

R code for kernel estimators

Gautam TRIPATHI

25th March 2019

smoothing-functions.R

```
1  # R code for kernel estimator of pdf_X(x) and E(Y|X=x).
2  # Original code and idea: Gautam Tripathi, 2017-03-08
3  # Rewrite: Andrei V. Kostyrka
4  # v1.0: 2019-03-07
5  # v1.1: 2019-03-18 (fixed an error in DCV that caused severe over-smoothing)
6
7  kernelFun <- function(x, # The values to compute the kernel
8                        kernel = "gaussian", # Kernel type: uniform, epanechnikov, triangular, quartic,
9                        ↪ gaussian
10                       rescale = TRUE, # Rescale to unit variance:  $\int_{-\infty}^{\infty} x^2 k(x) dx = \sigma^2_K =$ 
11                       ↪ 1
12                       convolution = FALSE # Return the convolution kernel? Used for CV
13 ) {
14   adj.factor <- switch(kernel,
15     uniform = sqrt(3),
16     triangular = sqrt(6),
17     epanechnikov = sqrt(5),
18     quartic = sqrt(7),
19     gaussian = 1
20   )
21   if (!rescale) adj.factor <- 1
22   x <- x / adj.factor
23   if (!convolution) {
24     k <- switch(kernel,
25       uniform = 1 / 2 * (abs(x) < 1),
26       triangular = (1 - abs(x)) * (abs(x) < 1),
27       epanechnikov = 3 / 4 * (1 - x^2) * (abs(x) < 1),
28       quartic = 15 / 16 * (1 - x^2)^2 * (abs(x) < 1),
29       gaussian = dnorm(x)
30     )
31   } else {
32     k <- switch(kernel,
33       uniform = 1 / 4 * (2 - abs(x)) * (abs(x) < 2),
34       triangular = 1 / 6 * ((3 * abs(x)^3 - 6 * x^2 + 4) * (abs(x) <= 1) + (8 - 12 * abs(x) + 6 * x^2
35       ↪ - abs(x)^3) * (abs(x) > 1 & abs(x) < 2)),
36       epanechnikov = 3 / 160 * (32 - 40 * x^2 + 20 * abs(x)^3 - abs(x)^5) * (abs(x) < 2),
37       quartic = 225 / 256 * (-128 / 105 * x^2 + 16 / 15 * x^4 + 256 / 315 + 4 / 105 * abs(x)^7 - 1 /
38       ↪ 630 * abs(x)^9 - 8 / 15 * abs(x)^5) * (abs(x) < 2),
39       gaussian = dnorm(x, sd = sqrt(2))
40     )
41   }
42   k <- k / adj.factor
43   return(k)
44 }
45
46 kernelWeights <- function(x, # A numeric vector or numeric matrix
47                          xgrid = NULL, # A numeric vector or numeric matrix with ncol(xgrid) =
48                          ↪ ncol(x)
49                          bw, # Bandwidth: a scalar or a vector of the same length as ncol(x)
50                          kernel = "gaussian", # Passed to kernelFun
51                          rescale = TRUE, # Passed to kernelFun
52                          convolution = FALSE # Passed to kernelFun
53 ) {
54   if (is.null(xgrid)) xgrid <- x # If no grid was passed, use existing data points as the grid
55   one.dim <- is.vector(x) # Are our data one-dimensional?
56   if (one.dim) {
57     if (length(bw) > 1) stop("For one-dimensional kernel weights, the bandwidth must be a scalar.")
58   }
59 }
```

```

53     diffs <- outer(x, xgrid, "-")
54     PK <- kernelFun(diffs / bw, kernel = kernel, rescale = rescale, convolution = convolution)
55   } else {
56     if (!is.matrix(x)) stop("x should be either a numeric vector or a numeric matrix.")
57     s <- ncol(x) # The dimension of data
58     if (ncol(x) != ncol(xgrid)) stop("x and xgrid must have the same number of columns.")
59     if (length(bw) == 1) bw <- rep(bw, s)
60     if (length(bw) != s) stop("For multi-dimensional kernel weights, the bandwidth must have the same
    ↪ length as ncol(x).")
61     nx <- nrow(x)
62     nxgrid <- nrow(xgrid)
63     PK <- matrix(1, nrow = nx, ncol = nxgrid) # Initialising the product kernel
64     for (i in 1:s) {
65       K <- kernelFun(outer(x[, i], xgrid[, i], "-") / bw[i], kernel = kernel, rescale = rescale,
    ↪ convolution = convolution)
66       PK <- PK * K
67     }
68   }
69   return(PK)
70 }
71
72 # Function for estimating pdf_X(x).
73 kernelDensity <- function(x, xgrid = NULL, bw, kernel = "gaussian", rescale = TRUE # Arguments passed
    ↪ to kernelWeights(...))
74 ) {
75   one.dim <- is.vector(x) # Are our data one-dimensional?
76   if (!one.dim & length(bw) == 1) bw <- rep(bw, ncol(x))
77   K <- kernelWeights(x = x, xgrid = xgrid, bw = bw, kernel = kernel, rescale = rescale)
78   dens <- colSums(K) / (nrow(K) * prod(bw))
79   return(dens)
80 }
81
82 # Function for estimating E(Y|X=x) (possibly leaving one observation out).
83 kernelSmooth <- function(x, # Passed to kernelWeights(...)
    y, # A vector of observations
    xgrid = NULL, bw, kernel = "gaussian", rescale = TRUE, # Passed to
    ↪ kernelWeights(...)
    LOO = FALSE # Return the leave-one-out estimator?
84 ) {
85   if (is.numeric(xgrid)) {
86     if (LOO & !isTRUE(all.equal(x, xgrid))) stop("The Leave-one-out estimator must use the same xgrid
    ↪ as x or NULL.")
87   }
88   K <- kernelWeights(x = x, xgrid = xgrid, bw = bw, kernel = kernel, rescale = rescale)
89   if (LOO) diag(K) <- 0
90   num <- colSums(sweep(K, 1, y, "*"))
91   den <- colSums(K)
92   muhat <- num / den
93   return(muhat)
94 }
95
96 # Density cross-validation for the estimator of f_X(x).
97 DCV <- function(x, bw, kernel = "gaussian", rescale = TRUE) {
98   one.dim <- is.vector(x) # Are our data one-dimensional?
99   if (one.dim) {
100     n <- length(x)
101   } else {
102     n <- nrow(x)
103     if (length(bw) == 1) bw <- rep(bw, ncol(x))
104   }
105   KK <- kernelWeights(x, x, bw = bw, kernel = kernel, rescale = rescale, convolution = TRUE)
106   term1 <- sum(KK) / (n^2 * prod(bw))
107   # Computing the LOO estimator efficiently: fhat_i(x) = n/(n-1) * fhat(x) - 1/((n-1)*b^s) * K((X[i] -
    ↪ x)/b)
108   fhat <- kernelDensity(x, bw = bw, kernel = kernel, rescale = rescale)
109   fhat.LOO <- (n * fhat - kernelFun(0, kernel = kernel, rescale = rescale) / (prod(bw))) / (n - 1)
110   term2 <- -2 * mean(fhat.LOO)
111   return(term1 + term2)
112 }
113
114 # Least-squares cross-validation function for Nadaraya-Watson estimator of E(Y|X).
115 LSCV <- function(x, y, bw, kernel = "gaussian", rescale = TRUE) {
116   muhat_i <- kernelSmooth(x = x, y = y, bw = bw, kernel = kernel, rescale = rescale, LOO = TRUE)

```

```

120 ASE <- mean((y - muhat_i)^2)
121 return(ASE)
122 }

```

smoothing-simulation-01-univariate.R

```

1  # R code for kernel estimator of pdf_X and E(Y|X).
2  # Original code and idea: Gautam Tripathi
3  # Rewrite: Andrei V. Kostyrka
4  # v1.0: 2019-03-07
5  # v1.1: 2019-03-18 (made plotting optional)
6
7  # This file is expected to run in 2 seconds for n = 100, in 6 s for n = 200, in 40 s for n = 400, and
8  #   180 s for n = 1000
9  rm(list = ls()) # Clear workspace.
10 source("smoothing-functions.R") # Load the functions.
11 start.time <- Sys.time() # Start clock.
12 set.seed(12345678) # Set seed for replication.
13 write.pdf <- TRUE # Do we want plots to be shown on the screen, or to be written in PDF files?
14
15 n <- 100 # Number of observations.
16
17 all.kernels <- c("uniform", "triangular", "epanechnikov", "quartic", "gaussian")
18 my.colours <- c("black", "blue", "red", "forestgreen", "darkorange1")
19
20 # Visualise the kernels. We want to save the plots in PDF.
21 if (write.pdf) pdf(file = "10-kernels.pdf", width = 7, height = 5) # Open PDF file for writing.
22 curve(kernelFun(x, kernel = "uniform", rescale = FALSE), -2, 2, ylim = c(0, 1.1), col = my.colours[1],
23       ↪ lwd = 2, ylab = "Kernel", main = "Various kernels used in smoothing", bty = "n")
24 for (i in 2:5) curve(kernelFun(x, kernel = all.kernels[i], rescale = FALSE), -2, 2, add = TRUE, col =
25       ↪ my.colours[i], lwd = 2)
26 legend("topright", legend = all.kernels, lwd = 1, col = my.colours, bty = "n")
27 dev.off() # Close the graphical device (if a PDF is being written, then finalises it).
28
29 # Visualise the kernels so that the integral of x^2 k(x) dx over R be 1.
30 if (write.pdf) pdf(file = "11-kernels-rescaled.pdf", width = 7, height = 5)
31 curve(kernelFun(x, kernel = "uniform", rescale = TRUE), -3, 3, ylim = c(0, 1.1), col = my.colours[1],
32       ↪ lwd = 2, ylab = "Rescaled kernel", bty = "n", main = "Rescaled kernels")
33 for (i in 2:5) curve(kernelFun(x, kernel = all.kernels[i], rescale = TRUE), -3, 3, add = TRUE, col =
34       ↪ my.colours[i], lwd = 2)
35 legend("topright", legend = all.kernels, lwd = 1, col = my.colours, bty = "n")
36 dev.off()
37
38 for (k in all.kernels) print(integrate(function(x) kernelFun(x, kernel = k, rescale = FALSE), -Inf,
39       ↪ Inf)) # Integrates to one.
40 for (k in all.kernels) print(integrate(function(x) kernelFun(x, kernel = k, rescale = TRUE), -Inf,
41       ↪ Inf)) # Integrates to one.
42 for (k in all.kernels) print(integrate(function(x) x^2 * kernelFun(x, kernel = k, rescale = FALSE),
43       ↪ -Inf, Inf)) # Depends on the shape of the kernel.
44 for (k in all.kernels) print(integrate(function(x) x^2 * kernelFun(x, kernel = k, rescale = TRUE),
45       ↪ -Inf, Inf)) # Integrates to 1.
46
47 if (write.pdf) pdf(file = "12-convolutions.pdf", width = 7, height = 5)
48 par(oma = c(0, 0, 3, 0)) # Set margins: bottom = left = right = 0, top = 3 for the main title.
49 par(mfrow = c(2, 3)) # Plot a 2x3 array of plots.
50 par(mar = c(2, 1, 4, 1)) # Set margins: bottom = 2, left = 1, top = 4, right = 1 for the sub-plots.
51 curve(kernelFun(x, kernel = "uniform", rescale = FALSE, convolution = FALSE), -3, 3, ylim = c(0,
52       ↪ 1.01), col = my.colours[1], lwd = 1, lty = 2, xlab = "", ylab = "", main = all.kernels[1],
53       ↪ font.main = 1, bty = "n", yaxt = "n") # Individual title with normal (roman, not bold) title.
54 curve(kernelFun(x, kernel = "uniform", rescale = FALSE, convolution = TRUE), -3, 3, col =
55       ↪ my.colours[1], lwd = 2, add = TRUE)
56 for (i in 2:5) {
57   curve(kernelFun(x, kernel = all.kernels[i], rescale = FALSE, convolution = FALSE), ylim = c(0,
58         ↪ 1.01), -3, 3, add = FALSE, col = my.colours[i], lwd = 1, lty = 2, ylab = "", main =
59         ↪ all.kernels[i], font.main = 1, bty = "n", yaxt = "n")
60   curve(kernelFun(x, kernel = all.kernels[i], rescale = FALSE, convolution = TRUE), -3, 3, add = TRUE,
61         ↪ col = my.colours[i], lwd = 2)
62 }
63 mtext("Convolutions of kernels", outer = TRUE) # Main title on top.
64 dev.off()
65
66 # Generate the data.

```

```

52 X <- rnorm(n, mean = 0, sd = 1)
53 U <- rnorm(n, mean = 0, sd = 1)
54 Y <- X^2 + U
55 # Generate the values of x at which pdf_X(x) and E(Y|X=x) are to be estimated.
56 Xgrid <- seq(-5, 5, length.out = 301) # Array of length 301.
57
58 # Estimate the basic kernel density with a good value (Silverman's RoT), with one too low and one too
  ↪ high
59 bws <- c(bw.nrd(X), bw.nrd(X) / 10, bw.nrd(X) * 10)
60 myfhat <- kernelDensity(X, Xgrid, bws[1], "gaussian")
61 myfhat.under <- kernelDensity(X, Xgrid, bws[2], "gaussian")
62 myfhat.over <- kernelDensity(X, Xgrid, bws[3], "gaussian")
63
64 if (write.pdf) pdf(file = "13-bandwidth-choice-consequences.pdf", width = 7, height = 5)
65 plot(Xgrid, myfhat, type = "l", ylim = c(0, max(myfhat.under)), ylab = "Density", xlab = "x", bty =
  ↪ "n", lwd = 2, main = "Effect of bandwidth on the density estimator")
66 lines(Xgrid, dnorm(Xgrid), lty = 2, lwd = 2)
67 rug(X)
68 lines(Xgrid, myfhat.under, type = "l", col = "red", lwd = 2)
69 lines(Xgrid, myfhat.over, type = "l", col = "blue", lwd = 2)
70 legend("topright", paste0(c("True density", "Optimal smthng (b=", "Under-smthng (b=", "Over-smthng
  ↪ (b="), c("", round(bws, 3)), c("", rep(")", 3))), bty = "n", col = c("black", "black", "red",
  ↪ "blue"), lty = c(2, 1, 1, 1), lwd = 2)
71 dev.off()
72
73 # Checking equality to one via numeric integration
74 stepsize <- Xgrid[2] - Xgrid[1]
75 sum(myfhat) * stepsize
76 sum(myfhat.under) * stepsize
77
78 # Cross-validating the bandwidth for the density---using exponential spacing for bandwidths and
  ↪ constructing the grid around our initial RoT guess plus or minus four times
79 bw.grid <- exp(seq(log(bws[1] / 4), log(bws[1] * 4), length.out = 101))
80 DCV.values <- lapply(all.kernels, function(k) sapply(bw.grid, function(b) DCV(X, bw = b, kernel = k)))
81 DCV.values <- matrix(unlist(DCV.values), ncol = 5)
82 min.dcv.indices <- apply(DCV.values, 2, function(x) which.min(x))
83 opt.bw.dcv <- bw.grid[min.dcv.indices]
84 names(opt.bw.dcv) <- all.kernels
85 min.cv <- apply(DCV.values, 2, function(x) min(x, na.rm = TRUE))
86 print(opt.bw.dcv)
87
88 if (write.pdf) pdf(file = "14-DCV.pdf", width = 7, height = 5)
89 par(oma = c(0, 0, 3, 0))
90 par(mfrow = c(2, 3))
91 par(mar = c(4, 4, 4, 1))
92 for (i in 1:5) {
93   plot(bw.grid, DCV.values[, i], ylim = range(DCV.values), col = my.colours[i], type = "l", lwd = 2,
    ↪ xlab = "Bandwidth b", ylab = "DCV(b)", main = all.kernels[i], bty = "n", font.main = 1, log =
    ↪ "x")
94   points(opt.bw.dcv[i], min.cv[i], cex = 2, col = "black", pch = 16)
95 }
96 mtext("Density cross-validation", outer = TRUE)
97 dev.off()
98
99 # Or can find the exact minimum using the minimiser now that we know that this function has an optimum
  ↪ in this range
100 opt.bw.dcv <- sapply(all.kernels, function(k) optimise(function(b) DCV(X, bw = b, kernel = k), c(0.1,
  ↪ 5))$minimum)
101
102 density.optimal <- lapply(all.kernels, function(k) kernelDensity(X, Xgrid, bw = opt.bw.dcv[k], kernel
  ↪ = k))
103 density.optimal <- matrix(unlist(density.optimal), ncol = 5)
104
105 if (write.pdf) pdf(file = "15-density-optimal.pdf", width = 7, height = 5)
106 par(oma = c(0, 0, 3, 0))
107 par(mfrow = c(2, 3))
108 par(mar = c(4, 5, 4, 1))
109 for (i in 1:5) {
110   plot(Xgrid, density.optimal[, i], ylim = range(density.optimal, 0.5), col = my.colours[i], type =
    ↪ "l", lwd = 2, xlab = "x", ylab = expression(hat(f)[X](x)), main = all.kernels[i], bty = "n",
    ↪ font.main = 1)
111   lines(Xgrid, dnorm(Xgrid), lty = 2)
112   rug(X)

```

```

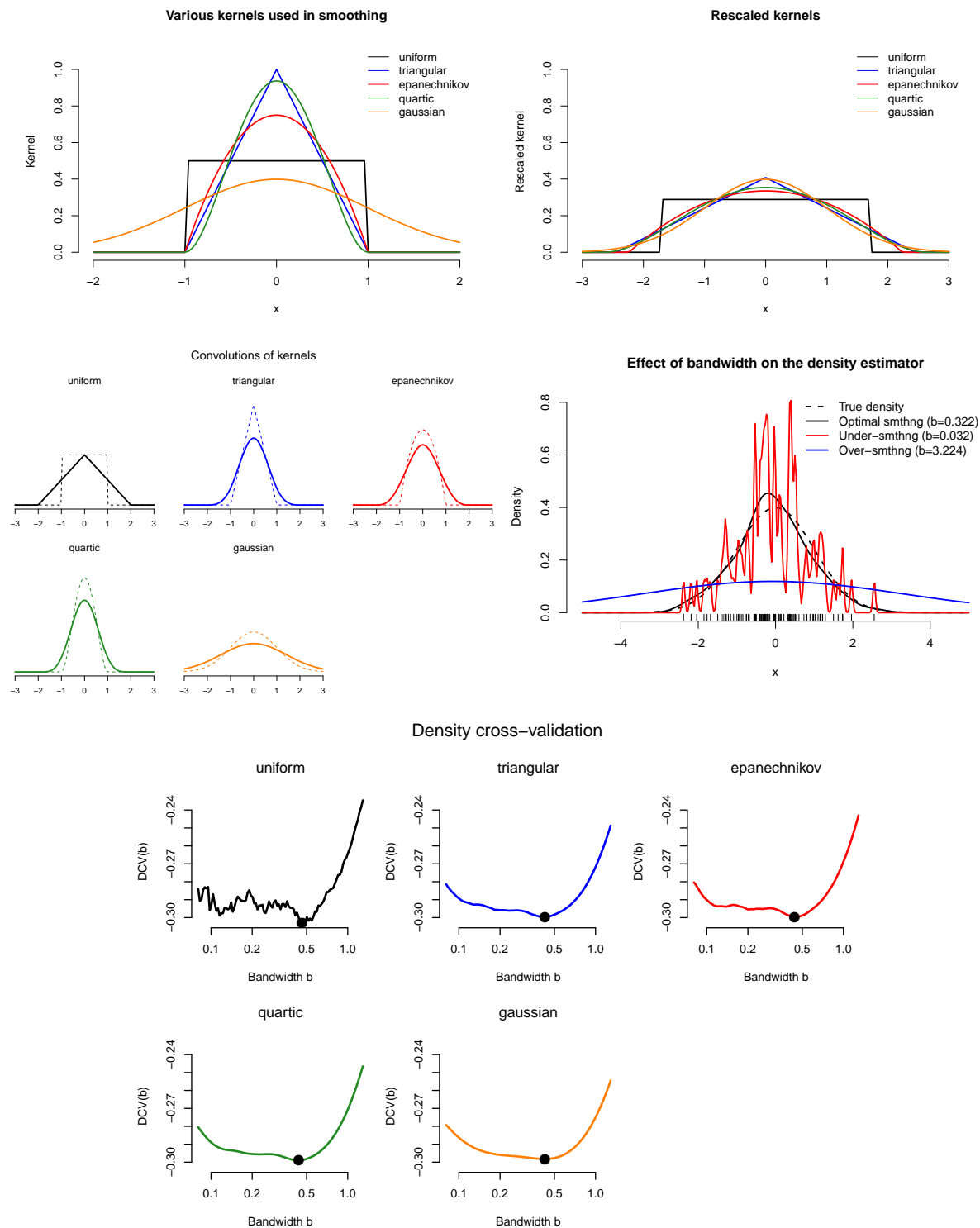
113   legend("topright", paste0("b_CV = ", round(opt.bw.dcv[i], 3)), bty = "n")
114 }
115 mtext("Density estimators with cross-validated bandwidths", outer = TRUE)
116 dev.off()
117
118 # Now, we estimate the best predictor of Y given X by Nadaraya--Watson estimator for
119 mymuhat <- kernelSmooth(X, Y, Xgrid, opt.bw.dcv["gaussian"], "gaussian")
120
121 if (write.pdf) pdf(file = "16-NW-estimator-optimal.pdf", width = 7, height = 5)
122 par(mar = c(4, 5, 4, 1))
123 plot(X, Y, xlab = "x", ylab = expression(hat(E)(Y ~ "|" ~ X == x)), bty = "n", main =
124   ↪ paste0("Nadaraya-Watson estimator (b=", round(opt.bw.dcv["gaussian"], 3), ")"))
125 lines(Xgrid, mymuhat)
126 dev.off()
127
128 # Computing least-squares cross-validation function value for bandwidth 0.42
129 LSCV(X, Y, opt.bw.dcv["gaussian"], "gaussian")
130 LSCV(X, Y, opt.bw.dcv["gaussian"] * 1.5, "gaussian") # It is smaller!
131
132 # Cross-validating the bandwidth for the Nadaraya--Watson estimator using Least Squares
133 LSCV.values <- lapply(all.kernels, function(k) sapply(bw.grid, function(b) LSCV(X, Y, bw = b, kernel =
134   ↪ k)))
135 LSCV.values <- matrix(unlist(LSCV.values), ncol = 5)
136 min.cv.indices <- apply(LSCV.values, 2, function(x) which.min(x))
137 opt.bw.lscv <- bw.grid[min.cv.indices]
138 names(opt.bw.lscv) <- all.kernels
139 min.cv <- apply(LSCV.values, 2, function(x) min(x, na.rm = TRUE))
140
141 if (write.pdf) pdf("17-LSCV.pdf", width = 7, height = 5)
142 plot(bw.grid, LSCV.values[, 1], type = "l", ylim = range(LSCV.values, na.rm = TRUE), ylab = "CV", xlab =
143   ↪ "Bandwidth", lwd = 2, main = "LS cross-validation function for the Nadaraya-Watson estimator",
144   ↪ bty = "n", log = "x")
145 for (i in 2:5) lines(bw.grid, LSCV.values[, i], col = my.colours[i], lwd = 2)
146 points(opt.bw.lscv, min.cv, cex = 2, col = my.colours, pch = 16)
147 legend("topleft", legend = all.kernels, lwd = 2, col = my.colours, bty = "n")
148 dev.off()
149
150 # Getting the exact minimum is troublesome in this case since the solution is on the boundary, but we
151 ↪ can find one for the normal kernel
152 opt.bw.lscv["gaussian"] <- optimise(function(b) LSCV(X, Y, bw = b, kernel = "gaussian"), c(0.1,
153   ↪ 0.55))$minimum
154
155 fhat.opt <- sapply(all.kernels, function(k) kernelDensity(X, Xgrid, opt.bw.lscv[k], kernel = k))
156 muhat.opt <- sapply(all.kernels, function(k) kernelSmooth(X, Y, Xgrid, opt.bw.lscv[k], kernel = k))
157
158 end.time <- Sys.time() # Start clock.
159 seconds.taken <- difftime(end.time, start.time, units = "s")
160
161 if (write.pdf) pdf(file = "18-density-and-regression.pdf", width = 7, height = 6)
162 par(oma = c(0, 0, 3, 0))
163 par(mfrow = c(2, 2))
164 par(mar = c(2, 5, 4, 2))
165
166 plot(Xgrid, fhat.opt[, "gaussian"], type = "l", lwd = 2, xlab = "x", ylab = expression(hat(f)[X](x)),
167   ↪ ylim = range(fhat.opt, na.rm = TRUE), bty = "n")
168 lines(Xgrid, dnorm(Xgrid), lty = 2)
169 rug(X)
170 title(main = "Gaussian kernel", font.main = 1)
171
172 plot(Xgrid, fhat.opt[, "uniform"], type = "l", lwd = 2, xlab = "x", ylab = expression(hat(f)[X](x)),
173   ↪ bty = "n")
174 lines(Xgrid, dnorm(Xgrid), lty = 2)
175 rug(X)
176 title(main = "Uniform kernel", font.main = 1)
177
178 plot(Xgrid, muhat.opt[, "gaussian"], type = "l", xlab = "x", lwd = 2, ylab = expression(hat(mu)(x)),
179   ↪ ylim = range(muhat.opt, Y, na.rm = TRUE), bty = "n")
180 points(X, Y, cex = 0.4, pch = 16, col = "#00000088") # Semi-transparent points
181 lines(Xgrid, Xgrid^2, lty = 2)
182 title(main = paste0("bandwidth = ", round(opt.bw.lscv["gaussian"], 3)), font.main = 1)
183
184 plot(Xgrid, muhat.opt[, "uniform"], type = "l", xlab = "x", lwd = 2, ylab = expression(hat(mu)(x)),
185   ↪ ylim = range(muhat.opt, Y, na.rm = TRUE), bty = "n")

```

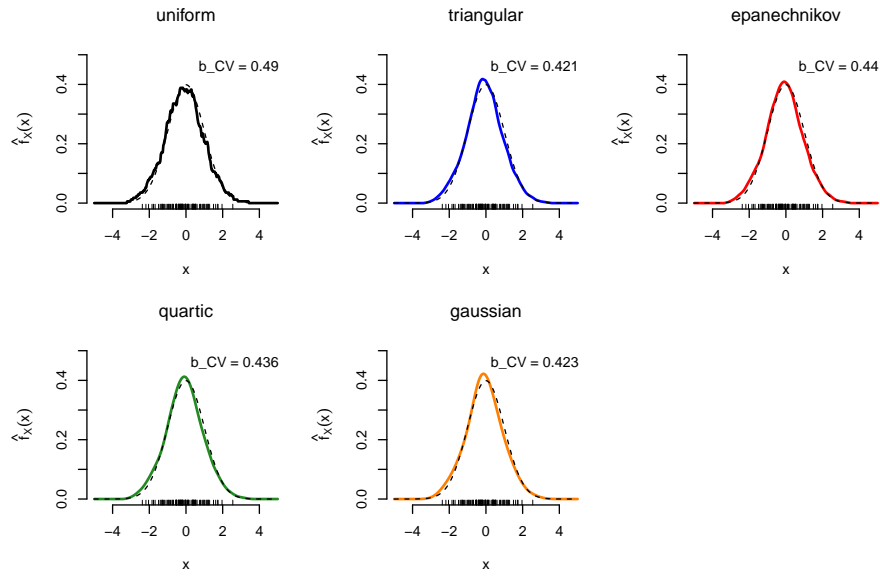
```

176 points(X, Y, cex = 0.4, pch = 16, col = "#00000088")
177 lines(Xgrid, Xgrid^2, lty = 2)
178 title(main = paste0("bandwidth = ", round(opt.bw.lscv["uniform"], 3)), font.main = 1)
179
180 top.plot.title <- c("Density and regression estimates", paste0("n = ", n), paste0("Time(sec) = ",
  ↪ round(as.numeric(seconds.taken), 3)))
181 mtext(top.plot.title, outer = TRUE, line = 1:-1)
182 dev.off()

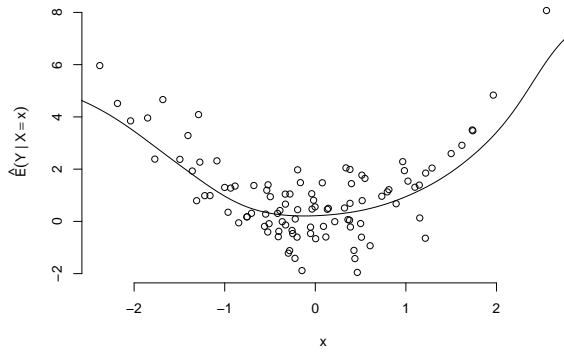
```



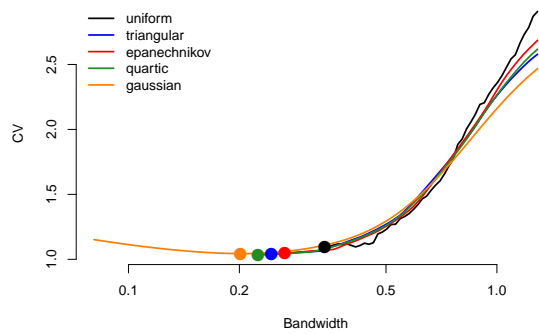
Density estimators with cross-validated bandwidths



Nadaraya-Watson estimator ($b=0.423$)



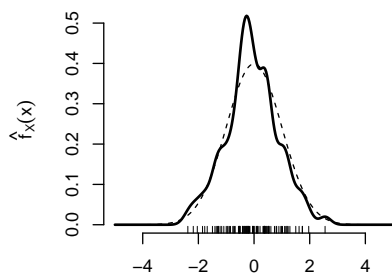
LS cross-validation function for the Nadaraya-Watson estimator



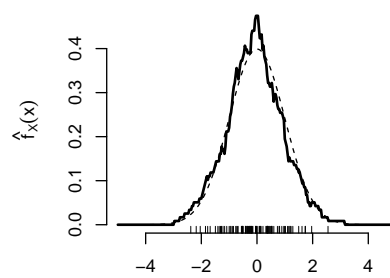
Density and regression estimates

$n = 100$
Time(sec) = 1.386

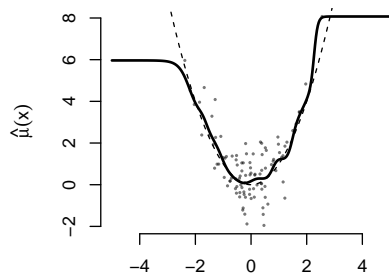
Gaussian kernel



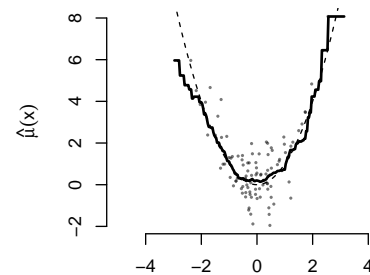
Uniform kernel



bandwidth = 0.204



bandwidth = 0.341



smoothing-simulation-02-multivariate.R

```

1  # R code for multi-dimensional density estimation and regression.
2  # Author: Andrei V. Kostyrka
3  # v1.0: 2019-03-08
4  # v1.1: 2019-03-18 (made plotting optional)
5
6  rm(list = ls()) # Clear workspace.
7  source("smoothing-functions.R") # Load the functions.
8  start.time <- Sys.time() # Start clock.
9  set.seed(20190308) # Set seed for replication.
10 write.pdf <- TRUE # Do we want plots to be shown on the screen, or to be written in PDF files?
11
12 ker <- "gaussian" # In this simulation; we shall use the Gaussian kernel only; you can change this for
13   ↪ any one you want provided that you do not run into the zero denominator problem
14
15 # Visualise the two-dimensional kernel in 3D.
16 x1 <- x2 <- seq(-4, 4, length.out = 51)
17 kernel.grid <- expand.grid(x1 = x1, x2 = x2)
18 product.kernel <- kernelFun(kernel.grid$x1, ker) * kernelFun(kernel.grid$x2, ker)
19 if (write.pdf) pdf(file = "20-kernel.pdf", width = 5, height = 5)
20 par(mar = c(1, 1, 3, 1))
21 persp(x1, x2, matrix(product.kernel, nrow = length(x1)), theta = 30, phi = 30, ticktype = "detailed",
22   ↪ zlab = "K(X)", main = "2D Gaussian product kernel")
23 dev.off()
24
25 n <- 1000
26 f.fun <- function(x1, x2) dchisq(x1, 5) * dchisq(x2, 8) # This is the true density
27 mu.fun <- function(x1, x2) 1 + x1 + x2 + 3 * sin(x1) + 3 * cos(0.5 * x2) # This is the true function
28
29 X1 <- rchisq(n, 5)
30 X2 <- rchisq(n, 8)
31 X <- cbind(X1, X2)
32 X1max <- quantile(X1, 0.995) # End of grid for visualisation
33 X2max <- quantile(X2, 0.995)
34 mu <- mu.fun(X1, X2)
35 Y <- mu + rnorm(n, sd = 2) # These are the observed values
36
37 ngrid <- 41
38 X1grid <- seq(0, X1max, length.out = ngrid)
39 X2grid <- seq(0, X2max, length.out = ngrid)
40 mygrid <- as.matrix(expand.grid(X1grid, X2grid))
41
42 f.matrix <- matrix(f.fun(mygrid[, 1], mygrid[, 2]), nrow = ngrid) # True density
43 mu.matrix <- matrix(mu.fun(mygrid[, 1], mygrid[, 2]), nrow = ngrid) # True conditional expectation
44
45 if (!("plot3D" %in% rownames(installed.packages()))) install.packages("plot3D")
46 library(plot3D) # We can make a wireframe plot using this extra library
47
48 if (write.pdf) pdf(file = "21-theoretical-values-3d.pdf", width = 9, height = 5)
49 par(mfrow = c(1, 2))
50 par(mar = c(1, 2, 3, 1))
51 p <- persp3D(X1grid, X2grid, f.matrix, xlim = c(0, X1grid[ngrid]), ylim = c(0, X2grid[ngrid]), zlim =
52   ↪ range(f.matrix), xlab = "X1", ylab = "X2", zlab = "Density", theta = 30, phi = 20, colvar = NULL,
53   ↪ alpha = 0.5, border = "black", facets = NA, ticktype = "detailed", main = "True density")
54 points3D(X1, X2, rep(0, n), pch = 16, col = "red", cex = 0.5, alpha = 0.75, add = TRUE)
55
56 p <- persp3D(X1grid, X2grid, mu.matrix, xlim = c(0, X1grid[ngrid]), ylim = c(0, X2grid[ngrid]), zlim =
57   ↪ range(Y), xlab = "X1", ylab = "X2", zlab = "mu", theta = 30, phi = 20, colvar = NULL, alpha = 0.5,
58   ↪ border = "black", facets = NA, ticktype = "detailed", main = "True regression function")
59 points3D(X1, X2, Y, pch = 16, col = "red", cex = 0.5, alpha = 0.75, add = TRUE)
60 dev.off()
61
62 # Sometimes, a 3D plot is hard to read, so people provide 2D contour lines at a higher grid
63   ↪ resolution. We make them pretty
64 nfine <- 101
65 X1grid.fine <- seq(0, X1max, length.out = nfine)
66 X2grid.fine <- seq(0, X2max, length.out = nfine)
67 mygrid.fine <- as.matrix(expand.grid(X1grid.fine, X2grid.fine))
68
69 nlev <- 25 # Using 25 contour levels
70 tcol <- rev(rainbow(nlev, start = 0, end = 0.7, v = 0.9))

```



```

65 if (write.pdf) pdf(file = "22-theoretical-values-2d.pdf", width = 9, height = 5)
66 par(mfrow = c(1, 2))
67 contour(X1grid.fine, X2grid.fine, matrix(f.fun(mygrid.fine[, 1], mygrid.fine[, 2]), nrow = nfine),
  → xlab = expression(X[1]), ylab = expression(X[2]), bty = "n", levels = round(seq(0, max(f.matrix) *
  → 0.99, length.out = nlev), 5), col = tcol, main = "True density levels")
68 contour(X1grid.fine, X2grid.fine, matrix(mu.fun(mygrid.fine[, 1], mygrid.fine[, 2]), nrow = nfine),
  → xlab = expression(X[1]), ylab = expression(X[2]), bty = "n", levels = round(seq(min(mu.matrix) *
  → 1.01, max(mu.matrix) * 0.99, length.out = nlev), 2), col = tcol, main = "True conditional
  → expectation levels")
69 dev.off()
70
71 # Now, we can plot a preliminary estimator for some reasonable bandwidth
72 bnaive <- sd(c(X1, X2)) * n-1/6 # Getting the bandwidth for visualisation with Silverman's rule
73 fhat <- kernelDensity(x = cbind(X1, X2), xgrid = mygrid, bw = bnaive)
74 muhat <- kernelSmooth(x = cbind(X1, X2), y = Y, xgrid = mygrid, bw = bnaive)
75 if (write.pdf) pdf(file = "23-estimator.pdf", width = 9, height = 5)
76 par(mfrow = c(1, 2))
77 par(mar = c(1, 2, 3, 1))
78 persp3D(X1grid, X2grid, matrix(fhat, nrow = ngrid), xlab = "X1", ylab = "X2", zlab = "Density", colvar
  → = NULL, alpha = 0.5, border = "black", facets = NA, main = paste0("Density estimator (b=",
  → round(bnaive, 3), ")"))
79 persp3D(X1grid, X2grid, matrix(muhat, nrow = ngrid), xlab = "X1", ylab = "X2", zlab = "mu", colvar =
  → NULL, alpha = 0.5, border = "black", facets = NA, main = paste0("Regression estimator (b=",
  → round(bnaive, 3), ")"))
80 dev.off()
81
82
83 # The tricky part: density=cross-validating two bandwidths or one bandwidth
84 # Method 1, more computationally difficult: each bandwidth separately
85 # This section takes 60 seconds on 12 cores under Mac/Linux; on Windows, 60*12 seconds
86 nbgrid <- 25
87 b1grid <- b2grid <- exp(seq(log(bnaive / 4), log(bnaive * 4), length.out = nbgrid))
88 step.size <- log(b1grid[2]) / b1grid[1]
89 bw.grid <- expand.grid(b1 = b1grid, b2 = b2grid)
90 system.time({ # Takes around 60 seconds on 12 cores
91   if (.Platform$OS.type == "windows") { # On Windows, this must be computed single-threadedly
92     DCV.values <- apply(bw.grid, 1, function(b) DCV(X, bw = b, kernel = ker))
93   } else {
94     library(parallel) # On everything else, for this plot, we evaluate DCV on a grid in parallel
95     DCV.values <- unlist(mclapply(1:nrow(bw.grid), function(i) DCV(X, bw = as.numeric(bw.grid[i, ]),
  → kernel = ker, rescale = TRUE), mc.cores = detectCores()))
96   }
97 })
98
99 DCV.values <- matrix(DCV.values, ncol = nbgrid)
100 min.dcv.index <- which.min(DCV.values)
101 opt.bw.dcv <- as.numeric(bw.grid[min.dcv.index, ])
102 cat(opt.bw.dcv, "- DCV =", min(DCV.values), "\n") # We can improve this values a bit now that we see
  → there are no multiple optima
103 opt.bw.dcv.fine <- optim(opt.bw.dcv, function(b) DCV(X, bw = b, kernel = ker), control = list(trace =
  → 2))
104 cat(opt.bw.dcv.fine$par, "- DCV =", opt.bw.dcv.fine$value, "\n") # We were really close
105 plotcomplex <- function() {
106   p <- persp(log(b1grid), log(b2grid), DCV.values, theta = 45, phi = 40, main = "DCV of two
  → bandwidths", axes = FALSE)
107   points(trans3D(log(opt.bw.dcv[1]), log(opt.bw.dcv[2]), min(DCV.values), p), pch = 16, col = "red",
  → cex = 2)
108   points(trans3D(log(opt.bw.dcv.fine$par[1]), log(opt.bw.dcv.fine$par[2]), opt.bw.dcv.fine$value, p),
  → pch = 16, col = "orange", cex = 2)
109   textpos1 <- trans3D(log(b1grid[seq(1, nbgrid, 3)]), rep(min(log(b2grid) - 2 * step.size),
  → nbgrid)[seq(1, nbgrid, 3)], rep(min(DCV.values), nbgrid)[seq(1, nbgrid, 3)], p)
110   text(textpos1$x, textpos1$y, labels = as.character(round(b1grid, 2))[seq(1, nbgrid, 3)], cex = 0.7)
111   textpos2 <- trans3D(rep(max(log(b2grid)) + 2 * step.size, nbgrid)[seq(1, nbgrid, 3)],
  → log(b2grid)[seq(1, nbgrid, 3)], rep(min(DCV.values), nbgrid)[seq(1, nbgrid, 3)], p)
112   text(textpos2$x, textpos2$y, labels = as.character(round(b2grid, 2))[seq(1, nbgrid, 3)], cex = 0.7)
113   textpos3 <- trans3D(c(mean(log(b1grid)), max(log(b1grid)) + 4 * step.size), c(min(log(b2grid)) - 4 *
  → step.size, mean(log(b2grid))), rep(min(DCV.values), 2), p)
114   text(textpos3$x, textpos3$y, labels = c("b1", "b2"))
115   return(p)
116 }
117
118 # Method 2, assuming the bandwidth is the same---searching on the diagonal of the plot
119 DCV.values.same <- sapply(b1grid, function(b) DCV(X, bw = b, kernel = ker))

```

```

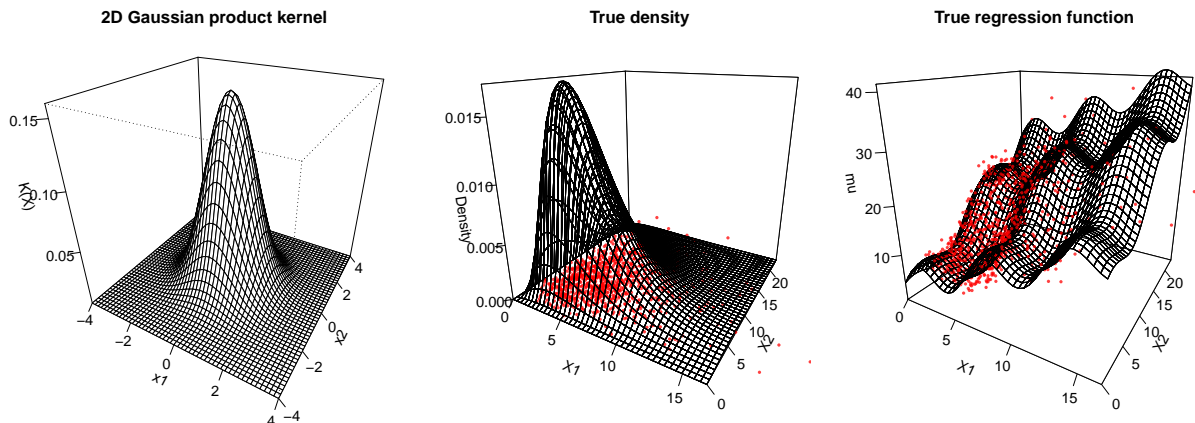
120 all(diag(DCV.values) == DCV.values.same)
121 opt.bw.dcv.same.fine <- optimise(function(b) DCV(X, bw = b, kernel = ker), interval = range(b1grid))
122 if (write.pdf) pdf("24-DCV-3D.pdf", 9, 5)
123 par(mfrow = c(1, 2))
124 plot(b1grid, DCV.values.same, type = "l", ylab = "DCV", xlab = "Unique bandwidth", bty = "n", log =
  ↪ "x", col = "blue", main = "DCV of one bandwidth")
125 points(b1grid[which.min(DCV.values.same)], min(DCV.values.same), col = "forestgreen", pch = 16, cex =
  ↪ 2)
126 points(opt.bw.dcv.same.fine$minimum, opt.bw.dcv.same.fine$objective, col = "green", pch = 16, cex = 2)
127 par(mar = c(1, 1, 3, 1))
128 p <- plotcomplex()
129 lines(trans3D(log(b1grid), log(b1grid), DCV.values.same, p), lwd = 2, col = "blue")
130 points(trans3D(log(opt.bw.dcv.same.fine$minimum), log(opt.bw.dcv.same.fine$minimum),
  ↪ opt.bw.dcv.same.fine$objective, p), pch = 16, col = "forestgreen", cex = 1.5)
131 dev.off()
132
133 # Cross-validating the bandwidths for Nadaraya---Watson estimator via Least Squares
134 # Method 1, more computationally difficult: each bandwidth separately
135 # This section takes 80 seconds on 12 cores under Mac/Linux; on Windows, 80*12 seconds
136 # Here, we are using the density-cross-validated bandwidth as the initial values
137 nbgrid <- 31
138 b1grid <- opt.bw.dcv.fine$par[1] * exp(seq(-2, 0.3, length.out = nbgrid))
139 b2grid <- opt.bw.dcv.fine$par[2] * exp(seq(-2, 0.3, length.out = nbgrid))
140 step.size <- log(b1grid[2] / b1grid[1])
141 bw.grid <- expand.grid(b1 = b1grid, b2 = b2grid)
142
143 system.time({
144   if (.Platform$OS.type == "windows") {
145     LSCV.values <- apply(bw.grid, 1, function(b) LSCV(X, Y, bw = b, kernel = ker)) # Might take two
146     ↪ minutes
147   } else {
148     LSCV.values <- unlist(mclapply(1:nrow(bw.grid), function(i) LSCV(X, Y, bw = as.numeric(bw.grid[i,
149     ↪ ]), kernel = ker, rescale = TRUE), mc.cores = detectCores()))
150   })
151 LSCV.values <- matrix(LSCV.values, ncol = nbgrid)
152 min.lscv.index <- which.min(LSCV.values)
153 opt.bw.lscv <- as.numeric(bw.grid[min.lscv.index, ])
154 cat(opt.bw.lscv, "- LSCV =", min(LSCV.values), "\n")
155 opt.bw.lscv.fine <- optim(opt.bw.lscv, function(b) LSCV(X, Y, bw = b, kernel = ker), control =
  ↪ list(trace = 2))
156 cat(opt.bw.lscv.fine$par, "- LSCV =", opt.bw.lscv.fine$value, "\n") # We were really close
157
158 plotcomplex <- function() {
159   p <- persp(log(b1grid), log(b2grid), LSCV.values, theta = 45, phi = 40, main = "LSCV of two
  ↪ bandwidths", axes = FALSE)
160   points(trans3D(log(opt.bw.lscv[1]), log(opt.bw.lscv[2]), min(LSCV.values), p), pch = 16, col =
  ↪ "red", cex = 2)
161   points(trans3D(log(opt.bw.lscv.fine$par[1]), log(opt.bw.lscv.fine$par[2]), opt.bw.lscv.fine$value,
  ↪ p), pch = 16, col = "orange", cex = 2)
162   textpos1 <- trans3D(log(b1grid[seq(1, nbgrid, 3)]), rep(min(log(b2grid) - 2 * step.size),
  ↪ nbgrid)[seq(1, nbgrid, 3)], rep(min(LSCV.values), nbgrid)[seq(1, nbgrid, 3)], p)
163   text(textpos1$x, textpos1$y, labels = as.character(round(b1grid, 2))[seq(1, nbgrid, 3)], cex = 0.7)
164   textpos2 <- trans3D(rep(max(log(b1grid)) + 2 * step.size, nbgrid)[seq(1, nbgrid, 3)],
  ↪ log(b2grid)[seq(1, nbgrid, 3)], rep(min(LSCV.values), nbgrid)[seq(1, nbgrid, 3)], p)
165   text(textpos2$x, textpos2$y, labels = as.character(round(b2grid, 2))[seq(1, nbgrid, 3)], cex = 0.7)
166   textpos3 <- trans3D(c(mean(log(b1grid)), max(log(b1grid)) + 5 * step.size), c(min(log(b2grid)) - 5 *
  ↪ step.size, mean(log(b2grid))), rep(min(LSCV.values), 2), p)
167   text(textpos3$x, textpos3$y, labels = c("b1", "b2"))
168   return(p)
169 }
170
171 # Method 2, assuming the bandwidth is the same---searching on the diagonal of the plot
172 LSCV.values.same <- sapply(b2grid, function(b) LSCV(X, Y, bw = b, kernel = ker))
173 opt.bw.lscv.same.fine <- optimise(function(b) LSCV(X, Y, bw = b, kernel = ker), interval =
  ↪ range(b1grid))
174 if (write.pdf) pdf("25-LSCV-3D.pdf", 9, 5)
175 par(mfrow = c(1, 2))
176 plot(b2grid, LSCV.values.same, type = "l", ylab = "LSCV", bty = "n", xlab = "Unique bandwidth", log =
  ↪ "x", col = "blue", main = "LSCV of one bandwidth")
177 points(b2grid[which.min(LSCV.values.same)], min(LSCV.values.same), col = "forestgreen", pch = 16, cex
  ↪ = 2)

```

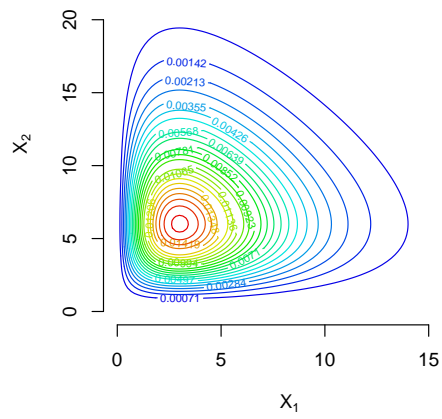
```

178 points(opt.bw.lscv.same.fine$minimum, opt.bw.lscv.same.fine$objective, col = "green", pch = 16, cex =
    ↪ 2)
179 par(mar = c(1, 1, 3, 1))
180 p <- plotcomplex()
181 lines(trans3D(log(b2grid), log(b2grid), LSCV.values.same, p), lwd = 2, col = "blue")
182 points(trans3D(log(opt.bw.lscv.same.fine$minimum), log(opt.bw.lscv.same.fine$minimum),
    ↪ opt.bw.lscv.same.fine$objective, p), pch = 16, col = "forestgreen", cex = 1.5)
183 dev.off()
184
185 # We take the LSCV bandwidth
186 fhat.opt <- kernelDensity(X, mygrid, opt.bw.lscv.fine$par, kernel = ker)
187 muhat.opt <- kernelSmooth(X, Y, mygrid, opt.bw.lscv.fine$par, kernel = ker)
188
189 end.time <- Sys.time() # Start clock.
190 seconds.taken <- difftime(end.time, start.time, units = "s")
191
192 if (write.pdf) pdf(file = "26-density-and-regression-multi.pdf", width = 9, height = 5)
193 par(oma = c(0, 0, 3, 0))
194 par(mfrow = c(1, 2))
195 par(mar = c(1, 1, 3, 1))
196
197 p <- persp3D(X1grid, X2grid, matrix(fhat.opt, nrow = ngrid), xlim = c(0, X1grid[ngrid]), ylim = c(0,
    ↪ X2grid[ngrid]), zlim = range(f.matrix), xlab = "X1", ylab = "X2", zlab = "Density", theta = 30,
    ↪ phi = 20, colvar = NULL, alpha = 0.5, border = "black", facets = NA, ticktype = "detailed", main =
    ↪ "")
198 points3D(X1, X2, rep(0, n), pch = 16, col = "red", cex = 0.5, alpha = 0.75, add = TRUE)
199
200 p <- persp3D(X1grid, X2grid, matrix(muhat, nrow = ngrid), xlim = c(0, X1grid[ngrid]), ylim = c(0,
    ↪ X2grid[ngrid]), zlim = range(Y), xlab = "X1", ylab = "X2", zlab = expression(mu), theta = 30, phi
    ↪ = 20, colvar = NULL, alpha = 0.5, border = "black", facets = NA, ticktype = "detailed", main = "")
201 points3D(X1, X2, Y, pch = 16, col = "red", cex = 0.5, alpha = 0.75, add = TRUE)
202
203 top.plot.title <- c("Density and regression estimates", paste0("n = ", n, ", time(sec) = ",
    ↪ round(as.numeric(seconds.taken), 1)), paste0("LSCV bandwidth = (",
    ↪ paste(round(opt.bw.lscv.fine$par, 3), collapse = ", "), ")"))
204 mtext(top.plot.title, outer = TRUE, line = 1:-1)
205 dev.off()
206
207 if (write.pdf) pdf(file = "27-density-and-regression-multi-levels.pdf", width = 9, height = 5)
208 par(oma = c(0, 0, 3, 0))
209 par(mfrow = c(1, 2))
210 par(mar = c(4, 4, 2, 1))
211 contour(X1grid.fine, X2grid.fine, matrix(kernelDensity(X, mygrid.fine, opt.bw.lscv.fine$par, kernel =
    ↪ ker), nrow = nfine), xlab = expression(X[1]), ylab = expression(X[2]), bty = "n", levels =
    ↪ round(seq(0, max(fhat.opt) * 0.99, length.out = nlev), 5), col = tcol)
212 contour(X1grid.fine, X2grid.fine, matrix(kernelSmooth(X, Y, mygrid.fine, opt.bw.lscv.fine$par, kernel
    ↪ = ker), nrow = nfine), xlab = expression(X[1]), ylab = expression(X[2]), bty = "n", levels =
    ↪ round(seq(min(muhat.opt) * 1.01, max(muhat.opt) * 0.99, length.out = nlev), 2), col = tcol)
213 top.plot.title <- c("Density and regression estimates", paste0("n = ", n, ", time(sec) = ",
    ↪ round(as.numeric(seconds.taken), 1)), paste0("LSCV bandwidth = (",
    ↪ paste(round(opt.bw.lscv.fine$par, 3), collapse = ", "), ")"))
214 mtext(top.plot.title, outer = TRUE, line = 1:-1)
215 dev.off()

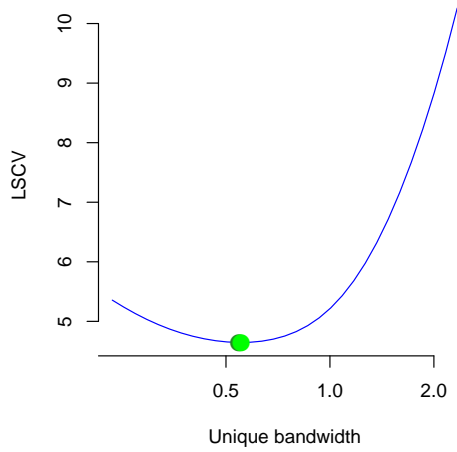
```



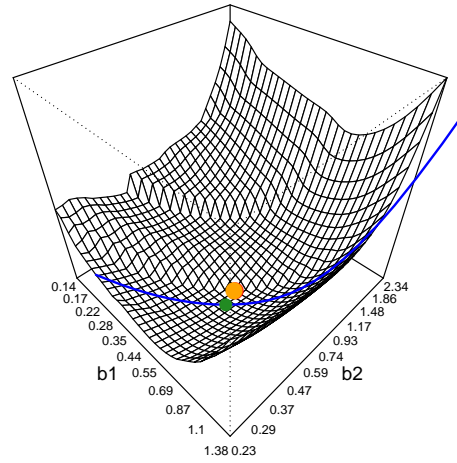
True density levels



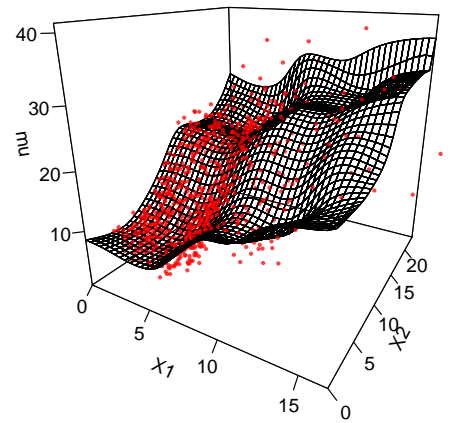
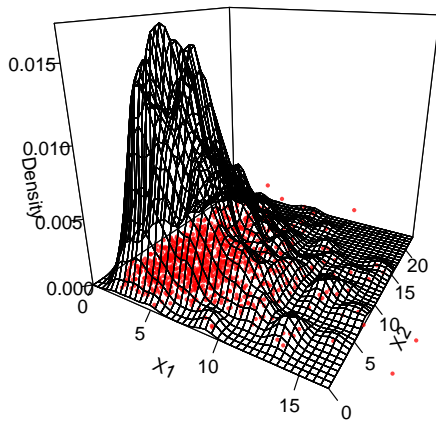
LSCV of one bandwidth



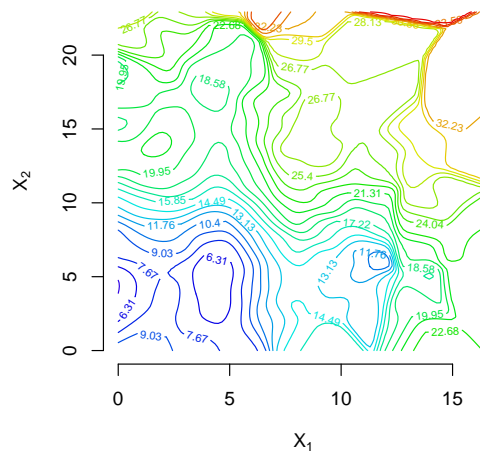
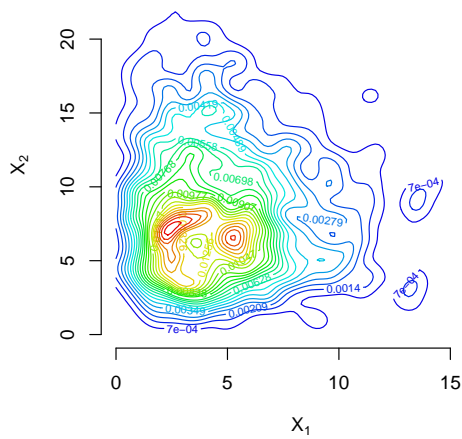
LSCV of two bandwidths



Density and regression estimates
 $n = 1000$, time(sec) = 197
 LSCV bandwidth = (0.509, 0.674)



Density and regression estimates
 $n = 1000$, time(sec) = 197
 LSCV bandwidth = (0.509, 0.674)



smoothing-simulation-03-distribution.R

```

1  # R code for simulating the conditional distribution of the kernel estimator of pdf_X and E(Y|X).
2  # Code: Andrei V. Kostyrka
3  # March 18, 2019
4  # v1.0: 2019-03-18
5
6  # Make sure you have ImageMagick installed on your computer, otherwise the animation magic will not
  ↪ happen!
7  # Go to https://www.imagemagick.org/script/download.php and install the latest version.
8
9  rm(list = ls()) # Clear workspace.
10 source("smoothing-functions.R") # Load the functions.
11 n <- 200 # Number of observations.
12 MC <- 10000 # Number of Monte-Carlo simulations
13 Xgrid <- seq(0, qchisq(0.99, 3), length.out = 101)
14 fhat <- muhat <- matrix(NA, nrow = length(Xgrid), ncol = MC)
15 mu <- function(x) 0.2 * x^2 + 2 * sin(x)
16 mybw <- 0.7
17 write.img <- TRUE # We shall be using PNG and PDF image formats because PDF cannot handle 10000 lines
  ↪ well
18
19 # We are using the following DGP:
20 # X is chi-squared with 3 df
21 # U is centred Poisson with lambda 4 (discrete!)
22 # Y = 0.2*X^2 + 2*sin(X)
23
24 # This simulation takes about 20 seconds
25 system.time({
26   for (i in 1:MC) {
27     set.seed(i)
28     X <- rchisq(n, 3)
29     U <- rpois(n, lambda = 4) - 4
30     Y <- mu(X) + U
31     # plot(X, Y)
32     # curve(mu(x), 0, 10, add=TRUE)
33     myfhat <- kernelDensity(X, Xgrid, bw = mybw)
34     mymuhat <- kernelSmooth(X, Y, Xgrid, bw = mybw)
35     fhat[, i] <- myfhat
36     muhat[, i] <- mymuhat
37     if (i %% 100 == 0) cat(i, "\n")
38   }
39 })
40
41 # Substantial productivity gains can be achieved if parallelisation (unavailable on Windows) is used!
42 library(parallel)
43 ncores <- if (.Platform$OS.type == "windows") 1 else detectCores()
44 if (.Platform$OS.type != "windows") {
45   system.time({
46     a <- mclapply(1:MC, function(i) {
47       set.seed(i)
48       X <- rchisq(n, 3)
49       U <- rpois(n, lambda = 4) - 4
50       Y <- mu(X) + U
51       myfhat <- kernelDensity(X, Xgrid, bw = mybw)
52       mymuhat <- kernelSmooth(X, Y, Xgrid, bw = mybw)
53       if (i %% 100 == 0) cat(i, "\n")
54       return(list(myfhat, mymuhat))
55     }, mc.cores = ncores)
56     fhat.parallel <- matrix(unlist(lapply(a, "[", 1)), ncol = MC)
57     muhat.parallel <- matrix(unlist(lapply(a, "[", 2)), ncol = MC)
58   })
59   print(all.equal(fhat, fhat.parallel))
60   print(all.equal(muhat, muhat.parallel))
61 }
62
63 # Quickly look at the optimal bandwidths
64 opt.bw.dcv <- optimise(function(b) DCV(X, bw = b), c(0.1, 5))$minimum
65 opt.bw.lscv <- optimise(function(b) LSCV(X, Y, bw = b), c(0.1, 5))$minimum
66 cat("Optimal DCV BW is", round(opt.bw.dcv, 3), "and LSCV BW is", round(opt.bw.lscv, 3), ".\n")
67
68 # Quickly look at what one density and conditional expectation estimation looked like
69 if (write.img) pdf("30-estimation-example.pdf", 9, 5)

```

```

70 par(mfrow = c(1, 2))
71 par(oma = c(0, 0, 0, 0))
72 par(mar = c(4, 4, 2, 1))
73 plot(Xgrid, myfhat, ylim = c(0, 0.25), main = "Estimated density in simulation #10000", bty = "n",
  ↪ ylab = "Density", xlab = "x")
74 curve(dchisq(x, 3), 0, 11, add = TRUE, lwd = 2, lty = 2, col = "blue")
75 rug(X)
76 plot(Xgrid, mymuhat, ylim = c(-2, 27), main = "Estimated mu in simulation #10000", bty = "n", ylab =
  ↪ expression(mu(x)), xlab = "x")
77 curve(0.2 * x^2 + 2 * sin(x), 0, 11, add = TRUE, lwd = 2, lty = 2, col = "red")
78 rug(X)
79 points(X, Y, pch = 16, cex = 0.5, col = "red")
80 dev.off()
81
82 # Plot the distribution of the density and regression function estimator
83 if (write.img) pdf("31-estimator-bands.pdf", 9, 5)
84 par(mfrow = c(1, 2))
85 par(oma = c(0, 0, 0, 0))
86 par(mar = c(4, 4, 2, 1))
87 plot(Xgrid, dchisq(Xgrid, 3), ylim = c(0, 0.3), col = "blue", ylab = "Density", xlab = "x", type =
  ↪ "l", lty = 2, lwd = 2, bty = "n", main = "Distribution of the density estimator")
88 lines(Xgrid, apply(fhat, 1, mean), lwd = 2)
89 for (q in c(0.250, 0.750)) lines(Xgrid, apply(fhat, 1, function(x) quantile(x, q)), lwd = 1, lty = 2)
90 for (q in c(0.025, 0.975)) lines(Xgrid, apply(fhat, 1, function(x) quantile(x, q)), lwd = 1, lty = 3)
91 for (q in c(0.005, 0.995)) lines(Xgrid, apply(fhat, 1, function(x) quantile(x, q)), lwd = 1, lty = 4)
92 legend("topright", c("True density", "Average estimate", "50% (IQ) range", "95% range", "99% range"),
  ↪ lty = c(2, 1, 2, 3, 4), lwd = c(2, 2, 1, 1, 1), col = c("blue", "black", "black", "black",
  ↪ "black"), bty = "n")
93 plot(Xgrid, mu(Xgrid), ylim = c(-2, 27), col = "red", type = "l", lwd = 2, lty = 2, bty = "n", ylab =
  ↪ expression(mu(x)), xlab = "x", main = "Distribution of the estimator of mu")
94 lines(Xgrid, apply(muhat, 1, mean), lwd = 2)
95 for (q in c(0.250, 0.750)) lines(Xgrid, apply(muhat, 1, function(x) quantile(x, q)), lwd = 1, lty = 2)
96 for (q in c(0.025, 0.975)) lines(Xgrid, apply(muhat, 1, function(x) quantile(x, q)), lwd = 1, lty = 3)
97 for (q in c(0.005, 0.995)) lines(Xgrid, apply(muhat, 1, function(x) quantile(x, q)), lwd = 1, lty = 4)
98 legend("topleft", c("True mu(x)", "Average estimate", "50% (IQ) range", "95% range", "99% range"), lty
  ↪ = c(2, 1, 2, 3, 4), lwd = c(2, 2, 1, 1, 1), col = c("red", "black", "black", "black", "black"),
  ↪ bty = "n")
99 dev.off()
100
101 # We can also plot all lines with semi-transparency at once and look at the shades
102 if (write.img) png("32-estimation-lines.png", 1200, 600, pointsize = 14, type = "cairo")
103 par(mfrow = c(1, 2))
104 par(oma = c(0, 0, 0, 0))
105 par(mar = c(4, 4, 2, 1))
106 plot(NULL, NULL, xlim = range(Xgrid), ylim = c(0, 0.3), ylab = "Density", xlab = "x", bty = "n", main
  ↪ = "Distribution of the density estimator")
107 for (l in 1:MC) lines(Xgrid, fhat[, l], lwd = 1, col = "#00000002")
108 lines(Xgrid, dchisq(Xgrid, 3), col = "blue", lty = 2, lwd = 2)
109 lines(Xgrid, apply(fhat, 1, mean), lwd = 2, col = "black")
110 plot(NULL, NULL, xlim = range(Xgrid), ylim = c(-2, 27), bty = "n", ylab = expression(mu(x)), xlab =
  ↪ "x", main = "Distribution of the estimator of mu")
111 for (l in 1:MC) lines(Xgrid, muhat[, l], lwd = 1, col = "#00000002")
112 lines(Xgrid, mu(Xgrid), col = "red", lwd = 2, lty = 2)
113 lines(Xgrid, apply(muhat, 1, mean), lwd = 2, col = "black")
114 curve(dchisq(x, 3) * 50, 0, 11, add = TRUE, lty = 3, col = "blue")
115 dev.off()
116
117 # And now---film
118 den.grid <- seq(0, 0.3, length.out = 401)
119 if (write.img) png("33-density-estimator-distribution.png", 1200, 600, pointsize = 14, type = "cairo")
120 par(mfrow = c(1, 2))
121 par(oma = c(0, 0, 0, 0))
122 par(mar = c(4, 4, 2, 1))
123 i <- 1
124 plot(den.grid, kernelDensity(fhat[i, ], den.grid, 0.0004), type = "l", main = paste0("Distribution of
  ↪ KDE at X = ", round(Xgrid[i, 2])), ylab = "Frequency across simulations", xlab = "Density
  ↪ estimator", bty = "n", lwd = 2)
125 abline(v = dchisq(Xgrid[i, ], 3), lwd = 2, col = "blue", lty = 2)
126 rug(fhat[i, ], col = "#00000011")
127 legend("topright", "True density value", col = "blue", lwd = 2, lty = 2, bty = "n")
128 i <- 30

```



```

129 plot(den.grid, kernelDensity(fhat[i, ], den.grid, 0.0004), type = "l", main = paste0("Distribution of
  ↳ KDE at X = ", round(Xgrid[i], 2)), ylab = "Frequency across simulations", xlab = "Density
  ↳ estimator", bty = "n", lwd = 2)
130 abline(v = dchisq(Xgrid[i], 3), lwd = 2, lty = 2, col = "blue")
131 rug(fhat[i, ], col = "#00000011")
132 dev.off()
133
134 if (!("animation" %in% rownames(installed.packages()))) install.packages("animation")
135 library(animation) # We can make a wireframe plot using this extra library
136 options(bitmapType = "cairo")
137
138 # Density estimator
139 saveGIF({ # Might takes from 30 seconds to 5 minutes depenging on the performance of ImageMagick
140   for (i in 1:101) {
141     plot(den.grid, kernelDensity(fhat[i, ], den.grid, 0.0004), type = "l", main = paste0("Distribution
  ↳ of KDE at X=", round(Xgrid[i], 2)), bty = "n", lwd = 2, ylab = "Frequency across simulations",
  ↳ xlab = "Density")
142     abline(v = dchisq(Xgrid[i], 3), lwd = 2, col = "blue", lty = 2)
143     rug(fhat[i, ], col = "#00000011")
144     legend("topright", legend = c(expression(f[X](x)), expression(hat(f)[X](x))), col = c("blue",
  ↳ "black"), lty = c(2, 1), lwd = 2, bty = "n")
145     if (i %% 10 == 0) cat(i, "\n")
146   }
147 }, movie.name = "34-density-estimators-the-film.gif", interval = 0.25, ani.width = 640, ani.height =
  ↳ 480, ani.dev = if (.Platform$OS.type=="windows") function(...) png(..., type="cairo") else "png")
148
149 # Regression estimator
150 mu.grid <- seq(0, 27, length.out = 401)
151 saveGIF({ # Might takes from 50 seconds to 5 minutes depenging on the performance of ImageMagick
152   for (i in 1:101) {
153     plot(mu.grid, kernelDensity(muhat[i, ], mu.grid, bw.nrd(muhat[i, ]) / 5), type = "l", main =
  ↳ paste0("Distribution of NW estimator at X =", round(Xgrid[i], 2)), lwd = 2, ylab = "Frequency
  ↳ across simulations", xlab = expression(mu))
154     abline(v = mu(Xgrid[i]), lwd = 2, col = "red")
155     rug(muhat[i, ], col = "#00000011")
156     legend("topright", legend = expression(mu(x)), col = "red", lwd = 2, bty = "n")
157     if (i %% 10 == 0) cat(i, "\n")
158   }
159 }, movie.name = "35-regression-estimators-the-film.gif", interval = 0.25, ani.width = 640, ani.height
  ↳ = 480, ani.dev = if (.Platform$OS.type=="windows") function(...) png(..., type="cairo") else
  ↳ "png")
160
161 # In 3D, we can use a coarser grig, but generating 101 densities still takes a while, so we
  ↳ parallelise while we can
162 den.grid <- seq(0, 0.3, length.out = 201)
163 kdes <- mclapply(1:101, function(i) kernelDensity(fhat[i, ], den.grid, bw.nrd(fhat[i, ]) / 5),
  ↳ mc.cores = ncores)
164 kdes <- lapply(kdes, function(x) x / max(x)) # Normalising them to one for nice plotting
165 saveGIF({ # Might takes from 20 seconds to 5 minutes depenging on the performance of ImageMagick
166   for (a in 1:360) {
167     par(mar = c(2, 2, 2, 2))
168     p <- persp(c(0, 11), c(0, 0.3), matrix(rep(0, 4), 2), col = "white", zlim = c(0, 1.01), theta = a,
  ↳ phi = 40, xlab = "x", ylab = "Density", zlab = "KDE concentration")
169     lines(trans3d(Xgrid, dchisq(Xgrid, 3), rep(0, 101), pmat = p), lwd = 1, col = "blue")
170     for (i in 1:101) lines(trans3d(rep(Xgrid[i], length(den.grid)), den.grid, kdes[[i]], pmat = p),
  ↳ lwd = 1, col = "#00000077")
171     if (a %% 5 == 0) cat(a, "/ 360\n")
172   }
173 }, movie.name = "36-kde-3d.gif", interval = 1 / 25, ani.width = 480, ani.height = 480, ani.dev = if
  ↳ (.Platform$OS.type=="windows") function(...) png(..., type="cairo") else "png")
174
175 mu.grid <- seq(0, 27, length.out = 201)
176 mus <- mclapply(1:101, function(i) kernelDensity(muhat[i, ], mu.grid, bw.nrd(muhat[i, ]) / 5),
  ↳ mc.cores = ncores)
177 mus <- lapply(mus, function(x) x / max(x))
178 saveGIF({ # Might takes from 20 seconds to 5 minutes depenging on the performance of ImageMagick
179   for (a in 1:360) {
180     par(mar = c(2, 2, 2, 2))
181     p <- persp(c(0, 11), c(0, 25), matrix(rep(0, 4), 2), col = "white", zlim = c(0, 1.01), theta = a,
  ↳ phi = 40, xlab = "x", ylab = "mu", zlab = "Estimator density")
182     lines(trans3d(Xgrid, mu(Xgrid), rep(0, 101), pmat = p), lwd = 2, col = "red")
183     for (i in 1:101) lines(trans3d(rep(Xgrid[i], length(mu.grid)), mu.grid, mus[[i]], pmat = p), lwd =
  ↳ 1, col = "#00000077")

```

```

184     if (a %% 5 == 0) cat(a, "/ 360\n")
185   }
186 }, movie.name = "37-mu-3d.gif", interval = 1 / 25, ani.width = 480, ani.height = 480, ani.dev = if
  ↪  (.Platform$OS.type=="windows") function(...) png(..., type="cairo") else "png")

```

