HUMBOLDT-UNIVERSITÄT ZU BERLIN

Chair of Statistics

The RODEO Approach for Nonparametric Density Estimation

Master's Thesis in Statistics in partial fullfillment for the degree

Master of Science (M.Sc.)

submitted by

Lukas Mödl

(581096)

Advisor: Prof. Dr. Nadja Klein

Co-advisor: Prof. Dr. Stefan Lessmann

Berlin, December 11, 2020

Statement of Authentication

I hereby declare that I have written the present thesis independently, without assistance from external parties and without use of other resources than those indicated. The ideas taken directly or indirectly from external sources are acknowledged in the text to the best of my knowledge. The thesis, either in full or in part, has not been previously submitted for grading at this or any other academic institution.

Place, Date

Signature

Abstract

The regularization of derivative expectation operator (RODEO) approach developed by Lafferty and Wasserman (2008) is a regularization technique designed for a wide range of nonparametric kernel smoothers. The approach applies regularization by penalizing the bias reduction associated with a bandwidth reduction along a smooth path of decreasing bandwidth parameter values in order to avoid overfitting. Dimensions with small local variation are effectively smoothed out, thus implicitly carrying out variable selection. Under certain conditions, faster rates of converges of convergence for the mean integrated square error can be achieved, which makes the approach attractive for applications in high dimensions. In this paper we apply the RODEO approach to local polynomial density estimation. We implemented the approach in the R package `lpderodeo`. We apply our implementation to a few examples, and evaluate its performance in a comparative study using a sample of eight other approaches for nonparametric density estimation. Our findings suggest that the approach does not work well in comparison to the other considered approaches with regard to the applied performance metrics. Furthermore, our implementation suffers from long computation time due to a naive query. Our main finding, however, concerns the fact that the theoretical framework proposed by Liu, Lafferty, and Wasserman (2007) has severe shortcomings. In fact, we demonstrate that a simple rotation of the data makes the algorithm fail in practice.

Contents

1 Introduction

Suppose that $X_1, X_2, \ldots, X_n$ is an independent and identically distributed (iid) sample drawn from some d-dimensional probability distribution with unknown density f. Using a nonparametric model has the advantage that no assumption about the functional form has to be made. However, if d is large, nonparametric models become very ineffective in practice due to large estimation errors. For instance, the optimal rate of convergence of the mean integrated square error (MISE) of the popular kernel density estimator (KDE) from Rosenblatt (1956) and Parzen (1962) is $\mathsf{O}\left(n^{-\frac{4}{4+d}}\right)$ under standard smoothness assumptions (Stone, 1980). That is, the rate of convergence is considerably slow even for moderate levels of d. Thus, the sample sizes necessary to bind the estimation error below an acceptable margin are usually neither available nor computationally processable. With the problem being inherent to nonparametric models, it is impossible to solve the problem in general. Since no functional form assumption is made, any information about the shape of the target density is extracted from the distribution of the given sample. However, the number of observations needed to adequately populate a d-dimensional domain grows exponentially with d (this phenomen is known as the "curse of dimensionality"; Bellman, 1957).

The regularization of derivative expectation operator (RODEO) approach attempts to circumvent this problem by regularization. The RODEO approach developed by Lafferty and Wasserman (2008) is a technique designed for a wide range of nonparametric kernel smoothers. The approach applies regularization by penalizing the bias reduction associated with a bandwidth reduction along a smooth path of decreasing bandwidth parameter values in order to avoid overfitting. Dimensions with small local variation are effectively smoothed out, thus implicitly carrying out variable selection.[1] If the local variation is small for all but $r < d$ dimensions, the approach is able to very likely achieve the faster rate of convergence of $\mathsf{O}\left(n^{-\frac{4}{4+r}}\right)$ up to a logarithmic factor, which makes the approach attractive for applications in high dimensions

Although promising, the RODEO approach for nonparametric density estimation has gained only little attention in the literature – we are only aware of Liu, Lafferty, and Wasserman (2007), Ram and Gray (2011), as well as Mahapatruni and Gray (2011) who have actually implemented the approach for nonparametric density estimation. The lack of attention given to a promising method thus provides a strong incentetive to further analyze the approach. We propose the R package `lpderodeo` which applies the RODEO approach to local polynomial density estimation. We apply our implementation to few examples, and evaluate its performance in a comparative study using a sample of eight other approaches for nonparametric density estimation. The examples include

[1] The RODEO approach could thus be seen as a nonparametric sister of the least squares shrinkage operator (LASSO) approach in parametric regression. The LASSO approach biases the regression coefficients towards zero, while the effect of sufficiently small coefficients is fully removed, thus carrying out variable selection.

data in $d = 1, 2, 10$, and 256 dimensions to analyze the efficacy at various levels of dimensionality. Our findings suggest that the approach does not work well in comparison to the other considered approaches with regard to the applied performance metrics. Our main finding, however, concerns the fact that the theoretical framework proposed by Liu, Lafferty, and Wasserman (2007) has severe shortcomings. In fact, we demonstrate that a simple rotation of the data make the algorithm fail in practice.

We proceed as follows. Section 2 gives an overview of the methodology; we review the theory of local polynomial density estimation, and we explain how the RODEO approach can be used for regularization of the estimate. Subsequently, Section 3 tests our implementation in practice using a few examples. Section 4 revisits our examples with different parametrizations. Section 5 discusses problems and limitations of the approach. Finally, Section 6 summarizes our findings and concludes. The appendix contains additional material; including the source code of our R package `lperodeo`.

2 Methodology

Suppose that f belongs to the Sobolev space $\mathcal{W}^{p+1,\infty}$, and let x be a point at which we want to estimate $f(x)$ based on the observed iid sample $X_1, X_2, \ldots, X_n$. To avoid technical issues, we assume that x lies in the interior of f's support. Furthermore, we assume, without loss of generality, that the given sample has zero mean, and a spherical unit variance-covariance matrix. This section briefly reviews the theory of local polynomial density estimation, derives the local polynomial density estimator (LPDE) for $f(x)$, and explains how the RODEO approach can be applied for regularization of the LPDE.

2.1 Local Polynomial Density Estimation

Since f is, by assumption, in $\mathcal{W}^{p+1,\infty}$, it is p-times continuously differentiable, and there exists a multivariate q-degree polynomial

$$\Pi(\mathrm{x}) = \alpha_0 \bullet \mathrm{x}^{\otimes 0} + \alpha_1 \bullet \mathrm{x}^{\otimes 1} + \cdots + \alpha_q \bullet \mathrm{x}^{\otimes q} \tag{2.1}$$

with $q < p$ that approximates the log density $\log\big(f(x)\big)$ well in a neighborhood around x. Here the $\bullet$ denotes an Euclidean inner product, and $\mathrm{x}^{\otimes \cdot}$ defines a tensor power (i.e. we put $\mathrm{x}^{\otimes 0} \coloneqq 1$, $\mathrm{x}^{\otimes 1} \coloneqq \mathrm{x}$, $\mathrm{x}^{\otimes 2} \coloneqq \mathrm{x} \otimes \mathrm{x}$ and so on, where $\otimes$ denotes the Kronecker product). Tensor powers are used here because – unlike component-wise powers – tensor powers include the cross products. The existence of such a polynomial is an immediate consequence of the Taylor theorem. The Taylor theorem states that any smooth function can be approximated locally around each query point by a polynomial whose coefficients are given by the function's (vectorized) partial derivatives evaluated at the query point. In other words, the Taylor theorem suggests that a polynomial is locally a reasonable parametric model for any smooth function, with the coefficients $\alpha_0, \alpha_1, \ldots, \alpha_q$ of the polynomial being the unknown model parameters.

The model parameters are estimated based on the observed sample using standard maximum likelihood techniques (Hjort and Jones, 1996; Loader, 1996; Loader, 2006). The corresponding objective function is given by

$$\mathcal{L}_x = -n^{-1} \sum_{i=1}^{n} K_H(X_i - x)\Pi(X_i - x) + \int_{\mathbb{R}^d} K_H(u - x) \exp\big(\Pi(u - x)\big)\, \mathrm{d}u, \tag{2.2}$$

where K_H is a kernel. The kernel weighs the observations, which is necessary as the approximation of $\log\big(f(x)\big)$ by a polynomial is, by construction, only valid within a neighborhood around x. Observations within the neighborhood should receive more weight, while observations further away

should receive only very little or no weight at all. The neighborhood's size, called its bandwidth, is controlled by the symmetric and positive definite $d \times d$ matrix H. The choice of the bandwidth parameter is crucial with regard to the variance-bias consideration of the estimate. We use the RODEO approach for bandwidth selection as described in the subsequent sections, and treat the bandwidth parameter for the remainder of this section as given. Approximating the log density – instead of the actual density – ensures nonnegativity of the estimate, and the additional term in (2.2) incorporates a constraint which ensures that the estimate integrates to unity (see Silverman, 1982).

Since (2.2) is smooth in the parameters $\alpha_0, \alpha_1, \dots, \alpha_q$, the first order condition of the optimization problem is – given some regularity conditions on the kernel – equivalent to solving the following system

$$\begin{aligned}
n^{-1} \sum_{i=1}^{n} K_H(X_i - x)(X_i - x)^{\otimes 0} &= \int_{\mathbb{R}^d} K_H(u - x)(u - x)^{\otimes 0} \exp\big(\Pi(u - x)\big)\, \mathrm{d}u \\
n^{-1} \sum_{i=1}^{n} K_H(X_i - x)(X_i - x)^{\otimes 1} &= \int_{\mathbb{R}^d} K_H(u - x)(u - x)^{\otimes 1} \exp\big(\Pi(u - x)\big)\, \mathrm{d}u \\
\vdots \qquad &= \qquad \vdots \\
n^{-1} \sum_{i=1}^{n} K_H(X_i - x)(X_i - x)^{\otimes q} &= \int_{\mathbb{R}^d} K_H(u - x)(u - x)^{\otimes q} \exp\big(\Pi(u - x)\big)\, \mathrm{d}u
\end{aligned}$$

of $1 + d + \cdots + d^q$ nonlinear equations for $\alpha_0, \alpha_1, \dots, \alpha_q$. A solution $\hat{\alpha}_0, \hat{\alpha}_1, \dots, \hat{\alpha}_q$ may not always exist, but if it does, the solution is unique, and it is guaranteed to be a maximizer of (2.2) (Loader, 2006). In case of an existing solution, the LPDE for $f(x)$ is defined by $\hat{f}_H(x) \coloneqq \exp(\hat{\alpha}_0)$. This definition follows immediately from the Taylor theorem, as the constant monomial in the Taylor expansion of a function around a query point equals the function itself evaluated at the query point. For notational covencience, we suppress the dependence of $\hat{f}_H(x)$ on the given sample. The LPDE has several appealing large sample properties such as consistency and asymptotic normality; see Loader (1996) and Loader (2006) for a rigorous formal treatment.

To ensure the existence of a closed-form solution for $\hat{f}_H(x)$, we assume in the following that $q \leq 2$ and use the normal kernel

$$K_H(u) = (2\pi)^{-\frac{d}{2}} \det(H^{-1}) \exp\left(-\frac{1}{2} H^{-1}u \bullet H^{-1}u\right). \tag{2.3}$$

This kernel allows the integrals on the right hand side of the first order condition to be evaluated analytically. Consequently, a closed-form solution for $\hat{f}_H(x)$ exists. The derivations for the LPDE with constant fitting ($q = 0$), linear fitting ($q = 1$), and quadratic fitting ($q = 2$) can be found in Appendix A.

The LPDE with constant fitting. When a constant polynomial ($q = 0$) is used to approximate the log density, the LPDE for $f(x)$ is given by

$$\hat{f}_H(x) = n^{-1} \sum_{i=1}^{n} K_H(X_i - x). \tag{2.4}$$

The LPDE with constant fitting is equivalent to the KDE from Rosenblatt (1956) and Parzen (1962), and has been widely studied; see, for example, Wand and Jones (1994) and Scott (2015). Since this estimator is based on a local constant approximation, it suffers from trimming of peaks or values near the boundary (Loader, 1996).

The LPDE with linear fitting. When a linear polynomial ($q = 1$) is used to approximate the log density, the LPDE for $f(x)$ is given by

$$\hat{f}_H(x) = n^{-1} \sum_{i=1}^{n} K_H(X_i - x) \exp\left(-\frac{1}{2} 1_n' W Q H^{-2} Q' W 1_n\right), \tag{2.5}$$

where 1_n is a vector of n ones, and Q and W are defined by

$$Q := \begin{bmatrix} (X_1 - x)' \\ (X_2 - x)' \\ \vdots \\ (X_n - x)' \end{bmatrix} \quad \text{and} \quad W := \begin{bmatrix} \frac{K_H(X_1 - x)}{\sum_{i=1}^n K_H(X_i - x)} & 0 & \cdots & 0 \\ 0 & \frac{K_H(X_2 - x)}{\sum_{i=1}^n K_H(X_i - x)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{K_H(X_n - x)}{\sum_{i=1}^n K_H(X_i - x)} \end{bmatrix},$$

respectively. Note that the LPDE with linear fitting equals the LPDE with constant fitting multiplied by an exponential bias correction term. This estimator thus has better properties compared to the LPDE with constant fitting around peaks and close to the boundary. We refer again to Loader (1996) for a thorough analysis.

The LPDE with quadratic fitting. When a quadratic polynomial ($q = 2$) is used to approximate the log density, the LPDE for $f(x)$ is given by

$$\hat{f}_H(x) = n^{-1} \sum_{i=1}^{n} K_H(X_i - x) \sqrt{\det(A)} \exp\left(-\frac{1}{2} 1_n' W Q H^{-1} A H^{-1} Q' W 1_n\right) \tag{2.6}$$

with $A := H(Q'WQ - Q'W1_n 1_n' WQ)^{-1} H$. Additional restrictions are necessary to ensure the existence of A (see Appendix A). This estimator has been shown to have similar good properties as the LPDE with linear fitting with regard to peaks and values near the boundary (Loader, 1996).

2.2 Applying the RODEO[2]

The choice of the bandwidth parameter H is crucial with regard to the variance and bias of $\hat{f}_H(x)$. If the bandwidth parameter is chosen to be large (in the sense of the Loewner order), the weight assigned by the kernel is distributed among many observations. This causes essential characteristics of $f(x)$ to be smoothed out effectively, resulting in a biased estimate $\hat{f}_H(x)$. Conversely, if the bandwidth parameter is chosen to be small, individual observations receive more weight which makes the kernel treat random fluctuations as genuine characteristics of $f(x)$. That is, $\hat{f}_H(x)$ overfits the given sample, resulting in a large variance. A formal version of this argument is given in Loader (1996) and Loader (2006). The goal is to select a bandwidth parameter which yields a good tradeoff between both variance and bias.

Let H_{init} be some large bandwidth parameter. Any real symmetric matrix is diagonalizable, so there exists an orthonormal $d \times d$ matrix P and a diagonal $d \times d$ matrix Ξ_{init} such that $H_{\text{init}} = P\Xi_{\text{init}}P'$. We have that Ξ_{init} is positive definite as H_{init} is, by definition, positive definite. Since the set of all positive definite diagonal $d \times d$ matrices together with the $d \times d$ zero matrix O_d is convex, there exists a smooth path φ defined on the positive real line including zero and infinity from Ξ_{init} to O_d such that $\varphi(t) \neq O_d$ for all finite t. It can further be shown that, as f is in $\mathcal{W}^{p+1,\infty}$, $\mathbb{E}[\hat{f}_H(x)]$ converges to $f(x)$ as the sample size increases and H converges to O_d (c.f. Loader, 1996; Loader, 2006). Since a real symmetric matrix converges to the zero matrix if and only if all of its eigenvalues converge to zero, we can likewise move along φ and $\mathbb{E}[\hat{f}_{P\varphi(\cdot)P'}(x)]$ converges to $f(x)$, i.e.

$$f(x) = \lim_{t\to\infty} \mathbb{E}[\hat{f}_{P\varphi(t)P'}(x)].$$

Adding the zero $\mathbb{E}[\hat{f}_{P\varphi(0)P'}(x)] - \mathbb{E}[\hat{f}_{P\varphi(0)P'}(x)]$ on the right hand side and rearranging the terms yields

$$\begin{aligned} f(x) &= \lim_{t\to\infty} \mathbb{E}[\hat{f}_{P\varphi(t)P'}(x)] + \mathbb{E}[\hat{f}_{P\varphi(0)P'}(x)] - \mathbb{E}[\hat{f}_{P\varphi(0)P'}(x)] \\ &= \mathbb{E}[\hat{f}_{P\varphi(0)P'}(x)] + \lim_{t\to\infty} \left(\mathbb{E}[\hat{f}_{P\varphi(t)P'}(x)] - \mathbb{E}[\hat{f}_{P\varphi(0)P'}(x)]\right). \end{aligned}$$

By definition of the path, we have that $\varphi(0) = \Xi_{\text{init}}$, so the first term equals $\mathbb{E}[\hat{f}_{H_{\text{init}}}(x)]$. For the second term, note that the function $\hat{f}_{P,P'}(x)$, defined in terms of a diagonal $d \times d$ matrix Ξ, is differentiable. Since derivative and expectation operators can be interchanged under mild conditions using the dominated convergence theorem, we have that the function $\mathbb{E}[\hat{f}_{P,P'}(x)]$ is differentiable

[2] The derivations presented in this section follow Lafferty and Wasserman (2008). Note that we use a different notation than Lafferty and Wasserman (2008) in this paper. Lafferty and Wasserman (2008), in our opinion, unnecessarily overload the meaning of symbols which we want to avoid. Furthermore, we extend the theory to unconstrained bandwidth selection using the eigendecomposition of real symmetric matrices.

as well. By convention, the derivative of a scalar-valued function with respect to a diagonal matrix is a diagonal matrix, so we put

$$\mu \coloneqq \operatorname{diag}\Big(\mathbb{E}\left[\frac{\mathrm{d}}{\mathrm{d}\Xi}\hat{f}_{P,P'}(x)\right]\Big) \qquad \text{and} \qquad \hat{\mu} \coloneqq \operatorname{diag}\Big(\frac{\mathrm{d}}{\mathrm{d}\Xi}\hat{f}_{P,P'}(x)\Big)$$

to gather the effective components in a vector; here diag denotes the operator that transforms the main diagonal of a matrix into a vector. Since φ is smooth, it is differentiable, and we put $\dot{\varphi} \coloneqq \operatorname{diag}\left(\frac{\mathrm{d}}{\mathrm{d}s}\varphi\right)$. The aforementioned second term can therefore be rewritten in terms of an integral using the fundamental theorem of calculus and standard matrix differentiation rules, i.e.

$$\begin{aligned} f(x) &= \mathbb{E}[\hat{f}_{H_{\text{init}}}(x)] + \lim_{t\to\infty}\left(\int_0^t \frac{\mathrm{d}}{\mathrm{d}s}\mathbb{E}[\hat{f}_{P\varphi(s)P'}(x)]\,\mathrm{d}s\right) \\ &= \mathbb{E}[\hat{f}_{H_{\text{inti}}}(x)] + \lim_{t\to\infty}\int_0^t \left(\frac{\mathrm{d}}{\mathrm{d}\,\Xi}\mathbb{E}[\hat{f}_{P\varphi(s)P'}(x)\right) : \left(\frac{\mathrm{d}}{\mathrm{d}s}\varphi(s)\right)\mathrm{d}s \\ &= \mathbb{E}[\hat{f}_{H_{\text{init}}}(x)] + \lim_{t\to\infty}\int_0^t \mu\big(\varphi(s)\big) \bullet \dot{\varphi}(s)\,\mathrm{d}s, \end{aligned}$$

where $:$ denotes the Frobenius inner product and $\bullet$ the Euclidean inner product.[3] Note that derivative and expectation operators were interchanged in the last step. To get an intuitive understanding of the the above expression, recall that $\hat{f}_{H_{\text{init}}}(x)$ is an inherently biased estimate for $f(x)$; the larger H_{init}, the larger the bias (Loader, 1996; Loader, 2006). Since $\hat{f}_{H_{\text{init}}}(x)$ is biased, the term

$$\lim_{t\to\infty}\int_0^t \mu\big(\varphi(s)\big) \bullet \dot{\varphi}(s)\,\mathrm{d}s$$

has to close the gap between $\mathbb{E}[\hat{f}_{H_{\text{init}}}(x)]$ and $f(x)$ for equality to hold. In other words, this term has to account for the total bias of $\hat{f}_{H_{\text{init}}}(x)$. So, for a given point $\varphi(s)$ along the path, we have, by definition of the derivative, that the amount of bias reduction is determined by $\mu\big(\varphi(s)\big)$. Furthermore, we have that $\dot{\varphi}(s)$ weighs the relative importance of the bias reduction according to the current position. Finally, the integral accumulates the total bias reduction along the whole path. The crucial observation made by Lafferty and Wasserman (2008) is that the bias reduction along φ can be manipulated by regularization of the derivative expectation operator μ, which motivates the name of the approach. The idea is to replace μ in the above expression of the bias by some integrable function ψ that penalizes the bias reduction along φ. The goal is to decrease the bias of $\hat{f}_{H_{\text{init}}}(x)$ without substantially increasing its variance.

[3] The formula derived here slightly differs from the one in Lafferty and Wasserman (2008) since they i) use a rather heuristic argument that ignores the limit, and ii) absorb a minus sign into their analogue of $\dot{\varphi}$.

Using $\hat{f}_{H_{\text{init}}}(x)$ as an estimator for $\mathbb{E}[\hat{f}_{H_{\text{init}}}(x)]$ and an appropriate estimator $\hat{\psi}$ for ψ defines the RODEO LPDE for $f(x)$:

$$\hat{f}_{\text{rodeo}}(x) \coloneqq \hat{f}_{H_{\text{init}}}(x) + \lim_{t\to\infty} \int_0^t \hat{\psi}\big(\varphi(s)\big) \bullet \dot{\varphi}(s)\, \mathrm{d}s. \tag{2.7}$$

In principle, there are many possible choices for $\hat{\psi}$ and φ. A particular specification for (2.7) is considered in the subsequent section.

2.3 Specifying the RODEO

In this paper, we follow Lafferty and Wasserman (2008) in that we consider a specification of (2.7) that applies regularization by thresholding the bias reduction along a path of exponentially decaying bandwidth parameter values.

Regularization by thresholding the bias reduction along φ is based on the idea that $\hat{\mu}$ can be used as a proxy to quantify the local variation of $f(x)$ in each dimension. Heuristically, the larger the local variation, the smaller the selected bandwidth should be in order to control the bias and to improve the goodness of fit. Conversely, the smaller the local variation, the larger the selected bandwidth should be in order to avoid overfitting. Thus, given a smooth path of decreasing bandwidths, we discard the bias reduction associated with a bandwidth decrease whenever the local variation quantified by $\hat{\mu}$ is below some threshold λ. That is, we use

$$\hat{\psi} = \theta \odot \mathsf{1}\big(|\hat{\mu}| > \lambda\big), \tag{2.8}$$

where $\mathsf{1}$ denotes the component-wise indicator function, $|\,\boldsymbol{.}\,|$ the component-wise absolute value function, and $\odot$ the component-wise multiplication. The parameter θ tunes the amount of bias reduction in each dimension. Based on theoretical considerations (see Theorem 1 below), Lafferty and Wasserman (2008) relate the threshold λ to the scale of $\hat{\mu}$ and suggest $\lambda = \zeta \odot \operatorname{diag}(\hat{\Sigma})$, where the $d \times d$ matrix $\hat{\Sigma}$ is an estimator for the scale of $\hat{\mu}$. The parameter ζ tunes the threshold in each dimension.

The choice of the smooth path φ is not too important as the starting point H_{init} and the end point O_d are, by assumption, fixed. The paths can therefore only differ in their directions and length. Consider the path

$$\varphi(s) = \Xi_{\text{init}} \operatorname{Diag}(\beta)^s \tag{2.9}$$

with exponential decay. Here Diag denotes the inverse operator of diag, i.e. Diag is the operator that transforms a vector into a diagonal matrix. The parameter β tunes the rate of decay in each dimension. The closer a component of β is to one, the slower the decay in that dimension; the closer a component is to zero, the faster the decay in that dimension.

Evaluation based on a greedy algorithm

Lafferty and Wasserman (2008) suggest evaluating (2.7) with the given choice of $\hat{\psi}$ and φ based on a greedy algorithm for reasons of computional efficiency. The algorithm moves iteratively along φ and deactivates a dimension by freezing its bandwidth at the current value as soon as the the corresponding component of $\hat{\psi}$ is equal to zero. The procedure is detailed in the pseudo-code Algorithm 1.

Algorithm 1: Greedy algorithm for evaluating (2.7) with $\hat{\psi}$ and φ given by (2.8) and (2.9), respectively.

Input parameters : H_{init}, β, θ, ζ, $\hat{\Sigma}$, q;

1 Set $\boldsymbol{\varphi}(\mathsf{s}) \leftarrow \Xi_{\text{init}}$;
2 Set $\hat{\mathsf{f}}_{\text{rodeo}}(\mathsf{x}) \leftarrow \hat{f}_{H_{\text{init}}}(x)$;
3 Set $\dot{\boldsymbol{\varphi}}(\mathsf{s}) \leftarrow \beta - 1_d$;
4 **while** $\dot{\boldsymbol{\varphi}}(\mathsf{s}) \neq 0_d$ **do**
5 Compute $\hat{\psi}\big(\boldsymbol{\varphi}(\mathsf{s})\big)$;
6 Set $\dot{\boldsymbol{\varphi}}(\mathsf{s}) \leftarrow \boldsymbol{\varphi}(\mathsf{s})(\beta - 1_d) \odot \mathbb{1}\big(\hat{\psi}\big(\boldsymbol{\varphi}(\mathsf{s})\big) \neq 0_d\big) \odot \mathbb{1}\big(\dot{\boldsymbol{\varphi}}(\mathsf{s}) \neq 0_d\big)$;
7 Set $\hat{\mathsf{f}}_{\text{rodeo}}(\mathsf{x}) \leftarrow \hat{\mathsf{f}}_{\text{rodeo}}(\mathsf{x}) + \hat{\psi}\big(\boldsymbol{\varphi}(\mathsf{s})\big) \bullet \dot{\boldsymbol{\varphi}}(\mathsf{s})$;
8 Set $\boldsymbol{\varphi}(\mathsf{s}) \leftarrow \boldsymbol{\varphi}(\mathsf{s}) + \text{Diag}\big(\dot{\boldsymbol{\varphi}}(\mathsf{s})\big)$;
9 **end**
10 **return** $\hat{\mathsf{f}}_{\text{rodeo}}(\mathsf{x}) \vee 0$;

The algorithm takes the initial bandwidth parameter H_{init}, the tuning parameters β, θ, and ζ, an estimator $\hat{\Sigma}$ for the scale of $\hat{\mu}$, and the functional form of the LPDE in terms of q as input parameters. Since the algorithm proceeds in a greedy fashion, the symbolic variables $\boldsymbol{\varphi}(\mathsf{s})$, $\hat{\mathsf{f}}_{\text{rodeo}}(\mathsf{x})$, and $\dot{\boldsymbol{\varphi}}(\mathsf{s})$ are introduced to emulate the movement along the path (lines 1, 2, 3). Note that $\boldsymbol{\varphi}(\mathsf{s})$, $\hat{\mathsf{f}}_{\text{rodeo}}(\mathsf{x})$, and $\dot{\boldsymbol{\varphi}}(\mathsf{s})$ are symbolic variables and should not be confused with their mathematical function analogues. The variables are initialized as follows:

- $\boldsymbol{\varphi}(\mathsf{s})$ is initialized by Ξ_{init} obtained from the eigendecomposition $H_{\text{init}} = P\Xi_{\text{init}}P'$.
- $\hat{\mathsf{f}}_{\text{rodeo}}(\mathsf{x})$ is initialized by the LPDE with H_{init} as bandwidth parameter.
- $\dot{\boldsymbol{\varphi}}(\mathsf{s})$ is initialized by $\beta - 1_d$.

After initializing the variables, the algorithm starts to move iteratively along the path. In each iteration, the algorithm computes $\hat{\psi}\big(\varphi(\mathsf{s})\big)$ (line 5). To find the value of $\hat{\psi}\big(\varphi(\mathsf{s})\big)$, those of $\hat{\mu}\big(\varphi(\mathsf{s})\big)$ and $\lambda\big(\varphi(\mathsf{s})\big)$ have to be computed:

- In general, $\hat{\mu}\big(\varphi(\mathsf{s})\big)$ is very difficult to compute analytically. Thus, we employ a numerical differentiation approach. The component-wise finite difference approximation of $\hat{\mu}\big(\varphi(\mathsf{s})\big)$ is given by

$$\hat{\mu}\big(\varphi(\mathsf{s})\big) \approx \delta^{-1} \begin{bmatrix} \hat{f}_{P(\varphi(\mathsf{s})+\delta E_{11})P'}(x) - \hat{f}_{P\varphi(\mathsf{s})P'}(x) \\ \hat{f}_{P(\varphi(\mathsf{s})+\delta E_{22})P'}(x) - \hat{f}_{P\varphi(\mathsf{s})P'}(x) \\ \vdots \\ \hat{f}_{P(\varphi(\mathsf{s})+\delta E_{dd})P'}(x) - \hat{f}_{P\varphi(\mathsf{s})P'}(x) \end{bmatrix}, \tag{2.10}$$

  where E_{jj} is a $d \times d$ matrix with zeros everywhere and a one at the (j, j)th position. Choosing a too large value for δ yields a poor approximation of the partial derivatives, while choosing δ too small results in arithmetic overflow due to large rounding errors. By default, $\delta = 0.0001$ is used, which yields a good tradeoff between approximation and rounding errors.

- To compute $\lambda\big(\varphi(\mathsf{s})\big)$ given by $\zeta \odot \hat{\Sigma}\big(\varphi(\mathsf{s})\big)$, the scale of $\hat{\mu}\big(\varphi(\mathsf{s})\big)$ has to be estimated. Since we have computed $\hat{\mu}\big(\varphi(\mathsf{s})\big)$ based on a finite difference approximation, we use a bootstrap procedure to estimate the scale of $\hat{\mu}$. That is, we simulate n draws with replacement from the given sample, and compute (2.10) for each draw. This step is replicated b times. By default, $b = 100$ replications are used; see the point on variation of $\hat{\Sigma}$ in Section 4. Finally, $\hat{\Sigma}\big(\varphi(\mathsf{s})\big)$ is computed based on the constructed sample.

After computing $\hat{\psi}\big(\varphi(\mathsf{s})\big)$, the algorithm updates the variables $\dot{\varphi}(\mathsf{s})$, $\hat{\mathsf{f}}_{\text{rodeo}}(\mathsf{x})$, and $\varphi(\mathsf{s})$ (lines 6, 7, 8). To simplify the procedure, Lafferty and Wasserman (2008) suggest replacing the continuum of bandwidths, i.e. the image set of φ under the nonnegative real line including zero and infinity, with a discrete set. The natural numbers including zero and infinity are – trivially – a subset of the positive real line including zero and infinity. Thus, the continuum of bandwidth parameters is straightforwardly discretized by restricting to the image set of φ under the natural numbers including zero and infinity. This simplification is of course at the expense of accuracy. Since integrals and derivatives on discrete domains behave like ordinary summations and finite differences, the variables are updated as follows:

- According to (2.9), it follows that $\dot{\varphi}(s)\,\mathrm{d}s = \varphi(s)(\beta^{\mathrm{d}s} - 1)$ using infinitesimal notation. Due to the discretization of the domain, $\mathrm{d}s = 1$, so $\dot{\varphi}(\mathsf{s})$ is set to $\varphi(\mathsf{s})(\beta - 1_d)$. The additional terms

$\mathbb{1}\big(\hat{\psi}\big(\varphi(\mathsf{s})\big) \neq 0_d\big)$ and $\mathbb{1}\big(\dot{\varphi}(\mathsf{s}) \neq 0_d\big)$ make the algorithm proceed in a greedy fashion.[4] Here 0_d denotes a vector of d zeros. Multiplying $\varphi(\mathsf{s})(\beta - 1_d)$ with the first term sets the bandwidth changes for those dimensions to zero for which the corresponding components of $\hat{\psi}\big(\varphi(\mathsf{s})\big)$ are equal to zero, and the other term ensures that the bandwidths for those dimensions remain unchanged in all future iterations of the algorithm.

- According to (2.7), the bias at a given point $\varphi(s)$ along the path is reduced by adding the bias correction $\hat{\psi}\big(\varphi(s)\big) \bullet \dot{\varphi}(s)\,\mathrm{d}s$ to the current value. Due to the discretization of the domain, $\mathrm{d}s = 1$, so $\hat{\mathsf{f}}_{\text{rodeo}}(\mathsf{x})$ is updated recursively by adding $\hat{\psi}\big(\varphi(\mathsf{s})\big) \bullet \dot{\varphi}(\mathsf{s})$.
- By definition of a finite difference, $\varphi(\mathsf{s})$ is updated recursively by adding the change $\dot{\varphi}(\mathsf{s})$.

Since bandwidth reductions increase the variance of $\hat{\mu}$, the algorithm will eventually reach a point where all components of $\dot{\varphi}(\mathsf{s})$ are equal to zero. In that case the algorithm stops and returns the maximum of zero and the current value of $\hat{\mathsf{f}}_{\text{rodeo}}(\mathsf{x})$ (line 10). Taking the maximum ensures nonnegativity of the estimate.

Theoretical properties

Liu, Lafferty, and Wasserman (2007), based on an earlier version of Lafferty and Wasserman (2008), propose a theoretical framework for the RODEO approach applied to local polynomial density estimation. It is based on the following:

> **Assumption 1: Local sparsity condition.** In a neighborhood around x, it holds that $p \geq 2$, and there exists a binary $r \times d$ matrix R of rank $r < d$ such that
>
> $$f(x) = g(Rx)h(x)$$
>
> for some twice continuously differentiable function g with nonzero second derivatives, and some function $h = c$ with $c > 0$.

The assumption states that $f(x)$ is sparse in the sense that its functional form is locally constant, and only varies (hence nonzero second derivatives) in some r-dimensional subspace spanned by the matrix R. We will refer to the r dimensions that span the subspace as the (local) relevant dimensions, while refering to the remaining $d - r$ as the (local) irrelevant dimensions.

If $\hat{\mu}$ is a reasonsable proxy for the local variation of $f(x)$ in each dimension, Algorithm 1 is expected to deactivate the irrelevant dimensions very early in the procedure. This is because the variation of

[4] Lafferty and Wasserman (2008) modify the index set to make the algorithm proceed in a greedy fashion. Our implementation is equivalent, but much simpler to implement, and in our opinion more intuitive.

a constant function is zero. The bandwidths for those dimensions remain large, causing essential characteristics of $f(x)$ in that dimension to be smoothed out. Heuristically, a large bandwidth is like ignoring a dimension, i.e. Algorithm 1 implicitly carries out variable selection in a nonparametric setting. Liu, Lafferty, and Wasserman (2007) formalize this argument by the following:

> **Theorem 1: Near-optimal convergence rate of the MISE.** Suppose that Assumption 1 holds. Furthermore, suppose that $H_{\text{init}} = \sqrt{h_0 \log\big(\log(n)\big)^{-1}}\, I_d$ for some $h_0 > 0$, $\beta = \beta_0 1_d$ for some $0 < \beta_0 < 1$, $\theta = \hat{\mu}$, $\zeta = \sqrt{2\log(\zeta_0)}1_d$ for $\zeta_0 = \mathsf{O}\big(\log(n)\big)$, $\hat{\Sigma}$ is chosen as the square root of $\hat{\mu}$'s variance-covariance matrix, and $q = 0$.[5] Then $\hat{f}_{\text{rodeo}}(x)$ output by Algorithm 1 satisfies
>
> $$\mathbb{E}\left[\int_{\mathbb{R}^d} \left|\hat{f}_{\text{rodeo}}(x) - f(x)\right|^2 \mathrm{d}x\right] = \mathsf{O}_{\mathbb{P}}\big(n^{-\frac{4}{4+r}} L_n\big),$$
>
> where L_n is a logarithmic sequence.

If the r relevant dimensions were isolated in advance, the optimal rate of convergence of the MISE would be $\mathsf{O}\big(n^{-\frac{4}{4+r}}\big)$. That is, Theorem 1 states that the MISE of the estimator output by Algorithm 1 is very likely to achieve the oracle convergence rate up to a logarithmic factor. The additional factor could be interpreted as the inevitable effort to identify the irrelevant dimensions. Theorem 1 justifies Algorithm 1 and motivate its application in high dimensions. If r is sufficiently small, good estimates can be obtained even if d is large.

However, as we will show in Section 5, there are a series of mistakes in the proof of Theorem 1 as shown in Liu, Lafferty, and Wasserman (2007). In particular, the proof is based on a wrong expressions for the mean and the variance of $\hat{\mu}$, which results in a fallacy. The validity of the theoretical framework proposed by Liu, Lafferty, and Wasserman (2007) is therefore brought into question.

[5] From the results of Liu, Lafferty, and Wasserman (2007) it becomes clear that the bandwidth parameter has to be the (matrix) square root of what they have used in their derivations. Consider, for instance, Example 1 in Liu, Lafferty, and Wasserman (2007). They claim that $\log\big(\log(200)\big)^{-1} \approx 0.59$ was chosen as initial bandwidth parameter. However, the lower right panel of Figure 3 in Liu, Lafferty, and Wasserman (2007) shows that the actual value is around $0.77 \approx \sqrt{\log\big(\log(200)\big)^{-1}}$.

3 Examples

Despite the theoretical problems we want to evaluate how the RODEO approach applied to local polynomial density estimation ($\hat{f}_{\text{rodeo}}$) performs in practice. We implemented the approach in the R package `lpderodeo` (see Appendix C). In the following, we apply our implementation to a few examples and evaluate its performance. The examples include data in $d = 1$, 2, 10, and 256 dimensions. To answer the question whether our implementation is a reasonable alternative to other approaches for nonparametric density estimation, we compare our implementation with a sample of other approaches. The other approaches we consider are:

- ($\hat{f}_{\text{nr}}$) a KDE with a normal kernel, where the bandwidth parameter is selected according to the normal reference rule from Silverman (1986);
- ($\hat{f}_{\text{pi}}$) a KDE with a normal kernel, where the bandwidth parameter is selected according to the plug-in rule from Chacón and Duong (2010);
- ($\hat{f}_{\text{hist}}$) a histogram with equally spaced bins, where the number of bins in each dimension is selected according to the normal reference rule from Freedman and Diaconis (1981);
- ($\hat{f}_{\text{nmm}}$) a normal mixture model;
- ($\hat{f}_{\text{knn}}$) the nearest neighbor density estimator, where the number of neighbors is chosen as the square root of the sample size;
- ($\hat{f}_{\text{det}}$) the decision tree density estimator from Ram and Gray (2011);
- ($\hat{f}_{\text{vine}}$) the vine copula density estimator from Nagler and Czado (2016); and
- ($\hat{f}_{\text{spline}}$) the smoothing spline density estimator from Gu (2013).

A short description for each approach is given on the next page. In the following, we choose the input parameters for each approach according to the default parameterization of the respective R implementation. By default, our implementation chooses the input parameters as suggested by Theorem 1, where the free parameters are chosen as $h_0 = 1$, $\beta_0 = 0.9$, and $\zeta_0 = d \log(n)$.

The raw data for each example in this section can be reproduced using the codes attached to Appendix D. We used R version 4.0.3 on a compute server with 80 cores at 2.0 GHz each, and 1024 GiB memory. The server access was provided by the Humboldt Lab for Empirical and Quantitative Research at Humboldt University of Berlin.

Approach	Description	R Package
$\hat{f}_{\text{nr}}$, $\hat{f}_{\text{pi}}$	Both rules select the bandwidth parameter such that the asymptotic MISE of the KDE is minimized. The only difference is in how the second derivative of the target density is estimated. The normal reference rule from Silverman (1986) assumes a Ricker wavelet, while the plug-in rule from Chacón and Duong (2010) estimates it from the data using a KDE with a pilot bandwidth.	`base`, `ks`
$\hat{f}_{\text{hist}}$	A histogram is a piece-wise constant estimator. The rule from Freedman and Diaconis (1981) selects the number of equally spaced bins so that the asymptotic MISE of the histogram is minimized.	`mvmesh` + `grDevices`
$\hat{f}_{\text{nmm}}$	A normal mixture model estimates the target density by a finite convex combination of normal densities. The model parameters (i.e. the means, variance-covariance matrices and mixture weights) are estimated using an expectation-maximization algorithm. The number of mixture components is determind by hierarchical model-based agglomerative clustering (Fraley and Raftery, 2002).	`mclust`
$\hat{f}_{\text{knn}}$	The nearest neighbor density estimator is a KDE with uniform kernel and varying bandwidth parameter. The bandwidth parameter is selected adaptively so that the number of observations in each neighborhood corresponds to the specified number of neighbors. Theoretical considerations motivate choosing the number of neighbors as the square root of sample size (Loftsgaarden and Quesenberry, 1965).	`TDA`
$\hat{f}_{\text{det}}$	The decision tree density estimator from Ram and Gray (2011) is essentially a histogram. However, the number of bins and the sizes of each bin are chosen adaptively according to a decision tree. The decision tree is trained so that the MISE of the estimator is minimized.	`detpack`
$\hat{f}_{\text{vine}}$	Any multivariate joint density can be expressed in terms of its univariate marginal densities and bivariate pair copula densities (Bedford and Cooke, 2002). The vine copula density estimator from Nagler and Czado (2016) makes use of this fact by estimating the marginal and the copula densities separately using KDE variants.	`kdevine`
$\hat{f}_{\text{spline}}$	A smoothing spline is a piece-wise polynomial function. The smoothing spline density estimator from Gu (2013) is the function which maximizes the negative likelihood functional with an added roughness penalty in a reproducing kernel Hilbert space.	`gss`

3.1 Univariate Normal Mixture Densities From Marron and Wand (1992)

In our first example we consider the 15 univariate normal mixture densities from Marron and Wand (1992). Normal mixture densities have the appealing property of being able to approximate any probability density arbitrarily well (Scott, 2015). Marron and Wand (1992) use this property to construct a sample of 15 densities that resemble many different challenges of density estimation, such as strong skewness, or multimodality. We do not give the full parameterization for each density, but instead offer a visual representation in Figure 3.1.

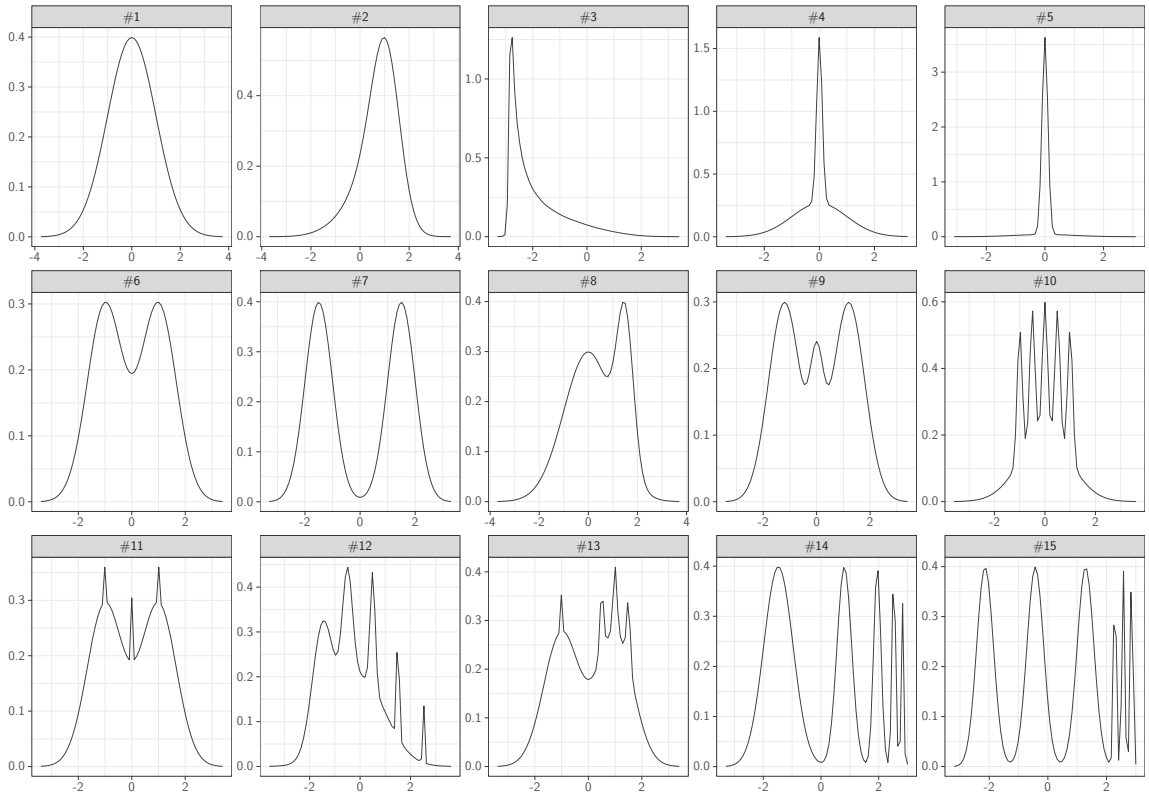


Figure 3.1: The 15 univariate normal mixture densities from Marron and Wand (1992).

We simulate $n = 200$ iid draws from each distribution. In each case, we center the data and scale it by the inverse sample standard deviation such that the sample has a mean of zero and a unit variance. Furthermore, we cut off a total of 0.1 percent of the probability mass in the tail regions to make the support of each density compact.

The estimated densities are visualized in Appendix B.[6] To quantify the goodness of fit beyond a mere sense of proportion we employ the MISE as a standard accuracy measure. Table 3.1 shows the

[6] Density #3 was also considered by Liu, Lafferty, and Wasserman (2007), as well as Ram and Gray (2011). Our estimates are nearly identical; see Figure B.1 and Figure B.2 in Appendix B. Unfortunately, exact replication of their results is not possible as neither Liu, Lafferty, and Wasserman (2007) nor Ram and Gray (2011) published their codes.

computed MISE for each approach. We use Monte Carlo integration to approximate the expected value. That is, we replicate the experiment m times, and compute in each replication the integrated square error

$$\int_{\mathbb{R}} \left|\hat{f}(x) - f(x)\right|^2 \mathrm{d}x$$

of the estimate $\hat{f}$ for the normal mixture density f. Finally, the MISE is obtained as the sample average over all m replications. The reported values in Table 3.1 are based on $m = 500$ replications.

Table 3.1: The MISE of each estimate for the 15 univariate normal mixture densities from Marron and Wand (1992). The reported values are scaled by 10^4. The approaches with the smallest MISE are highlighted.

Density	$\hat{f}_{\text{rodeo}}$	$\hat{f}_{\text{nr}}$	$\hat{f}_{\text{pi}}$	$\hat{f}_{\text{hist}}$	$\hat{f}_{\text{nmm}}$	$\hat{f}_{\text{knn}}$	$\hat{f}_{\text{det}}$	$\hat{f}_{\text{vine}}$	$\hat{f}_{\text{spline}}$
#1	49	33	36	118	13	282	237	36	23
#2	77	55	57	168	63	407	320	57	40
#3	543	1360	544	756	441	679	733	557	282
#4	536	1168	238	615	133	577	729	238	202
#5	1265	3646	425	1028	170	2372	2651	425	262
#6	72	55	48	119	40	240	254	48	47
#7	192	333	72	587	37	309	431	72	63
#8	81	82	67	142	95	272	245	67	67
#9	80	77	61	140	80	248	270	61	58
#10	542	527	518	589	577	454	588	518	456
#11	75	60	60	133	66	265	252	60	59
#12	205	191	182	235	257	320	363	182	165
#13	117	91	80	156	72	278	273	80	79
#14	798	754	396	792	382	478	662	396	305
#15	918	1058	501	998	279	568	679	502	322

$\hat{f}_{\text{nmm}}$ and $\hat{f}_{\text{spline}}$ have the best performance over all 15 densities. The two approaches took on average 500 and 300 milliseconds to fit a density on a $k = 75$ point grid. $\hat{f}_{\text{pi}}$ and $\hat{f}_{\text{vine}}$, which coincide

(up to numerical imprecisions) in the univariate case, yield also quite strong results. It took both approaches approximately 10 milliseconds to fit a density. With regard to its MISE, $\hat{f}_{\text{rodeo}}$ ranks consistently in the middle of the considered approaches, and it took 400 milliseconds to fit a density. $\hat{f}_{\text{hist}}$ and $\hat{f}_{\text{det}}$ are, by construction, piece-wise constant functions and thus fail to account for the smoothness of the normal mixture densities, which results in bad fits. The approaches took about 5 and 45 milliseconds to fit a density. $\hat{f}_{\text{nr}}$ tends to oversmooth the true density in nearly all cases, while $\hat{f}_{\text{knn}}$ tends to undersmooth it – with this resulting in large estimation errors for both approaches. It took the two approaches about 1 and 5 milliseconds, respectively, to fit a density.

3.2 Bivariate Density Satisfying Assumption 1

Assumption 1 is not meaningful for the univariate case. In this example we therefore consider a simple bivariate density that meets the requirements of Assumption 1. Our goal is to study how well the considered approaches fit the irrelevant dimension. For this purpose, consider the beta mixture density from Loader (1996) given by

$$f_1(x_1) = \left(\frac{2}{3} \frac{\Gamma(3)}{\Gamma(1)\,\Gamma(2)} (1 - x_1) + \frac{1}{3} \frac{\Gamma(20)}{\Gamma(10)\,\Gamma(10)} x_1^9 (1 - x_1)^9 \right) \mathbb{1}(0 \leq x_1 \leq 1). \tag{3.1}$$

Here Γ denotes the usual gamma function. Furthermore, consider the uniform density given by

$$f_2(x_2) = \mathbb{1}(0 \leq x_2 \leq 1). \tag{3.2}$$

The product of (3.1) and (3.2) defines a density (product density), which satisfies Assumption 1. To see why, let R be the transposed first canonical basis vector of $\mathbb{R}^2$. Furthermore, let $g = f_1$, and $h(x) = \mathbb{1}(0 \leq x_1 \leq 1)\,\mathbb{1}(0 \leq x_2 \leq 1)$. Then it holds that $g(Rx)h(x)$ is equal to the product density for each x in $\mathbb{R}^2$; the first dimension is the relevant dimension and the second dimension the irrelevant dimension. Figure 3.2 visualizes the product density.

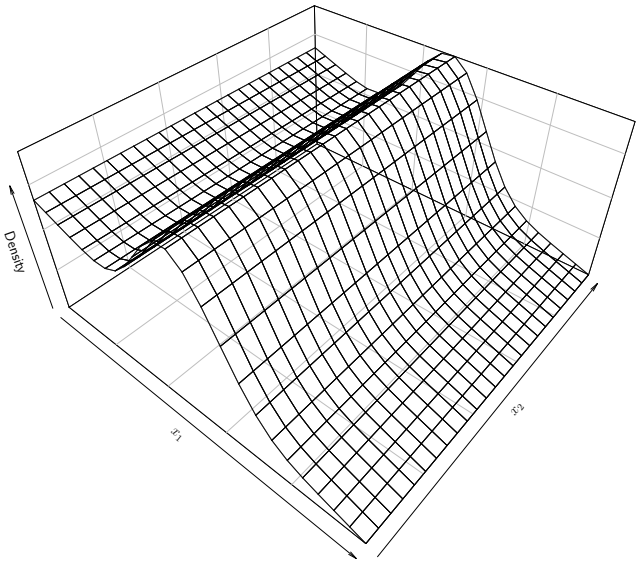


Figure 3.2: The product density of (3.1) and (3.2).

To simulate random draws from the distribution associated with the product density, we use the fact that the dimensions are, by construction, independent. Sample draws for the relevant dimension are generated from the distribution associated with (3.1), and sample draws for the irrelevant dimension are generated from the distribution associated with (3.2). We simulate $n = 500$ iid draws. To remove location, scale, and correlation effects from the sample, we center the data and rotate it using the PCA transformation. The PCA transformation is given by $U\Lambda^{-\frac{1}{2}}$, where U denotes the matrix of eigenvectors of the sample variance-covariance matrix and Λ the diagonal matrix of eigenvalues.

The estimates for the product density are visualized in Appendix B.[7] Since we are interested in how well the irrelevant dimension is fitted, we estimate the marginal density. To obtain an estimate $\hat{f}_1$ for the marginal density (3.2), we integrate the corresponding estimate $\hat{f}$ for the product density numerically with respect to the first dimension, i.e.

$$\hat{f}_2(x_2) = \int_{\mathbb{R}} \hat{f}\big((x_1, x_2)\big)\, \mathrm{d}x_1.$$

Figure 3.3 shows the numerically integrated estimates for (3.2). To quantify the goodness of fit of the numerically integrated estimates for (3.2), we employ again the MISE as accuracy measure.

[7] Ram and Gray (2011) also considered the product density. Our results are very similar; see Figure B.1 and Figure B.11 in Appendix B. Liu, Lafferty, and Wasserman (2007) also considered the density. However, they used a modified version of the RODEO approach, which is not covered in this paper. The results are thus not comparable.

We compute the MISE as described in Section 3.1. The reported values in Table 3.2 are based on $m = 500$ replications.

Table 3.2: The MISE of each numerically integrated estimate for (3.2). The reported values are scaled by 10^4. The respective approaches with the smallest MISE are highlighted.

Density	$\hat{f}_{\text{rodeo}}$	$\hat{f}_{\text{nr}}$	$\hat{f}_{\text{pi}}$	$\hat{f}_{\text{hist}}$	$\hat{f}_{\text{nmm}}$	$\hat{f}_{\text{knn}}$	$\hat{f}_{\text{det}}$	$\hat{f}_{\text{vine}}$	$\hat{f}_{\text{spline}}$
(3.2)	363	589	298	209	632	258	137	247	140

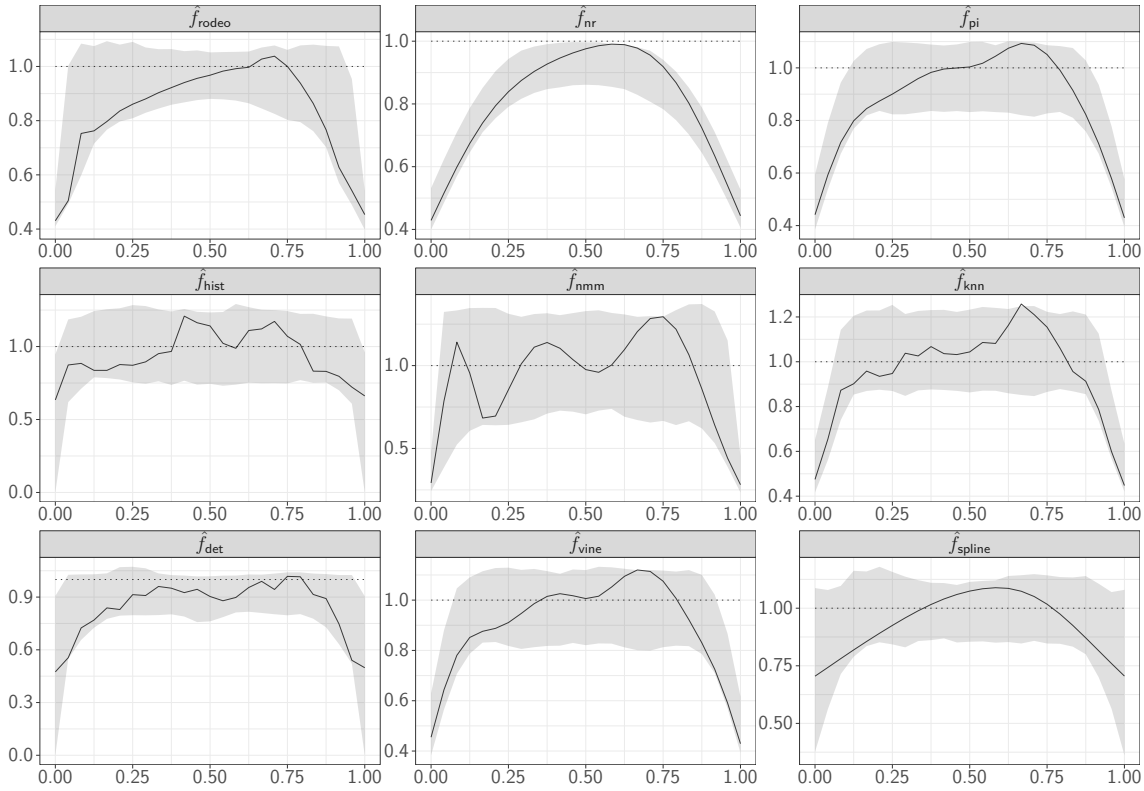


Figure 3.3: Numerically integrated estimates for (3.2). The dotted line indicates the true density. The shaded region corresponds to the 95 percent confidence region based on $m = 500$ replications.

$\hat{f}_{\text{det}}$ fits the irrelevant dimension best, closely followed by $\hat{f}_{\text{spline}}$. The two approaches took about 1 and 17 seconds to fit the product density on a $k = 25 \times 25$ point grid. $\hat{f}_{\text{nmm}}$ fits the marginal density the worst. It took, however, only 5 seconds to fit the density, while $\hat{f}_{\text{nr}}$, $\hat{f}_{\text{pi}}$, $\hat{f}_{\text{hist}}$, and $\hat{f}_{\text{vine}}$ each took about 20 seconds. $\hat{f}_{\text{knn}}$ took about 250 *milli*seconds to fit the density. $\hat{f}_{\text{rodeo}}$ took 10 minutes to fit the density, and yields the second worst fit. According to the way Theorem 1 works, we would have expected that the bandwidth of the irrelevant dimension remains close to the initial bandwidth

resulting in a good fit of the irrelevant dimension. However, this did not happen at all. We will discuss this issue further in Section 5.

3.3 Classification of Chernekov Radiation

An important application of density estimation is classification. In this example we consider a classification example from astrophysics. Our goal is to distinguish the Chernekov radiation, i.e. the faint light emitted by charged particles when they enter the atmosphere, that is caused by gamma rays from Chernekov radiation that is caused by common hadron showers. Identification of gamma rays is important in astrophysics, e.g., for observing annihilation of dark matter, the accretion of black holes, or the analysis of supernova remnants.

The data from Bock et al. (2004) imitates measurements taken on the Major Atmospheric Gamma Imaging Chernekov radiation telescopes located on the Canary Islands. The data contains $d = 10$ variables (telescope width, angle etc.) for $n = 19\,020$ observations, where 12 332 observations are gamma rays and the remaining 6688 observations are common hadron showers. Let $f(\,\cdot \mid \text{gamma ray})$ denote the density of gamma rays, and let $f(\,\cdot \mid \text{hadron shower})$ denote the density of hadron showers. Furthermore, assume that a gamma ray and a hadron shower have equal prior probabilities. Using Bayes theorem, the posterior probability for a gamma ray given the telescope data x is equal to

$$\frac{f(x \mid \text{gamma ray})}{f(x \mid \text{gamma ray}) + f(x \mid \text{hadron shower})}.$$

We use the following classification strategy. Whenever the posterior probability is larger than some threshold Δ, an observation is classified as caused by a gamma ray; otherwise, the observation is classified as caused by hadron showers. The threshold Δ controls the tradeoff between both Chernekov radiation falsely classified as caused by gamma rays (false positives), and Chernekov radiation correctly classified as caused by gamma rays (true positives). Larger values for Δ reduce the number of false positives, but larger values also reduce the number true positives. The tradeoff between both false positive and true positive rate is illustrated by the Receiver operating characteristic (ROC) curve. The larger the area under the curve (AUC), the higher the classification power. We use the AUC as performance metric to evaluate the approaches.

To implement the proposed classification strategy, we have to estimate the posterior probability by estimating $f(\,\cdot \mid \text{gamma ray})$ and $f(\,\cdot \mid \text{hadron shower})$. 12 680 observations, two thirds of the data, are used to estimate the densities. We center the training data for gamma rays and hadron showers and rotate it using the PCA transformation. The remaining 6340 observations are then used for evaluation. Figure 3.4 shows the ROC curves together with the corresponding AUC for the classifier trained by the respective approach.

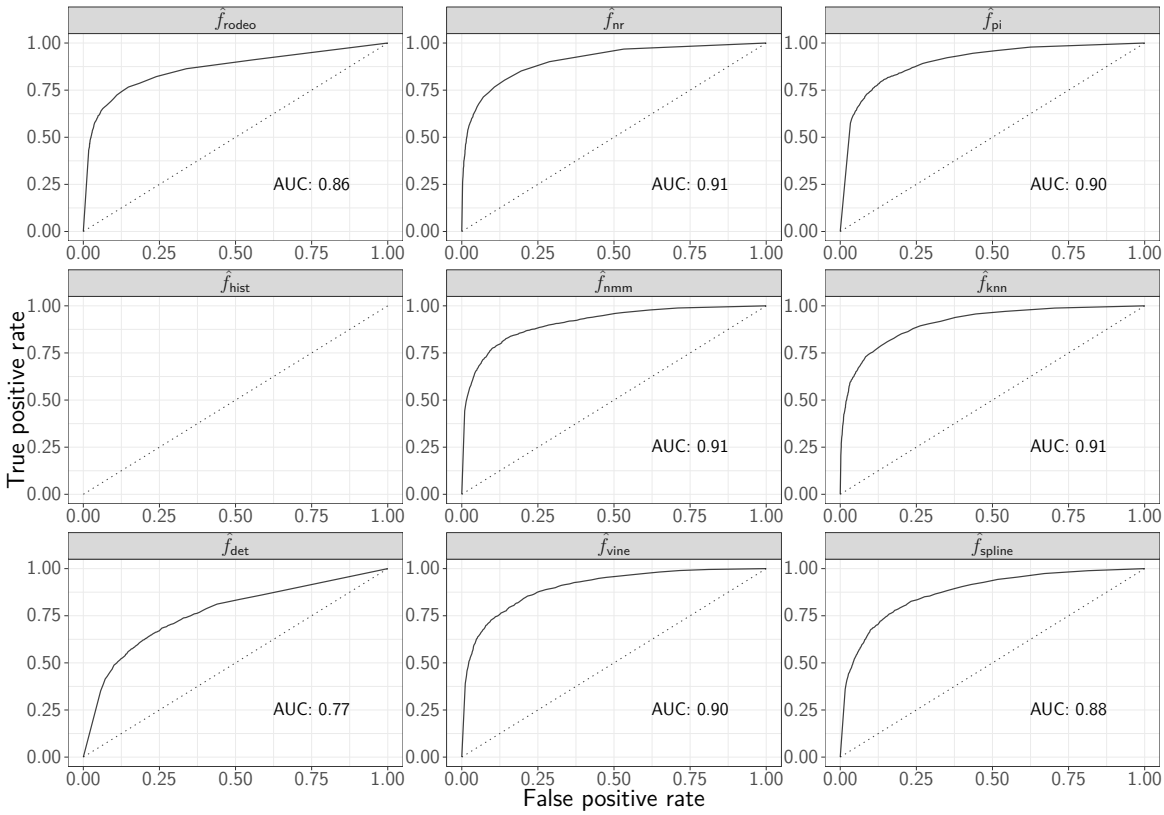


Figure 3.4: ROC curve and AUC for the classifiers trained by the respective approach.

The panel for $\hat{f}_{\text{hist}}$ is left empty on purpose to indicate that the implementation does not work for $d \geq 6$ dimensions. The classifier trained by $\hat{f}_{\text{det}}$ has by far the lowest AUC. Classifying all 6340 observations took about 1 minute. The classifier trained by $\hat{f}_{\text{rodeo}}$ has the second lowest AUC. Classifying the observations took almost four weeks. We will discuss this issue further in Section 5. The AUC for the classifier trained by $\hat{f}_{\text{spline}}$ is slightly higher. It took about 3.5 minutes to classify the observations. The classifiers trained by $\hat{f}_{\text{pi}}$ and $\hat{f}_{\text{vine}}$ achieve the same AUC. However, the computation time differs significantly – it took the former several hours to classify all observations, while the latter needed about 20 minutes. The classifiers trained by $\hat{f}_{\text{nr}}$, $\hat{f}_{\text{knn}}$, and $\hat{f}_{\text{nmm}}$ achieve the highest AUC. Classifying the observations took about 1, 7, and 90 minutes for these approaches respectively.

3.4 Pattern Recognition Using Handwritten Zip Code Digits

In our last example we consider 16×16 pixel images of handwritten zip codes that were automatically scanned from the envelopes by the US Postal Service. The data from Le Cun et al. (1990) contains 7291 scanned digits, which have been deslanted, had their size normalized, and finally are transformed to 16×16 pixel grayscale images. Figure 3.5 shows the per-pixel average for each digit in the data

set.

Figure 3.5: Per-pixel average for each digit in the data set from Le Cun et al. (1990).

We center and rotate the sample for each digit using a regularized ZCA transformation to remove location, scale, and correlation effects of the pixels. The regularized ZCA transformation is given by $U(\Lambda + \varkappa I_d)^{-\frac{1}{2}}U'$, where U denotes the matrix of eigenvectors of the sample variance-covariance matrix of the respective digit, and Λ the diagonal matrix of eigenvalues. Since the sample variance-covariance is numerically singular, we apply regularization to the eigenvalues by adding $\varkappa = 0.1$ to each eigenvalue as suggested by Pal and Sudeep (2016).

Each image in the data set can be seen as random draw from a 256-dimensional distribution. Suppose our goal is the generation of artificial handwritten digits by simulating sample draws from the the corresponding distributions. For that purpose, we estimate the associated densities and employ importance sampling to simulate random draws. That is, we simulate m iid draws $X_1, X_2, \ldots, X_m$ from a 256-dimensional normal distribution with zero mean and identity variance-covariance matrix (proposal distribution), and compute the ratios $\frac{\hat{f}(X_1)}{f(X_1)}, \frac{\hat{f}(X_2)}{f(X_2)}, \ldots, \frac{\hat{f}(X_m)}{f(X_m)}$, where $\hat{f}$ is an estimated density, and f is the density of the proposal distribution. The computed ratios certainly do not sum to one, so we normalize them to obtain probabilities. A sample draw is then simulated by sampling from $X_1, X_2, \ldots, X_m$ with the computed probabilities. Each draw is based on $m = 500$ iid draws

from the proposal distribution.

The per-pixel sample average of $l = 100$ iid draws from the distribution of the digit "6" estimated by $\hat{f}_{\text{nr}}$, $\hat{f}_{\text{nmm}}$, $\hat{f}_{\text{knn}}$, and $\hat{f}_{\text{det}}$, respectively, is visualized in Figure 3.6. All the other approaches did not work for this extremely high dimensional data. $\hat{f}_{\text{rodeo}}$ fails because of infeasibly long computation times and because of arithmetic underflow.[8] We will discuss this issue further in Section 5. Plots of per-pixel sample averages for the other digits can be found in Appendix B.

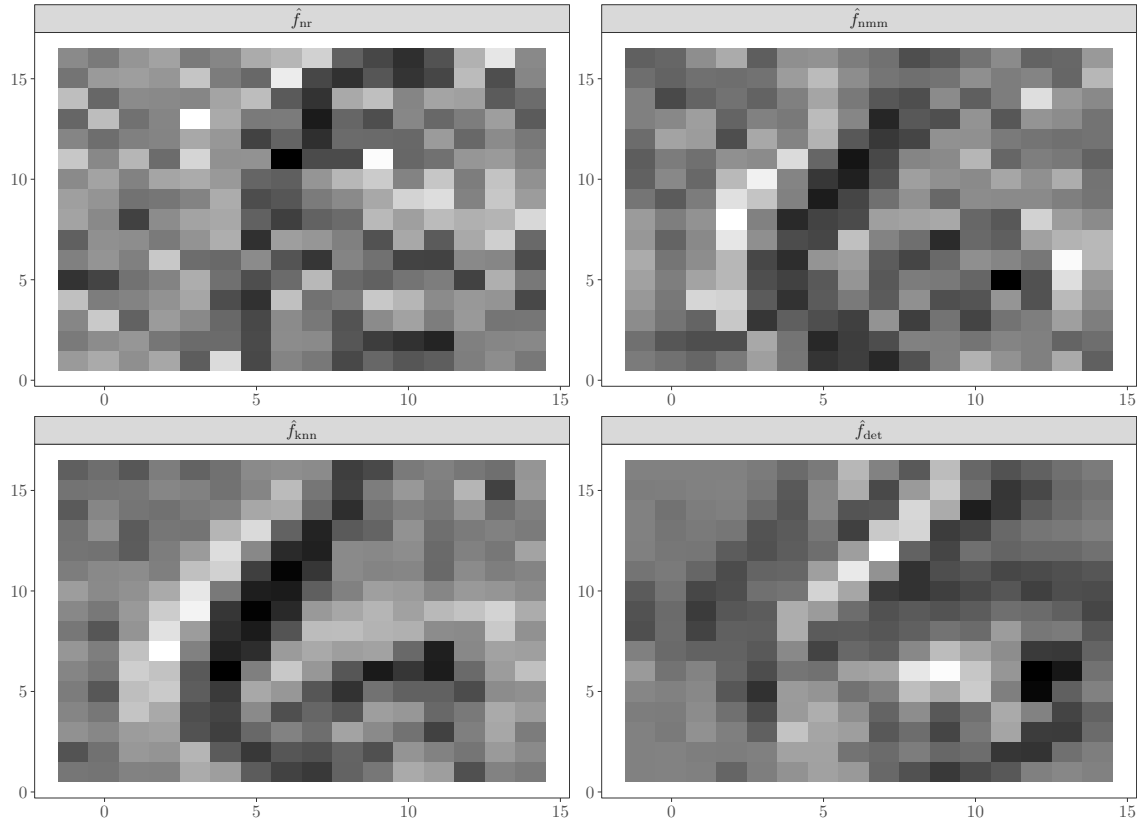


Figure 3.6: Per-pixel sample average based on $l = 100$ random draws from the distribution of the digit "6" estimated by the respective approaches.

The results are not very strong even for the approaches that do work for this high dimensional example. The respective digits are not always recognizable. This is likely due to the fact that only about 700 observation were available on average to estimate the density for each digit. Another potential source of poor results is that the normal distribution is possibly not a good choice for the proposal distribution. We leave it to the reader to decide which of the four working approaches yields the best results. About 10 days in total were needed to simulate the data for all digits.

[8] Liu, Lafferty, and Wasserman (2007) considered a similar example using their implementation. They were able to obtain estimates. However, given the limited documentation in Liu, Lafferty, and Wasserman (2007), it is not possible to reproduce their results using our implementation.

4 Variation of Input Parameters

Until now we have used the same values for the input parameters H_{init}, β, θ, ζ, $\hat{\Sigma}$, and q in all examples. This section revisits the 15 univariate normal mixture densities from Marron and Wand (1992) considered in Section 3.1 with different parameterizations. One parameter is varied at a time, and for each parameter how the results change is briefly discussed.

Variation of H_{init}. Until now we set this parameter to $H_{\text{init}} = \sqrt{h_0 \log\big(\log(n)\big)^{-1}}$ with $h_0 = 1$. Since we have transformed the data such that the sample has unit variance, choosing $h_0 = 1$ corresponds to choosing an initial bandwidth parameter slightly below the standard deviation of the sample. Larger values, unsurprisingly, caused significantly longer computation times, but eventually resulted in the same estimates (see Table B.1 in Appendix B). Smaller values were not considered as the virtue of the RODEO approach relies on large initial bandwidths.

Variation of β. $\beta = \beta_0$ with $\beta_0 = 0.9$ has been used to this point. We also considered the very small value $\beta_0 = 0.01$, the very large value $\beta_0 = 0.99$, and the intermediate value $\beta_0 = 0.5$ to analyze the behavior at various rates of decay. The small β_0 made the estimates very volatile, while the large β_0 resulted in slightly smoother estimates, but also significantly longer computation times. Results are slightly improved using the intermediate β_0 (see Table B.2 in Appendix B). An open question is how to set this parameter optimally.

Variation of θ. Our implementation features the following thresholds to tune the bias reduction:

- Hard: $\theta = \hat{\mu}$. The hard threshold is discontinuous at the threshold value λ and is thus very sensitive to small changes in the data.
- Soft: $\theta = |\hat{\mu}| - |\hat{\mu}| \odot \lambda \odot \hat{\mu}^{-1}$. The soft threshold is continuous, but unnecessarily biases the estimate for large inputs.
- Garrote: $\theta = \hat{\mu} - \lambda^2 \odot \hat{\mu}^{-1}$. The garrote threshold seeks to remedy the drawbacks of both hard and soft thresholds.
- Hyperbole: $\theta = (|\hat{\mu}| - |\hat{\mu}|^2 \odot \lambda^2 \odot \hat{\mu}^{-2})^{\frac{1}{2}}$. Similar to the garrote threshold, the hyperbole threshold is a compromise between the hard and soft thresholds.

In all of the above expressions the powers are understood component-wise. For some of the densities, the garrote threshold yields slightly better results than the hard threshold; the soft and hyperbole thresholds almost always perform worse (see Table B.3 in Appendix B). Contrary to our expectation, however, this parameter is apparently only of minor importance.

Variation of ζ. This parameter is important from a theoretical perspective, but is a simple scaling factor in practice. Small values resulted in small thresholds, which caused a cascade of decreasing bandwidths. Large values, on the other hand, made the algorithm stop after the first iteration. In general, Algorithm 1 is very sensitive to changes in this parameter. Until now we used $\zeta = \sqrt{2d\log(\log(n))}$. We also considered the slightly larger value $\zeta = \sqrt{2d\log(n)}$, and $\zeta = 1$. However, the results did not improve at all (see Table B.4 in Appendix B).

Variation of $\hat{\Sigma}$. We have implemented the following estimators for the scale of $\hat{\mu}$:

- The standard deviation.
- The median absolute deviation divided by 1.483. The median absolute deviation is less sensitive to outliers as compared to the standard deviation. Assuming the components of $\hat{\mu}$ are normally distributed, dividing the median absolute deviation by 1.483 makes it a consistent estimator for the standard deviation (Scott, 2015).
- The interquartile range divided by 1.349. As the median absolute deviation, the interquartile range is less sensitive to outliers. Assuming the components of $\hat{\mu}$ are normally distributed, dividing the interquartile range by 1.349 makes it a consistent estimator for the standard deviation (Scott, 2015).
- The minimum of standard deviation and interquartile range divided by 1.349.
- The standard deviation (variant 2). In case of constant fitting, an explicit expression for $\hat{\mu}$ can be obtained using standard matrix differentiation rules and the linearity of the diag operator:

$$
\begin{aligned}
\hat{\mu}(\Xi) &= \operatorname{diag}\Big(\frac{\mathrm{d}}{\mathrm{d}\Xi}\hat{f}_{P\Xi P'}(x)\Big) \\
&= \operatorname{diag}\left(n^{-1}\sum_{i=1}^{n}\frac{\mathrm{d}}{\mathrm{d}\Xi}K_{P\Xi P'}(x-X_i)\right) \\
&= \operatorname{diag}\left(n^{-1}\sum_{i=1}^{n}\left(\Xi^{-1}P'(x-X_i)(x-X_i)'P\Xi^{-2}-\Xi^{-1}\right)K_{P\Xi P'}(x-X_i)\right) \\
&= n^{-1}\sum_{i=1}^{n}\operatorname{diag}\left(\Xi^{-1}P'(x-X_i)(x-X_i)'P\Xi^{-2}-\Xi^{-1}\right)K_{P\Xi P'}(x-X_i) \\
&= n^{-1}\sum_{i=1}^{n}\tilde{X}_i,
\end{aligned}
$$

  where $\tilde{X}_i \coloneqq \operatorname{diag}\left(\Xi^{-1}P'(x-X_i)(x-X_i)'P\Xi^{-2}-\Xi^{-1}\right)K_{P\Xi P'}(x-X_i)$. Given that $P = I_d$, we have that the $\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_n$ are iid. Thus, a natural estimator for the variance-covariance

matrix of $\hat{\mu}$ is

$$\hat{\Sigma}(\Xi) = n^{-1}(n-1)^{-1} \sum_{i=1}^{n} (\tilde{X}_i - \hat{\mu})(\tilde{X}_i - \hat{\mu})',$$

because the variance of a sample mean in case of iid random variables is equal to the variance divided by the sample size.

- The standard deviation (variant 3). The disadvantage of variant 2 is that it is only valid in case of constant fitting and if $P = I_d$. Liu, Lafferty, and Wasserman (2007, full version) derive an expression for the variance-covariance matrix of $\hat{\mu}$ (ibid., equation 101 on page 25). This expression can be used to obtain the following estimator for the variance-covariance matrix of $\hat{\mu}$:

  $$\hat{\Sigma}(\Xi) = n^{-1}(4\pi)^{-\frac{d}{2}} \det(\Xi^{-1}) \hat{f}_{P\Xi P'}(x) P \Xi^{-2} P'.$$

  However, as we will show in Section 5.2, the initial expression for the variance-covariance matrix of $\hat{\mu}$ derived by Liu, Lafferty, and Wasserman (2007, full version) is wrong. We discuss this issue further in Section 5. The estimator based on this expression is thus invalid. Nevertheless, we implemented this variant.

Using the median absolute deviation as an estimator for the scale of $\hat{\mu}$ yields slightly better results compared to the standard deviation. The interquartile range, on the other hand, performs worse (see Table B.5 in Appendix B). Estimating the standard deviation of $\hat{\mu}$ based on the bootstrap procedure and based on variant 2 yields almost identical results, which suggests that $b = 100$ replications are sufficient to obtain a reasonable estimate for the standard deviation of $\hat{\mu}$, at least when using constant fitting. Estimating the standard deviation based on variant 3, on the other hand, lead to very different results. This is very likely due to the fact that the initial expression for the variance-covariance matrix of $\hat{\mu}$ derived by Liu, Lafferty, and Wasserman (2007, full version) is wrong. Variant 2 and variant 3 are not based on the bootstrap procedure and are thus substantially faster with regard to computation time.

Variation of q. Our implementation allows the following functional forms for the LPDE:

- Constant fitting ($q = 0$). The LPDE is given by (2.4).
- Linear fitting ($q = 1$). The LPDE is given by (2.5).
- Quadratic fitting ($q = 2$). The LPDE is given by (2.6).

The goodness of fit increases substantially when using linear or quadratic fitting. The goodness of fit is particularly good for the latter (see Figure B.24, Figure B.25, Table B.6 in Appendix B). However, we have encountered a situation in which the matrix A as defined in (2.6) does not exist because of numerical issues.[9] This shortcoming aside, changing q had the potential of producing the most positive effects on the results of all parameter changes considered in this section.

[9] A possible solution to this problem could be to apply Tikhonov regularization, i.e. blow up the main diagonal elements of $Q'WQ - Q'W1_n1_n'WQ$ additively by a positive constant to ensure it is invertible. We will consider implementing this in a future version of our package.

5 Problems

We have seen in Section 3 that our implementation is not consistent with the theoretical framework proposed by Liu, Lafferty, and Wasserman (2007) – in Section 3.2 we have considered a density that satisfies Assumption 1, but the bandwidth of the irrelevant dimension were reduced as opposed to how Theorem 1 works. Furthermore, in Section 3.3 we have encountered the problem that our implementation is practically infeasible in higher dimensions due to dramatically long computaton times, and in Section 3.4 because of arithmetic underflow. This section examines the problems and shows that the RODEO approach for nonparametric density estimation has severe theoretical and practical problems.

5.1 Computational Limitations

Despite being hard coded in C/C++, our implementation is relatively slow. The main reason for this is that our implementation is based on a naive query. Consider the parameter configuration from Section 3. Since Algorithm 1 is an iterative procedure, it is difficult to derive its time complexity. However, we know that there are $\Omega(nkdb)$ operations necessary to evaluate (2.7) using Algorithm 1, which is already very slow. Thus, as the sample size n, the number of query points k, the dimensions d, or the number of bootstrap replications b increases, our implementation quickly becomes practically infeasible due to dramatically long computation times.

Another problem with our implementation is arithmetic underflow. Arithmetic underflow occurs if the result of a computation is a magnitude smaller than the smallest number representable by the computer and thus automatically rounded to zero. In the example considered in Section 3.4, the values of $\hat{\mu}$ were of magnitude 10^{-156} and the variance was of magnitude 10^{-312}. The latter is, however, no longer representable. Consequently, the algorithm fails since the resulting scale estimates are rounded to zero.

5.2 Theoretical Shortcomings

Suppose we are in the setting of Theorem 1. The theorem is based on a concentration inequality to show that the probability that $\hat{\mu}$ exceeds λ in magnitude converges to zero for the irrelevant dimensions. The concentration inequality invokes the mean μ and the variance-covariance matrix Σ of $\hat{\mu}$. Liu, Lafferty, and Wasserman (2007, full version) therefore derive an explicit expression for μ based on the mean of (2.4). The mean of (2.4) is given by

$$\mathbb{E}[\hat{f}_H(x)] = f(x) + \frac{1}{2} H : \mathcal{H} f(x) H + \mathsf{o}(H : H), \tag{5.1}$$

where $:$ denotes the Frobenius inner product and $\mathcal{H}f(x)$ the Hessian of $f(x)$ (see e.g., Wand and Jones, 1994; Scott, 2015). Taking the derivative with respect to H, and interchanging derivative and expectation operators on the right hand side yields

$$\mathbb{E}\left[\frac{\mathrm{d}}{\mathrm{d}H}\hat{f}_H(x)\right] = \mathcal{H}f(x)H + \frac{\mathrm{d}}{\mathrm{d}H}\,\mathsf{o}(H : H) \tag{5.2}$$

which equals μ up to vectorization using the diag operator. Liu, Lafferty, and Wasserman (2007, full version) make the mistake of also interchanging the derivative operator and the Landau symbol o (ibid., equation 24 on page 24). Interchanging the derivative operator and o is, in general, not valid and can have severe consequences – the derivative operator is not continuous which implies that the derivative of a convergent function may not necessarily be convergent. For instance, consider the function $T(z) = z^2 \sin(z^{-1})$ which is clearly $\mathsf{o}(z)$ as z approaches zero. But

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}z}T(z) &= 2z\sin(z^{-1}) - \cos(z^{-1}) \\ &= -\cos(z^{-1}) + \mathsf{o}(z)\end{aligned}$$

diverges as z approaches zero, i.e. $\frac{\mathrm{d}}{\mathrm{d}z}T$ is *not* $\mathsf{o}(1)$ as one would expect if the derivative operator and o were interchangeable. By definition of Landau symbols, we ultimately cannot rule out that the o term in (5.1) behaves similar to the term in this example.

The theoretical properties of Algorithm 1 are thus unlikely to hold. In fact, we can demonstrate how the above error leads to a fallacy in the proof of Theorem 1 as presented in Liu, Lafferty, and Wasserman (2007, full version). They deduce that "for sufficiently large n it's obvious that $\lambda_j \geq 2\mu_j$" and "for sufficiently large n [and without loss of generality], $\mu_j \geq ch_j f_{jj}(x)\,[> 0]$" (ibid., page 26). The notation comes from the original paper; the meaning of each symbol is not too important as these are all real numbers. They further derive the inequality "$ch_j f_{jj}(x) \geq 2\lambda_j$" (ibid., page 27). However, taking all three inequalities together – that is, $\lambda_j \geq 2\mu_j \geq 2ch_j f_{jj}(x) \geq 4\lambda_j$ – we arrive at $1 \geq 4$ which is clearly a contradiction. Thus, the proof of Theorem 1 as shown in Liu, Lafferty, and Wasserman (2007, full version) is invalid.

5.3 Rotating the Data Makes Algorithm 1 Fail

There are considerable doubts about the validity of the approach for nonparametric density estimation not only from a theoretical point of view, but also considering practical applications. If Algorithm 1 indeed works as described in Section 2.3, a rotation of the data should not have any consequences on the algorithm as rotating the data does not affect the local variation of the target density.[10] However,

[10] The only effect rotating the data has on the target density is that it scales it by a constant, which is given by the determinant of the rotation.

we now demonstrate that a simple rotation of the data will make Algorithm 1 fail. For that, we replicate Example 4 from Liu, Lafferty, and Wasserman (2007). The density considered in that example is given by:

$$f(x) = \prod_{j=1}^{5} \frac{1}{\sqrt{2\pi\omega^2}} \exp\left(-\frac{1}{2}\frac{x_j^2}{2\omega^2}\right) \prod_{j=6}^{30} 1(0 \leq x_j \leq 1), \tag{5.3}$$

where $\omega_j = 0.02j$. Like Liu, Lafferty, and Wasserman (2007), we simulate $n = 100$ iid draws from the corresponding distribution, and replicate the experiment $m = 30$ times. Since (5.3) is the product of univariate normal densities and univariate standard uniform densities, we can simulate samples from the corresponding marginal distribution of each dimension.

We estimate (5.3) at the query point $x = 0_{30}$. Figure 5.1 shows boxplots of bandwidth parameter value in the last iteration of Algorithm 1 when using constant, linear, and quadratic fitting. The other input parameters are chosen as in Section 3.

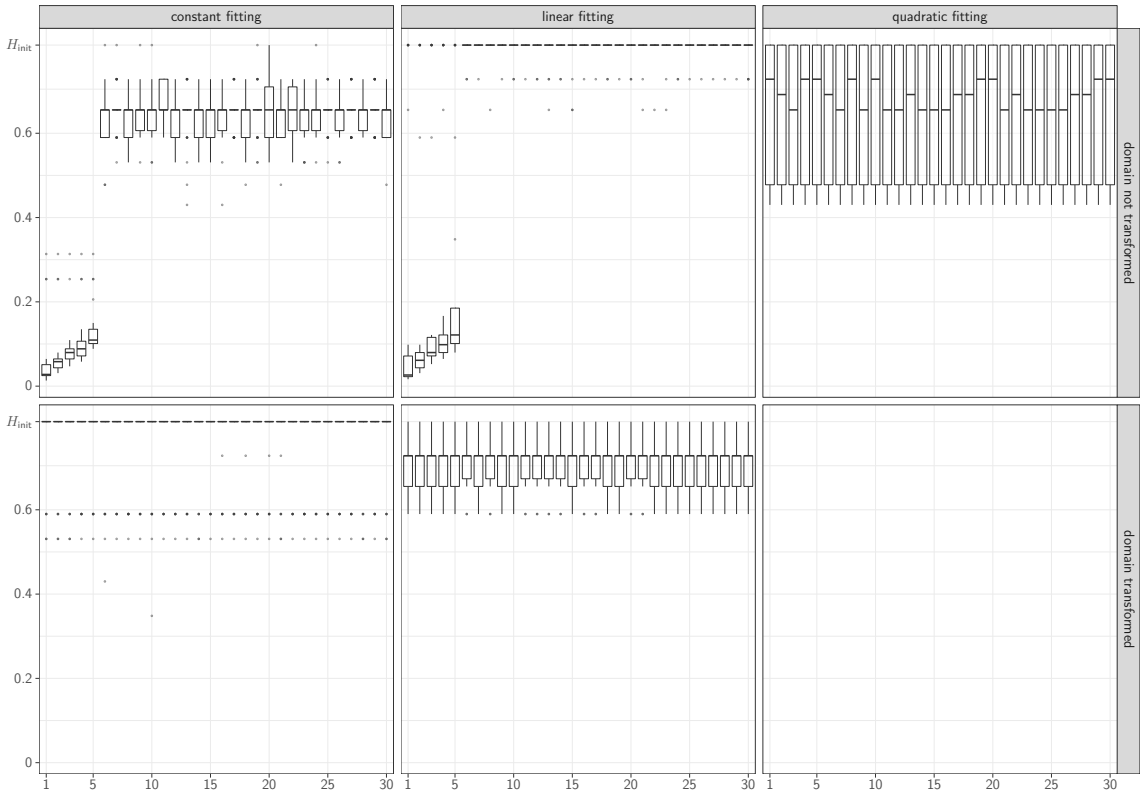


Figure 5.1: Boxplots of the bandwidth parameter value in the last iteration of Algorithm 1. The boxplots are based on $m = 30$ replications.

Since (5.3) satisfies Assumption 1 at the query point 0_{30}, we would expect the algorithm to shrink the bandwidths of the first five dimensions, while the bandwidths of the other 25 dimension remain

close to the initial bandwidth parameter $H_{\text{init}} = \sqrt{\log\big(\log(100)\big)^{-1}} I_{30} \approx 0.81 I_{30}$. Consider the upper panels first. The algorithm does in fact shrink the bandwidths for the relevant dimensions in the constant and linear fitting case – the linear fitting case is consistent with the result reported by Liu, Lafferty, and Wasserman (2007). In the quadratic fitting case, no pattern can be recognized since all dimensions are affected equally. Now consider the lower panels. In the quadratic fitting case, no results are reported due to numerical issues regarding the existence of the LPDE (see Section 4). Once the domain of (5.3) is rotated using, for instance, the PCA transformation, the algorithm no longer works as expected in either the constant or the linear fitting case – the bandwidths are reduced for all dimensions. Changing the evaluation point and increasing the sample size does not change this finding.

We suspect that the previously witnessed bandwidth reductions were solely driven by different scales. Note that an irrelevant dimension has a variance of about 0.02, while the relevant dimensions have a much smaller variance of $(0.02j)^2$. The smaller the variance, the smaller the variance of the corresponding component of $\hat{\mu}$. This in turn results in small thresholds which encourages a bandwidth reduction. This effect disappears once scale effects are removed. This argument is, however, only of heuristic nature. A formal proof for our claim stands out.

6 Conclusion

In this paper we considerd the RODEO approach applied to local polynomial density estimation. We developed the R package `lpderodeo`, which implements the approach. The package can be built from source using the files attached to Appendix C. We applied our implementation to a few examples at various levels of dimensionality, and evaluated its performance in a comparative study. Our findings suggest that our implementation does not work well in comparison to most of the other considered approaches with regard to the applied performance metrics. Furthermore, our implementation suffers from long computation times due to a naive query.

Our main finding is, however, that there are severe shortcomings with the theoretical framework proposed by Liu, Lafferty, and Wasserman (2007) for the RODEO approach applied to local polynomial density estimation. Specifically, we can show that their proof of Theorem 1, which justifes the application of the approach in high dimensions, is flawed – interchanging the derivative operator and the Landau symbol o resulted in a fallacy. There is thus no reason to assume that the estimator output by Algorithm 1 achieves the faster rate of convergence of $\mathsf{O}\left(n^{-\frac{4}{4+r}}\right)$ as claimed. In fact, we demonstrated that a simple rotation of the data made the algorithm fail in practice.

Nevertheless, by varying some of the input parameters, very good results were obtained, which motivates us to not yet give up on our implementation. To further improve it, the following points should be considered in future versions:

- **Implement multi-core processing.** To improve computational efficiency, multi-core processing should be implemented. Algorithm 1 could then process multiple dimension at the same time rather than iteratively, thus speeding up computation time.

- **Implement a k-d query.** Instead of using a naive query, a query based on a k-d tree could be implemented as described in Gray and Moore (2003). This hopefully leads to major improvements in compuation time.

- **Derive the first and second moments of $\hat{\mu}$.** If an expression for the variance of $\hat{\mu}$ was known, we could use the formula instead and avoid the expensive bootstrap procedure.

- **Combat arithmetic underflow.** To solve the arithmetic underflow problem it may already suffice to take logs on both sides of (2.7), noting that the constant term $(2\pi)^{-\frac{d}{2}}$ can be factored out.

Considering the suggestions, we are convinced that our implementation could develop into a strong approach for nonparametric density estimation once it is fully mature.

References

[1] Bedford, T. and Cooke, R. "Vines – A New Graphical Model for Dependent Random Variables." *Annals of Statistics* 30.4 (2002).

[2] Bellman, R. *Dynamic Programming*. Princeton University Press, 1957.

[3] Bock, R., Chilingarian, A., Gaug, M., et al. "Methods for Multidimensional Event Classification: A Case Study Using Images From a Cherenkov Gamma-ray Telescope." *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 516.2-3 (2004).

[4] Chacón, José and Duong, Tarn. "Multivariate Plug-in Bandwidth Selection With Unconstrained Pilot Bandwidth Matrices." *TEST* 19.2 (2010).

[5] Fraley, C. and Raftery, A. "Model-based Clustering, Discriminant Analysis, and Density Estimation." *Journal of the American Statistical Association* 97.458 (2002).

[6] Freedman, David and Diaconis, Persi. "On the Histogram As a Density Estimator: L_2 Theory." *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete* 57.4 (1981).

[7] Gray, A. and Moore, A. "Rapid Evaluation of Multiple Density Models." *AISTATS*. 2003.

[8] Gu, C. *Smoothing Spline ANOVA Models*. Springer Science & Business Media, 2013.

[9] Hjort, N. and Jones, C. "Locally Parametric Nonparametric Density Estimation." *Annals of Statistics* 24.4 (1996).

[10] Lafferty, J. and Wasserman, L. "Rodeo: Sparse, Greedy Nonparametric Regression." *Annals of Statistics* 36.1 (2008).

[11] Le Cun, Y., Matan, O., Boser, B., et al. "Handwritten Zip Code Recognition With Multilayer Networks." *Proceedings of the 10th International Conference on Pattern Recognition*. Vol. 2. 1990.

[12] Liu, H., Lafferty, J., and Wasserman, L. "Sparse Nonparametric Density Estimation in High Dimensions Using the Rodeo." *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2007.
Full version: http://www.cs.cmu.edu/~hanliu/papers/jsm07.pdf.

[13] Loader, C. "Local Likelihood Density Estimation." *Annals of Statistics* 24.4 (1996).

[14] Loader, C. *Local Regression and Likelihood*. Springer Science & Business Media, 2006.

[15] Loftsgaarden, D. and Quesenberry, C. "A Nonparametric Estimate of a Multivariate Density Function." *Annals of Mathematical Statistics* 36.3 (1965).

[16] Mahapatruni, G. and Gray, A. "Cake: Convex adaptive kernel density estimation." *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2011.

[17] Marron, J. and Wand, M. "Exact Mean Integrated Squared Error." *Annals of Statistics* 20.2 (1992).

[18] Nagler, T. and Czado, C. "Evading the Curse of Dimensionality in Nonparametric Density Estimation With Simplified Vine Copulas." *Journal of Multivariate Analysis* 151 (2016).

[19] Pal, K. and Sudeep, K. "Preprocessing for Image Classification by Convolutional Neural Networks." *2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)* (2016).

[20] Parzen, E. "On Estimation of a Probability Density Function and Mode." *Annals of Mathematical Statistics* 33.3 (1962).

[21] Ram, P. and Gray, A. "Density Estimation Trees." *Proceedings of the 17th SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2011.

[22] Rosenblatt, M. "Remarks on Some Nonparametric Estimates of a Density Function." *Annals of Mathematical Statistics* 27.3 (1956).

[23] Scott, D. *Multivariate Density Estimation: Theory, Practice, and Visualization*. John Wiley & Sons, 2015.

[24] Silverman, B. "On the Estimation of a Probability Density Function by the Maximum Penalized Likelihood Method." *Annals of Statistics* 10.3 (1982).

[25] Silverman, B. *Density Estimation for Statistics and Data Analysis*. CRC Press, 1986.

[26] Stone, C. "Optimal Rates of Convergence for Nonparametric Estimators." *The Annals of Statistics* (1980).

[27] Wand, M. and Jones, C. *Kernel Smoothing*. CRC Press, 1994.

A Derivation of the LPDE

This appendix derives the LPDE with constant fitting (2.4), the LPDE with linear fitting (2.5), and the LPDE with quadratic fitting (2.6) in case a normal kernel is used.

Derivation of the LPDE with constant

When a constant polynomial $\Pi(\mathsf{x}) = \alpha_0$ is used to approximate the log density locally, maximizing (2.2) is equivalent to solving the equation

$$n^{-1} \sum_{i=1}^{n} K_H(X_i - x) = \int_{\mathbb{R}^d} K_H(u - x) \exp(\hat{\alpha}_0) \, \mathrm{d}u \tag{A.1}$$

for $\exp(\alpha_0)$. We immediately obtain (2.4) as the right hand side of (A.1) evaluates to $\exp(\hat{\alpha}_0)$, which is, by definition, the LPDE for $f(x)$.

Derivation of the LPDE with linear fitting

When a linear polynomial $\Pi(\mathsf{x}) = \alpha_0 + \alpha_1 \bullet \mathsf{x}$ is used to approximate the log density locally, maximizing (2.2) is equivalent to solving the following system

$$n^{-1} \sum_{i=1}^{n} K_H(X_i - x) = \int_{\mathbb{R}^d} K_H(u - x) \exp\big(\hat{\alpha}_0 + \hat{\alpha}_1 \bullet (u - x)\big) \, \mathrm{d}u \tag{A.2}$$

$$n^{-1} \sum_{i=1}^{n} K_H(X_i - x)(X_i - x) = \int_{\mathbb{R}^d} K_H(u - x)(u - x) \exp\big(\hat{\alpha}_0 + \hat{\alpha}_1 \bullet (u - x)\big) \, \mathrm{d}u \tag{A.3}$$

of $1 + d$ equations for $\exp(\hat{\alpha}_0)$. We evaluate the right hand side of (A.2) first:

$$\begin{aligned}
&\int_{\mathbb{R}^d} K_H(u - x) \exp\big(\hat{\alpha}_0 + \hat{\alpha}_1 \bullet (u - x)\big) \, \mathrm{d}u \\
&\quad = \int_{\mathbb{R}^d} (2\pi)^{-\frac{d}{2}} \det(H^{-1}) \exp\Big(-\frac{1}{2} H^{-1}(u - x) \bullet H^{-1}(u - x)\Big) \exp\big(\hat{\alpha}_0 + \hat{\alpha}_1 \bullet (u - x)\big) \, \mathrm{d}u \\
&\quad = \int_{\mathbb{R}^d} (2\pi)^{-\frac{d}{2}} \exp\Big(-\frac{1}{2} v \bullet v + \frac{1}{2} H\hat{\alpha}_1 \bullet H\hat{\alpha}_1 + \hat{\alpha}_0\Big) \, \mathrm{d}v \\
&\quad = \exp(\hat{\alpha}_0 + \frac{1}{2} H\hat{\alpha}_1 \bullet H\hat{\alpha}_1) \int_{\mathbb{R}^d} (2\pi)^{-\frac{d}{2}} \exp\Big(-\frac{1}{2} v \bullet v\Big) \, \mathrm{d}v \\
&\quad = \exp\Big(\hat{\alpha}_0 + \frac{1}{2} H\hat{\alpha}_1 \bullet H\hat{\alpha}_1\Big) .
\end{aligned}$$

Note that a change of variables ($u = x + Hv + H^2\hat{\alpha}_1$) was used in the second step. To evaluate (A.3) we differentiate the just derived expression with respect to $\hat{\alpha}_1$, and interchange derivative and integral operators using the dominated convergence theorem:

$$\begin{aligned}
&\int_{\mathbb{R}^d} K_H(u-x)(u-x)\exp\big(\hat{\alpha}_0 + \hat{\alpha}_1(u-x)\big) \\
&\quad = \int_{\mathbb{R}^d} \frac{\mathrm{d}}{\mathrm{d}\hat{\alpha}_1} K_H(u-x)\exp\big(\hat{\alpha}_0 + \hat{\alpha}_1(u-x)\big)\,\mathrm{d}u \\
&\quad = \frac{\mathrm{d}}{\mathrm{d}\hat{\alpha}_1}\int_{\mathbb{R}^d} K_H(u-x)\exp\big(\hat{\alpha}_0 + \hat{\alpha}_1(u-x)\big)\,\mathrm{d}u \\
&\quad = \frac{\mathrm{d}}{\mathrm{d}\hat{\alpha}_1}\exp\Big(\hat{\alpha}_0 + \frac{1}{2}H\hat{\alpha}_1 \bullet H\hat{\alpha}_1\Big) \\
&\quad = H^2\hat{\alpha}_1\exp\Big(\hat{\alpha}_0 + \frac{1}{2}H\hat{\alpha}_1 \bullet H\hat{\alpha}_1\Big).
\end{aligned}$$

Thus, dividing (A.3) by (A.2) yields $H^2\hat{\alpha}_1$ on the right hand side, and

$$n^{-1}\sum_{i=1}^{n}\frac{K_H(X_i - x)}{n^{-1}\sum_{i=1}^{n}K_H(X_i - x)}(X_i - x) = Q'W1_n$$

on the left hand side. Premultiplying H^{-1} on both sides yields $H\hat{\alpha}_1 = H^{-1}Q'W1_n$, which can be plugged into the right hand side of (A.2) to obtain:

$$n^{-1}\sum_{i=1}^{n}K_H(X_i - x) = \exp\Big(\hat{\alpha}_0 + \frac{1}{2}H\hat{\alpha}_1 \bullet H\hat{\alpha}_1\Big).$$

Multiplying both sides by $\exp\Big(-\frac{1}{2}H\hat{\alpha}_1 \bullet H\hat{\alpha}_1\Big)$ isolates $\exp(\hat{\alpha}_0)$ on the right hand side, thus obtaining (2.5). Recall that $\mathsf{a} \bullet \mathsf{a} = \mathsf{a}'\mathsf{a}$ for any vector a.

Derivation of the LPDE with quadratic fitting

When a quadratic polynomial $\Pi(\mathsf{x}) = \alpha_0 + \alpha_1 \bullet \mathsf{x} + \alpha_2 \bullet \mathsf{x}^{\otimes 2}$ is used to approximate the log density locally, maximizing (2.2) is equivalent to solving the following system

$$\begin{aligned}
&n^{-1}\sum_{i=1}^{n}K_H(X_i - x) \\
&\qquad = \int_{\mathbb{R}^d} K_H(u-x)\exp\big(\hat{\alpha}_0 + \hat{\alpha}_1 \bullet (u-x) + \hat{\alpha}_2(u-x)^{\otimes 2}\big)\,\mathrm{d}u
\end{aligned} \tag{A.4}$$

$$n^{-1}\sum_{i=1}^{n} K_H(X_i - x)(X_i - x) = \int_{\mathbb{R}^d} K_H(u-x)(u-x)\exp\big(\hat{\alpha}_0 + \hat{\alpha}_1 \bullet (u-x) + \hat{\alpha}_2(u-x)^{\otimes 2}\big)\,\mathrm{d}u \tag{A.5}$$

$$n^{-1}\sum_{i=1}^{n} K_H(X_i - x)(X_i - x)^{\otimes 2} = \int_{\mathbb{R}^d} K_H(u-x)(u-x)^{\otimes 2}\exp\big(\hat{\alpha}_0 + \hat{\alpha}_1 \bullet (u-x) + \hat{\alpha}_2(u-x)^{\otimes 2}\big)\,\mathrm{d}u \tag{A.6}$$

of $1 + d + d^2$ equations for $\exp(\hat{\alpha}_0)$. Let $A \coloneqq I - 2H\,\mathrm{Mat}(\hat{\alpha}_2)H$, where $\mathrm{Mat}(\hat{\alpha}_2)$ denotes the $d \times d$ matrix that arises when unstacking the d^2 components of $\hat{\alpha}_2$ into a $d \times d$ matrix. We have to make the additional assumption that A is a symmetric and positive definite $d \times d$ matrix to ensure the existence of the integrals on the right hand sides. That is, we (implicitly) add the restrictions i) $\mathrm{Mat}(\alpha_2)$ is symmetric and nonsingular and ii) $(\mathrm{Mat}(\alpha_2))^{-1} - H^2$ is nonsingular when estimating α_2. It turns out that, under these restrictions, A coincides with the definition of A given in Section 2. Evaluating the right hand side of (A.4) yields

$$\begin{aligned}
&\int_{\mathbb{R}^d} K_H(u-x)\exp\big(\hat{\alpha}_0 + \hat{\alpha}_1 \bullet (u-x) + \hat{\alpha}_2 \bullet (u-x)^{\otimes 2}\big)\,\mathrm{d}u \\
&\quad= \int_{\mathbb{R}^d} (2\pi)^{-\frac{d}{2}} \det(H^{-1})\exp\Big(-\frac{1}{2}H^{-1}(u-x) \bullet H^{-1}(u-x)\Big)\exp\big(\hat{\alpha}_0 + \hat{\alpha}_1 \bullet (u-x) + \hat{\alpha}_2 \bullet (u-x)^{\otimes 2}\big)\,\mathrm{d}u \\
&\quad= \int_{\mathbb{R}^d} (2\pi)^{-\frac{d}{2}} \sqrt{\det(A)^{-1}}\exp\Big(-\frac{1}{2}v \bullet v + \frac{1}{2}HA^{-\frac{1}{2}}\hat{\alpha}_1 \bullet HA^{-\frac{1}{2}}\hat{\alpha}_1 + \hat{\alpha}_0\Big)\,\mathrm{d}v \\
&\quad= \sqrt{\det(A)^{-1}}\exp\Big(\frac{1}{2}HA^{-\frac{1}{2}}\hat{\alpha}_1 \bullet HA^{-\frac{1}{2}}\hat{\alpha}_1 + \hat{\alpha}_0\Big)\int_{\mathbb{R}^d} (2\pi)^{-\frac{d}{2}}\exp\Big(-\frac{1}{2}v \bullet v\Big)\,\mathrm{d}v \\
&\quad= \sqrt{\det(A)^{-1}}\exp\Big(\frac{1}{2}A^{-\frac{1}{2}}H\hat{\alpha}_1 \bullet A^{-\frac{1}{2}}H\hat{\alpha}_1 + \hat{\alpha}_0\Big).
\end{aligned}$$

Note that a change of variables ($v = x + HA^{-\frac{1}{2}}u + A^{-1}H\hat{\alpha}_1$) was used in the second step. To evaluate the left hand side of (A.4) the differentiation under the integral sign method is used again to obtain

$$\begin{aligned}
&\int_{\mathbb{R}^d} K_H(u-x)(u-x)\exp\big(\hat{\alpha}_0 + \hat{\alpha}_1 \bullet (u-x) + \hat{\alpha}_2(u-x)^{\otimes 2}\big)\,\mathrm{d}u \\
&\quad= HA^{-1}H\hat{\alpha}_1\sqrt{\det(A)^{-1}}\exp\Big(\frac{1}{2}A^{-\frac{1}{2}}H\hat{\alpha}_1 \bullet A^{-\frac{1}{2}}H\hat{\alpha}_1 + \hat{\alpha}_0\Big).
\end{aligned}$$

Thus, dividing (A.5) by (A.4) yields $HA^{-1}H\hat{\alpha}_1 = Q'W1_n$. Premultiplying H^{-1} from the left yields $A^{-1}H\hat{\alpha}_1 = H^{-1}Q'W1_n$. Next, the differentiation under the integral sign method is used again to obtain

$$\begin{aligned}&\int K_H(u-x)(u-x)^{\otimes 2}\exp\big(\hat{\alpha}_0 + \hat{\alpha}_1 \bullet (u-x) + \hat{\alpha}_2(u-x)^{\otimes 2}\big)\,\mathrm{d}u \\ &\quad = \big(HA^{-1}H\hat{\alpha}_1\hat{\alpha}_1'HA^{-1}H + HA^{-1}\big)\sqrt{\det(A)^{-1}}\exp\Big(\frac{1}{2}A^{-\frac{1}{2}}H\hat{\alpha}_1 \bullet A^{-\frac{1}{2}}H\hat{\alpha}_1 + \hat{\alpha}_0\Big)\end{aligned}$$

for (A.6). Dividing (A.6) by (A.4) yields $HA^{-1}\hat{\alpha}_1\hat{\alpha}_1A^{-1}H + HA^{-1}H = Q'WQ$. Premultiplying H^{-1} from both sides yields $A^{-1}H\hat{\alpha}_1\hat{\alpha}_1HA^{-1} + A^{-1} = H^{-1}Q'WQH^{-1}$. Computing the outer product for (A.5) and subtracting it from (A.6) yields $A^{-1} = H^{-1}(Q'WQ - Q'W1_n1_n'WQ)H^{-1}$. The restrictions on α_2 ensure invertibility for A, i.e.

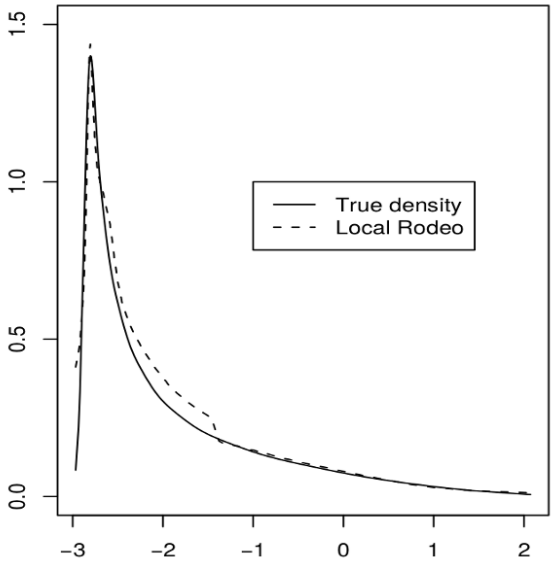
$$A = H\big(Q'WQ - Q'W1_n1_n'WQ\big)^{-1}H$$

as claimed. This expression is then plugged into the right hand side of (A.4). Multiplying both sides of (A.4) by $\sqrt{\det(A)}\exp\Big(-\frac{1}{2}A^{-\frac{1}{2}}H\hat{\alpha}_1 \bullet A^{-\frac{1}{2}}H\hat{\alpha}_1\Big)$ isolates $\exp(\hat{\alpha}_0)$ on the right hand side, thus obtaining (2.6). Recall that $\mathsf{a} \bullet \mathsf{a} = \mathsf{a}'\mathsf{a}$ for any vector a.

B Additional Figures and Tables

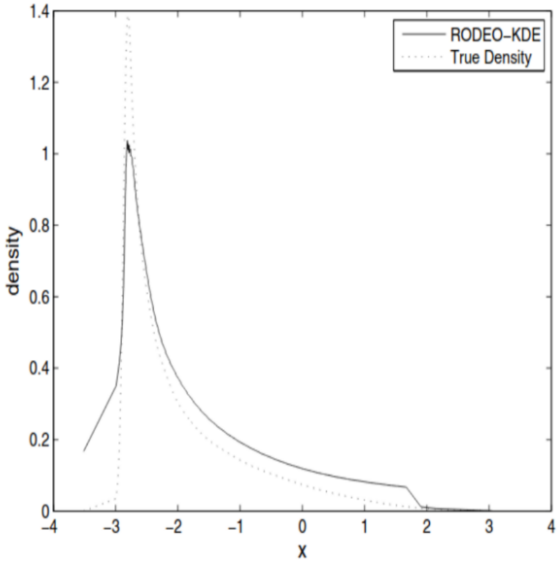
Liu, Lafferty, and Wasserman (2007):

- Density #3 from Marron and Wand (1992):



Ram and Gray (2011):

- Density #3 from Marron and Wand (1992):



- Product density of (3.1) and (3.2):

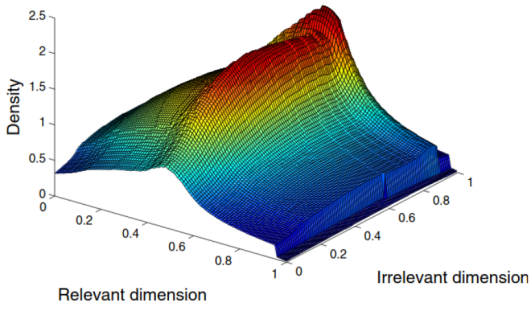


Figure B.1: Screenshots of the estimates for the densities also considered by Liu, Lafferty, and Wasserman (2007), as well as Ram and Gray (2011).

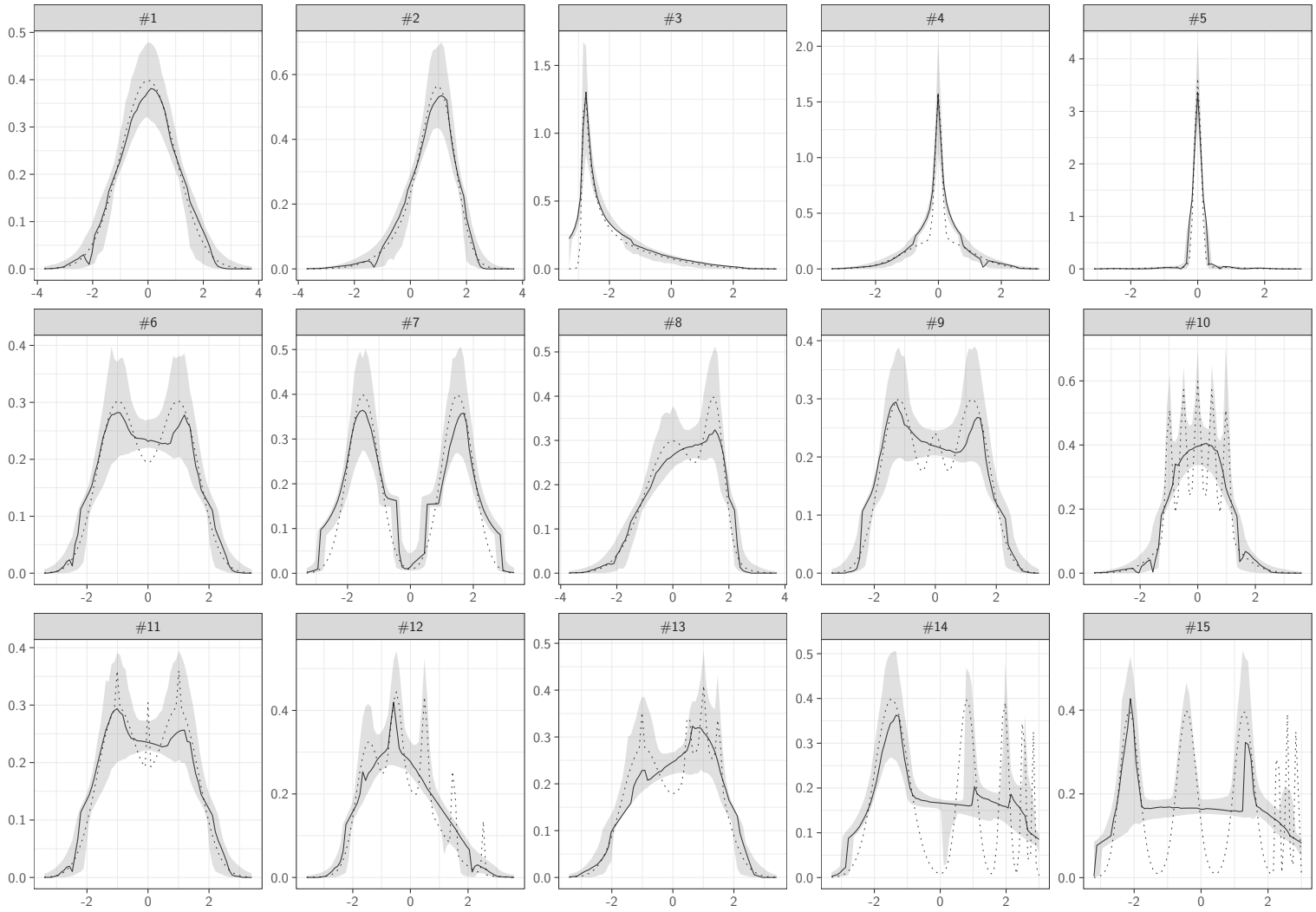


Figure B.2: The 15 normal mixture densities from Marron and Wand (1992) estimated by $\hat{f}_{\text{rodeo}}$. The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The shaded region corresponds to the 95 percent confidence region based on $m = 500$ replications.

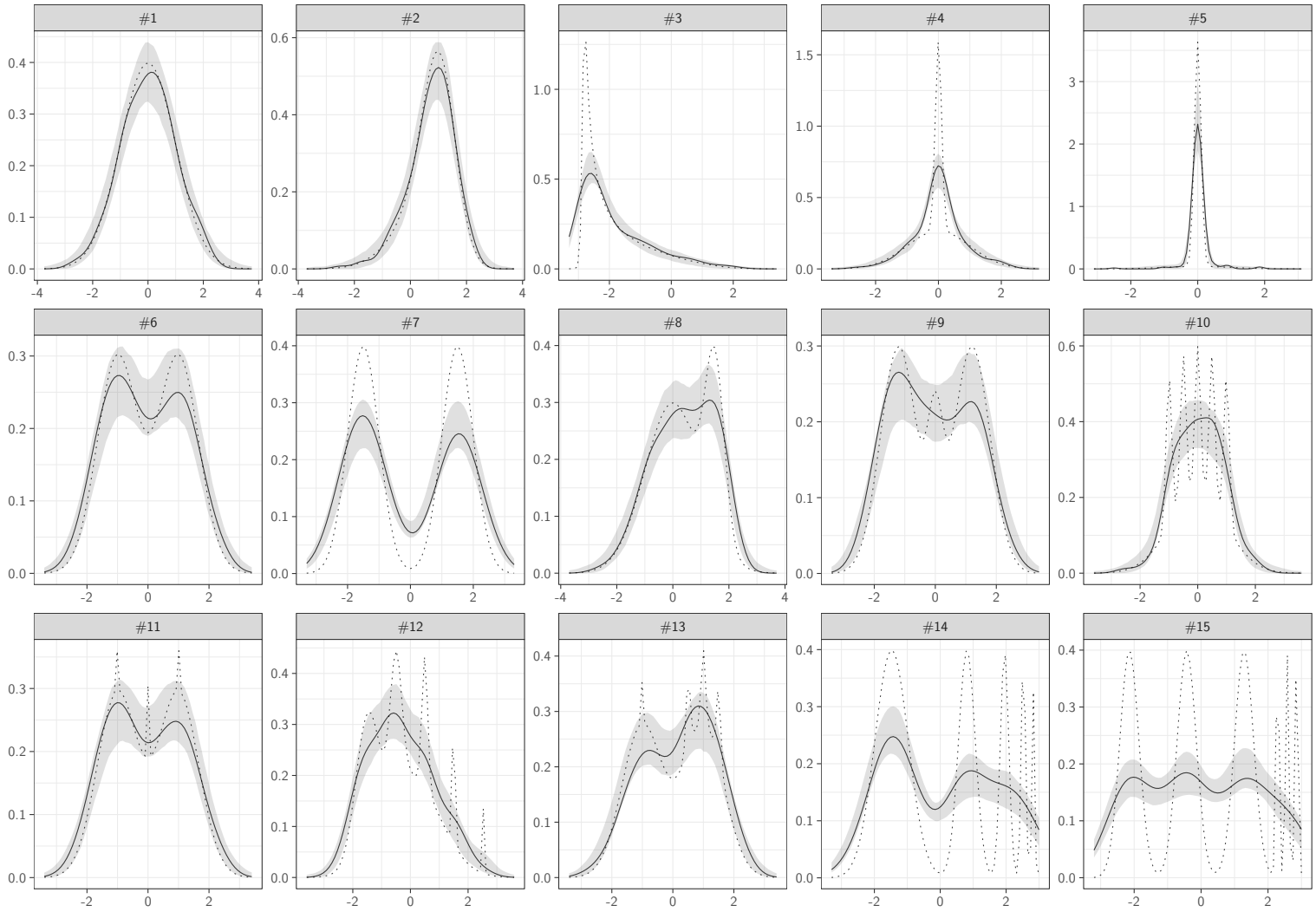


Figure B.3: The 15 normal mixture densities from Marron and Wand (1992) estimated by $\hat{f}_{\text{nr}}$. The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The shaded region corresponds to the 95 percent confidence region based on $m = 500$ replications.

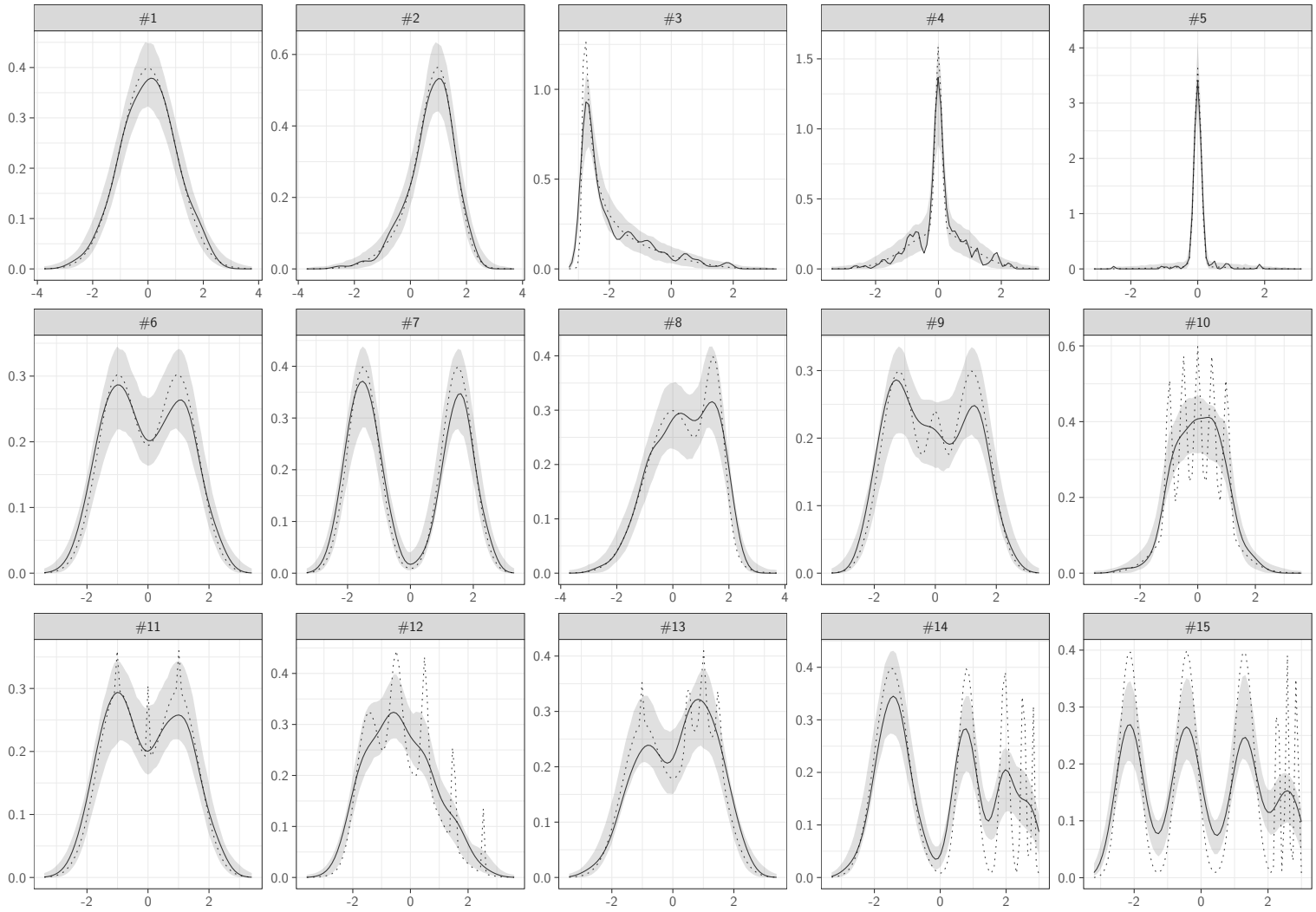


Figure B.4: The 15 normal mixture densities from Marron and Wand (1992) estimated by $\hat{f}_{\text{pi}}$. The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The shaded region corresponds to the 95 percent confidence region based on $m = 500$ replications.

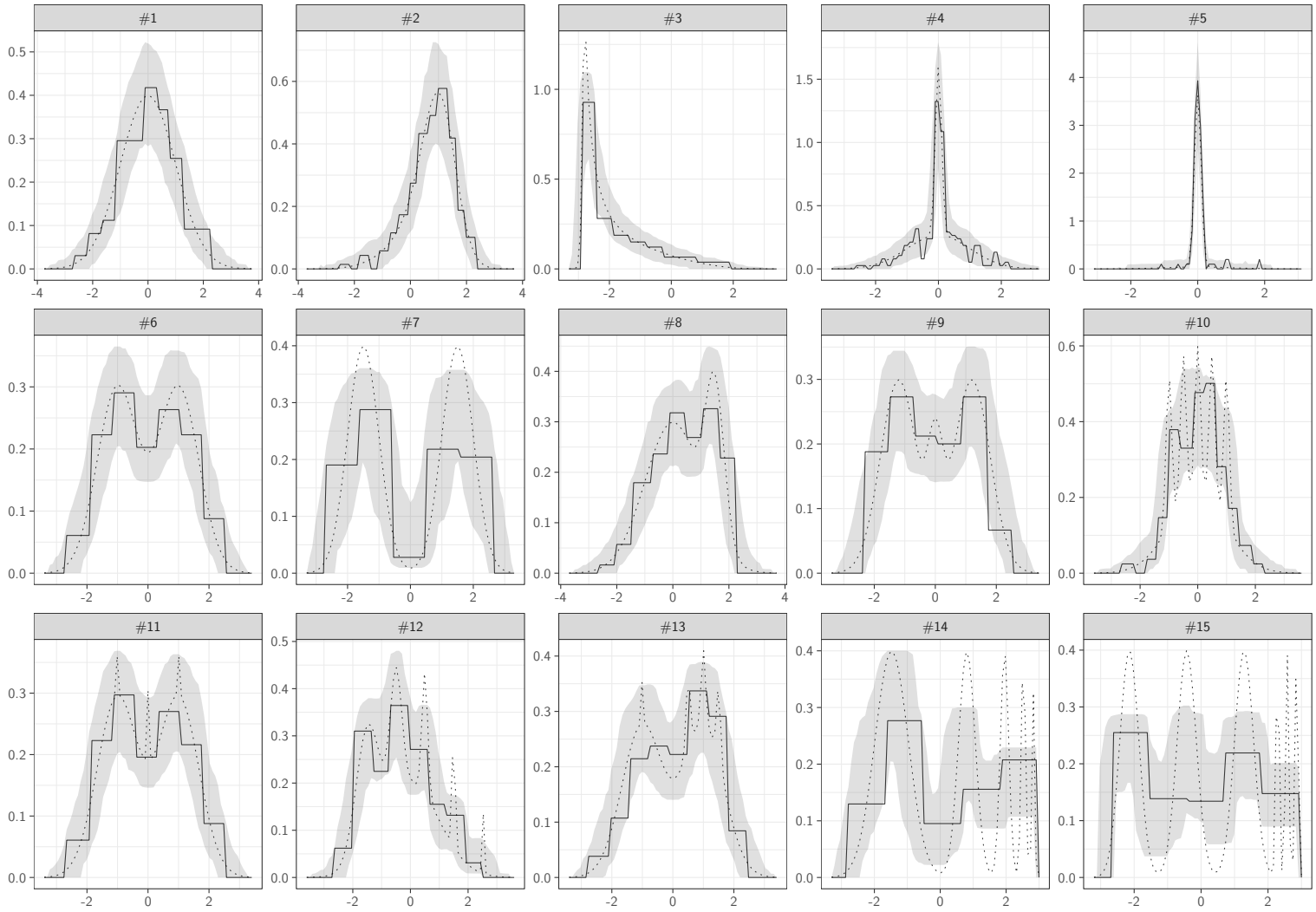


Figure B.5: The 15 normal mixture densities from Marron and Wand (1992) estimated by $\hat{f}_{\text{hist}}$. The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The shaded region corresponds to the 95 percent confidence region based on $m = 500$ replications.

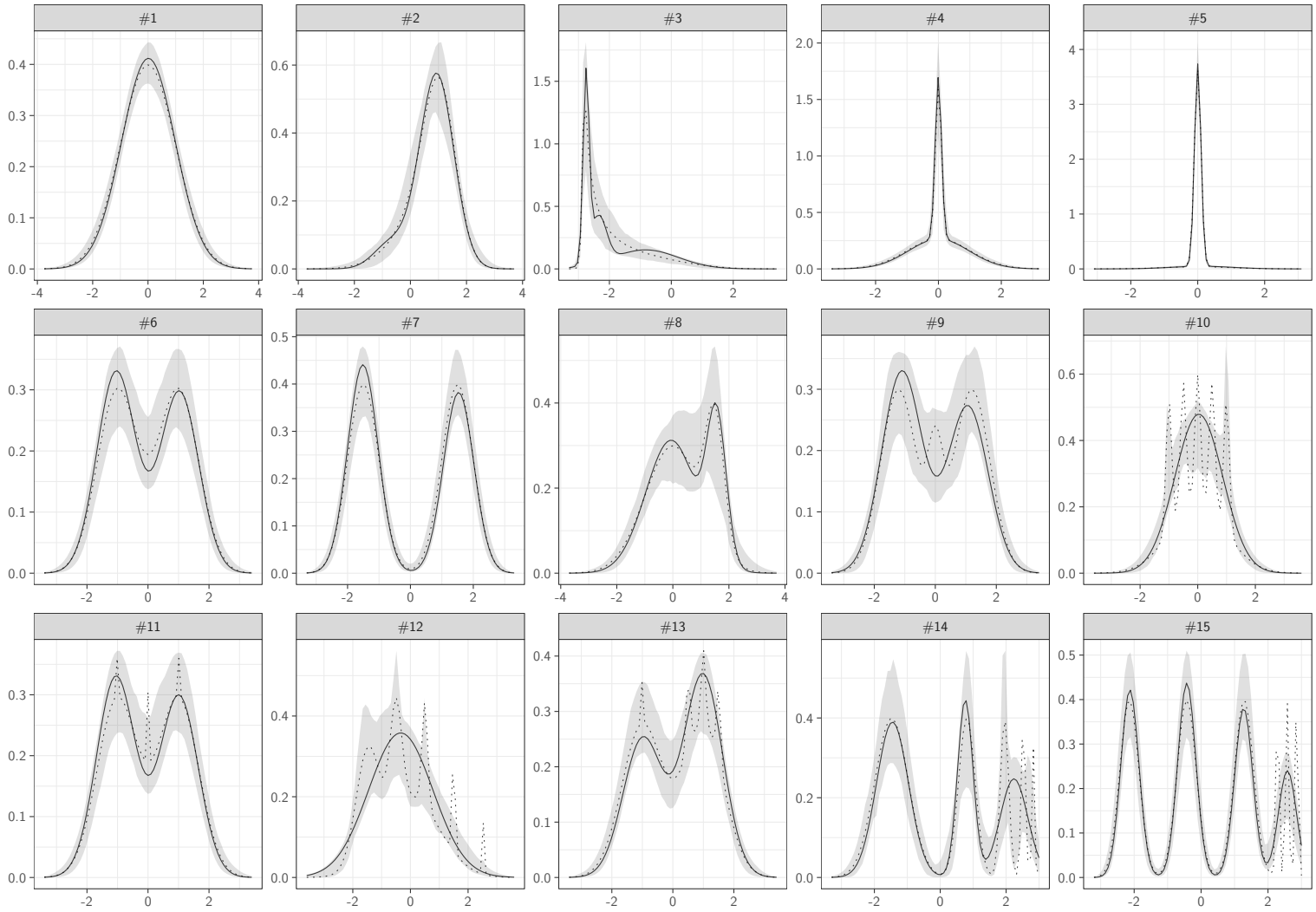


Figure B.6: The 15 normal mixture densities from Marron and Wand (1992) estimated by $\hat{f}_{\text{nmm}}$ as scale estimator. The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The shaded region corresponds to the 95 percent confidence region based on $m = 500$ replications.

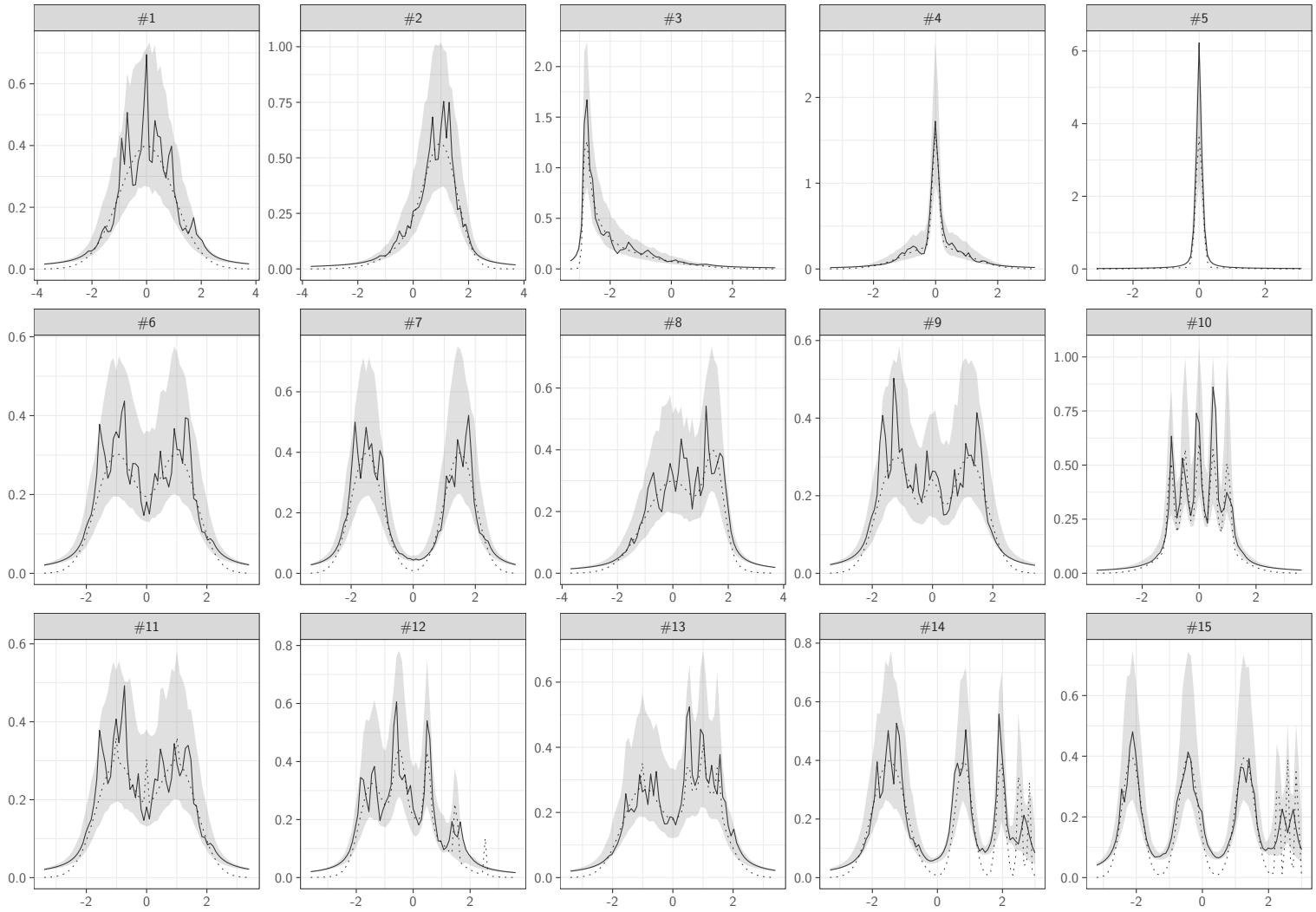


Figure B.7: The 15 normal mixture densities from Marron and Wand (1992) estimated by $\hat{f}_{\text{knn}}$. The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The shaded region corresponds to the 95 percent confidence region based on $m = 500$ replications.

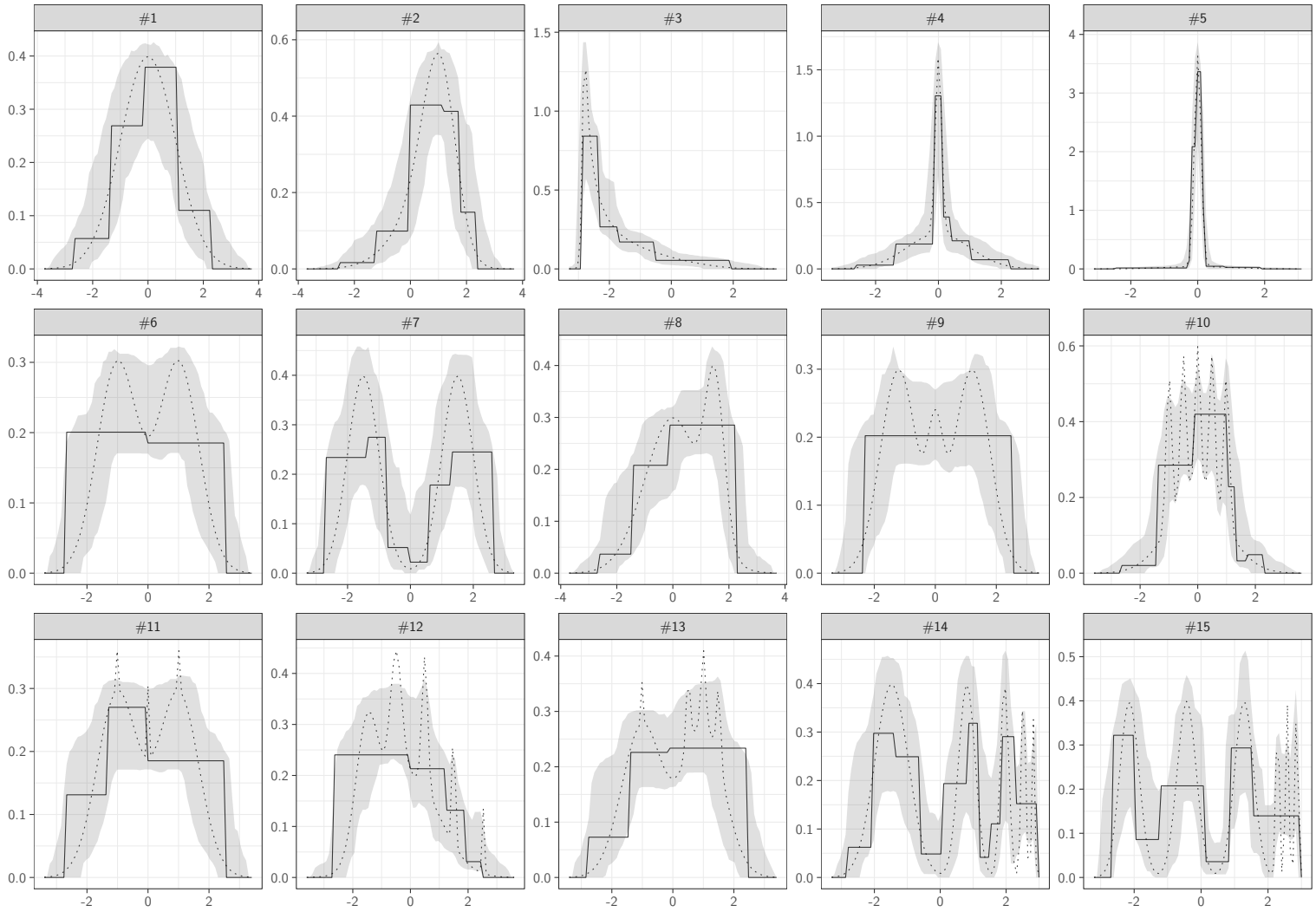


Figure B.8: The 15 normal mixture densities from Marron and Wand (1992) estimated by $\hat{f}_{\text{det}}$. The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The shaded region corresponds to the 95 percent confidence region based on $m = 500$ replications.

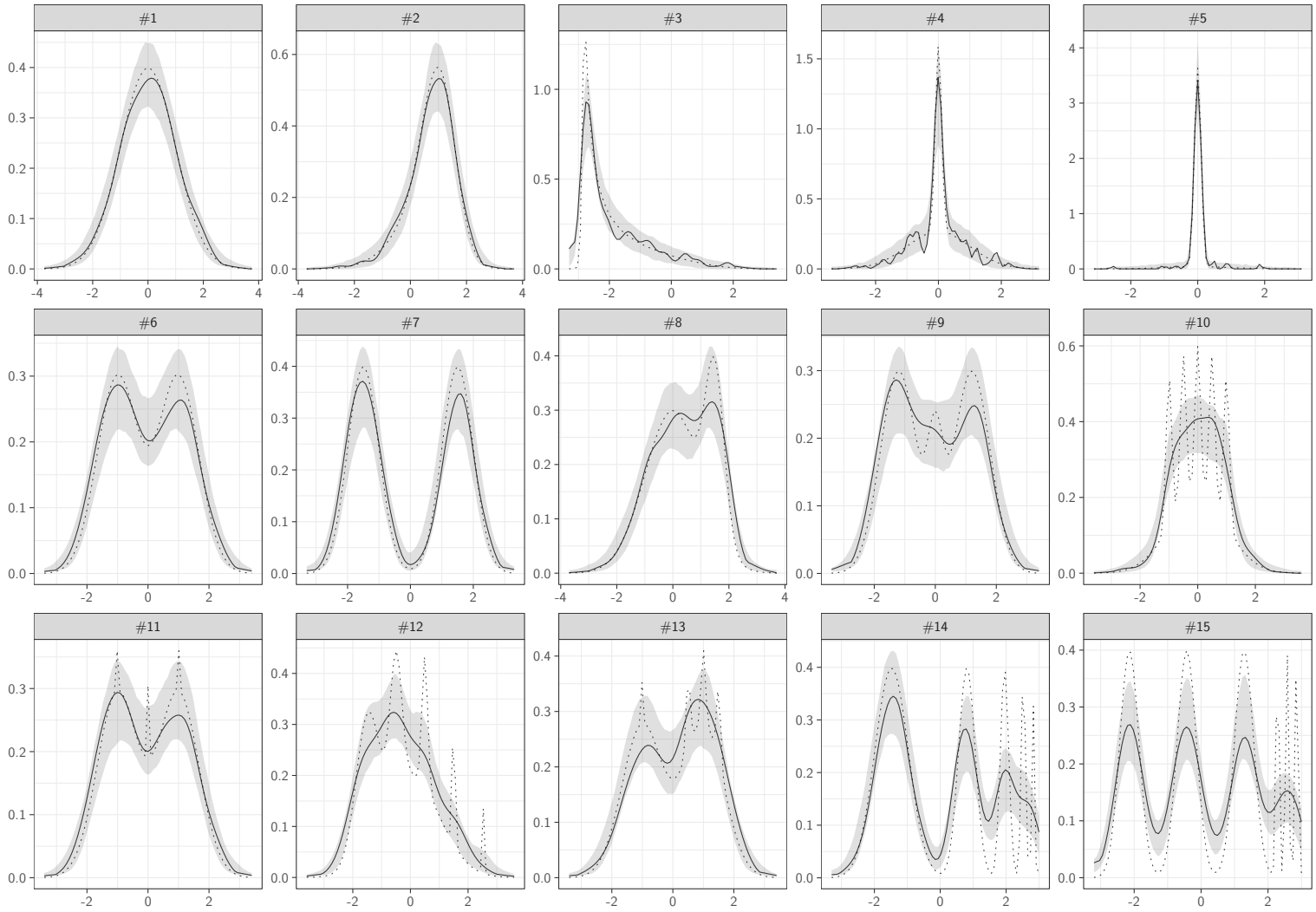


Figure B.9: The 15 normal mixture densities from Marron and Wand (1992) estimated by $\hat{f}_{\text{vine}}$. The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The shaded region corresponds to the 95 percent confidence region based on $m = 500$ replications.

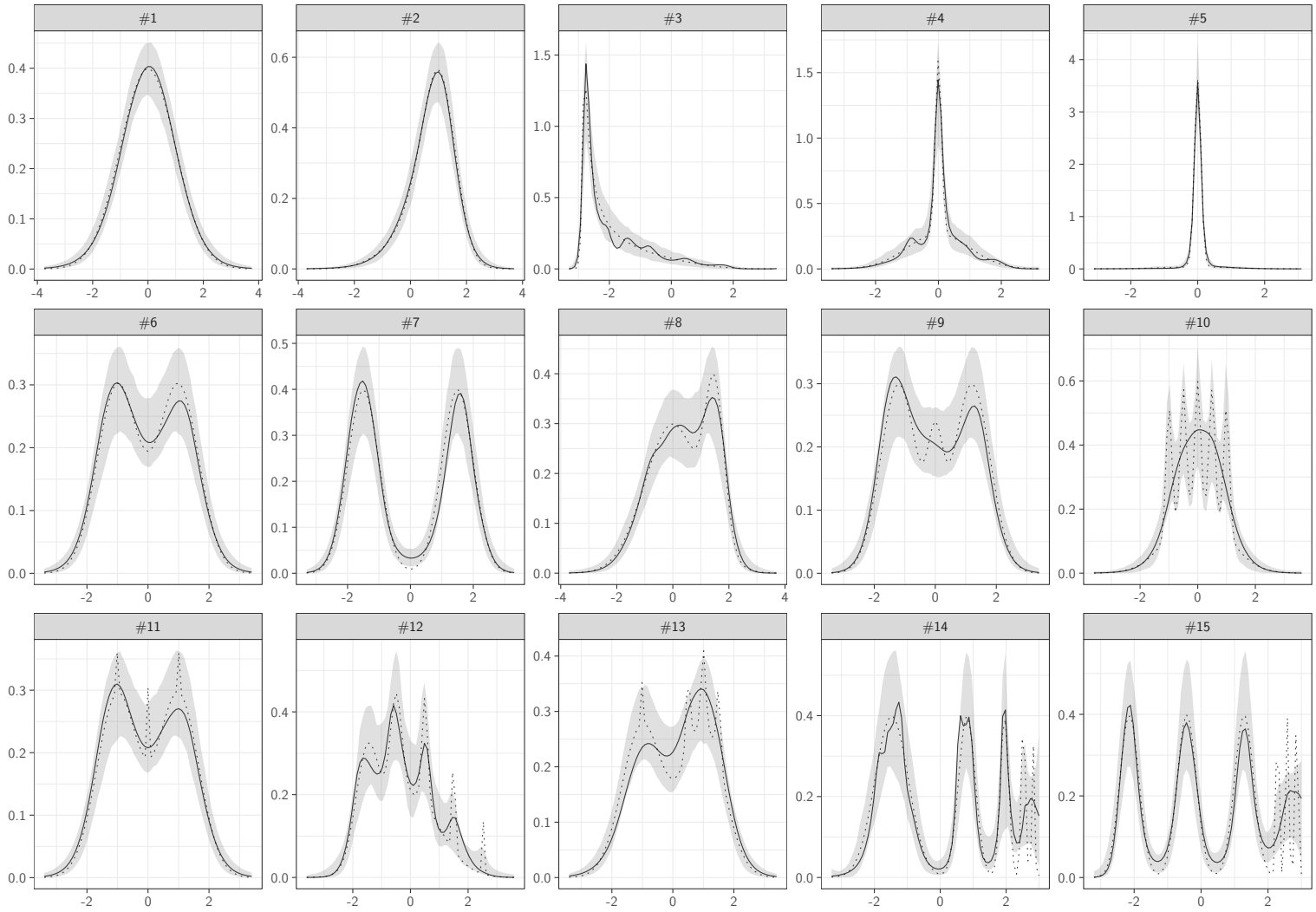


Figure B.10: The 15 normal mixture densities from Marron and Wand (1992) estimated by $\hat{f}_{\text{spline}}$. The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The shaded region corresponds to the 95 percent confidence region based on $m = 500$ replications.

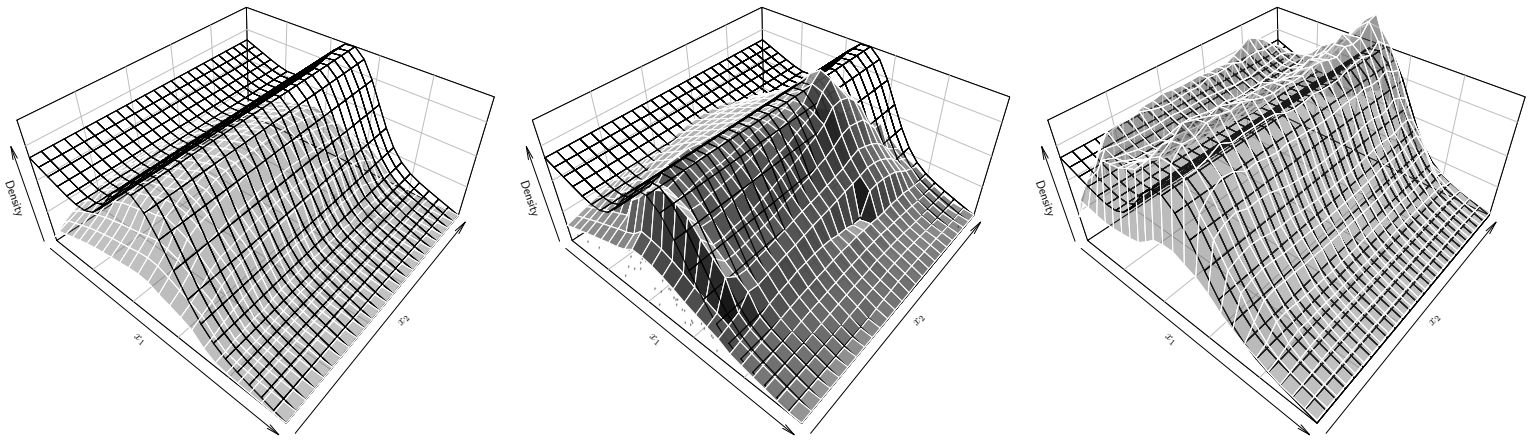


Figure B.11: The product density of (3.1) and (3.2) estimated by $\hat{f}_{\text{rodeo}}$ (middle). The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The plots on the left and right indicate the 95 percent confidence region based on $m = 500$ replications.

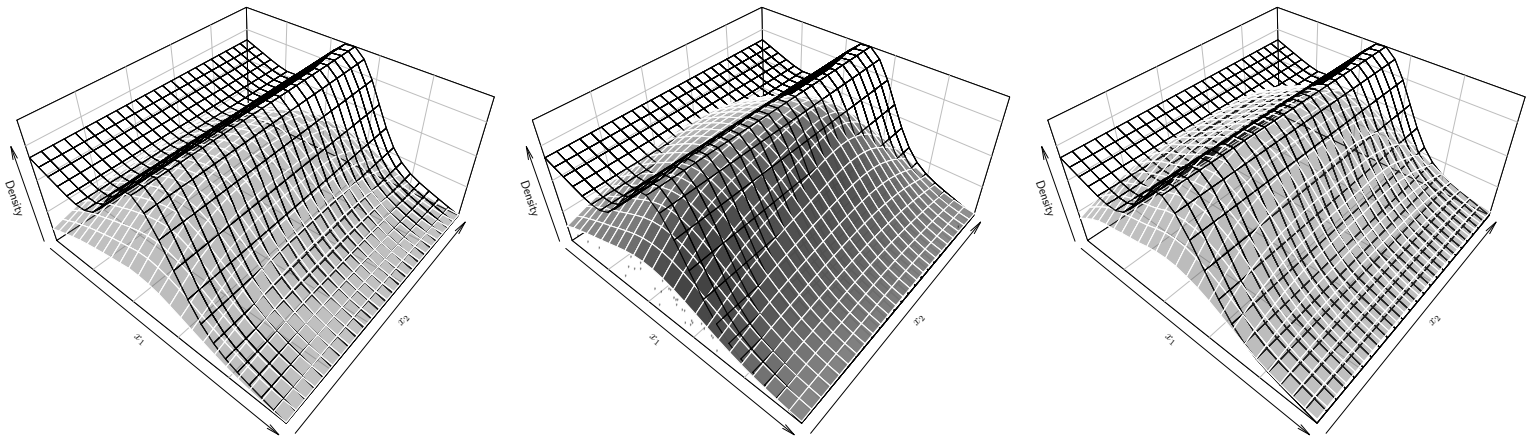


Figure B.12: The product density of (3.1) and (3.2) estimated by $\hat{f}_{\text{nr}}$ (middle). The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The plots on the left and right indicate the 95 percent confidence region based on $m = 500$ replications.

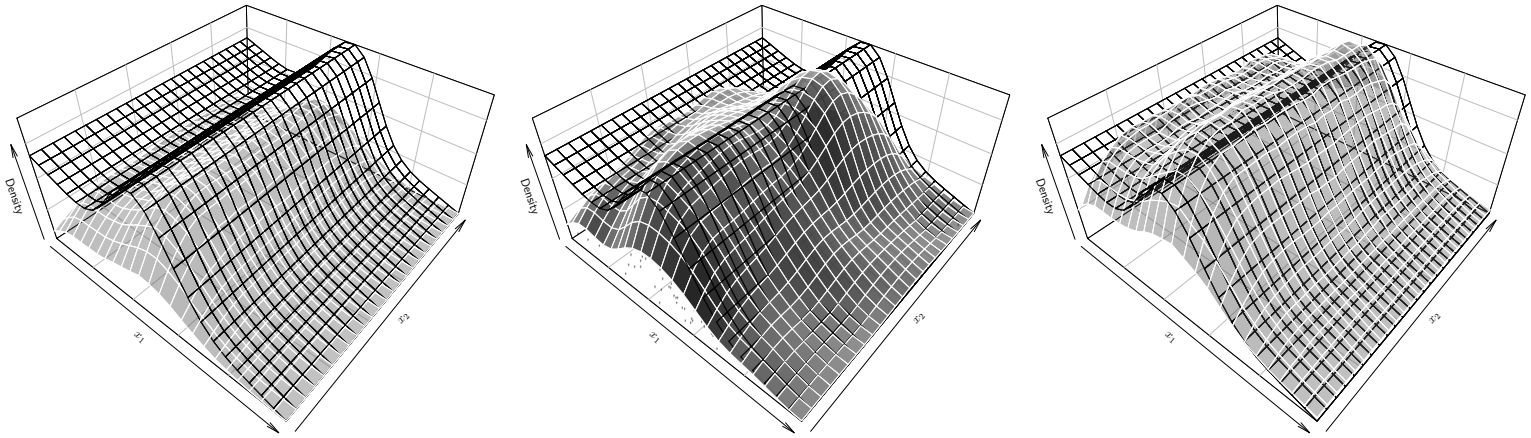


Figure B.13: The product density of (3.1) and (3.2) estimated by $\hat{f}_{\text{pi}}$ (middle). The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The plots on the left and right indicate the 95 percent confidence region based on $m = 500$ replications.

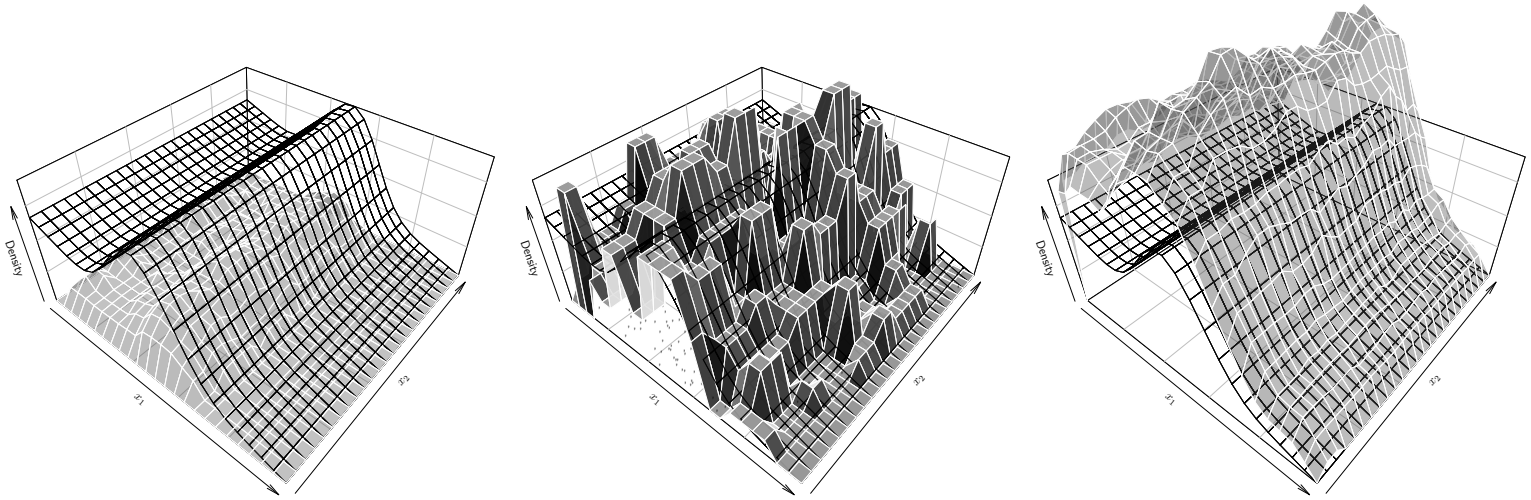


Figure B.14: The product density of (3.1) and (3.2) estimated by $\hat{f}_{\text{hist}}$ (middle). The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The plots on the left and right indicate the 95 percent confidence region based on $m = 500$ replications.

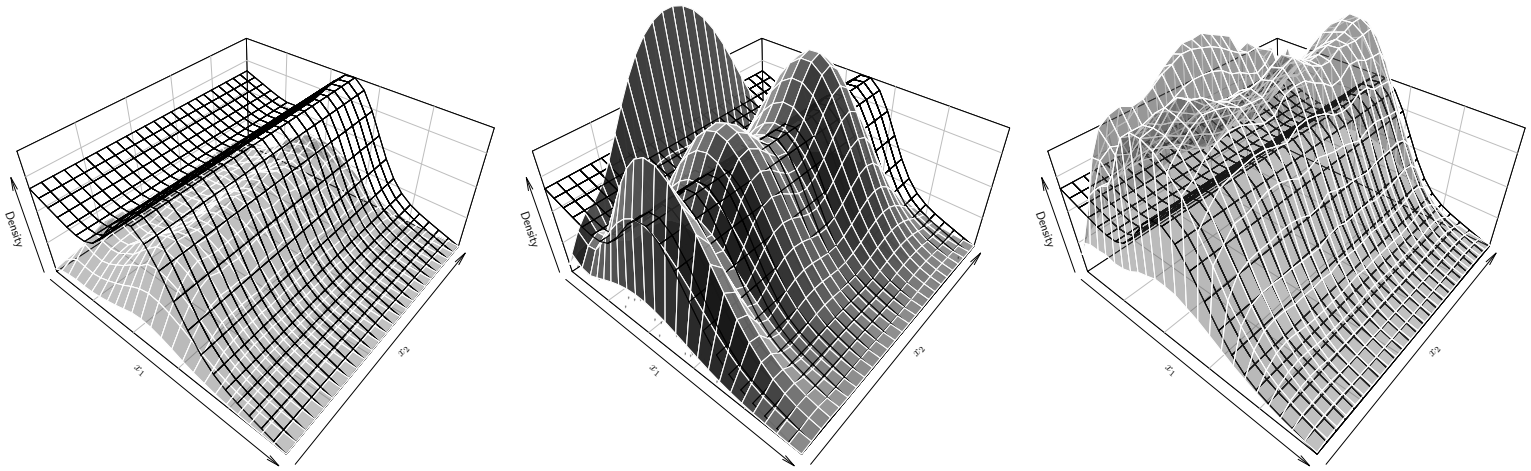


Figure B.15: The product density of (3.1) and (3.2) estimated by $\hat{f}_{\text{nmm}}$ (middle). The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The plots on the left and right indicate the 95 percent confidence region based on $m = 500$ replications.

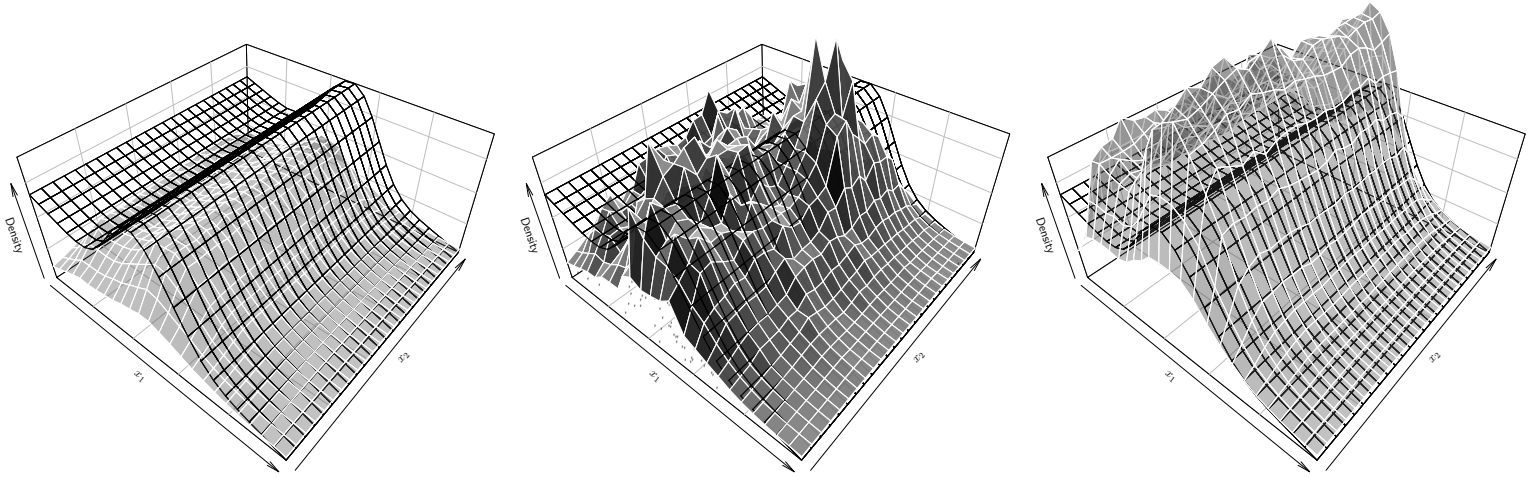


Figure B.16: The product density of (3.1) and (3.2) estimated by $\hat{f}_{\text{knn}}$ (middle). The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The plots on the left and right indicate the 95 percent confidence region based on $m = 500$ replications.

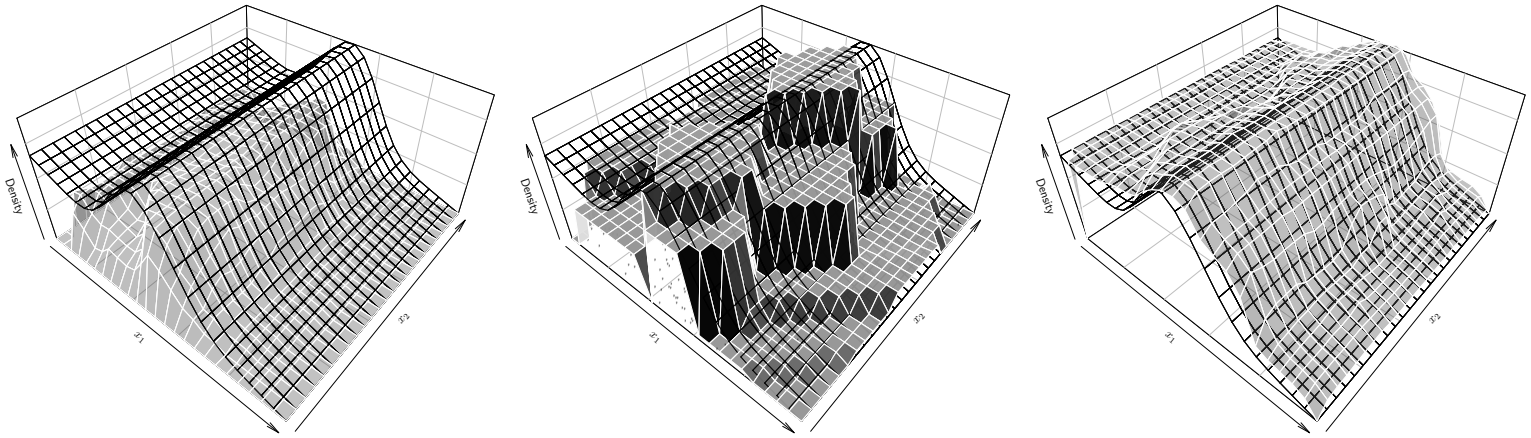


Figure B.17: The product density of (3.1) and (3.2) estimated by $\hat{f}_{\text{det}}$ (middle). The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The plots on the left and right indicate the 95 percent confidence region based on $m = 500$ replications.

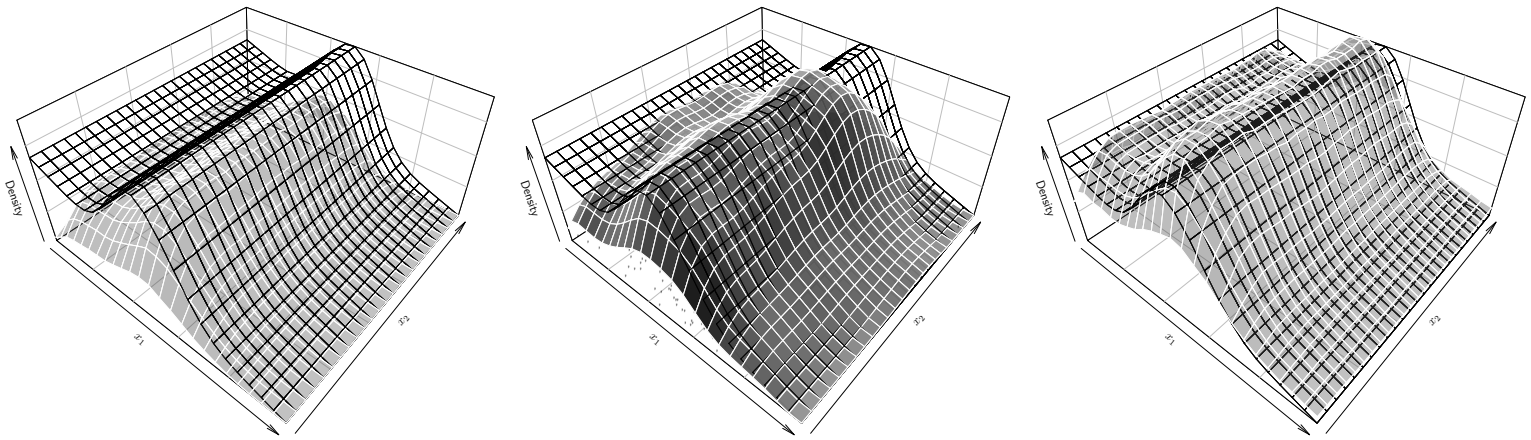


Figure B.18: The product density of (3.1) and (3.2) estimated by $\hat{f}_{\text{vine}}$ (middle). The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The plots on the left and right indicate the 95 percent confidence region based on $m = 500$ replications.

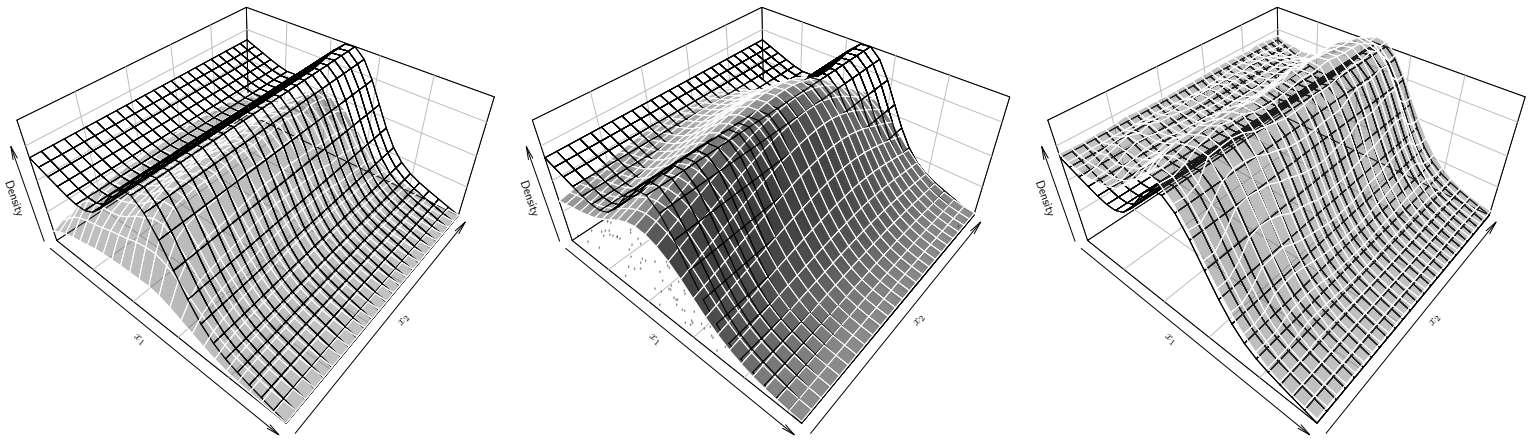


Figure B.19: The product density of (3.1) and (3.2) estimated by $\hat{f}_{\text{spline}}$ (middle). The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The plots on the left and right indicate the 95 percent confidence region based on $m = 500$ replications.

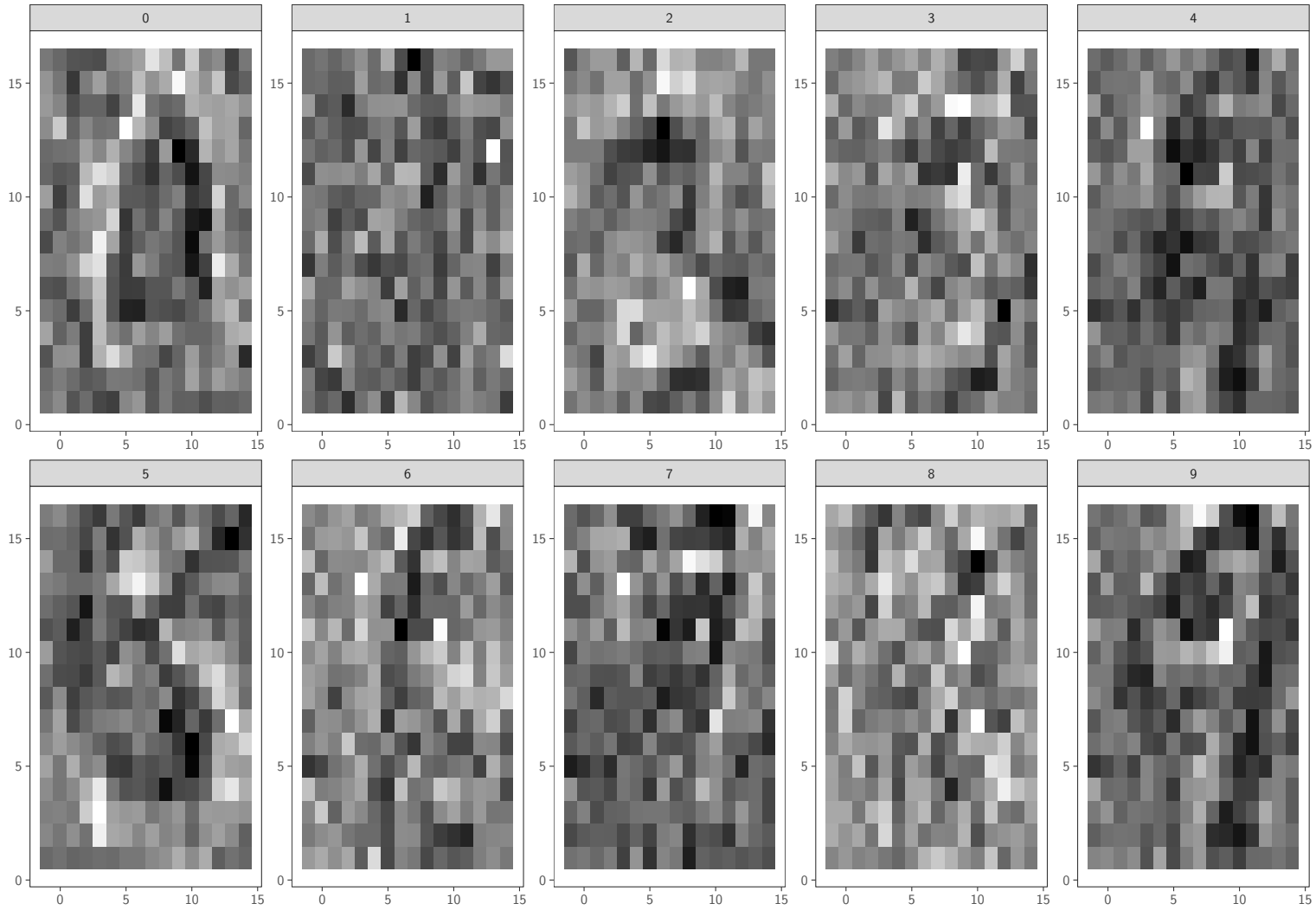


Figure B.20: Per-pixel average for $l = 100$ random draws from the distribution of the respective digit estimated by $\hat{f}_{\text{nr}}$.

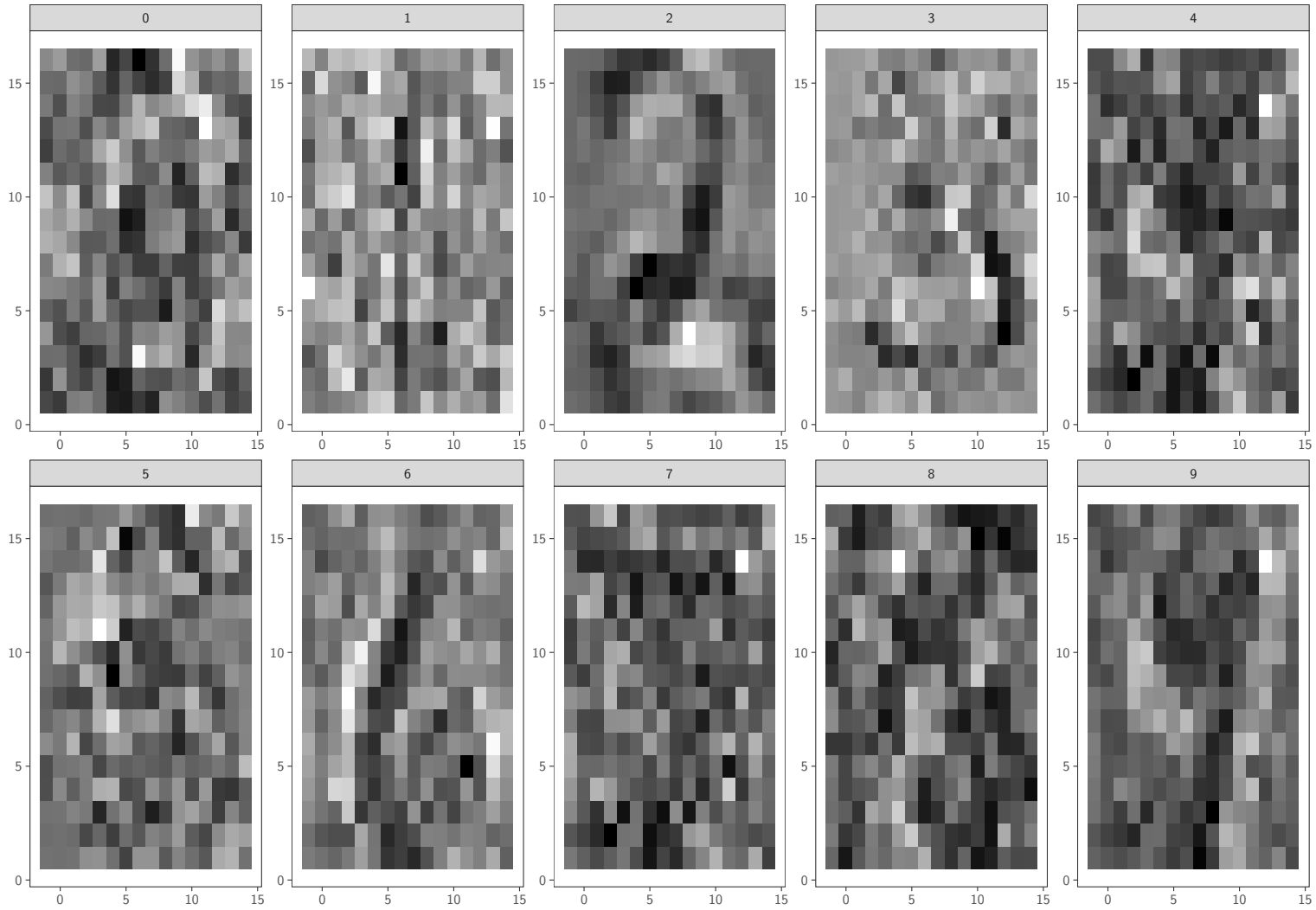


Figure B.21: Per-pixel average for $l = 100$ random draws from the distribution of the respective digit estimated by $\hat{f}_{\text{nmm}}$.

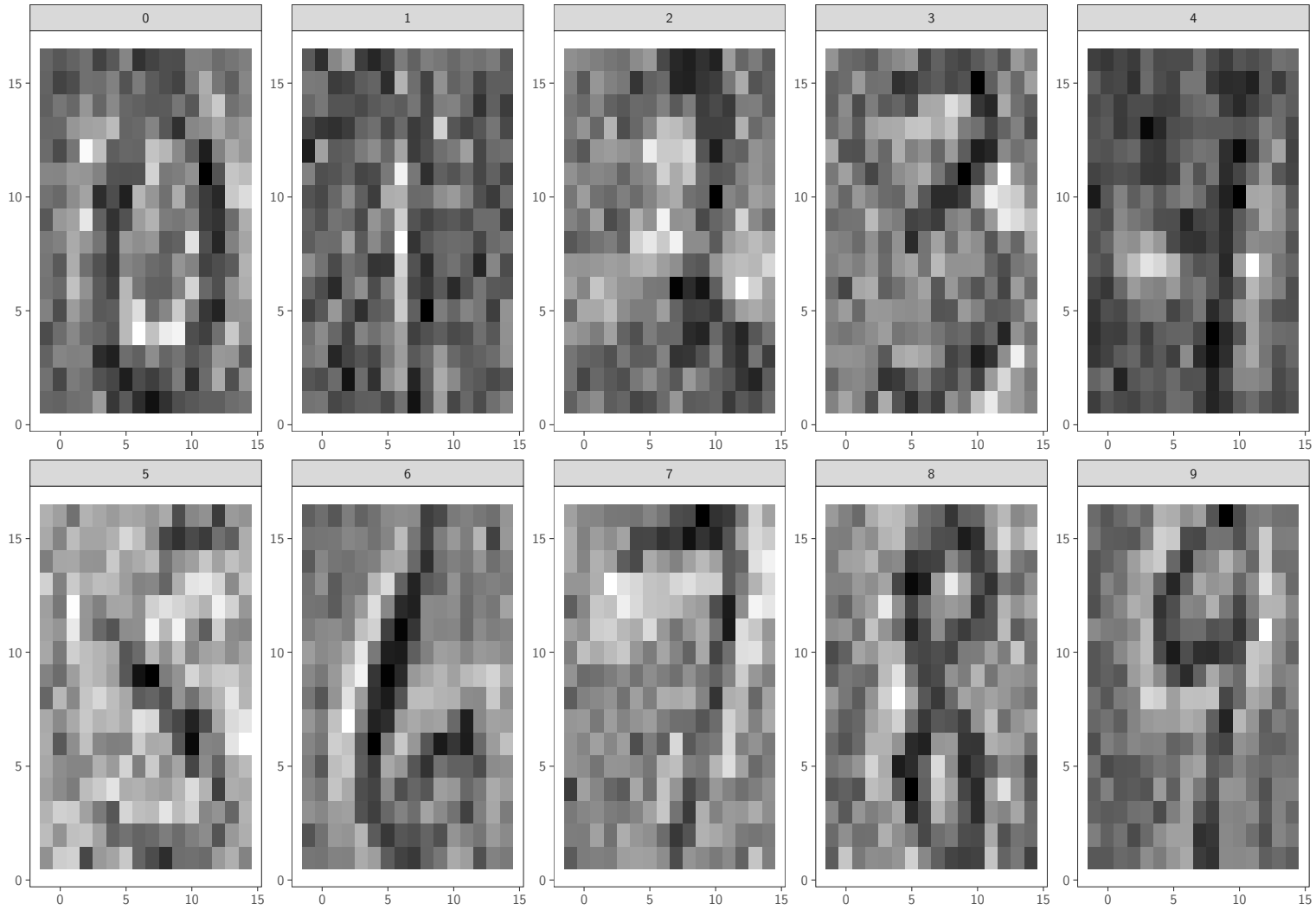


Figure B.22: Per-pixel average for $l = 100$ random draws from the distribution of the respective digit estimated by $\hat{f}_{\text{knn}}$.

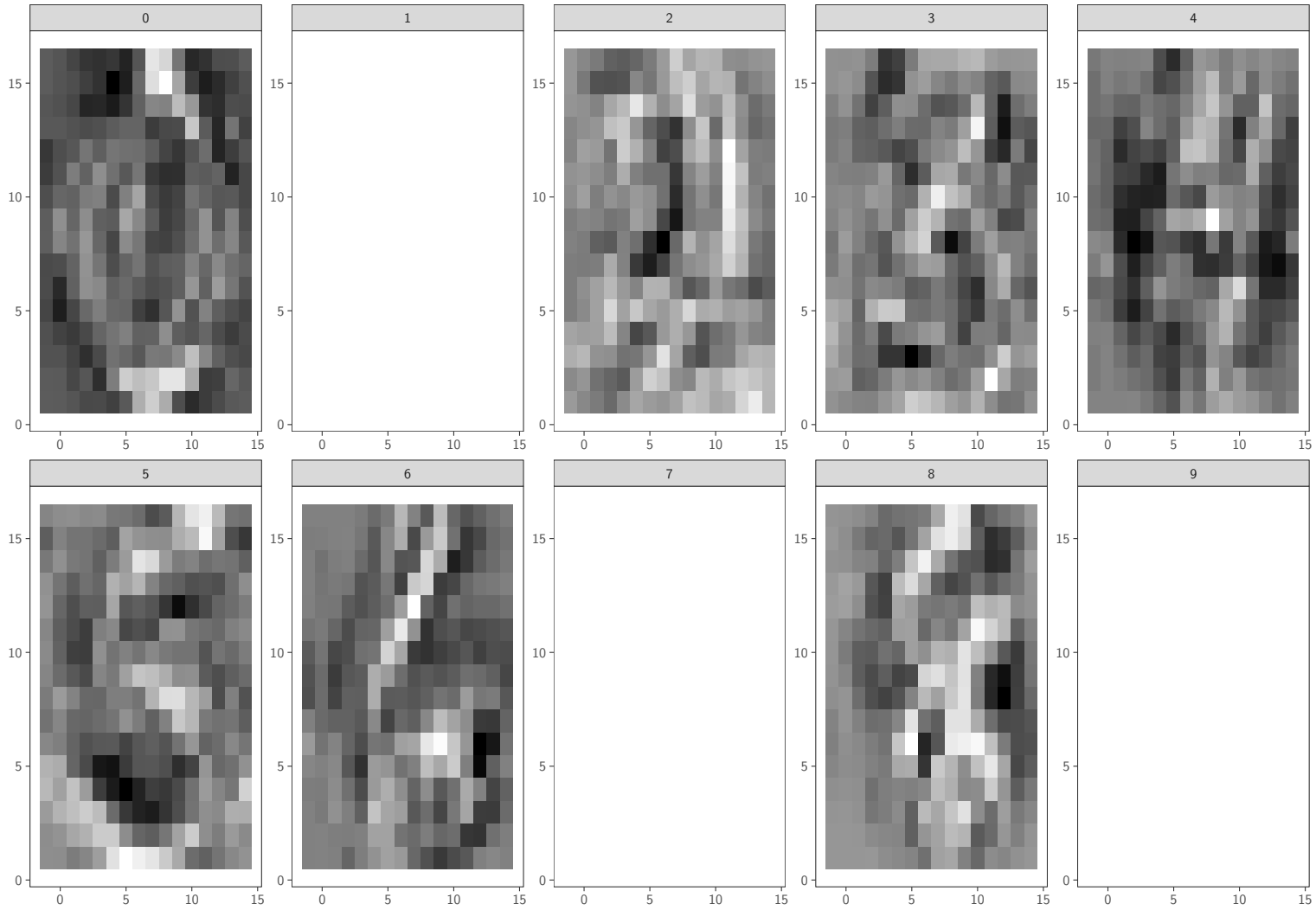


Figure B.23: Per-pixel average for $l = 100$ random draws from the distribution of the respective digit estimated by $\hat{f}_{\text{det}}$.

Table B.1: The MISE of $\hat{f}_{\text{rodeo}}$ (variation of H_{init}) for the 15 univariate normal mixture densities from Marron and Wand (1992). The reported values are scaled by 10^4. The approach with the smallest MISE is highlighted.

Density	$h_0 = 1$	$h_0 = 2$	$h_0 = 5$
#1	49	49	49
#2	77	77	77
#3	543	543	543
#4	536	536	536
#5	1265	1265	1265
#6	72	72	72
#7	192	192	192
#8	81	81	81
#9	80	80	80
#10	542	542	542
#11	75	75	75
#12	205	205	205
#13	117	117	117
#14	798	798	798
#15	918	918	918

Table B.2: The MISE of $\hat{f}_{\text{rodeo}}$ (variation of β) for the 15 univariate normal mixture densities from Marron and Wand (1992). The reported values are scaled by 10^4. The approach with the smallest MISE is highlighted.

Density	$\beta_0 = 0.9$	$\beta_0 = 0.01$	$\beta_0 = 0.5$	$\beta_0 = 0.99$
#1	49	211	42	43
#2	77	301	70	65
#3	543	1860	460	1173
#4	536	1916	417	960
#5	1265	5907	1445	2492
#6	72	223	73	72
#7	192	768	220	260
#8	81	297	84	78
#9	80	247	85	72
#10	542	850	512	559
#11	75	213	77	65
#12	205	370	194	202
#13	117	286	116	110
#14	798	835	798	832
#15	918	973	925	1062

Table B.3: The MISE of $\hat{f}_{\text{rodeo}}$ (variation of θ) for the 15 univariate normal mixture densities from Marron and Wand (1992). The reported values are scaled by 10^4. The approach with the smallest MISE is highlighted.

Density	Hard	Soft	Garrote	Hyperbole
#1	49	50	47	60
#2	77	80	75	104
#3	543	555	535	699
#4	536	556	535	694
#5	1265	1350	1289	1799
#6	72	82	74	99
#7	192	215	197	304
#8	81	90	82	111
#9	80	91	82	114
#10	542	535	531	552
#11	75	80	74	95
#12	205	207	202	219
#13	117	122	116	141
#14	798	798	793	835
#15	918	925	914	973

Table B.4: The MISE of $\hat{f}_{\text{rodeo}}$ (variation of ζ) for the 15 univariate normal mixture densities from Marron and Wand (1992). The reported values are scaled by 10^4. The approach with the smallest MISE is highlighted.

Density	$\sqrt{2\log\big(\log(n)\big)}$	$\sqrt{2\log(n)}$	1
#1	49	147	55
#2	77	171	87
#3	543	870	563
#4	536	662	560
#5	1265	1676	1411
#6	72	151	102
#7	192	299	266
#8	81	165	112
#9	80	177	114
#10	542	679	591
#11	75	166	98
#12	205	265	229
#13	117	221	141
#14	798	856	823
#15	918	1146	1118

Table B.5: The MISE of $\hat{f}_{\text{rodeo}}$ (variation of $\hat{\Sigma}$) for the 15 univariate normal mixture densities from Marron and Wand (1992). The reported values are scaled by 10^4. The approach with the smallest MISE is highlighted.

Density	Standard deviation	Median absolute deviation	Interquartile range	Minimum	Standard deviation variant 2	Standard deviation variant 3
#1	49	48	188	49	50	157
#2	77	75	224	79	78	173
#3	543	547	1108	547	545	856
#4	536	532	820	535	540	671
#5	1265	1266	1735	1264	1267	2289
#6	72	73	201	73	73	166
#7	192	188	346	190	195	315
#8	81	79	336	81	81	285
#9	80	79	275	80	81	214
#10	542	549	750	539	540	674
#11	75	74	239	75	76	206
#12	205	205	320	205	205	291
#13	117	116	1709	119	119	838
#14	798	799	994	797	798	980
#15	918	923	1175	918	917	1095

Table B.6: The MISE of $\hat{f}_{\text{rodeo}}$ (variation of q) for the 15 univariate normal mixture densities from Marron and Wand (1992). The reported values are scaled by 10^4. The approach with the smallest MISE is highlighted.

constant fitting	linear fitting	quadratic fitting	
#1	49	52	31
#2	77	74	48
#3	543	325	332
#4	536	406	518
#5	1265	474	— [a]
#6	72	70	59
#7	192	137	60
#8	81	74	69
#9	80	75	74
#10	542	456	532
#11	75	71	76
#12	205	196	204
#13	117	109	90
#14	798	784	617
#15	918	898	891

[a] estimator did not exist in this case

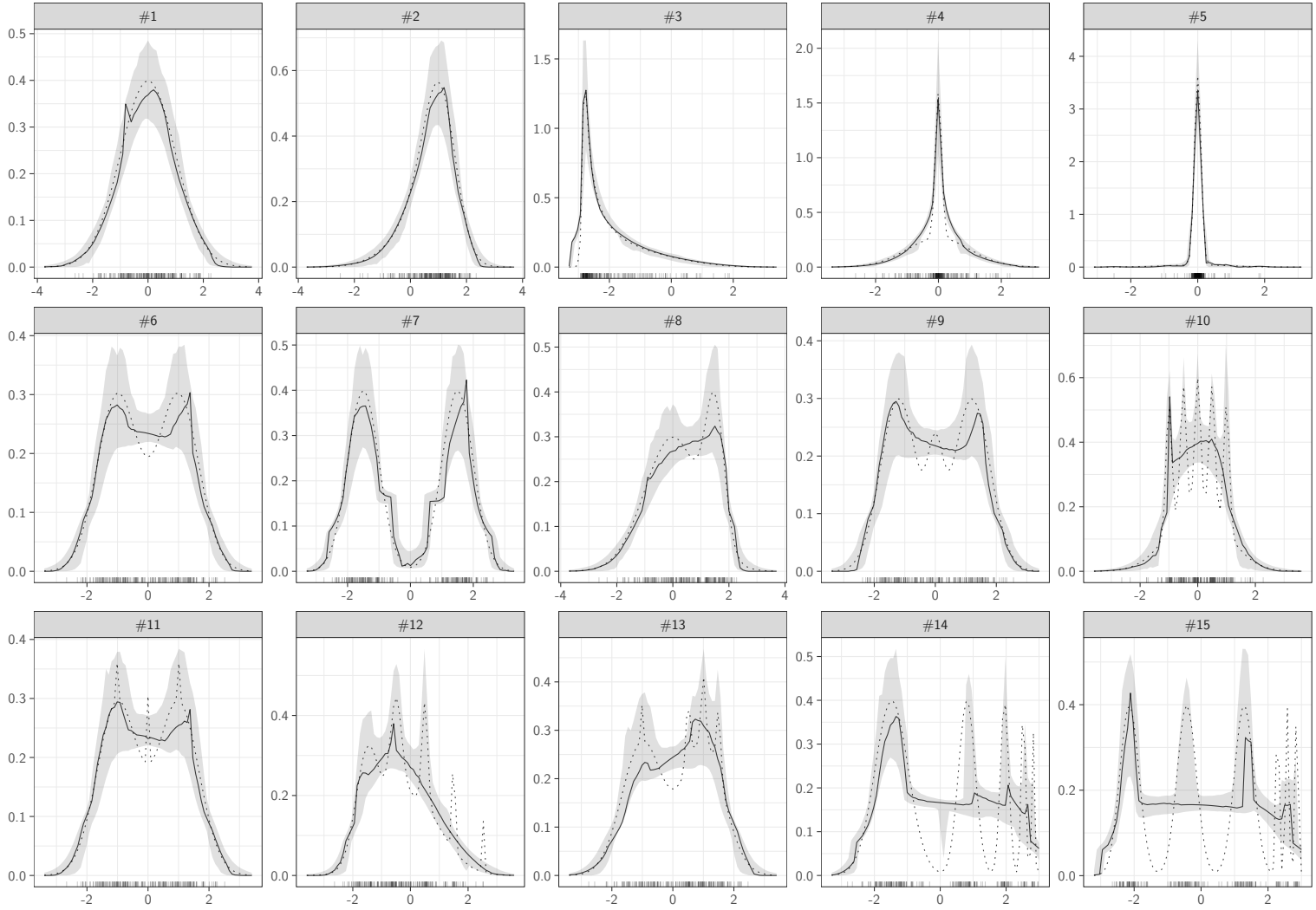


Figure B.24: The 15 normal mixture densities from Marron and Wand (1992) estimated by $\hat{f}_{\text{rodeo}}$ ($q = 1$). The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The shaded region corresponds to the 95 percent confidence region based on $m = 500$ replications.

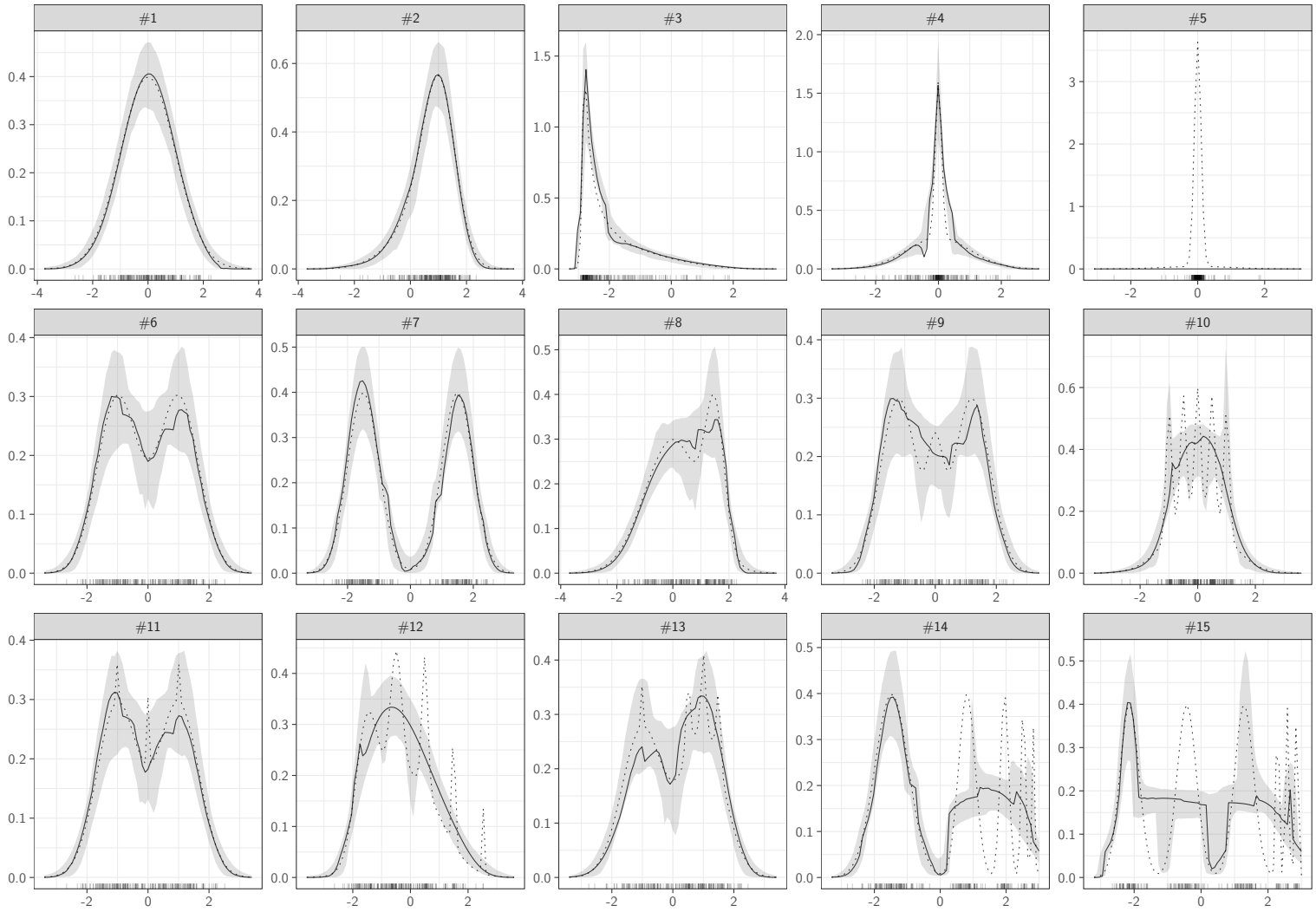


Figure B.25: The 15 normal mixture densities from Marron and Wand (1992) estimated by $\hat{f}_{\text{rodeo}}$ ($q = 2$). The dotted line indicates the true density. The distribution of the sample is indicated by the rugs at the bottom. The shaded region corresponds to the 95 percent confidence region based on $m = 500$ replications.

C R Package `lpderodeo`

This appendix contains the source code of our R package `lpderodeo` version 1.0.0. The package was originally built under R version 4.0.3. Our package is not yet available in the official repositories, and thus has to be installed manually. The guide below explains the procedure.

To build the package skeleton, the R package `RcppArmadillo` version 0.10.1 is required, because our code depends on C/C++ code. The package skeleton is built by running the command

```
#require("RcppArmadillo")
RcppArmadillo::RcppArmadillo.package.skeleton("lpderodeo")
\smallskip
```

in R. The command creates a folder called `lpderodeo` in the current working directory. Next, some files in the newly created folder have to be modified using the files attached to this appendix:

- replace `<current working directory>/lpderodeo/src/rcpp_hello_world.cpp` with `main.cpp`
- replace `<current working directory>/lpderodeo/DESCRIPTION` with `DESCRIPTION`
- replace `<current working directory>/lpderodeo/man/rcpp_hello_world.Rd` with `lpderodeo.Rd`
- delete `<current working directory>/lpderodeo/man/lpderodeo-package.Rd`

Note that unintended line breaks have to be removed. The package attributes are then set by running the command

```
#require("RcppArmadillo")
Rcpp::compileAttributes("lpderodeo/", TRUE)
```

in R. Then another file in the folder has to be modified:

- replace `<current working directory>/lpderodeo/RcppExports.R` with `RcppExports.R`

Finally, the package can be built and installed by executing

```
R CMD build lpderodeo
R CM INSTALL lpderodeo_1.0.tar.gz
```

in the console. After successfully installing the package, it is ready to use. Run the command

```
#require(lpderodeo)
?lpderodeo
```

in R to open the help page. The help page contains some examples and further information on how to use our package.

main.cpp

```
// [[Rcpp::plugins(cpp11)]]

#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]

typedef double (*funcPtr)(arma::mat Q, arma::mat H);

/* Gaussian weighting function */
arma::vec gaussianweight(arma::mat X, arma::mat H) {
  return(std::pow(arma::datum::sqrt2pi, -(int)X.n_cols) / arma::det(H) * arma::exp(-0.5 * arma::diagvec(X * arma::powmat(H.i(), 2) *
  X.t())));
}

/* Constant fitting */
double constantfitting(arma::mat Q, arma::mat H) {
  int n = Q.n_rows;
  arma::mat W = arma::diagmat(gaussianweight(Q, H));
  arma::vec ones = arma::ones(n);

  return(std::pow(n, -1) * arma::dot(ones, W * ones));
}

/* Linear fitting */
double linearfitting(arma::mat Q, arma::mat H) {
  int n = Q.n_rows;
  double cf = constantfitting(Q, H);
  arma::mat W = std::pow(n * cf, -1) * arma::diagmat(gaussianweight(Q, H));
  arma::vec ones = arma::ones(n);
  arma::mat Hinv = H.i();

  return(cf * std::exp(-0.5 * std::pow(arma::norm(Hinv * Q.t() * W * ones, 2), 2)));
}

/* Quadratic fitting */
double quadraticfitting(arma::mat Q, arma::mat H) {
  int n = Q.n_rows;
  double cf = constantfitting(Q, H);
  arma::mat W = std::pow(n * cf, -1) * arma::diagmat(gaussianweight(Q, H));
  arma::vec ones = arma::ones(n);
  arma::mat Asqrt = arma::sqrtmat_sympd(Q.t() * W * Q - Q.t() * W * ones * ones.t() * W * Q).i();

  return(cf * det(H * Asqrt) * std::exp(-0.5 * std::pow(arma::norm(Asqrt * Q.t() * W * ones, 2), 2)));
}

/* Local polynomial density estimation coupled with the RODEO approach for bandwidth selection */

// [[Rcpp::export]]
Rcpp::List lpderodeo(arma::vec x, arma::mat sample, arma::mat hinit, arma::vec beta, std::string theta, arma::vec zeta, std::string
scale, int q, int tmax, int b, double delta, double seed, bool verbose) {
  /* Initialize variables */
  int s = 1;
  int d = x.n_rows;
  int n = sample.n_rows;

  /* set a seed */
  Rcpp::Environment base_env("package:base");
  Rcpp::Function set_seed_r = base_env["set.seed"];
  set_seed_r(std::floor(std::fabs(seed)));

  /* A function pointer that points (by default) to constatfitting */
  funcPtr lpde = *Rcpp::XPtr<funcPtr>(new funcPtr(&constantfitting));

  /* If q = 1 linear fitting is used and quadratic fitting is used if q = 2  */
  if(q == 1) {
    lpde = *Rcpp::XPtr<funcPtr>(new funcPtr(&linearfitting));
  }

  if(q == 2) {
    lpde = *Rcpp::XPtr<funcPtr>(new funcPtr(&quadraticfitting));
```

```
}

/* a variable storing the path of bandwidth parameters (the path is stored in a matrix: each row is a step of the algorithm and each
column an element of the vectorised matrix of bandwidths) */
arma::mat path = arma::mat(arma::trans(arma::vectorise(hinit)));

arma::mat Xi = hinit; // the Xi matrix (eigenvalues)
arma::mat P = arma::eye(d, d); // the P matrix (orthonormal transformation)

/* if Xi is not a diagonal already compute its eigen decomposition */
if(!Xi.is_diagmat()) {
  arma::vec xi;
  arma::eig_sym(xi, P, Xi);

  Xi = arma::diagmat(xi);
}

/* create the Q matrix by subtracting x from all rows */
arma::mat Q = sample; Q.each_row() -= x.t();

/* compute the initial estimate */
double fhat = lpde(Q, P * Xi * P.t());

/* start the algorithm; in every iteration the diagonal components of Xi are reduced as long as the corresponding mu hat is above the
threshold */
while(1) {
  /* initialize mu hat and threshold */
  arma::vec muhat = arma::zeros(d);
  arma::vec threshold = arma::zeros(d);

  /* initialize container for scale estimation */
  arma::mat container;

  if(scale == "stdv2") {
    /* alternative scale computations (only valid for LPDE w const fitting and diagonal initial bandwidth parameter) */
    container = arma::repmat(arma::pow(arma::diagvec(Xi), -3), 1, n).t() % (arma::pow(Q, 2) - arma::repmat(arma::pow(arma::diagvec(Xi),
    2), 1, n).t()) % arma::repmat(gaussianweight(Q, Xi), 1, d);
  }
  else if(scale == "stdv3") {
    /* scale estimator based on the expression for the variance of mu derived by Liu, Lafferty, and Wasserman (2007) */
    container = std::pow(4 * arma::datum::pi, -0.5 * d) * lpde(Q, P * Xi * P.t()) / (4 * n * arma::det(Xi)) * (P * arma::pow(Xi.i(),
    2) * P.t());
  }
  else {
    /* initialize bootstrap sample */
    container = arma::randi<arma::mat>(b, n, arma::distr_param(0, n - 1));
  }

  /* loop over each dimension and compute mu hat and thresholds */
  for(int j = 0; j != d; ++j) {
    /* computations for a dimension can be skipped if already beta = 1 */
    if(beta.at(j) == 1) {
      continue;
    }

    /* Compute the derivative with respect to eigenvalue j numerically */
    arma::mat Ejj = arma::zeros(d, d);
    Ejj.at(j,j) = delta; // this is the naive numerical epsilon difference

    /* computation of the mu hat */
    muhat[j] = (lpde(Q, P * (Xi + Ejj) * P.t()) - lpde(Q, P * Xi * P)) / Ejj.at(j, j);

    /* computation of the threshold */
    double disp;

    if(scale == "stdv2") {
      disp = std::sqrt(arma::var(container.col(j)) / n);
    }
    else if(scale == "stdv3") {
      disp = std::sqrt(container.at(j, j));
    }
    else {
```

```
    /* scale computation based on bootstrap */
    arma::vec bootstraps = arma::zeros(b);

    /* compute bootstrap */ /* TODO there may be more efficient ways to do this */
    for(int l = 0; l != b; ++l) {
      arma::mat Qb = Q.rows(arma::conv_to<arma::uvec>::from(container.row(l)));
      bootstraps.at(l) = (lpde(Qb, P * (Xi + Ejj) * P.t()) - lpde(Qb, P * Xi * P.t())) / Ejj.at(j, j);
    }

    if(scale == "std") {
      /* standard deviation */
      disp = std::sqrt(arma::var(bootstraps));
    }
    else if(scale == "mad") {
      /* median absolute deviation */
      disp = arma::median(arma::abs(bootstraps - arma::median(bootstraps) * arma::ones(b))) / 1.483;
    }
    else if(scale == "iqr") {
      /* interquartile range */
      disp = arma::as_scalar(arma::quantile(bootstraps, (arma::vec){0.75}) - arma::quantile(bootstraps, (arma::vec){0.25})) / 1.349;
    }
    else if(scale == "min") {
      /* minimum of interquartile range and standard deviation */
      disp = std::min(std::sqrt(arma::var(bootstraps)), arma::as_scalar(arma::quantile(bootstraps, (arma::vec){0.75}) -
      arma::quantile(bootstraps, (arma::vec){0.25})) / 1.349);
    }
    else {
      Rcpp::warning("scale = " + scale + " is not supported. Defaulting to scale = 'std'.");

      /* default to std as scale estimator */
      disp = std::sqrt(arma::var(bootstraps));
    }
  }

  threshold[j] = zeta[j] * disp;

  // verbose
  if(verbose) {
    Rcpp::Rcout << "[ Debug ] Step " << s << " in dimension " << j + 1 << ": muhat = " << muhat[j] << " (scale estimator = " << disp
    << "), threshold = " << threshold[j] << ", h = " << Xi.at(j, j) << std::endl;
  }

  /* test if the mu hat is above the threshold */
  if(std::abs(muhat[j]) < threshold[j]) {
    /* set beta in dimension j to 1 so that the bandwidths in that dimension remain unchanged in all future steps */
    beta[j] = 1;

    // verbose
    if(verbose) {
      Rcpp::Rcout << "[ Debug ] Step " << s << " dimension " << j + 1 << ": |muhat| < threshold (done)" << std::endl;
    }
  }

  /* is the mu hat a number at all? If not freeze the current bandwidth for that dimension */
  if(!std::isnormal(muhat[j])) {
    Rcpp::warning("In step " + std::to_string(s) + " in dimension " + std::to_string(j + 1) + ": the value of muhat = " +
    std::to_string(threshold[j]) + " is either zero, approx zero or nan/infinite. Bandwidths for this dimension are frozen.");

    /* the bandwidth parameter is frozen by setting beta for that dimension to 1 */
    beta[j] = 1;
  }
}

/* compute the derivative along the path */
arma::vec dotphi = (beta - arma::ones(d)) % arma::diagvec(Xi);

/* compute the derivative with resepect to the eigenvalue j numerically */
arma::vec dfhatdxi = muhat;

/* Update the rodeo estimator */
if(theta == "hard") {
  fhat = fhat + arma::dot(dfhatdxi, dotphi);
```

```
    }
    else if(theta == "soft") {
      fhat = fhat + arma::dot(arma::sign(dfhatdxi) % (arma::abs(dfhatdxi) - threshold), dotphi);
    }
    else if(theta == "garrote") {
      fhat = fhat + arma::dot(dfhatdxi - arma::pow(threshold, 2) / dfhatdxi, dotphi);
    }
    else if(theta == "hyperbole") {
      /* TODO using abs here is a dirty fix to avoid nans in the last iteration */
      fhat = fhat + arma::dot(arma::sign(dfhatdxi) % arma::sqrt(arma::abs(arma::pow(dfhatdxi, 2) - arma::pow(threshold, 2))), dotphi);
    }
    else {
      Rcpp::warning("theta = " + theta + " is not supported. Defaulting to theta = 'hard'.");

      /* default to hard threshold */
      fhat = fhat + arma::dot(dfhatdxi, dotphi);
    }

    // verbose
    if(verbose) {
      Rcpp::Rcout << "[ Debug ] Step " << s << " rodeo lpde = " << fhat << std::endl;
    }

    /* if beta = 1 in every dimension exit the loop */
    if(arma::sum(beta) == d) {
      // verbose
      if(verbose) {
        Rcpp::Rcout << "[ Debug ] Exit loop after " << s << " steps." << std::endl << std::endl;
      }

      break;
    }

    /* exit the loop if maximum number of iterations is reached */
    if(s > tmax) {
      Rcpp::warning("Maximum number of iterations reached.");

      break;
    }

    /* otherwise update the bandwidths and path */
    Xi = Xi % arma::diagmat(beta);
    path = arma::join_vert(path, arma::trans(arma::vectorise(P * Xi * P.t())));

    /* and update running index */
    ++s;
  }

  /* return the rodeo lpde and the path of bandwidths as list element */
  return(Rcpp::List::create(std::max(fhat, 0.0), path));
}
```

lpderodeo.Rd

```
\name{lpderodeo}\alias{lpderodeo}\title{Using the RODEO Approach for Local Polynomial Density Estimation}\author{Lukas
Moedl}\description{This function estimates an unknown probability density function based on an observed sample. The unknown density is
approximated locally at some evaluation point by a q-order polynomial. The bandwidth parameter for the local polynomial approximation is
selected using the RODEO approach of Lafferty and Wasserman (2008) and Lafferty, Liu and Wasserman (2007).}\usage{
lpderodeo(x, sample,
    hinit = 1 / sqrt(log(log(nrow(sample)))) * diag(ncol(sample)),
    beta = rep(0.9, length(x)),
    theta = "hard",
    zeta = rep(sqrt(2 * log(ncol(sample) * log(nrow(sample)))), length(x)),
    scale = "std",
    q = 0,
```

```
    tmax = 100,
    b = 100,
    delta = 0.0001,
    seed = runif(1, 1, 161),
    verbose = FALSE
  )
}
\arguments{\item{x}{An evaluation point in a d-dimensional domain.}\item{sample}{A n × d matrix containing the observed
sample.}\item{hinit}{The initial bandwidth parameter.}\item{beta}{The tuning parameter controls the bandwidth reduction
rate.}\item{theta}{The tuning parameter controls the amount of bias reduction. Allowed values are: "hard", "soft", "garrote", and
"hyperbole".}\item{zeta}{The tuning parameter controls the threshold values.}\item{scale}{Specifies the scale estimator. Possible values
are "std" for standard deviation (default), "mad" for median absolute deviation, "iqr" for interquartile range based on bootstrap,
"stdv2" for the alternative variance estimator based on the functional form of the LPDE with constant fitting, and "stdv3" for the
variance estimator based on the (wrong) expression derived by Liu, Lafferty, and Wasserman (2007).}\item{q}{Order of the polynomial used
for the approximation: q = 0 (constant), q = 1 (linear) and q = 2 (quadratic) are supported.}\item{tmax}{The maximum number of
iterations for the bandwidth selection algorithm.}\item{b}{Number of replications used to bootstrap the scale estimator of mu
hat.}\item{seed}{Seed to reproduce the results.}\item{delta}{}\item{verbose}{Verbose for debugging.}}\value{A list containing the value
of the local polynomial density estimator at x and a matrix representing the path of selected bandwidth parameters. The rows correspond
to the iterations of the algorithm and the columns contain (vectorized) bandwidth parameter.}\details{Starting from the initial
bandwidth parameter, the algorithm carries out a check in each iteration on whether the bandwidths of a dimension must be shrunk or
not.}\references{Lafferty and Wasserman (2008), Lafferty, Liu and Wasserman (2007).}
\examples{
  # A univariate example
  x = 0
  sample = matrix(rnorm(200))

  lpderodeo(x, sample) # estimate (with constant fitting)
  dnorm(0)             # true value

  # A 10-variate example
  x = rep(0, 10)
  sample = matrix(rnorm(10 * 200), ncol = 10)

  lpderodeo(x, sample, q = 1) # estimate (with linear fitting)
  prod(dnorm(x))              # true value
}
```

DESCRIPTION

```
Package: lpderodeo
Type: Package
Title: The RODEO Approach for Nonparametric Density Estimation
Version: 1.0.0
Date: 2020-08-05
Author: Lukas Moedl
Maintainer: Lukas Moedl <lukas.moedl@hu-berlin.de>
Description: Local polynomial density estimation combined with the RODEO approach for regularization.
License: GPL
Imports: Rcpp (>= 1.0.5)
LinkingTo: Rcpp, RcppArmadillo
```

RcppExports.R

```
lpderodeo <- function(x, sample, hinit = 1 / sqrt(log(log(nrow(sample)))) * diag(ncol(sample)), beta = rep(0.9, length(x)), theta =
"hard", zeta = rep(sqrt(2 * log(ncol(sample) * log(nrow(sample)))), length(x)), scale = "std", q = 0, tmax = 100, b = 100, delta =
0.0001, seed = runif(1, 1, 161), verbose = FALSE) {
  # TODO process input and throw exceptions if necessary for better use experience
  .Call(`_lpderodeo_lpderodeo`, x, sample, hinit, beta, theta, zeta, scale, q, tmax, b, delta, seed, verbose)
}
```

D Used R Codes

The following R codes allow replication of the raw data used for the figures and tables in Section 3. Note that the codes only replicate the raw data – the code for generating the figures and tables itself is not included.

The codes require R version 4.0.3. In addition, the following have to be installed: `mvQuad` version 1.0.6, `lpderodeo` version 1.0.0 (see Appendix C), `ks` version 1.11.7, `mvmesh` version 1.6.0, `mclust` version 5.4.6, `TDA` version 1.6.9, `detpack` version 1.1.3, `kdevine` version 0.4.2, and `gss` version 2.2.2. Furthermore, the R packages `tidyverse` version 1.3.0, and `doParallel` version 1.0.16 are required.

`example1.R`

```
# Libraries
library("lpderodeo")
library("ks")
library("mclust")
library("kdevine")
library("detpack")
library("gss")
library("TDA")
#library("mvmesh")

library("mvQuad")

library("tidyverse")

library("doParallel")
cores = 80 # adjust

# Little helper functions
inrange  = function(x, rng) all(apply(rbind(x, rng), 2, function(x) return(x[1] >= x[2] & x[1] <= x[3])))
dmixnorm = function(x, p, mus, sds) sum(p * mapply(function(m, s) dnorm(x, m, s), mus, sds))
rmixnorm = function(n, p, mus, sds) {s = sample(1:length(p), n, 1, p); rnorm(n, mus[s], sds[s])}

# Accuracy metrics
ise = function(f, g, grid) quadrature(function(x) (f(x) - g(x))^2, grid)

# =========
# Example 1
# =========

# parameter for the 15 test densities
params = list(
  no01 = list(1, 0, 1, c(-3.75, 3.75)),
  no02 = list(c(1/5, 1/5, 3/5), c(0, 1/2, 13/12), c(1, 2/3, 5/9), c(-3.7, 3.7)),
  no03 = list(rep(1/8, 8), 3 * (2/3)^(0:7) - 3, (2/3)^(0:7), c(-3.3, 3.4)),
  no04 = list(c(2/3, 1/3), c(0, 0), c(1, 1/10), c(-3.4, 3.2)),
  no05 = list(c(1/10, 9/10), c(0, 0), c(1, 1/10), c(-3.1, 3.1)),
  no06 = list(c(1/2, 1/2), c(-1, 1), c(2/3, 2/3), c(-3.4, 3.4)),
  no07 = list(c(1/2, 1/2), c(-3/2, 3/2), c(1/2, 1/2), c(-3.3, 3.3)),
  no08 = list(c(3/4, 1/4), c(0, 3/2),  c(1, 1/3), c(-3.7, 3.7)),
  no09 = list(c(9/20, 9/20, 1/10), c(-6/5, 6/5, 0), c(3/5, 3/5, 1/4), c(-3.4, 3.4)),
  no10 = list(c(1/2, rep(1/10, 5)), c(0, (0:4)/2 - 1), c(1, rep(1/10, 5)), c(-3.6, 3.6)),
  no11 = list(c(49/100, 49/100, rep(1/350, 7)), c(-1, 1, (0:6)/2 - 3/2), c(2/3, 2/3, rep(1/100, 7)), c(-3.4, 3.4)),
  no12 = list(c(1/2, 2^(1 - (-2:2))/31), c(0, (-2:2) + 1/2), c(1, 2^(-1 * (-2:2))/10), c(-3.6, 3.6)),
  no13 = list(c(46/100, 46/100, rep(1/300, 3), rep(7/300, 3)), c(-1, 1, -1 * (1:3)/2, (1:3)/2), c(2/3, 2/3, rep(1/100, 3), rep(7/100,
  3)), c(-3.4, 3.4)),
  no14 = list(2^(5 - (0:5))/63, (65 - 96 * 1/2^(0:5))/21, 32/63 * 1/2^(0:5), c(-3.3, 3)),
```

```
  no15 = list(c(2/7, 2/7, 2/7, 1/21, 1/21, 1/21), c((12 * (0:2) - 15)/7, 2/7 * (8:10)), c(2/7, 2/7, 2/7, 1/21, 1/21, 1/21), c(-3.2, 3))
)

# replications and sample size
m = 500
n = 200
d = 1

# data
df = tibble()

# running index of densities no
no = 1

while(no  <= 15) {
  # set seed
  set.seed(161)

  # replicate m times n random draws from corresponding normal mixture distributions
  S = map(1:m, ~as.matrix(rmixnorm(n, params[[no]][[1]], params[[no]][[2]], params[[no]][[3]])))

  # create a grid for plotting and integration
  grid1 = as.matrix(seq(params[[no]][[4]][1], params[[no]][[4]][2], l = 75))
  grid2 = createNIGrid(d, "nLe", 25, "sparse")
  rescale.NIGrid(grid2, domain = c(params[[no]][[4]][1], params[[no]][[4]][2]))
  X = grid1

  # the true function
  f = function(x) map_dbl(array_branch(x, 1), ~dmixnorm(.x, params[[no]][[1]], params[[no]][[2]], params[[no]][[3]]))

  # parallel computation of runs
  runs = mclapply(1:m, function(run) {
    # whiten sample
    scl = 1 / sd(S[[run]])
    cnt = mean(S[[run]])
    s   = scl * (S[[run]] - cnt)

    # estimation objects
    hns   = (4 / (n * (d + 2)))^(1 / (d + 4)) * diag(d)
    hpi   = ks::hpi(s)
    hhist = graphics::hist(s, breaks = seq(min(s), max(s), l = nclass.FD(s)), plot = F)
    hgmm  = mclust::Mclust(s, verbose = F)
    hknn  = floor(sqrt(n))
    hdet  = detpack::det.construct(t(s), mode = 1, progress = F, cores = cores)
    hvine = kdevine::kde1d(s)
    hss   = gss::ssden(~ ., data = as.data.frame(s), domain = as.data.frame(as.matrix(apply(rbind(s, sweep(X, 2, cnt) * scl), 2,
    range))))

    # Vectorized stimatates
    fhats = list(
      function(x) map_dbl(array_branch(x, 1),
        ~scl * lpderodeo::lpderodeo((.x - cnt) * scl, s)[[1]]),
      function(x) map_dbl(array_branch(x, 1),
        ~scl * dmixnorm((.x - cnt) * scl, rep(1 / n, n), s, rep(hns, n))),
      function(x) map_dbl(array_branch(x, 1),
        ~scl * dmixnorm((.x - cnt) * scl, rep(1 / n, n), s, rep(hpi, n))),
      function(x) map_dbl(array_branch(x, 1),
        ~scl * ifelse(inrange((.x - cnt) * scl, as.matrix(range(hhist$breaks))), hhist$density[graphics::hist((.x - cnt) * scl, breaks =
        hhist$breaks, plot = F)$density > 0], 0)),
      function(x) map_dbl(array_branch(x, 1),
        ~scl * dmixnorm((.x - cnt) * scl, hgmm$parameters$pro, hgmm$parameters$mean, sqrt(hgmm$parameters$variance$sigmasq))),
      function(x) map_dbl(array_branch(x, 1),
        ~scl * TDA::knnDE(s, (.x - cnt) * scl, hknn)),
      function(x) map_dbl(array_branch(x, 1),
        ~scl * detpack::det.query(hdet, (.x - cnt) * scl)),
      function(x) map_dbl(array_branch(x, 1),
        ~scl * kdevine::dkdevine((.x - cnt) * scl, hvine)),
      function(x) map_dbl(array_branch(x, 1),
        ~scl * ifelse(inrange((.x - cnt) * scl, hss$domain), gss::dssden(hss, as.data.frame(t(c((.x - cnt) * scl)))), 0))
    )

    # create data
```

```
    tibble(no, run, grid1 = grid1) %>%
      mutate(
        fhat1 = fhats[[1]](X),
        fhat2 = fhats[[2]](X),
        fhat3 = fhats[[3]](X),
        fhat4 = fhats[[4]](X),
        fhat5 = fhats[[5]](X),
        fhat6 = fhats[[6]](X),
        fhat7 = fhats[[7]](X),
        fhat8 = fhats[[8]](X),
        fhat9 = fhats[[9]](X),
        ftrue = f(X)
      ) %>%
      mutate(across(grid1, map(fhats, ~function(x) ise(.x, f, grid2)), .names = "ise_{fn}")) %>%
      do.call(data.frame, .) %>% as_tibble() %>%
      return();

      message("no: ", no, " run: ", run)

  }, mc.cores = cores)

  # update data
  df = runs %>% bind_rows() %>% bind_rows(df)

  # update running index
  no = no + 1
}

# write raw data to file
df %>%
  write_csv("data/example1_raw.csv")
```

example2.R

```
# Libraries
library("lpderodeo")
library("ks")
library("mclust")
library("kdevine")
library("detpack")
library("gss")
library("mvmesh")
library("TDA")

library("mvQuad")

library("tidyverse")

library("doParallel")
cores = 80 # adjust

# Little helper functions
inrange    = function(x, rng) all(apply(rbind(x, rng), 2, function(x) return(x[1] >= x[2] & x[1] <= x[3])))
dmixmvnorm = function(x, p, mus, sds) sum(p * mapply(function(m, s) mclust::dmvnorm(x, m, s), mus, sds))
margf      = function(f, y, grid) quadrature(function(x) f(cbind(x, y)), grid)
mvhist     = function(x, h) {
  a = mvmesh::TallyHrep(x, mvmesh::V2Hrep(h$mesh$S), report = "none")
  # return relative frequence divided by area of simplex to obtain an estimate for the density
  return(h$rel.freq[a$rel.freq > 0] / det(matrix(c(-1,0,1,-1,0,0,0,1), ncol = 4) %*% h$mesh$S[,,a$which.simplex]))
}

# Accuracy metrics
ise = function(f, g, grid) return(quadrature(function(x) (f(x) - g(x))^2, grid))

# =========
```

```r
# Example 2
# =========

# Beta mixture routines
dmixbeta = function(x, p, shape1, shape2) sapply(x, function(x) sum(p * dbeta(x, shape1, shape2)))
rmixbeta = function(n, p, shape1, shape2) {cmpnts = sample(1:length(p), n, 1, p); rbeta(n, shape1[cmpnts], shape2[cmpnts])}

# replications and sample size
m = 500
n = 500
d = 2

# data
df = tibble()

# set seed
set.seed(161)

# replicate m times n random draws from corresponding Beta mixture distributions
S = map(1:m, ~cbind(rmixbeta(n, c(2/3, 1/3), c(1, 10), c(2, 10)), matrix(runif(n, 0, 1))))

# create a grid for plotting and integration
grid1 = as.matrix(seq(0, 1, l = 25))
grid21 = createNIGrid(1, "nLe", 15, "sparse")
grid22 = createNIGrid(2, "nLe", 25, "sparse")
X = as.matrix(expand.grid(grid1, grid1))

# the true functions
f   = function(x) map_dbl(array_branch(x, 1), ~dmixbeta(.x[1], c(2/3, 1/3), c(1, 10), c(2, 10)) * dunif(.x[2], 0, 1))
m1f = function(y) map_dbl(array_branch(y, 1), ~dunif(.x, 0, 1))

# parallel computation of runs
runs = mclapply(1:m, function(run) {
  # whiten sample
  eig = eigen(cov(sweep(S[[run]], 2, apply(S[[run]], 2, mean))), T)
  cnt = apply(S[[run]], 2, mean)
  scl = eig$vectors %*% diag(1 / sqrt(eig$values)) #%*% t(eig$vectors)
  s   = sweep(S[[run]], 2, cnt) %*% scl

  # estimation objects
  hns   = (4 / (n * (d + 2)))^(1 / (d + 4)) * diag(d)
  hpi   = ks::Hpi(s)
  hhist = mvmesh::histRectangular(s, apply(s, 2, nclass.FD), plot = "none")
  hgmm  = mclust::Mclust(s, verbose = F)
  hknn  = floor(sqrt(n))
  hdet  = detpack::det.construct(t(s), mode = 1, progress = F, cores = cores)
  hvine = kdevine::kdevine(s)
  hss   = gss::ssden1(~ ., data = as.data.frame(s), domain = as.data.frame(as.matrix(apply(rbind(s, sweep(X, 2, cnt) %*% scl), 2,
  range))))

  # Vectorized estimatates
  fhats = list(
    function(x) map_dbl(array_branch(x, 1),
      ~abs(det(scl)) * lpderodeo::lpderodeo((.x - cnt) %*% scl, s)[[1]]),
    function(x) map_dbl(array_branch(x, 1),
      ~abs(det(scl)) * dmixmvnorm((.x - cnt) %*% scl, rep(1 / n, n), asplit(s, 1), replicate(n, hns, FALSE))),
    function(x) map_dbl(array_branch(x, 1),
      ~abs(det(scl)) * dmixmvnorm((.x - cnt) %*% scl, rep(1 / n, n), asplit(s, 1), replicate(n, hpi, FALSE))),
    function(x) map_dbl(array_branch(x, 1),
      ~abs(det(scl)) * ifelse(inrange((.x - cnt) %*% scl, rbind(hhist$mesh$a, hhist$mesh$b)), mvhist((.x - cnt) %*% scl, hhist), 0)),
    function(x) map_dbl(array_branch(x, 1),
      ~abs(det(scl)) * dmixmvnorm((.x - cnt) %*% scl, hgmm$parameters$pro, asplit(hgmm$parameters$mean, 2),
      asplit(hgmm$parameters$variance$sigma, 3))),
    function(x) map_dbl(array_branch(x, 1),
      ~abs(det(scl)) * TDA::knnDE(s, (.x - cnt) %*% scl, hknn)),
    function(x) map_dbl(array_branch(x, 1),
      ~abs(det(scl)) * detpack::det.query(hdet, t((.x - cnt) %*% scl))),
    function(x) map_dbl(array_branch(x, 1),
      ~abs(det(scl)) * kdevine::dkdevine((.x - cnt) %*% scl, hvine)),
    function(x) map_dbl(array_branch(x, 1),
      ~abs(det(scl)) * ifelse(inrange((.x - cnt) %*% scl, hss$domain), gss::dssden(hss, as.data.frame(t(c((.x - cnt) %*% scl)))), 0))
  )
```

```
  # Vectorized marginal estimatates
  m1fhats = list(
    function(y) map_dbl(array_branch(y, 1), ~margf(fhats[[1]], .x, grid21)),
    function(y) map_dbl(array_branch(y, 1), ~margf(fhats[[2]], .x, grid21)),
    function(y) map_dbl(array_branch(y, 1), ~margf(fhats[[3]], .x, grid21)),
    function(y) map_dbl(array_branch(y, 1), ~margf(fhats[[4]], .x, grid21)),
    function(y) map_dbl(array_branch(y, 1), ~margf(fhats[[5]], .x, grid21)),
    function(y) map_dbl(array_branch(y, 1), ~margf(fhats[[6]], .x, grid21)),
    function(y) map_dbl(array_branch(y, 1), ~margf(fhats[[7]], .x, grid21)),
    function(y) map_dbl(array_branch(y, 1), ~margf(fhats[[8]], .x, grid21)),
    function(y) map_dbl(array_branch(y, 1), ~margf(fhats[[9]], .x, grid21))
  )

  # create data
  df1 = tibble(run, plt = 1, grid1 = X) %>%
    mutate(
      fhat1 = fhats[[1]](X),
      fhat2 = fhats[[2]](X),
      fhat3 = fhats[[3]](X),
      fhat4 = fhats[[4]](X),
      fhat5 = fhats[[5]](X),
      fhat6 = fhats[[6]](X),
      fhat7 = fhats[[7]](X),
      fhat8 = fhats[[8]](X),
      fhat9 = fhats[[9]](X),
      ftrue = f(X)
    ) %>%
    mutate(across(grid1, map(fhats, ~function(x) ise(.x, f, grid22)), .names = "ise_{fn}"))

  df2 = tibble(run, plt = 2, grid1 = grid1) %>%
    mutate(
      fhat1 = m1fhats[[1]](grid1),
      fhat2 = m1fhats[[2]](grid1),
      fhat3 = m1fhats[[3]](grid1),
      fhat4 = m1fhats[[4]](grid1),
      fhat5 = m1fhats[[5]](grid1),
      fhat6 = m1fhats[[6]](grid1),
      fhat7 = m1fhats[[7]](grid1),
      fhat8 = m1fhats[[8]](grid1),
      fhat9 = m1fhats[[9]](grid1),
      ftrue = m1f(grid1)
    ) %>%
    mutate(across(grid1, map(m1fhats, ~function(x) ise(.x, m1f, grid21)), .names = "ise_{fn}"))

  bind_rows(df1, df2) %>% return()

  message("run: ", run)

}, mc.cores = cores)

# write raw data to file
runs %>%
  bind_rows() %>%
  write_csv("data/example2_raw.csv")
```

example3.R

```
# Libraries
library("lpderodeo")
library("ks")
library("mclust")
library("kdevine")
library("detpack")
library("gss")
```

```r
library("TDA")
#library("mvmesh")

library("tidyverse")

library("doParallel")
cores = 80 # adjust

# Little helper functions
inrange    = function(x, rng) all(apply(rbind(x, rng), 2, function(x) return(x[1] >= x[2] & x[1] <= x[3])))
dmixmvnorm = function(x, p, mus, sds) sum(p * mapply(function(m, s) mclust::dmvnorm(x, m, s), mus, sds))

# =========
# Example 3
# =========

# Load data
raw = "https://archive.ics.uci.edu/ml/machine-learning-databases/magic/magic04.data" %>%
    read_csv(col_names = FALSE) %>%
    mutate(id = row_number(), class = case_when(X11 == "g" ~ 1, X11 == "h" ~ 0)) %>%
    select(id, class, X1:X10)

# set seed
set.seed(161)

# train test split
train = raw %>% group_by(class) %>% sample_frac(0.66) %>% ungroup()
test  = raw %>% anti_join(train, "id")

# select relevant variables
S = train %>% split(train$class) %>% map(~.x %>% select(-id, -class) %>% as.matrix())
X = test %>% select(-id, -class) %>% as.matrix()

# parallel computation for each class
df = mclapply(1:2, function(class) {
  # dimension and sample size
  n = nrow(S[[class]])
  d = ncol(S[[class]])

  # white the sample
  eig = eigen(cov(sweep(S[[class]], 2, apply(S[[class]], 2, mean))), T)
  cnt = apply(S[[class]], 2, mean)
  scl = eig$vectors %*% diag(1 / sqrt(eig$values)) %*% t(eig$vectors)
  s   = sweep(S[[class]], 2, cnt) %*% scl

  # estimation objects
  hns   = (4 / (n * (d + 2)))^(1 / (d + 4)) * diag(d)
  hpi   = round(ks::Hpi(s), 12)
  hhist = NA # doesnt work for 10-variate data
  hgmm  = mclust::Mclust(s, verbose = T)
  hknn  = sqrt(n)
  hdet  = detpack::det.construct(t(s), mode = 1, progress = T, cores = cores)
  hvine = kdevine::kdevine(s)
  hss   = gss::ssden1(~ ., data = as.data.frame(s), domain = as.data.frame(as.matrix(apply(rbind(s, sweep(X, 2, cnt) %*% scl), 2,
  range))))

  # Vectorized stimatates
  fhats = list(
    function(x) map_dbl(array_branch(x, 1),
      -abs(det(scl)) * lpderodeo::lpderodeo((.x - cnt) %*% scl, s, scale = "stdv2", verbose = T)[[1]]),
    function(x) map_dbl(array_branch(x, 1),
      -abs(det(scl)) * dmixmvnorm((.x - cnt) %*% scl, rep(1 / n, n), asplit(s, 1), replicate(n, hns, FALSE))),
    function(x) map_dbl(array_branch(x, 1),
      -abs(det(scl)) * dmixmvnorm((.x - cnt) %*% scl, rep(1 / n, n), asplit(s, 1), replicate(n, hpi, FALSE))),
    function(x) map_dbl(array_branch(x, 1),
      -abs(det(scl)) * 0),
    function(x) map_dbl(array_branch(x, 1),
      -abs(det(scl)) * dmixmvnorm((.x - cnt) %*% scl, hgmm$parameters$pro, asplit(hgmm$parameters$mean, 2),
      asplit(hgmm$parameters$variance$sigma, 3))),
    function(x) map_dbl(array_branch(x, 1),
      -abs(det(scl)) * TDA::knnDE(s, (.x - cnt) %*% scl, hknn)),
    function(x) map_dbl(array_branch(x, 1),
```

```
      -abs(det(scl)) * detpack::det.query(hdet, (.x - cnt) %*% scl)),
    function(x) map_dbl(array_branch(x, 1),
      -abs(det(scl)) * kdevine::dkdevine((.x - cnt) %*% scl, hvine)),
    function(x) map_dbl(array_branch(x, 1),
      -abs(det(scl)) * ifelse(inrange((.x - cnt) %*% scl, hss$domain), gss::dssden(hss, as.data.frame(t(c((.x - cnt) %*% scl)))), 0))
  )

  # create data
  test %>%
    select(id, class) %>%
    mutate(
      fhat1 = fhats[[1]](X),
      fhat2 = fhats[[2]](X),
      fhat3 = fhats[[3]](X),
      fhat4 = fhats[[4]](X),
      fhat5 = fhats[[5]](X),
      fhat6 = fhats[[6]](X),
      fhat7 = fhats[[7]](X),
      fhat8 = fhats[[8]](X),
      fhat9 = fhats[[9]](X)
    ) %>%
    gather(key, val, fhat1:fhat9) %>%
    return()

}, mc.cores = cores)

# write raw data to file
df %>% reduce(inner_join, by = c("id", "class", "key")) %>%
  mutate(bayes = ifelse(val.x + val.y == 0, 0, val.x / val.x + val.y)) %>%
  write_csv("example3_raw.csv")
```

example4.R

```
# Libraries
#library("lpderodeo")
library("mclust")
#library("kdevine")
library("detpack")
#library("gss")
library("TDA")
#library("mvmesh")

library("tidyverse")

library("doParallel")
cores = 8 # adjust

# Little helper functions
dmixmvnorm = function(x, p, mus, sds) sum(p * mapply(function(m, s) mclust::dmvnorm(x, m, s), mus, sds))
logknnde   = function(x, k, data) log(k / nrow(data)) - (ncol(data) / 2) * log(pi) + lgamma(ncol(data) / 2 + 1) - ncol(data) *
log(apply(FNN::knnx.dist(data, t(x), k, "kd_tree"), 1, max))

# importance sampler
fhat.rnd = function(prop, estimate) {
  # importance sampling probabilities
  P = map_dbl(asplit(prop, 2), ~ estimate(.x) - mclust::dmvnorm(t(.x), rep(0, nrow(prop)), diag(nrow(prop)), TRUE))

  return(prop[, sample(1:ncol(prop), 1, 1, exp(P) / sum(exp(P)))])
}

# =========
# Example 4
# =========

# load data and transform it into handy format
```

```
raw = "https://web.stanford.edu/~hastie/ElemStatLearn/datasets/zip.train.gz" %>%
    read_delim(col_names = FALSE, delim = " ") %>%
    mutate(id = row_number(), digit = X1) %>%
    select(id, digit, X2:X257) %>%
    mutate(across(c(-id, -digit), ~ (1 - .x) / 2)) %>%
    nest(data = c(-id, -digit)) %>%
    transmute(id, digit, data = unname(as.matrix(bind_rows(data))))

# random draws
l = 100

# parallel computation for each digit
df = mclapply(0:9, function(no) {
  # set seed
  set.seed(161)

  # train data set
  S = raw %>% filter(digit == no)

  # ZCA transform
  eig = eigen(cov(sweep(S$data, 2, apply(S$data, 2, mean))), T)
  cnt = apply(S$data, 2, mean)
  scl = eig$vectors %*% diag(1 / sqrt(0.1 + eig$values)) %*% t(eig$vectors)

  S = S %>% mutate(datazca = sweep(S$data, 2, cnt) %*% scl)

  # replications, dimension and sample size
  m = 500
  n = nrow(S$datazca)
  d = ncol(S$datazca)

  # proposal
  X = map(1:l, ~ t(MASS::mvrnorm(m, rep(0, d), diag(d))))

  # estimation objects
  hns   = (4 / (n * (d + 2)))^(1 / (d + 4)) * diag(d)
  hhist = NA
  hgmm  = mclust::Mclust(S$datazca, verbose = T)
  hknn  = sqrt(n)
  hdet  = detpack::det.construct(t(S$datazca), mode = 1, progress = T, cores = cores)
  hvine = NA
  hss   = NA

  # sample from the estimated densities and update data
  tibble(digit = no, pix = 1:d) %>%
    mutate(
      fhat1 = matrix(0, ncol = d),
      fhat2 = t(matrix(unlist(map(1:l, ~fhat.rnd(X[.x], function(x) log(dmixmvnorm(x, rep(1 / n, n), asplit(S$datazca, 1), replicate(n,
      hns, FALSE)))))), ncol = d, byrow = T)),
      fhat3 = matrix(0, ncol = d),
      fhat4 = matrix(0, ncol = d),
      fhat5 = t(matrix(unlist(map(1:l, ~fhat.rnd(X[.x], function(x) log(dmixmvnorm(x, hgmm$parameter$pro, asplit(hgmm$parameter$mean, 2),
      asplit(hgmm$parameter$variance$sigma, 3)))))), ncol = d, byrow = T)),
      fhat6 = t(matrix(unlist(map(1:l, ~fhat.rnd(X[.x], function(x) logknnde(x, hknn, S$datazca)))), ncol = d, byrow = T)),
      fhat7 = t(matrix(tryCatch(unlist(map(1:m, ~det.rnd(1, hdet))), error = function(x) 0), ncol = d, byrow = T)),
      fhat8 = matrix(0, ncol = d),
      fhat9 = matrix(0, ncol = d),
      ftrue = S$data
    ) %>%
    mutate(across(starts_with("fhat"),  ~c(solve(scl) %*% rowMeans(.x)))) %>%
    return()

    message("digit: ", no)

}, mc.cores = cores)

# write raw data to file
df %>% bind_rows() %>%
  write_csv("data/example4_raw.csv")
```

llw2007example4.R

```
# Libraries
library("lpderodeo")

library("tidyverse")

library("doParallel")
cores = 75 # adjust

# replications and sample size
m = 30
n = 100

# data
df = tibble()

# set a seed
set.seed(161)

# replicate m times n random draws from the synthetic normal distribution with irrelevant components
S = map(1:m, ~cbind(rnorm(n, 0, 0.02), rnorm(n, 0, 0.04), rnorm(n, 0, 0.06), rnorm(n, 0, 0.08), rnorm(n, 0, 0.1), matrix(runif(25 * n, 0,
1), ncol = 25)))

# evaluation point
x = rep(0, 30)

# bandwidth selection
runs = mclapply(1:m, function(run) {
  s = S[[run]]

  # whiten sample
  cnt = apply(s, 2, mean)
  scl = eigen(cov(sweep(s, 2, cnt)))$vectors %*% diag(1 / sqrt(eigen(cov(sweep(s, 2, cnt)))$values))

  # create data
  tibble(run = rep(run, ncol(s)), dim = 1:30) %>%
    mutate(Hrodeo0 = tryCatch(diag(matrix(tail(pluck(lpderodeo(x, s, q = 0), 2), 1), ncol = 30)), error = function(x) NA)) %>%
    mutate(Hrodeo1 = tryCatch(diag(matrix(tail(pluck(lpderodeo(x, s, q = 1), 2), 1), ncol = 30)), error = function(x) NA)) %>%
    mutate(Hrodeo2 = tryCatch(diag(matrix(tail(pluck(lpderodeo(x, s, q = 2), 2), 1), ncol = 30)), error = function(x) NA)) %>%
    mutate(Hrodeo0v = tryCatch(diag(matrix(tail(pluck(lpderodeo(scl %*% (x - cnt), sweep(s, 2, cnt) %*% scl , q = 0), 2), 1), ncol =
    30)), error = function(x) NA)) %>%
    mutate(Hrodeo1v = tryCatch(diag(matrix(tail(pluck(lpderodeo(scl %*% (x - cnt), sweep(s, 2, cnt) %*% scl, q = 1), 2), 1), ncol = 30)),
    error = function(x) NA)) %>%
    mutate(Hrodeo2v = tryCatch(diag(matrix(tail(pluck(lpderodeo(scl %*% (x - cnt), sweep(s, 2, cnt) %*% scl, q = 2), 2), 1), ncol = 30)),
    error = function(x) NA)) %>%
    do.call(data.frame, .) %>% as_tibble() %>%
    return()
}, mc.cores = cores)

# update data
df = runs %>% bind_rows() %>% bind_rows(df)

# save raw data to file
df %>% write_csv("data/example4llw2007_raw.csv")
```