag{A.6}$$

Remark. Several points regarding (A.6) require proper awareness:

1. As announced, the CI is meant for $\mathbb{E}[\hat{f}(x;h)] = (K_h * f)(x)$, not $f(x)$, as it does not account for the smoothing bias.
2. This is a **pointwise** CI: $\mathbb{P}\left[\mathbb{E}[\hat{f}(x;h)] \in I\right] \approx 1-\alpha$ for *each* $x \in \mathbb{R}$. That is, $\mathbb{P}\left[\mathbb{E}[\hat{f}(x;h)] \in I, \forall x \in \mathbb{R}\right] \neq 1-\alpha$.
3. The CI relies on an approximation of $f(x)$ in the variance, done by $\hat{f}(x;h) = f(x) + O_{\mathbb{P}}\left(h^2 + (nh)^{-1/2}\right)$. Additionally, the convergence to a normal distribution happens at rate $\sqrt{nh}$. Hence, both h and nh need to be small and large, respectively, for a good coverage.
4. The CI is built using a **deterministic bandwidth** h (i.e., not data-driven), which is not usually the case in practice. If a bandwidth selector is employed, the coverage may be affected, especially for small n.

We illustrate the construction of the CI in (A.6) for the situation where the reference density is a $\mathcal{N}(\mu, \sigma^2)$ and the normal kernel is employed. This allows to use (A.4) and (A.5), in combination with (2.12) and (2.13), to obtain

$$\begin{aligned}\mathbb{E}[\hat{f}(x;h)] &= \phi_{(h^2+\sigma^2)^{1/2}}(x-\mu), \\ \mathbb{V}\mathrm{ar}[\hat{f}(x;h)] &= \frac{1}{n}\left(\frac{\phi_{(h^2/2+\sigma^2)^{1/2}}(x-\mu)}{2\pi^{1/2}h} - (\phi_{(h^2+\sigma^2)^{1/2}}(x-\mu))^2\right).\end{aligned}$$

These *exact* results are useful to benchmark (A.6) with the asymptotic CI that employs the exact variance of $\hat{f}(x;h)$:

$$\left(\hat{f}(x;h) \pm z_{\alpha/2}\sqrt{\mathbb{V}\mathrm{ar}[\hat{f}(x;h)]}\right). \tag{A.7}$$

[2] If we wanted to obtain a CI with the second result in Theorem 2.2 we would need to estimate $f''(x)$.

The following chunk of code evaluates the proportion of times that $\mathbb{E}[\hat{f}(x;h)]$ belongs to I for each $x \in \mathbb{R}$, both with estimated or known variance in the asymptotic distribution.

```
# R(K) for a normal
Rk <- 1 / (2 * sqrt(pi))

# Generate a sample from a N(mu, sigma^2)
n <- 100
mu <- 0
sigma <- 1
set.seed(123456)
x <- rnorm(n = n, mean = mu, sd = sigma)

# Compute the kde (NR bandwidth)
kde <- density(x = x, from = -4, to = 4, n = 1024, bw = "nrd")

# Selected bandwidth
h <- kde$bw

# Estimate the variance
var_kde_hat <- kde$y * Rk / (n * h)

# True expectation and variance (because the density is a normal)
E_kde <- dnorm(x = kde$x, mean = mu, sd = sqrt(sigma^2 + h^2))
var_kde <- (dnorm(kde$x, mean = mu, sd = sqrt(h^2 / 2 + sigma^2)) /
              (2 * sqrt(pi) * h) - E_kde^2) / n

# CI with estimated variance
alpha <- 0.05
z_alpha2 <- qnorm(1 - alpha / 2)
ci_low_1 <- kde$y - z_alpha2 * sqrt(var_kde_hat)
ci_up_1 <- kde$y + z_alpha2 * sqrt(var_kde_hat)

# CI with known variance
ci_low_2 <- kde$y - z_alpha2 * sqrt(var_kde)
ci_up_2 <- kde$y + z_alpha2 * sqrt(var_kde)

# Plot estimate, CIs and expectation
plot(kde, main = "Density and CIs", ylim = c(0, 1))
lines(kde$x, ci_low_1, col = "gray")
lines(kde$x, ci_up_1, col = "gray")
lines(kde$x, ci_low_2, col = "gray", lty = 2)
lines(kde$x, ci_up_2, col = "gray", lty = 2)
lines(kde$x, E_kde, col = "red")
legend("topright", legend = c("Estimate", "CI estimated var",
                              "CI known var", "Smoothed density"),
       col = c("black", "gray", "gray", "red"), lwd = 2, lty = c(1, 1, 2, 1))
```

The above experiment shows the CI, but it does not give any insight into the effective coverage of the CI. The following simulation exercise precisely addresses this issue. As seen from Figure A.2, the estimation of the kde's variance has a considerable impact in the coverage of the CI in the low-density regions, and it is reasonable fine for the higher density regions.

```
# Simulation setting
n <- 200; h <- 0.15
mu <- 0; sigma <- 1 # Normal parameters
M <- 5e2 # Number of replications in the simulation
n_grid <- 512 # Number of x's for computing the kde
alpha <- 0.05; z_alpha2 <- qnorm(1 - alpha / 2) # alpha for CI

# Compute expectation and variance
kde <- density(x = 0, bw = h, from = -4, to = 4, n = n_grid) # Just for kde$x
E_kde <- dnorm(x = kde$x, mean = mu, sd = sqrt(sigma^2 + h^2))
```

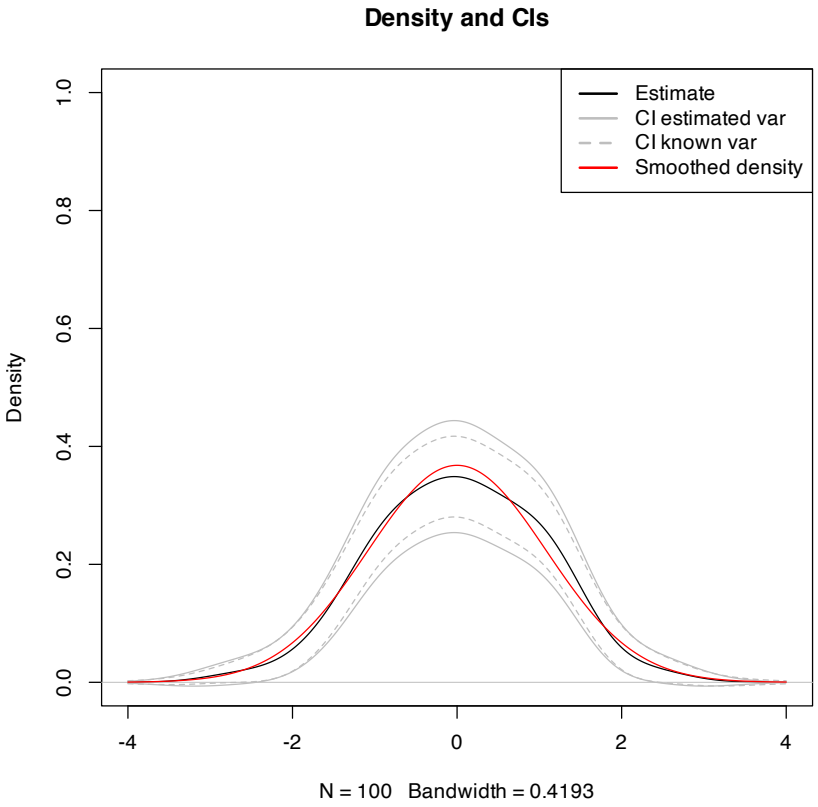


Figure A.1: The CIs (A.6) and (A.7) for $\mathbb{E}[\hat{f}(x;h)]$ with estimated and known variances.

```
var_kde <- (dnorm(kde$x, mean = mu, sd = sqrt(h^2 / 2 + sigma^2)) /
              (2 * sqrt(pi) * h) - E_kde^2) / n

# For storing if the mean is inside the CI
inside_ci_1 <- inside_ci_2 <- matrix(nrow = M, ncol = n_grid)

# Simulation
set.seed(12345)
for (i in 1:M) {

  # Sample & kde
  x <- rnorm(n = n, mean = mu, sd = sigma)
  kde <- density(x = x, bw = h, from = -4, to = 4, n = n_grid)
  sd_kde_hat <- sqrt(kde$y * Rk / (n * h))

  # CI with estimated variance
  ci_low_1 <- kde$y - z_alpha2 * sd_kde_hat
  ci_up_1 <- kde$y + z_alpha2 * sd_kde_hat

  # CI with known variance
  ci_low_2 <- kde$y - z_alpha2 * sqrt(var_kde)
  ci_up_2 <- kde$y + z_alpha2 * sqrt(var_kde)

  # Check if for each x the mean is inside the CI
  inside_ci_1[i, ] <- E_kde > ci_low_1 & E_kde < ci_up_1
  inside_ci_2[i, ] <- E_kde > ci_low_2 & E_kde < ci_up_2

}

# Plot results
plot(kde$x, colMeans(inside_ci_1), ylim = c(0.25, 1), type = "l",
     main = "Empirical coverage of CIs", xlab = "x", ylab = "Coverage")
lines(kde$x, colMeans(inside_ci_2), col = 4)
abline(h = 1 - alpha, col = 2)
```

```
abline(h = 1 - alpha + c(-1, 1) * qnorm(0.975) *
         sqrt(alpha * (1 - alpha) / M), col = 2, lty = 2)
legend(x = "bottom", legend = c("CI estimated var", "CI known var",
                                "Nominal level",
                                "95% CI for the nominal level"),
       col = c(1, 4, 2, 2), lwd = 2, lty = c(1, 1, 1, 2))
```

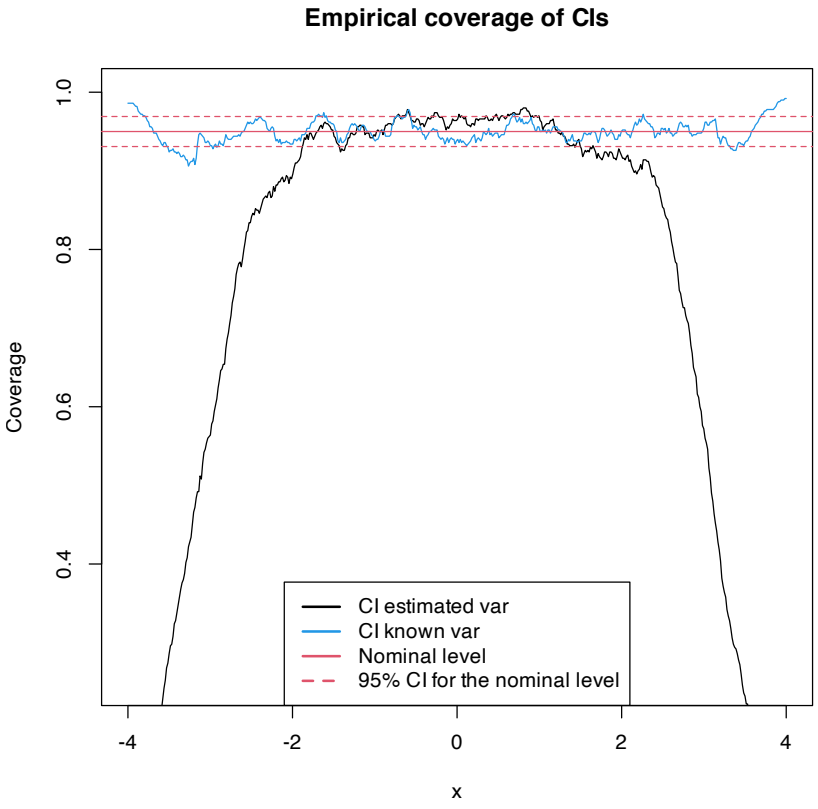


Figure A.2: Empirical coverage of the CIs (A.6) and (A.7) for $\mathbb{E}[\hat{f}(x;h)]$ with estimated and known variances.

Exercise A.1. Explore the coverage of the asymptotic CI for varying values of h. To that end, adapt the previous code to work in a `manipulate` environment like the example given below.

```
# Sample
x <- rnorm(100)

# Simple plot of kde for varying h
manipulate::manipulate({

  kde <- density(x = x, from = -4, to = 4, bw = h)
  plot(kde, ylim = c(0, 1), type = "l", main = "")
  curve(dnorm(x), from = -4, to = 4, col = 2, add = TRUE)
  rug(x)

}, h = manipulate::slider(min = 0.01, max = 2, initial = 0.5, step = 0.01))
```

Exercise A.2 (Exercise 6.9.5 in Wasserman (2006))**.** Data on the salaries of the chief executive officer of 60 companies is available at `http://lib.stat.cmu.edu/DASL/Datafiles/ceodat.html` (alternative link). Investigate the distribution of salaries using a kde. Use $\hat{h}_{\text{LSCV}}$ to choose the amount of smoothing. Also, consider $\hat{h}_{\text{RT}}$. There appear to be a few bumps in the density. Are they real? Use

confidence bands to address this question. Finally, comment on the resulting estimates.

B
Review on parametric regression

We review now a couple of useful parametric regression models that will be used in the construction of nonparametric estimators.

B.1 Linear regression

B.1.1 Model formulation and least squares

The multiple linear regression employs *multiple* predictors $X_1, \ldots, X_p$ [1] for explaining a single response Y by *assuming* that a linear relation of the form

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon \tag{B.1}$$

holds between the predictors $X_1, \ldots, X_p$ and the response Y. In (B.1), β_0 is the *intercept* and $\beta_1, \ldots, \beta_p$ are the *slopes*, respectively. ε is a random variable with mean zero and independent from $X_1, \ldots, X_p$. Another way of looking at (B.1) is

$$\mathbb{E}[Y|X_1 = x_1, \ldots, X_p = x_p] = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p, \tag{B.2}$$

since $\mathbb{E}[\varepsilon|X_1 = x_1, \ldots, X_p = x_p] = 0$. Therefore, the expectation of Y is changing in a *linear* fashion with respect to the values of $X_1, \ldots, X_p$. Hence the interpretation of the coefficients:

- β_0: is the expectation of Y when $X_1 = \cdots = X_p = 0$.
- β_j, $1 \leq j \leq p$: is the **additive** increment in expectation of Y for an increment of one unit in $X_j = x_j$, provided that the remaining variables *do not change*.

Figure B.1 illustrates the geometrical interpretation of a multiple linear model: a plane in the $(p+1)$-dimensional space. If $p = 1$, the plane is the regression line for simple linear regression. If $p = 2$, then the plane can be visualized in a three-dimensional plot.

The estimation of $\beta_0, \beta_1, \ldots, \beta_p$ is done by minimizing the so-called *Residual Sum of Squares* (RSS). We first need to introduce some helpful notation for this and the next section:

- A sample of $(X_1, \ldots, X_p, Y)$ is denoted by $(X_{11}, \ldots, X_{1p}, Y_1), \ldots, (X_{n1}, \ldots, X_{np}, Y_n)$, where X_{ij} denotes the i-th observation of the j-th predictor X_j. We denote with $\mathbf{X}_i = (X_{i1}, \ldots, X_{ip})'$ to the i-th observation of $(X_1, \ldots, X_p)$, so the sample is $(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)$.

[1] Not to confuse with a sample!

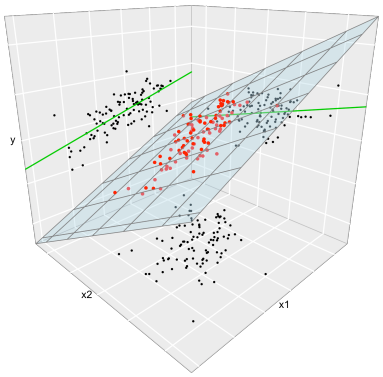


Figure B.1: The regression plane (blue) of Y on X_1 and X_2, and its relation with the regression lines (green lines) of Y on X_1 (left) and of Y on X_2 (right). The red points represent the sample for (X_1, X_2, Y) and the black points the projections for (X_1, X_2) (bottom), (X_1, Y) (left), and (X_2, Y) (right). Note that the regression plane is not a direct extension of the marginal regression lines.

- The *design matrix* contains all the information of the predictors and a column of ones

$$\mathbf{X} = \begin{pmatrix} 1 & X_{11} & \cdots & X_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & X_{n1} & \cdots & X_{np} \end{pmatrix}_{n\times(p+1)}.$$

- The *vector of responses* $\mathbf{Y}$, the *vector of coefficients* $\boldsymbol{\beta}$, and the *vector of errors* are, respectively,

$$\mathbf{Y} = \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}_{n\times 1}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}_{(p+1)\times 1}, \text{ and } \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}_{n\times 1}.$$

Thanks to the matrix notation, we can turn the sample version of the multiple linear model, namely

$$Y_i = \beta_0 + \beta_1 X_{i1} + \ldots + \beta_p X_{ip} + \varepsilon_i, \quad i = 1, \ldots, n,$$

into something as compact as

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}.$$

The RSS for the multiple linear regression is

$$\begin{aligned} \text{RSS}(\boldsymbol{\beta}) &:= \sum_{i=1}^{n} (Y_i - \beta_0 - \beta_1 X_{i1} - \ldots - \beta_p X_{ip})^2 \\ &= (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}). \end{aligned} \tag{B.3}$$

RSS($\boldsymbol{\beta}$) aggregates the *squared vertical distances* from the data to a regression plane given by $\boldsymbol{\beta}$. Note that the *vertical distances* are considered because we want to minimize the error in the *prediction* of Y. Thus, the treatment of the variables is *not symmetrical*[2]; see Figure B.2. The least squares estimators are the minimizers of (B.3):

$$\hat{\boldsymbol{\beta}} := \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^{p+1}} \text{RSS}(\boldsymbol{\beta}).$$

Luckily, thanks to the matrix form of (B.3), it is simple to compute a closed-form expression for the least squares estimates:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}. \tag{B.4}$$

Exercise B.1. $\hat{\boldsymbol{\beta}}$ can be obtained by differentiating (B.3). Prove it using that $\frac{\partial \mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = \mathbf{A}$ and $\frac{\partial f(\mathbf{x})' g(\mathbf{x})}{\partial \mathbf{x}} = f(\mathbf{x})' \frac{\partial g(\mathbf{x})}{\partial \mathbf{x}} + g(\mathbf{x})' \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$ for two vector-valued functions f and g.

Let's check that indeed the coefficients given by R's `lm` are the ones given by (B.4) in a toy linear model.

```
# Generates 50 points from a N(0, 1): predictors and error
set.seed(34567)
x1 <- rnorm(50)
x2 <- rnorm(50)
```

[2] If that were the case, we would consider perpendicular distances, which yield Principal Component Analysis (PCA).

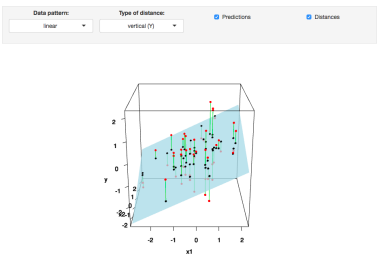


Figure B.2: The least squares regression plane $y = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2$ and its dependence on the kind of squared distance considered. Application available here.

```r
x3 <- x1 + rnorm(50, sd = 0.05) # Make variables dependent
eps <- rnorm(50)

# Responses
y_lin <- -0.5 + 0.5 * x1 + 0.5 * x2 + eps
y_qua <- -0.5 + x1^2 + 0.5 * x2 + eps
y_exp <- -0.5 + 0.5 * exp(x2) + x3 + eps

# Data
data_animation <- data.frame(x1 = x1, x2 = x2, y_lin = y_lin,
                             y_qua = y_qua, y_exp = y_exp)

# Call lm
# lm employs formula = response ~ predictor1 + predictor2 + ...
# (names according to the data frame names) for denoting the regression
# to be done
mod <- lm(y_lin ~ x1 + x2, data = data_animation)
summary(mod)
##
## Call:
## lm(formula = y_lin ~ x1 + x2, data = data_animation)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.37003 -0.54305  0.06741  0.75612  1.63829
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.5703     0.1302  -4.380 6.59e-05 ***
## x1            0.4833     0.1264   3.824 0.000386 ***
## x2            0.3215     0.1426   2.255 0.028831 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9132 on 47 degrees of freedom
## Multiple R-squared:  0.276,  Adjusted R-squared:  0.2452
## F-statistic: 8.958 on 2 and 47 DF,  p-value: 0.0005057

# mod is a list with a lot of information
# str(mod) # Long output

# Coefficients
mod$coefficients
## (Intercept)          x1          x2
##  -0.5702694   0.4832624   0.3214894

# Application of formula (3.4)

# Matrix X
X <- cbind(1, x1, x2)

# Vector Y
Y <- y_lin

# Coefficients
beta <- solve(t(X) %*% X) %*% t(X) %*% Y
beta
##          [,1]
##    -0.5702694
## x1  0.4832624
## x2  0.3214894
```

Exercise B.2. Compute $\boldsymbol{\beta}$ for the regressions `y_lin ~ x1 + x2`, `y_qua ~ x1 + x2`, and `y_exp ~ x2 + x3` using equation (B.4) and the function `lm`. Check that the fitted plane and the coefficient estimates are coherent.

Once we have the least squares estimates $\hat{\boldsymbol{\beta}}$, we can define the following two concepts:

- The *fitted values* $\hat{Y}_1, \ldots, \hat{Y}_n$, where

$$\hat{Y}_i := \hat{\beta}_0 + \hat{\beta}_1 X_{i1} + \cdots + \hat{\beta}_p X_{ip}, \quad i = 1, \ldots, n.$$

  They are the vertical projections of $Y_1, \ldots, Y_n$ into the fitted line (see Figure B.2). In a matrix form, inputting (B.3)

$$\hat{\mathbf{Y}} = \mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y} = \mathbf{H}\mathbf{Y},$$

  where $\mathbf{H} := \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$ is called the *hat matrix* because it "puts the hat into $\mathbf{Y}$". What it does is to project $\mathbf{Y}$ into the regression plane (see Figure B.2).

- The *residuals* (or estimated errors) $\hat{\varepsilon}_1, \ldots, \hat{\varepsilon}_n$, where

$$\hat{\varepsilon}_i := Y_i - \hat{Y}_i, \quad i = 1, \ldots, n.$$

  They are the vertical distances between actual and fitted data.

B.1.2 Model assumptions

Observe that $\hat{\boldsymbol{\beta}}$ was derived from purely geometrical arguments, not probabilistic ones. That is, we have not made any probabilistic assumption on the data generation process. However, some probabilistic assumptions are required to infer the *unknown* population coefficients $\boldsymbol{\beta}$ from the sample $(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)$.

The assumptions of the multiple linear model are:

i. **Linearity**: $\mathbb{E}[Y|X_1 = x_1, \ldots, X_p = x_p] = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$.
ii. **Homoscedasticity**: $\mathbb{V}\mathrm{ar}[\varepsilon_i] = \sigma^2$, with σ^2 constant for $i = 1, \ldots, n$.
iii. **Normality**: $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ for $i = 1, \ldots, n$.
iv. **Independence of the errors**: $\varepsilon_1, \ldots, \varepsilon_n$ are independent.

A good one-line summary of the linear model is the following (independence is assumed):

$$Y|(X_1 = x_1, \ldots, X_p = x_p) \sim \mathcal{N}(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p, \sigma^2). \quad \text{(B.5)}$$

Inference on the parameters $\boldsymbol{\beta}$ and σ can be done, conditionally[3] on $X_1, \ldots, X_n$, from (B.5). We do not explore this further, referring the interested reader to, e.g., Section 2.4 in García-Portugués (2022). Instead, we remark the connection between least squares estimation and the maximum likelihood estimator derived from (B.5).

First, note that (B.5) is the *population version* of the linear model (it is expressed in terms of the random variables). The *sample version* that summarizes assumptions i–iv is

$$\mathbf{Y}|\mathbf{X} \sim \mathcal{N}_n(\mathbf{X}\boldsymbol{\beta}, \sigma^2\mathbf{I}_n).$$

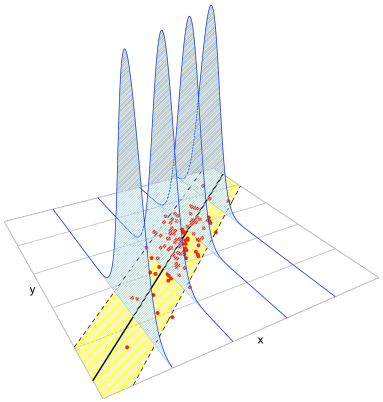


Figure B.3: The key concepts of the simple linear model. The blue densities denote the conditional density of Y for each cut in the X axis. The yellow band denotes where the 95% of the data is, according to the model. The red points represent a sample following the model.

[3] We assume that the randomness is on the response only.

Using this result, it is easy to obtain the log-likelihood function of $Y_1, \ldots, Y_n$ conditionally on $\mathbf{X}_1, \ldots, \mathbf{X}_n$ as[4]

$$\ell(\boldsymbol{\beta}) = \log \phi_{\sigma^2 \mathbf{I}_n}(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) = \sum_{i=1}^{n} \log \phi_\sigma(Y_i - (\mathbf{X}\boldsymbol{\beta})_i). \tag{B.6}$$

[4] Recall that $\phi_{\boldsymbol{\Sigma}}(\cdot - \boldsymbol{\mu})$ and $\phi_\sigma(\cdot - \mu)$ denote the pdf of a $\mathcal{N}_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and a $\mathcal{N}(\mu, \sigma^2)$, respectively.

Finally, the following result justifies the consideration of the least squares estimate: it equals the maximum likelihood estimator derived under assumptions i–iv.

Theorem B.1. *Under assumptions i–iv, the maximum likelihood estimator of $\boldsymbol{\beta}$ is the least squares estimate* (B.4)*:*

$$\hat{\boldsymbol{\beta}}_{\mathrm{ML}} = \arg \max_{\boldsymbol{\beta} \in \mathbb{R}^{p+1}} \ell(\boldsymbol{\beta}) = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}\mathbf{Y}.$$

Proof. Expanding the first equality at (B.6) gives[5]

$$\ell(\boldsymbol{\beta}) = -\log((2\pi)^{n/2}\sigma^n) - \frac{1}{2\sigma^2}(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}).$$

[5] Since $|\sigma^2 \mathbf{I}_n|^{1/2} = \sigma^n$.

Optimizing ℓ does not require knowledge on σ^2, since differentiating with respect to $\boldsymbol{\beta}$ and equating to zero gives (see Exercise B.1) $\frac{1}{\sigma^2}(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})'\mathbf{X} = \mathbf{0}$. Solving the equation gives the form for $\hat{\boldsymbol{\beta}}_{\mathrm{ML}}$. □

Exercise B.3. Conclude the proof of Theorem B.1.

B.2 Logistic regression

B.2.1 Model formulation

When the response Y can take two values only, codified for convenience as 1 (success) and 0 (failure), Y is called a *binary* variable. A binary variable, known also as a *Bernoulli variable*, is a $\mathrm{B}(1, p)$.[6]

[6] Recall that $\mathbb{E}[\mathrm{B}(1,p)] = \mathbb{P}[\mathrm{B}(1,p) = 1] = p$.

If Y is a binary variable and $X_1, \ldots, X_p$ are predictors associated with Y, the purpose in *logistic regression* is to estimate

$$\begin{aligned} p(x_1, \ldots, x_p) &:= \mathbb{P}[Y = 1 | X_1 = x_1, \ldots, X_p = x_p] \\ &= \mathbb{E}[Y | X_1 = x_1, \ldots, X_p = x_p], \end{aligned} \tag{B.7}$$

this is, how the probability of $Y = 1$ is changing according to particular values, denoted by $x_1, \ldots, x_p$, of the predictors $X_1, \ldots, X_p$. A tempting possibility is to consider a linear model for (B.7), $p(x_1, \ldots, x_p) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$. However, such a model will run into serious problems inevitably: negative probabilities and probabilities greater than one will arise.

A solution is to consider a function to encapsulate the value of $z = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$, in $\mathbb{R}$, and map it back to $[0, 1]$. There are several alternatives to do so, based on distribution functions $F : \mathbb{R} \longrightarrow [0, 1]$ that deliver $y = F(z) \in [0, 1]$. Different choices of F give rise to different models, the most common one being the *logistic distribution function*:

$$\mathrm{logistic}(z) := \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}}.$$

Its inverse, $F^{-1} : [0,1] \longrightarrow \mathbb{R}$, known as the *logit function*, is

$$\text{logit}(p) := \text{logistic}^{-1}(p) = \log \frac{p}{1-p}.$$

This is a *link function*, that is, a function that maps a given space (in this case $[0,1]$) onto $\mathbb{R}$. The term link function is used in *generalized linear models*, which follow exactly the same philosophy of the logistic regression – mapping the domain of Y to $\mathbb{R}$ in order to apply there a linear model. As said above, different link functions are possible, but we will concentrate here exclusively on the logit as a link function.

The *logistic model* is defined as the next parametric form for (B.7):

$$\begin{aligned} p(x_1, \ldots, x_p) &= \text{logistic}(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p) \\ &= \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)}}. \end{aligned} \tag{B.8}$$

The linear form inside the exponent has a clear interpretation:

- If $\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p = 0$, then $p(x_1, \ldots, x_p) = \frac{1}{2}$ ($Y = 1$ and $Y = 0$ are equally likely).
- If $\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p < 0$, then $p(x_1, \ldots, x_p) < \frac{1}{2}$ ($Y = 1$ less likely).
- If $\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p > 0$, then $p(x_1, \ldots, x_p) > \frac{1}{2}$ ($Y = 1$ more likely).

To be more precise on the interpretation of the coefficients $\beta_0, \ldots, \beta_p$ we need to introduce the concept of *odds*. The odds is an equivalent form of expressing the distribution of probabilities in a binary variable. Since $\mathbb{P}[Y = 1] = p$ and $\mathbb{P}[Y = 0] = 1 - p$, both the success and failure probabilities can be inferred from p. Instead of using p to characterize the distribution of Y, we can use

$$\text{odds}(Y) = \frac{p}{1-p} = \frac{\mathbb{P}[Y = 1]}{\mathbb{P}[Y = 0]}. \tag{B.9}$$

The odds is the *ratio between the probability of success and the probability of failure*. It is extensively used in betting due to its better interpretability.[7] Conversely, if the odds of Y is given, we can easily know what is the probability of success p, using the inverse[8] of (B.9):

$$p = \mathbb{P}[Y = 1] = \frac{\text{odds}(Y)}{1 + \text{odds}(Y)}.$$

Remark. Recall that the odds is a number in $[0, +\infty]$. The 0 and $+\infty$ values are attained for $p = 0$ and $p = 1$, respectively. The log-odds (or logit) is a number in $[-\infty, +\infty]$.

We can rewrite (B.8) in terms of the odds (B.9). If we do so, then

$$\begin{aligned} &\text{odds}(Y | X_1 = x_1, \ldots, X_p = x_p) \\ &\quad = \frac{p(x_1, \ldots, x_p)}{1 - p(x_1, \ldots, x_p)} \\ &\quad = e^{\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p} \\ &\quad = e^{\beta_0} e^{\beta_1 x_1} \ldots e^{\beta_p x_p}. \end{aligned}$$

[7] For example, if a horse Y has a probability $p = 2/3$ of winning a race ($Y = 1$), then the odds of the horse is odds $= \frac{p}{1-p} = \frac{2/3}{1/3} = 2$. This means that the horse has a probability of winning that is *twice* larger than the probability of losing. This is sometimes written as a $2:1$ or 2×1 (spelled "two-to-one").

[8] For example, if the odds of the horse was 5, that would correspond to a probability of winning $p = 5/6$.

This provides the following interpretation of the coefficients:

- e^{β_0}: is the odds of $Y = 1$ when $X_1 = \cdots = X_p = 0$.
- e^{β_j}, $1 \leq j \leq k$: is the **multiplicative** increment of the odds for an increment of one unit in $X_j = x_j$, provided that the remaining variables *do not change*. If the increment in X_j is of r units, then the multiplicative increment in the odds is $(e^{\beta_j})^r$.

B.2.2 *Model assumptions and estimation*

Some probabilistic assumptions are required to perform inference on the model parameters $\boldsymbol{\beta}$ from a sample $(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)$. These assumptions are somehow simpler than the ones for linear regression.

The assumptions of the logistic model are the following:

i. **Linearity in the logit**[9]: $\text{logit}(p(\mathbf{x})) = \log \frac{p(\mathbf{x})}{1-p(\mathbf{x})} = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$.
ii. **Binariness**: $Y_1, \ldots, Y_n$ are binary variables.
iii. **Independence**: $Y_1, \ldots, Y_n$ are independent.

A good one-line summary of the logistic model is the following (independence is assumed):

$$Y|(X_1 = x_1, \ldots, X_p = x_p) \sim \text{Ber}\left(\text{logistic}(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)\right) = \text{Ber}\left(\frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)}}\right).$$

Since $Y_i \sim \text{Ber}(p(\mathbf{X}_i))$, $i = 1, \ldots, n$, the log-likelihood of $\boldsymbol{\beta}$ is

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} \log\left(p(\mathbf{X}_i)^{Y_i}(1 - p(\mathbf{X}_i))^{1-Y_i}\right) = \sum_{i=1}^{n} \{Y_i \log(p(\mathbf{X}_i)) + (1 - Y_i)\log(1 - p(\mathbf{X}_i))\}. \quad \text{(B.10)}$$

Unfortunately, due to the nonlinearity of the optimization problem, there are no explicit solutions for $\hat{\boldsymbol{\beta}}$. These have to be obtained numerically by means of an iterative procedure, which may run into problems in low-sample situations with perfect classification. Unlike in the linear model, inference is not exact from the assumptions, but rather approximate in terms of maximum likelihood theory. We do not explore this further and refer the interested reader to, e.g., Section 5.3 in García-Portugués (2022).

Figure B.5 shows how the log-likelihood changes with respect to the values for (β_0, β_1) in three data patterns. The data of the illustration has been generated with the following code.

```
# Data
set.seed(34567)
x <- rnorm(50, sd = 1.5)
y1 <- -0.5 + 3 * x
y2 <- 0.5 - 2 * x
y3 <- -2 + 5 * x
y1 <- rbinom(50, size = 1, prob = 1 / (1 + exp(-y1)))
```

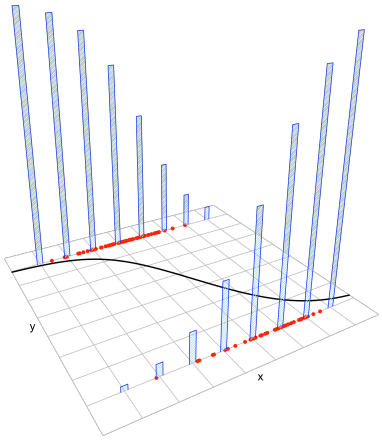


Figure B.4: The key concepts of the logistic model. The blue bars represent the conditional distribution of probability of Y for each cut in the X axis. The red points represent a sample following the model.

[9] An equivalent way of stating this assumption is $p(\mathbf{x}) = \text{logistic}(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)$.

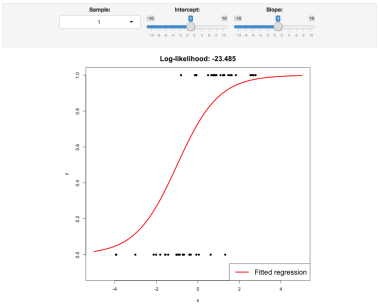
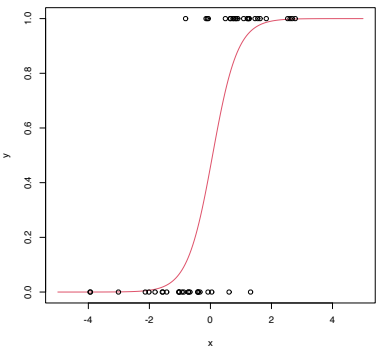


Figure B.5: The logistic regression fit and its dependence on β_0 (horizontal displacement) and β_1 (steepness of the curve). Recall the effect of the sign of β_1 on the curve: if positive, the logistic curve has an "s" form; if negative, the form is a reflected "s". Application available here.

```
y2 <- rbinom(50, size = 1, prob = 1 / (1 + exp(-y2)))
y3 <- rbinom(50, size = 1, prob = 1 / (1 + exp(-y3)))

# Data
data_mle <- data.frame(x = x, y1 = y1, y2 = y2, y3 = y3)
```

Let's check that indeed the coefficients given by R's `glm` are the ones that maximize the likelihood of the animation of Figure B.5. We do so for `y ~ x1`.

```
# Call glm
# glm employs formula = response ~ predictor1 + predictor2 + ...
# (names according to the data frame names) for denoting the regression
# to be done. We need to specify family = "binomial" to make a
# logistic regression
mod <- glm(y1 ~ x, family = "binomial", data = data_mle)
summary(mod)
##
## Call:
## glm(formula = y1 ~ x, family = "binomial", data = data_mle)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.47853  -0.40139   0.02097   0.38880   2.12362
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.1692     0.4725  -0.358 0.720274
## x             2.4282     0.6599   3.679 0.000234 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 69.315  on 49  degrees of freedom
## Residual deviance: 29.588  on 48  degrees of freedom
## AIC: 33.588
##
## Number of Fisher Scoring iterations: 6

# mod is a list with a lot of information
# str(mod) # Long output

# Coefficients
mod$coefficients
## (Intercept)           x
##  -0.1691947   2.4281626

# Plot the fitted regression curve
x_grid <- seq(-5, 5, l = 200)
y_grid <- 1 / (1 + exp(-(mod$coefficients[1] + mod$coefficients[2] * x_grid)))
plot(x_grid, y_grid, type = "l", col = 2, xlab = "x", ylab = "y")
points(x, y1)
```



Exercise B.4. For the regressions `y ~ x2` and `y ~ x3`, do the following:

a. Check that $\boldsymbol{\beta}$ is indeed maximizing the likelihood as compared with Figure B.5.
b. Plot the fitted logistic curve and compare it with the one in Figure B.5.

C
Informal review on hypothesis testing

The process of hypothesis testing has an interesting analogy with a trial. The analogy helps to understand the elements present in a formal hypothesis test in an *intuitive* way.[1]

Hypothesis test	Trial
Null hypothesis H_0	The **defendant**: an individual **accused** of committing a crime. He[2] is backed up by the **presumption of innocence**, which means that he is *not guilty* until there is enough evidence to support his guilt.
Sample $X_1, \ldots, X_n$	Collection of **evidence supporting innocence and guilt** of the defendant. This evidence contains a certain degree of uncontrollable randomness due to how it is collected and the context regarding the case[3].
Test statistic[4] T_n	**Summary of the evidence presented** by the prosecutor and defense lawyer.
Distribution of T_n under H_0	The **judge** conducting the trial. He evaluates and measures the evidence presented by both sides and presents a verdict for the defendant.
Significance level α	$1-\alpha$ is the **strength of the evidence** required by the judge for condemning the defendant. The judge allows evidence that, on average, condemns $100\alpha\%$ of the innocents, due to the randomness inherent to the evidence collection process. $\alpha = 0.05$ is considered to be a reasonable level[5].

[1] That is not intended to replace a formal introduction to hypothesis tests. The interested reader can find one, e.g., in Chapter 6 in Molina-Peralta and García-Portugués (2022).

[2] The masculin pronoun in no case indicates gender ascription. It is used as a neutral form and could be substituted for any personal pronoun.

[3] Think about phenomena that may randomly support defendant's innocence or guilt, irrespective of his true condition. For example: spurious coincidences ("happen to be in the wrong place at the wrong time"), loss of evidence during the case, previous past statemets of the defendant, dubious identification by witness, imprecise witness testimonies, unverificable alibi, etc.

[4] Usually simply referred to as *statistic*.

[5] As the judge must have the power of condemning a guilty defendant. Setting $\alpha = 0$ (no innocents are declared guilty) would result in a judge that systematically declares everybody *not guilty*. Therefore, a compromise is needed.

Hypothesis test	Trial
p-value	**Decision** of the judge that measures the degree of compatibility, in a scale 0–1, of the presumption of innocence with the summary of the evidence presented. If p-value $< \alpha$, the defendant is declared *guilty*. Otherwise, he is declared *not guilty*.
H_0 is rejected	The defendant is declared *guilty*: there is **strong evidence supporting his guilt**.
H_0 is not rejected	The defendant is declared *not guilty*: either he is **innocent or there is not enough evidence supporting his guilt**.

More formally, the p-value of a hypothesis test about H_0 is defined as:

> The p-value is the probability of obtaining a test statistic more unfavorable to H_0 than the observed, assuming that H_0 is true.

Therefore, **if the p-value is small** (smaller than the chosen level α), **it is unlikely that the evidence against H_0 is due to randomness**. As a consequence, H_0 is rejected. If the p-value is large (larger than α), then it is more possible that the evidence against H_0 is merely due to the randomness of the data. In this case, we do not reject H_0.

If H_0 holds, then the p-value (which is a random variable) is distributed uniformly in $(0, 1)$. If H_0 does not hold, then the distribution of the p-value is not uniform but concentrated at 0 (where the rejections of H_0 take place).

D
References

Aitchison, J. and Aitken, C. G. G. (1976). Multivariate binary discrimination by the kernel method. *Biometrika*, 63(3):413–420.

Allaire, J. J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., and Iannone, R. (2020). *rmarkdown: Dynamic Documents for R*. R package version 2.5.

Anderson, T. W. (1962). On the distribution of the two-sample Cramer-von Mises criterion. *The Annals of Mathematical Statistics*, 33(3):1148–1159.

Benjamini, Y. (1988). Opening the box of a boxplot. *The American Statistician*, 42(4):257–262.

Blum, J. R., Kiefer, J., and Rosenblatt, M. (1961). Distribution free tests of independence based on the sample distribution function. *The Annals of Mathematical Statistics*, 32(2):485–498.

Chacón, J. E. (2015). A population background for nonparametric density-based clustering. *Statistical Science*, 30(4):518–532.

Chacón, J. E. and Duong, T. (2018). *Multivariate Kernel Smoothing and its Applications*, volume 160 of *Monographs on Statistics and Applied Probability*. CRC Press, Boca Raton.

Cuesta-Albertos, J. A., Gordaliza, A., and Matrán, C. (1997). Trimmed *k*-means: an attempt to robustify quantizers. 25(2):553–576.

D'Agostino, R. B. and Stephens, M. A., editors (1986). *Goodness-of-Fit Techniques*, volume 68 of *Statistics: Textbooks and Monographs*. Marcel Dekker, New York.

Dallal, G. E. and Wilkinson, L. (1986). An analytic approximation to the distribution of Lilliefors's test statistic for normality. *The American Statistician*, 40(4):294–296.

DasGupta, A. (2008). *Asymptotic Theory of Statistics and Probability*. Springer Texts in Statistics. Springer, New York.

del Barrio, E. (2007). Empirical and quantile processes in the asymptotic theory of goodness-of-fit tests. In *Lectures on Empirical Processes*, EMS Series of Lectures in Mathematics, pages 1–92. European Mathematical Society, Zürich.

Devroye, L. and Györfi, L. (1985). *Nonparametric Density Estimation*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, New York.

Duong, T. (2020). *ks: Kernel Smoothing*. R package version 1.11.7.

ESA (1997). *The Hipparcos and Tycho Catalogues*, volume 1200 of *ESA SP*. ESA Publication Division, Noordwijk.

Fan, J. and Gijbels, I. (1996). *Local Polynomial Modelling and its Applications*, volume 66 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, London.

Fukunaga, K. and Hostetler, L. D. (1975). The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40.

García-Portugués, E. (2022). *Notes for Predictive Modeling*. Version 5.9.8. ISBN 978-84-09-29679-8.

Genovese, C. R., Perone-Pacifico, M., Verdinelli, I., and Wasserman, L. (2014). Nonparametric ridge estimation. *The Annals of Statistics*, 42(4):1511–1545.

González-Manteiga, W. and Crujeiras, R. M. (2013). An updated review of goodness-of-fit tests for regression models. *TEST*, 22(3):361–411.

Hardle, W. and Marron, J. S. (1991). Bootstrap simultaneous error bars for nonparametric regression. *The Annals of Statistics*, 19(2):778–796.

Hayfield, T. and Racine, J. S. (2008). Nonparametric econometrics: The np package. *Journal of Statistical Software*, 27(5):1–32.

Hintze, J. L. and Nelson, R. D. (1998). Violin plots: a box plot-density trace synergism. *The American Statistician*, 52(2):181–184.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning*, volume 103 of *Springer Texts in Statistics*. Springer, New York.

Li, Q. and Racine, J. S. (2007). *Nonparametric Econometrics*. Princeton University Press, Princeton.

Lilliefors, H. W. (1967). On the Kolmogorov-Smirnov test for normality with mean and variance unknown. *Journal of the American Statistical Association*, 62(318):399–402.

Liu, R. Y. (1988). Bootstrap procedures under some non-i.i.d. models. *The Annals of Statistics*, 16(4):1696–1708.

Loader, C. (1999). *Local Regression and Likelihood*. Statistics and Computing. Springer, New York.

Mann, H. B. and Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60.

Marron, J. S. and Wand, M. P. (1992). Exact mean integrated squared error. *The Annals of Statistics*, 20(2):712–736.

Molina-Peralta, I. and García-Portugués, E. (2022). *A First Course on Statistical Inference*. Lecture notes. Version 0.9.1.

Nadaraya, E. A. (1964). On estimating regression. *Teoriya Veroyatnostei i ee Primeneniya*, 9(1):157–159.

Nelsen, R. B. (2006). *An Introduction to Copulas*. Springer Series in Statistics. Springer-Verlag, New York, second edition.

Parzen, E. (1962). On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33(3):1065–1076.

Petersen, K. B. and Pedersen, M. S. (2012). *The Matrix Cookbook*. Technical University of Denmark. Version 20121115.

Pettitt, A. N. (1976). A two-sample Anderson-Darling rank statistic. *Biometrika*, 63(1):161–168.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. Vienna.

Rizzo, M. L. and Székely, G. J. (2016). Energy distance. *WIREs Computational Statistics*, 8(1):27–38.

Rosenblatt, M. (1956). Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics*, 27(3):832–837.

Royston, P. (1993). A pocket-calculator algorithm for the Shapiro-Francia test for non-normality: an application to medicine. *Statistics in Medicine*, 12(2):181–184.

Ruppert, D., Sheather, S. J., and Wand, M. P. (1995). An effective bandwidth selector for local least squares regression. *Journal of the American Statistical Association*, 90(432):1257–1270.

Scarsini, M. (1984). On measures of concordance. *Stochastica*, 8:201–218.

Scott, D. W. (2015). *Multivariate Density Estimation*. Wiley Series in Probability and Statistics. John Wiley & Sons, Hoboken, second edition.

Scott, D. W. and Terrell, G. R. (1987). Biased and unbiased cross-validation in density estimation. *Journal of the American Statistical Association*, 82(400):1131–1146.

Shao, J. (1999). *Mathematical Statistics*. Springer Texts in Statistics. Springer, New York.

Sheather, S. J. and Jones, M. C. (1991). A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society, Series B (Methodological)*, 53(3):683–690.

Sheskin, D. J. (2011). *Handbook of Parametric and Nonparametric Statistical Procedures*. CRC Press, Boca Raton, fifth edition.

Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. Monographs on Statistics and Applied Probability. Chapman & Hall, London.

Stephens, M. A. (1974). EDF statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69(347):730–737.

Székely, G. J. and Rizzo, M. L. (2013). Energy statistics: A class of statistics based on distances. *Journal of Statistical Planning and Inference*, 143(8):1249–1272.

Székely, G. J. and Rizzo, M. L. (2017). The energy of data. *Annual Review of Statistics and its Application*, 4(1):447–479.

Székely, G. J., Rizzo, M. L., and Bakirov, N. K. (2007). Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35(6):2769–2794.

Tukey, J. W. (1977). *Exploratory Data Analysis*. Behavioral Science: Quantitative Methods. Addison-Wesley Publishing Company, Reading.

Úcar, I. (2018). *Energy Efficiency in Wireless Communications for Mobile User Devices*. PhD thesis, Universidad Carlos III de Madrid.

van der Maaten, L. and Hinton, G. (2008). Visualizing data using *t*-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605.

van der Vaart, A. W. (1998). *Asymptotic Statistics*, volume 3 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, Cambridge.

Velilla, S. (1994). A goodness-of-fit test for autoregressive moving-average models based on the standardized sample spectral distribution of the residuals. *Journal of Time Series Analysis*, 15(6):637–647.

Wand, M. P. and Jones, M. C. (1995). *Kernel Smoothing*, volume 60 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, London.

Wasserman, L. (2004). *All of Statistics*. Springer Texts in Statistics. Springer, New York.

Wasserman, L. (2006). *All of Nonparametric Statistics*. Springer Texts in Statistics. Springer, New York.

Watson, G. S. (1964). Smooth regression analysis. *Sankhyā, Series A*, 26(4):359–372.

Wickham, H. and Stryjewski, L. (2011). 40 years of boxplots. Technical report.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83.

Wu, C. F. J. (1986). Jackknife, bootstrap and other resampling methods in regression analysis. *The Annals of Statistics*, 14(4):1261–1295.

Xie, Y. (2016). *bookdown: Authoring Books and Technical Documents with R Markdown*. The R Series. CRC Press, Boca Raton.

Xie, Y. (2020). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.30.

Xie, Y. and Allaire, J. J. (2020). *tufte: Tufte's Styles for R Markdown Documents*. R package version 0.8.