R code for kernel estimators

Gautam Tripathi

19th March 2019

smoothing-functions.R

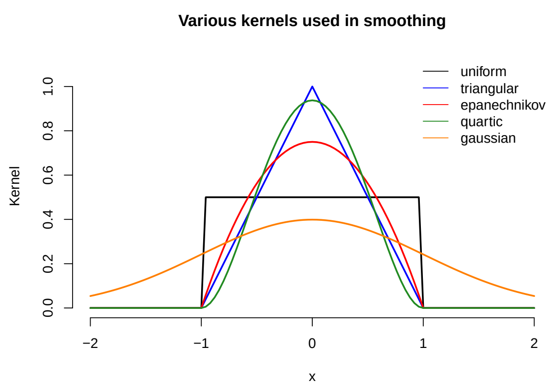
```r
# R code for kernel estimator of pdf_X(x) and E(Y|X=x).
# Original code and idea: Gautam Tripathi, 2017-03-08
# Rewrite: Andreï V. Kostyrka
# v1.0: 2019-03-07
# v1.1: 2019-03-18 (fixed an error in DCV that caused severe over-smoothing)

kernelFun <- function(x, # The values to compute the kernel
                      kernel = "gaussian", # Kernel type: uniform, epanechnikov, triangular, quartic,
                      ↪ gaussian
                      rescale = TRUE, # Rescale to unit variance: int_-Inf^+Inf x^2 k(x) = sigma^2_K =
                      ↪ 1
                      convolution = FALSE # Return the convolution kernel? Used for CV
) {
  adj.factor <- switch(kernel,
    uniform = sqrt(3),
    triangular = sqrt(6),
    epanechnikov = sqrt(5),
    quartic = sqrt(7),
    gaussian = 1
  )
  if (!rescale) adj.factor <- 1
  x <- x / adj.factor
  if (!convolution) {
    k <- switch(kernel,
      uniform = 1 / 2 * (abs(x) < 1),
      triangular = (1 - abs(x)) * (abs(x) < 1),
      epanechnikov = 3 / 4 * (1 - x^2) * (abs(x) < 1),
      quartic = 15 / 16 * (1 - x^2)^2 * (abs(x) < 1),
      gaussian = dnorm(x)
    )
  } else {
    k <- switch(kernel,
      uniform = 1 / 4 * (2 - abs(x)) * (abs(x) < 2),
      triangular = 1 / 6 * ((3 * abs(x)^3 - 6 * x^2 + 4) * (abs(x) <= 1) + (8 - 12 * abs(x) + 6 * x^2
      ↪ - abs(x)^3) * (abs(x) > 1 & abs(x) < 2)),
      epanechnikov = 3 / 160 * (32 - 40 * x^2 + 20 * abs(x)^3 - abs(x)^5) * (abs(x) < 2),
      quartic = 225 / 256 * (-128 / 105 * x^2 + 16 / 15 * x^4 + 256 / 315 + 4 / 105 * abs(x)^7 - 1 /
      ↪ 630 * abs(x)^9 - 8 / 15 * abs(x)^5) * (abs(x) < 2),
      gaussian = dnorm(x, sd = sqrt(2))
    )
  }
  k <- k / adj.factor
  return(k)
}

kernelWeights <- function(x, # A numeric vector or numeric matrix
                          xgrid = NULL, # A numeric vector or numeric matrix with ncol(xgrid) =
                          ↪ ncol(x)
                          bw, # Bandwidth: a scalar or a vector of the same length as ncol(x)
                          kernel = "gaussian", # Passed to kernelFun
                          rescale = TRUE, # Passed to kernelFun
                          convolution = FALSE # Passed to kernelFun
) {
  if (is.null(xgrid)) xgrid <- x # If no grid was passed, use existing data points as the grid
  one.dim <- is.vector(x) # Are our data one-dimensional?
  if (one.dim) {
    if (length(bw) > 1) stop("For one-dimensional kernel weights, the bandwidth must be a scalar.")
    diffs <- outer(x, xgrid, "-")
    PK <- kernelFun(diffs / bw, kernel = kernel, rescale = rescale, convolution = convolution)
  } else {
    if (!is.matrix(x)) stop("x should be either a numeric vector or a numeric matrix.")
    s <- ncol(x) # The dimension of data
    if (ncol(x) != ncol(xgrid)) stop("x and xgrid must have the same number of columns.")
    if (length(bw) == 1) bw <- rep(bw, s)
    if (length(bw) != s) stop("For multi-dimensional kernel weights, the bandwidth must have the same length as ncol(x).")
    nx <- nrow(x)
    nxgrid <- nrow(xgrid)
    PK <- matrix(1, nrow = nx, ncol = nxgrid) # Initialising the product kernel
    for (i in 1:s) {
      K <- kernelFun(outer(x[, i], xgrid[, i], "-") / bw[i], kernel = kernel, rescale = rescale, convolution = convolution)
      PK <- PK * K
    }
  }
  return(PK)
}

# Function for estimating pdf_X(x).
kernelDensity <- function(x, xgrid = NULL, bw, kernel = "gaussian", rescale = TRUE # Arguments passed to kernelWeights(...)
) {
  one.dim <- is.vector(x) # Are our data one-dimensional?
  if (!one.dim & length(bw) == 1) bw <- rep(bw, ncol(x))
  K <- kernelWeights(x = x, xgrid = xgrid, bw = bw, kernel = kernel, rescale = rescale)
  dens <- colSums(K) / (nrow(K) * prod(bw))
  return(dens)
}

# Function for estimating E(Y|X=x) (possibly leaving one observation out).
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?
) {
  if (is.numeric(xgrid)) {
    if (LOO & !isTRUE(all.equal(x, xgrid))) stop("The Leave-one-out estimator must use the same xgrid as x or NULL.")
  }
  K <- kernelWeights(x = x, xgrid = xgrid, bw = bw, kernel = kernel, rescale = rescale)
  if (LOO) diag(K) <- 0
  num <- colSums(sweep(K, 1, y, "*"))
  den <- colSums(K)
  muhat <- num / den
  return(muhat)
}

# Density cross-validation for the estimator of f_X(x).
DCV <- function(x, bw, kernel = "gaussian", rescale = TRUE) {
  one.dim <- is.vector(x) # Are our data one-dimensional?
  if (one.dim) {
    n <- length(x)
  } else {
    n <- nrow(x)
    if (length(bw) == 1) bw <- rep(bw, ncol(x))
  }
  KK <- kernelWeights(x, x, bw = bw, kernel = kernel, rescale = rescale, convolution = TRUE)
  term1 <- sum(KK) / (n^2 * prod(bw))
  # Computing the LOO estimator efficiently: fhat_i(x) = n/(n-1) * fhat(x) - 1/((n-1)*b^s) * K((X[i] - x)/b)
  fhat <- kernelDensity(x, bw = bw, kernel = kernel, rescale = rescale)
  fhat.LOO <- (n * fhat - kernelFun(0, kernel = kernel, rescale = rescale) / (prod(bw))) / (n - 1)
  term2 <- -2 * mean(fhat.LOO)
  return(term1 + term2)
}

# Least-squares cross-validation function for Nadaraya-Watson estimator of E(Y|X).
LSCV <- function(x, y, bw, kernel = "gaussian", rescale = TRUE) {
  muhat_i <- kernelSmooth(x = x, y = y, bw = bw, kernel = kernel, rescale = rescale, LOO = TRUE)
  ASE <- mean((y - muhat_i)^2)
  return(ASE)
}
```

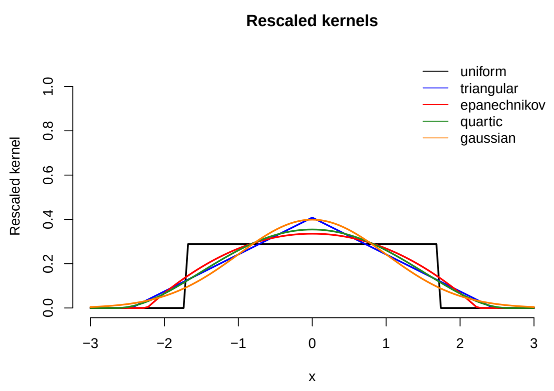
smoothing-simulation-01-univariate.R

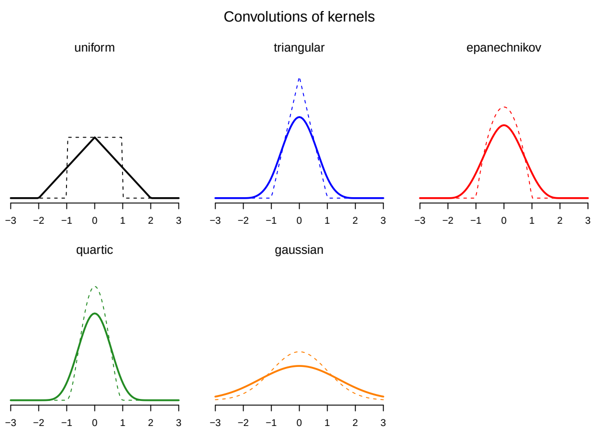
```r
# R code for kernel estimator of pdf_X and E(Y|X).
# Original code and idea: Gautam Tripathi
# Rewrite: Andreï V. Kostyrka
# v1.0: 2019-03-07
# v1.1: 2019-03-18 (made plotting optional)

# 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
↪  180 s for n = 1000
rm(list = ls()) # Clear workspace.
source("smoothing-functions.R") # Load the functions.
start.time <- Sys.time() # Start clock.
set.seed(12345678) # Set seed for replication.
write.pdf <- TRUE # Do we want plots to be shown on the screen, or to be written in PDF files?

n <- 100 # Number of observations.

all.kernels <- c("uniform", "triangular", "epanechnikov", "quartic", "gaussian")
my.colours <- c("black", "blue", "red", "forestgreen", "darkorange1")

# Visualise the kernels. We want to save the plots in PDF.
if (write.pdf) pdf(file = "10-kernels.pdf", width = 7, height = 5) # Open PDF file for writing.
curve(kernelFun(x, kernel = "uniform", rescale = FALSE), -2, 2, ylim = c(0, 1.1), col = my.colours[1],
↪  lwd = 2, ylab = "Kernel", main = "Various kernels used in smoothing", bty = "n")
for (i in 2:5) curve(kernelFun(x, kernel = all.kernels[i], rescale = FALSE), -2, 2, add = TRUE, col =
↪  my.colours[i], lwd = 2)
legend("topright", legend = all.kernels, lwd = 1, col = my.colours, bty = "n")
dev.off() # Close the graphical device (if a PDF is being written, then finalises it).

# Visualise the kernels so that the integral of x^2 k(x) dx over R be 1.
if (write.pdf) pdf(file = "11-kernels-rescaled.pdf", width = 7, height = 5)
curve(kernelFun(x, kernel = "uniform", rescale = TRUE), -3, 3, ylim = c(0, 1.1), col = my.colours[1],
↪  lwd = 2, ylab = "Rescaled kernel", bty = "n", main = "Rescaled kernels")
for (i in 2:5) curve(kernelFun(x, kernel = all.kernels[i], rescale = TRUE), -3, 3, add = TRUE, col =
↪  my.colours[i], lwd = 2)
legend("topright", legend = all.kernels, lwd = 1, col = my.colours, bty = "n")
dev.off()

for (k in all.kernels) print(integrate(function(x) kernelFun(x, kernel = k, rescale = FALSE), -Inf,
↪  Inf)) # Integrates to one.
for (k in all.kernels) print(integrate(function(x) kernelFun(x, kernel = k, rescale = TRUE), -Inf,
↪  Inf)) # Integrates to one.
for (k in all.kernels) print(integrate(function(x) x^2 * kernelFun(x, kernel = k, rescale = FALSE),
↪  -Inf, Inf)) # Depends on the shape of the kernel.
for (k in all.kernels) print(integrate(function(x) x^2 * kernelFun(x, kernel = k, rescale = TRUE),
↪  -Inf, Inf)) # Integrates to 1.

if (write.pdf) pdf(file = "12-convolutions.pdf", width = 7, height = 5)
par(oma = c(0, 0, 3, 0)) # Set margins: bottom = left = right = 0, top = 3 for the main title.
par(mfrow = c(2, 3)) # Plot a 2x3 array of plots.
par(mar = c(2, 1, 4, 1)) # Set margins: bottom = 2, left = 1, top = 4, right = 1 for the sub-plots.
curve(kernelFun(x, kernel = "uniform", rescale = FALSE, convolution = FALSE), -3, 3, ylim = c(0,
↪  1.01), col = my.colours[1], lwd = 1, lty = 2, xlab = "", ylab = "", main = all.kernels[1],
↪  font.main = 1, bty = "n", yaxt = "n") # Individual title with normal (roman, not bold) title.
curve(kernelFun(x, kernel = "uniform", rescale = FALSE, convolution = TRUE), -3, 3, col =
↪  my.colours[1], lwd = 2, add = TRUE)
for (i in 2:5) {
  curve(kernelFun(x, kernel = all.kernels[i], rescale = FALSE, convolution = FALSE), ylim = c(0,
  ↪  1.01), -3, 3, add = FALSE, col = my.colours[i], lwd = 1, lty = 2, ylab = "", main =
  ↪  all.kernels[i], font.main = 1, bty = "n", yaxt = "n")
  curve(kernelFun(x, kernel = all.kernels[i], rescale = FALSE, convolution = TRUE), -3, 3, add = TRUE,
  ↪  col = my.colours[i], lwd = 2)
}
mtext("Convolutions of kernels", outer = TRUE) # Main title on top.
dev.off()

# Generate the data.
```

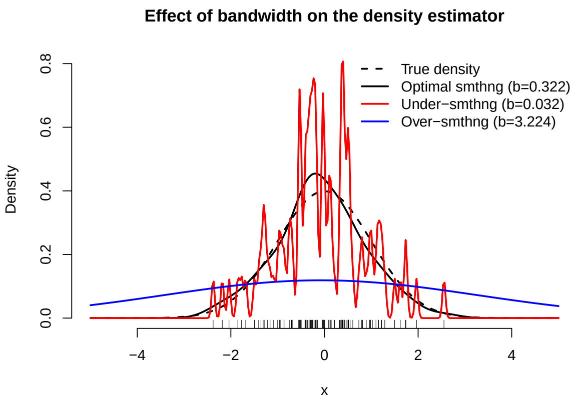
```r
X <- rnorm(n, mean = 0, sd = 1)
U <- rnorm(n, mean = 0, sd = 1)
Y <- X^2 + U
# Generate the values of x at which pdf_X(x) and E(Y|X=x) are to be estimated.
Xgrid <- seq(-5, 5, length.out = 301) # Array of length 301.

# Estimate the basic kernel density with a good value (Silverman's RoT), with one too low and one too high
bws <- c(bw.nrd(X), bw.nrd(X) / 10, bw.nrd(X) * 10)
myfhat <- kernelDensity(X, Xgrid, bws[1], "gaussian")
myfhat.under <- kernelDensity(X, Xgrid, bws[2], "gaussian")
myfhat.over <- kernelDensity(X, Xgrid, bws[3], "gaussian")

if (write.pdf) pdf(file = "13-bandwidth-choice-consequences.pdf", width = 7, height = 5)
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")
lines(Xgrid, dnorm(Xgrid), lty = 2, lwd = 2)
rug(X)
lines(Xgrid, myfhat.under, type = "l", col = "red", lwd = 2)
lines(Xgrid, myfhat.over, type = "l", col = "blue", lwd = 2)
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)
dev.off()

# Checking equality to one via numeric integration
stepsize <- Xgrid[2] - Xgrid[1]
sum(myfhat) * stepsize
sum(myfhat.under) * stepsize

# Cross-validating the bandwidth for the density---using exponential spacing for bandwidths and constructing the grid aroud our initial RoT guess plus or minus four times
bw.grid <- exp(seq(log(bws[1] / 4), log(bws[1] * 4), length.out = 101))
DCV.values <- lapply(all.kernels, function(k) sapply(bw.grid, function(b) DCV(X, bw = b, kernel = k)))
DCV.values <- matrix(unlist(DCV.values), ncol = 5)
min.dcv.indices <- apply(DCV.values, 2, function(x) which.min(x))
opt.bw.dcv <- bw.grid[min.dcv.indices]
names(opt.bw.dcv) <- all.kernels
min.cv <- apply(DCV.values, 2, function(x) min(x, na.rm = TRUE))
print(opt.bw.dcv)

if (write.pdf) pdf(file = "14-DCV.pdf", width = 7, height = 5)
par(oma = c(0, 0, 3, 0))
par(mfrow = c(2, 3))
par(mar = c(4, 4, 4, 1))
for (i in 1:5) {
  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")
  points(opt.bw.dcv[i], min.cv[i], cex = 2, col = "black", pch = 16)
}
mtext("Density cross-validation", outer = TRUE)
dev.off()

# Or can find the exact minimum using the minimiser now that we know that this function has an optimum in this range
opt.bw.dcv <- sapply(all.kernels, function(k) optimise(function(b) DCV(X, bw = b, kernel = k), c(0.1, 5))$minimum)

density.optimal <- lapply(all.kernels, function(k) kernelDensity(X, Xgrid, bw = opt.bw.dcv[k], kernel = k))
density.optimal <- matrix(unlist(density.optimal), ncol = 5)

if (write.pdf) pdf(file = "15-density-optimal.pdf", width = 7, height = 5)
par(oma = c(0, 0, 3, 0))
par(mfrow = c(2, 3))
par(mar = c(4, 5, 4, 1))
for (i in 1:5) {
  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)
  lines(Xgrid, dnorm(Xgrid), lty = 2)
  rug(X)
  legend("topright", paste0("b_CV = ", round(opt.bw.dcv[i], 3)), bty = "n")
}
mtext("Density estimators with cross-validated bandwidths", outer = TRUE)
dev.off()

# Now, we estimate the best predictor of Y given X by Nadaraya---Watson estimator for
mymuhat <- kernelSmooth(X, Y, Xgrid, opt.bw.dcv["gaussian"], "gaussian")

if (write.pdf) pdf(file = "16-NW-estimator-optimal.pdf", width = 7, height = 5)
par(mar = c(4, 5, 4, 1))
plot(X, Y, xlab = "x", ylab = expression(hat(E)(Y ~ "|" ~ X == x)), bty = "n", main = paste0("Nadaraya-Watson estimator (b=", round(opt.bw.dcv["gaussian"], 3), ")"))
lines(Xgrid, mymuhat)
dev.off()

# Computing least-squares cross-validation function value for bandwidth 0.42
LSCV(X, Y, opt.bw.dcv["gaussian"], "gaussian")
LSCV(X, Y, opt.bw.dcv["gaussian"] * 1.5, "gaussian") # It is smaller!

# Cross-validating the bandwidth for the Nadaraya---Watson estimator using Least Squares
LSCV.values <- lapply(all.kernels, function(k) sapply(bw.grid, function(b) LSCV(X, Y, bw = b, kernel = k)))
LSCV.values <- matrix(unlist(LSCV.values), ncol = 5)
min.cv.indices <- apply(LSCV.values, 2, function(x) which.min(x))
opt.bw.lscv <- bw.grid[min.cv.indices]
names(opt.bw.lscv) <- all.kernels
min.cv <- apply(LSCV.values, 2, function(x) min(x, na.rm = TRUE))

if (write.pdf) pdf("17-LSCV.pdf", width = 7, height = 5)
plot(bw.grid, LSCV.values[, 1], type = "l", ylim = range(LSCV.values, na.rm = TRUE), ylab = "CV", xlab = "Bandwidth", lwd = 2, main = "LS cross-validation function for the Nadaraya-Watson estimator", bty = "n", log = "x")
for (i in 2:5) lines(bw.grid, LSCV.values[, i], col = my.colours[i], lwd = 2)
points(opt.bw.lscv, min.cv, cex = 2, col = my.colours, pch = 16)
legend("topleft", legend = all.kernels, lwd = 2, col = my.colours, bty = "n")
dev.off()

# Getting the exact minimum is troublesome in this case since the solution is on the boundary, but we can find one for the normal kernel
opt.bw.lscv["gaussian"] <- optimise(function(b) LSCV(X, Y, bw = b, kernel = "gaussian"), c(0.1, 0.55))$minimum

fhat.opt <- sapply(all.kernels, function(k) kernelDensity(X, Xgrid, opt.bw.lscv[k], kernel = k))
muhat.opt <- sapply(all.kernels, function(k) kernelSmooth(X, Y, Xgrid, opt.bw.lscv[k], kernel = k))

end.time <- Sys.time() # Start clock.
seconds.taken <- difftime(end.time, start.time, units = "s")

if (write.pdf) pdf(file = "18-density-and-regression.pdf", width = 7, height = 6)
par(oma = c(0, 0, 3, 0))
par(mfrow = c(2, 2))
par(mar = c(2, 5, 4, 2))

plot(Xgrid, fhat.opt[, "gaussian"], type = "l", lwd = 2, xlab = "x", ylab = expression(hat(f)[X](x)), ylim = range(fhat.opt, na.rm = TRUE), bty = "n")
lines(Xgrid, dnorm(Xgrid), lty = 2)
rug(X)
title(main = "Gaussian kernel", font.main = 1)

plot(Xgrid, fhat.opt[, "uniform"], type = "l", lwd = 2, xlab = "x", ylab = expression(hat(f)[X](x)), bty = "n")
lines(Xgrid, dnorm(Xgrid), lty = 2)
rug(X)
title(main = "Uniform kernel", font.main = 1)

plot(Xgrid, muhat.opt[, "gaussian"], type = "l", xlab = "x", lwd = 2, ylab = expression(hat(mu)(x)), ylim = range(muhat.opt, Y, na.rm = TRUE), bty = "n")
points(X, Y, cex = 0.4, pch = 16, col = "#00000088") # Semi-transparent points
lines(Xgrid, Xgrid^2, lty = 2)
title(main = paste0("bandwidth = ", round(opt.bw.lscv["gaussian"], 3)), font.main = 1)

plot(Xgrid, muhat.opt[, "uniform"], type = "l", xlab = "x", lwd = 2, ylab = expression(hat(mu)(x)), ylim = range(muhat.opt, Y, na.rm = TRUE), bty = "n")
```

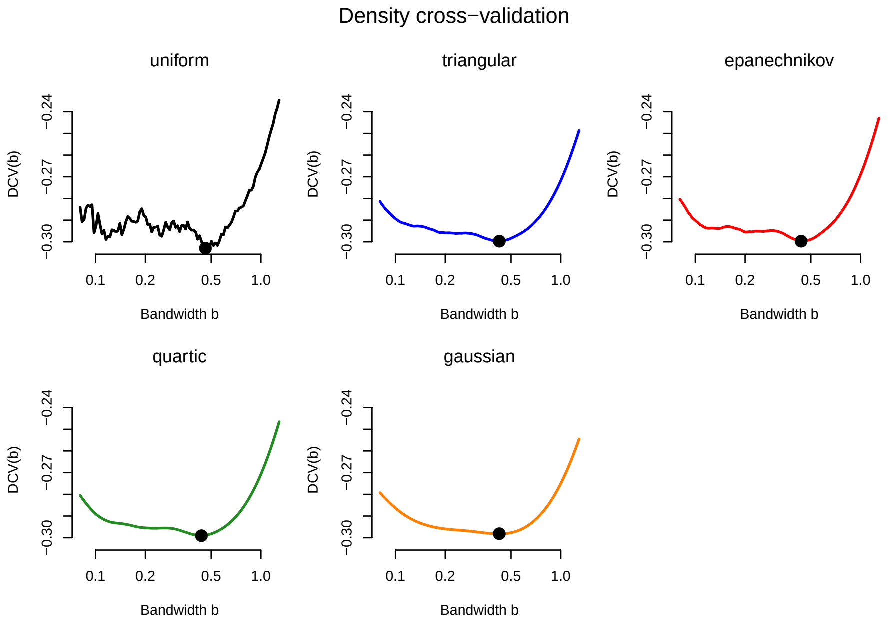
```r
points(X, Y, cex = 0.4, pch = 16, col = "#00000088")
lines(Xgrid, Xgrid^2, lty = 2)
title(main = paste0("bandwidth = ", round(opt.bw.lscv["uniform"], 3)), font.main = 1)

top.plot.title <- c("Density and regression estimates", paste0("n = ", n), paste0("Time(sec) = ",
↪  round(as.numeric(seconds.taken), 3)))
mtext(top.plot.title, outer = TRUE, line = 1:-1)
dev.off()
```

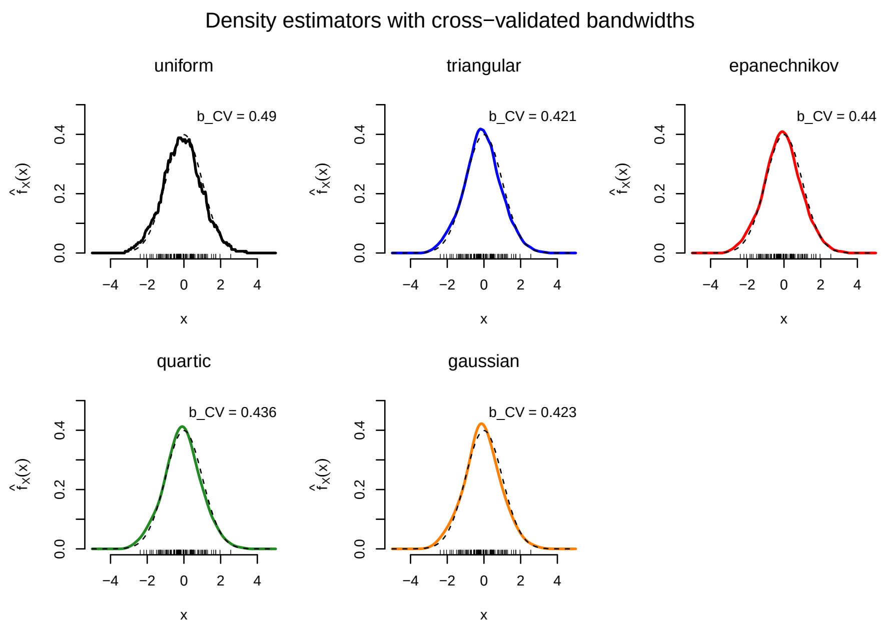


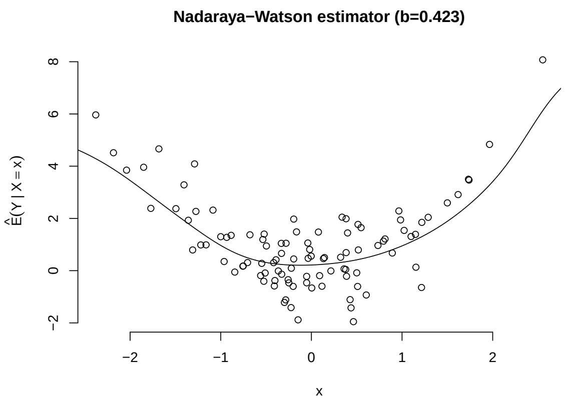


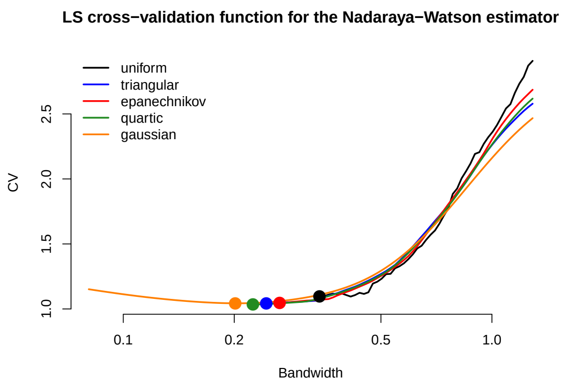


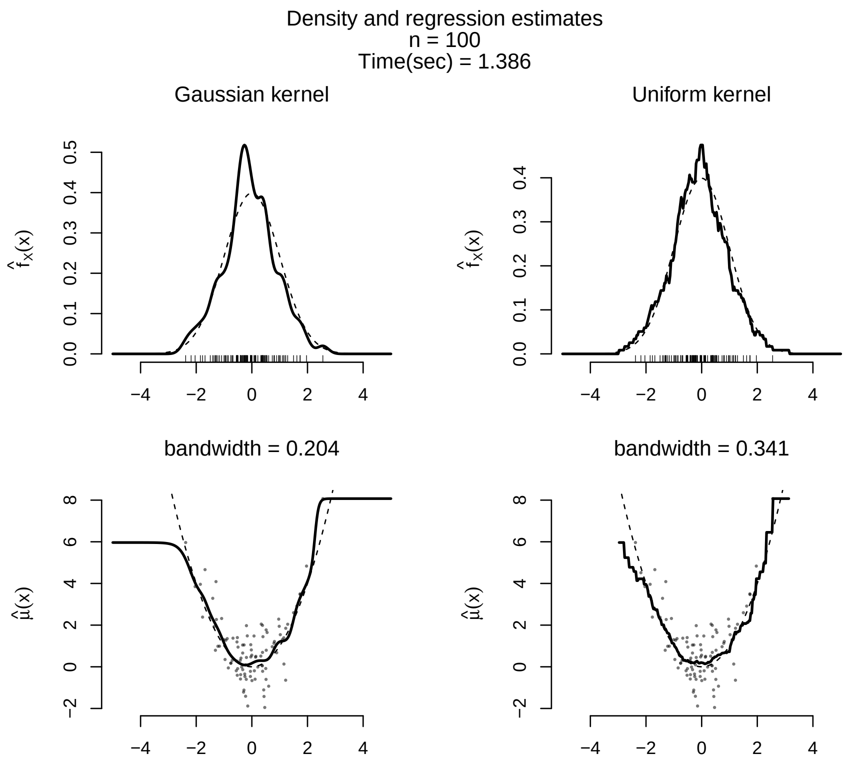





Density estimators with cross−validated bandwidths
uniform
b_CV = 0.49
triangular
b_CV = 0.421
epanechnikov
b_CV = 0.44
quartic
b_CV = 0.436
gaussian
b_CV = 0.423
$\hat{f}_X(x)$
x
0.0
0.2
0.4
−4
−2
0
2
4


Nadaraya−Watson estimator (b=0.423)
$\hat{E}(Y \mid X = x)$
x
−2
0
2
4
6
8
−2
−1
0
1
2


LS cross−validation function for the Nadaraya−Watson estimator
uniform
triangular
epanechnikov
quartic
gaussian
CV
Bandwidth
1.0
1.5
2.0
2.5
0.1
0.2
0.5
1.0


Density and regression estimates
n = 100
Time(sec) = 1.386
Gaussian kernel
Uniform kernel
$\hat{f}_X(x)$
0.0
0.1
0.2
0.3
0.4
0.5
−4
−2
0
2
4
bandwidth = 0.204
bandwidth = 0.341
$\hat{\mu}(x)$
−2
0
2
4
6
8

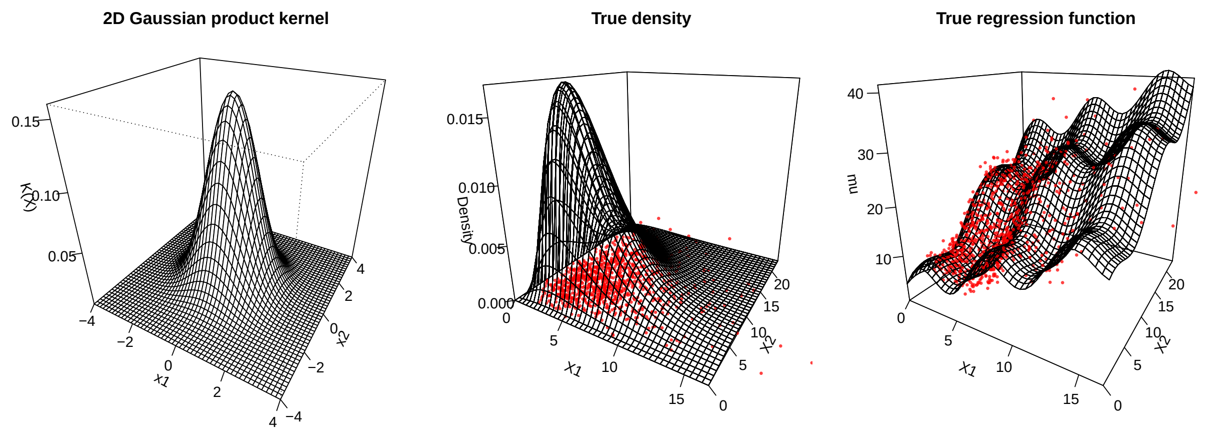
smoothing-simulation-02-multivariate.R

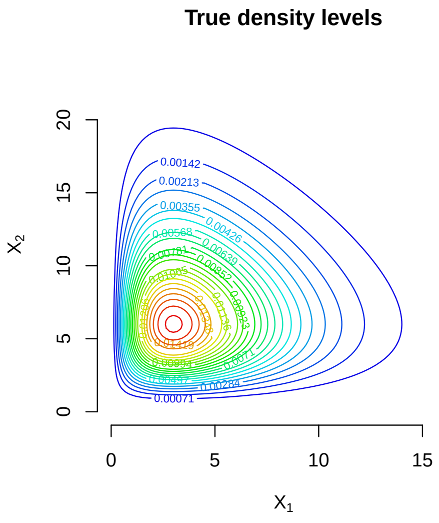
```r
# R code for multi-dimensional density estimation and regression.
# Author: Andreï V. Kostyrka
# v1.0: 2019-03-08
# v1.1: 2019-03-18 (made plotting optional)

rm(list = ls()) # Clear workspace.
source("smoothing-functions.R") # Load the functions.
start.time <- Sys.time() # Start clock.
set.seed(20190308) # Set seed for replication.
write.pdf <- TRUE # Do we want plots to be shown on the screen, or to be written in PDF files?

ker <- "gaussian" # In this simulation; we shall use the Gaussian kernel only; you can change this for
↪  any one you want provided that you do not run into the zero denominator problem

# Visualise the two-dimensional kernel in 3D.
x1 <- x2 <- seq(-4, 4, length.out = 51)
kernel.grid <- expand.grid(x1 = x1, x2 = x2)
product.kernel <- kernelFun(kernel.grid$x1, ker) * kernelFun(kernel.grid$x2, ker)
if (write.pdf) pdf(file = "20-kernel.pdf", width = 5, height = 5)
par(mar = c(1, 1, 3, 1))
persp(x1, x2, matrix(product.kernel, nrow = length(x1)), theta = 30, phi = 30, ticktype = "detailed",
↪  zlab = "K(X)", main = "2D Gaussian product kernel")
dev.off()

n <- 1000
f.fun <- function(x1, x2) dchisq(x1, 5) * dchisq(x2, 8) # This is the true density
mu.fun <- function(x1, x2) 1 + x1 + x2 + 3 * sin(x1) + 3 * cos(0.5 * x2) # This is the true function

X1 <- rchisq(n, 5)
X2 <- rchisq(n, 8)
X <- cbind(X1, X2)
X1max <- quantile(X1, 0.995) # End of grid for visualisation
X2max <- quantile(X2, 0.995)
mu <- mu.fun(X1, X2)
Y <- mu + rnorm(n, sd = 2) # These are the observed values

ngrid <- 41
X1grid <- seq(0, X1max, length.out = ngrid)
X2grid <- seq(0, X2max, length.out = ngrid)
mygrid <- as.matrix(expand.grid(X1grid, X2grid))

f.matrix <- matrix(f.fun(mygrid[, 1], mygrid[, 2]), nrow = ngrid) # True density
mu.matrix <- matrix(mu.fun(mygrid[, 1], mygrid[, 2]), nrow = ngrid) # True conditional expectation

if (!("plot3D" %in% rownames(installed.packages()))) install.packages("plot3D")
library(plot3D) # We can make a wireframe plot using this extra library

if (write.pdf) pdf(file = "21-theoretical-values-3d.pdf", width = 9, height = 5)
par(mfrow = c(1, 2))
par(mar = c(1, 2, 3, 1))
p <- persp3D(X1grid, X2grid, f.matrix, 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 = "True density")
points3D(X1, X2, rep(0, n), pch = 16, col = "red", cex = 0.5, alpha = 0.75, add = TRUE)

p <- persp3D(X1grid, X2grid, mu.matrix, xlim = c(0, X1grid[ngrid]), ylim = c(0, X2grid[ngrid]), zlim =
↪  range(Y), xlab = "X1", ylab = "X2", zlab = "mu", theta = 30, phi = 20, colvar = NULL, alpha = 0.5,
↪  border = "black", facets = NA, ticktype = "detailed", main = "True regression function")
points3D(X1, X2, Y, pch = 16, col = "red", cex = 0.5, alpha = 0.75, add = TRUE)
dev.off()

# Sometimes, a 3D plot is hard to read, so people provide 2D contour lines at a higher grid
↪  resolution. We make them pretty
nfine <- 101
X1grid.fine <- seq(0, X1max, length.out = nfine)
X2grid.fine <- seq(0, X2max, length.out = nfine)
mygrid.fine <- as.matrix(expand.grid(X1grid.fine, X2grid.fine))

nlev <- 25 # Using 25 contour levels
tcol <- rev(rainbow(nlev, start = 0, end = 0.7, v = 0.9))

```

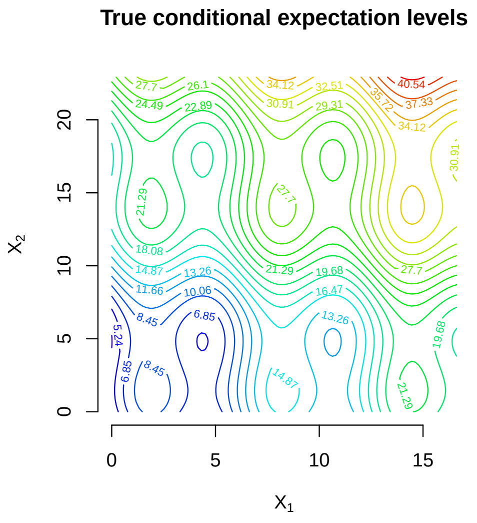
```r
if (write.pdf) pdf(file = "22-theoretical-values-2d.pdf", width = 9, height = 5)
par(mfrow = c(1, 2))
contour(X1grid.fine, X2grid.fine, matrix(f.fun(mygrid.fine[, 1], mygrid.fine[, 2]), nrow = nfine), xlab = expression(X[1]), ylab = expression(X[2]), bty = "n", levels = round(seq(0, max(f.matrix) * 0.99, length.out = nlev), 5), col = tcol, main = "True density levels")
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")
dev.off()

# Now, we can plot a preliminary estimator for some reasonable bandwidth
bnaive <- sd(c(X1, X2)) * n^(-1 / 6) # Getting the bandwidth for visualisation with Silverman's rule
fhat <- kernelDensity(x = cbind(X1, X2), xgrid = mygrid, bw = bnaive)
muhat <- kernelSmooth(x = cbind(X1, X2), y = Y, xgrid = mygrid, bw = bnaive)
if (write.pdf) pdf(file = "23-estimator.pdf", width = 9, height = 5)
par(mfrow = c(1, 2))
par(mar = c(1, 2, 3, 1))
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), ")"))
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), ")"))
dev.off()


# The tricky part: density=cross-validating two bandwidths or one bandwidth
# Method 1, more computationally difficult: each bandwidth separately
# This section takes 60 seconds on 12 cores under Mac/Linux; on Windows, 60*12 seconds
nbgrid <- 25
b1grid <- b2grid <- exp(seq(log(bnaive / 4), log(bnaive * 4), length.out = nbgrid))
step.size <- log(b1grid[2] / b1grid[1])
bw.grid <- expand.grid(b1 = b1grid, b2 = b2grid)
system.time({ # Takes around 60 seconds on 12 cores
  if (.Platform$OS.type == "windows") { # On Windows, this must be computed single-threadedly
    DCV.values <- apply(bw.grid, 1, function(b) DCV(X, bw = b, kernel = ker))
  } else {
    library(parallel) # On everything else, for this plot, we avaluate DCV on a grid in parallel
    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()))
  }
})

DCV.values <- matrix(DCV.values, ncol = nbgrid)
min.dcv.index <- which.min(DCV.values)
opt.bw.dcv <- as.numeric(bw.grid[min.dcv.index, ])
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
opt.bw.dcv.fine <- optim(opt.bw.dcv, function(b) DCV(X, bw = b, kernel = ker), control = list(trace = 2))
cat(opt.bw.dcv.fine$par, "- DCV =", opt.bw.dcv.fine$value, "\n") # We were really close
plotcomplex <- function() {
  p <- persp(log(b1grid), log(b2grid), DCV.values, theta = 45, phi = 40, main = "DCV of two bandwidths", axes = FALSE)
  points(trans3D(log(opt.bw.dcv[1]), log(opt.bw.dcv[2]), min(DCV.values), p), pch = 16, col = "red", cex = 2)
  points(trans3D(log(opt.bw.dcv.fine$par[1]), log(opt.bw.dcv.fine$par[2]), opt.bw.dcv.fine$value, p), pch = 16, col = "orange", cex = 2)
  textpos1 <- trans3D(log(b1grid[seq(1, nbgrid, 3)]), rep(min(log(b2grid) - 2 * step.size), nbgrid)[seq(1, nbgrid, 3)], rep(min(DCV.values), nbgrid)[seq(1, nbgrid, 3)], p)
  text(textpos1$x, textpos1$y, labels = as.character(round(b1grid, 2))[seq(1, nbgrid, 3)], cex = 0.7)
  textpos2 <- trans3D(rep(max(log(b2grid)) + 2 * step.size, nbgrid)[seq(1, nbgrid, 3)], log(b2grid)[seq(1, nbgrid, 3)], rep(min(DCV.values), nbgrid)[seq(1, nbgrid, 3)], p)
  text(textpos2$x, textpos2$y, labels = as.character(round(b2grid, 2))[seq(1, nbgrid, 3)], cex = 0.7)
  textpos3 <- trans3D(c(mean(log(b1grid)), max(log(b1grid)) + 4 * step.size), c(min(log(b2grid)) - 4 * step.size, mean(log(b2grid))), rep(min(DCV.values), 2), p)
  text(textpos3$x, textpos3$y, labels = c("b1", "b2"))
  return(p)
}

# Method 2, assuming the bandwidth is the same---searching on the diagonal of the plot
DCV.values.same <- sapply(b1grid, function(b) DCV(X, bw = b, kernel = ker))
```

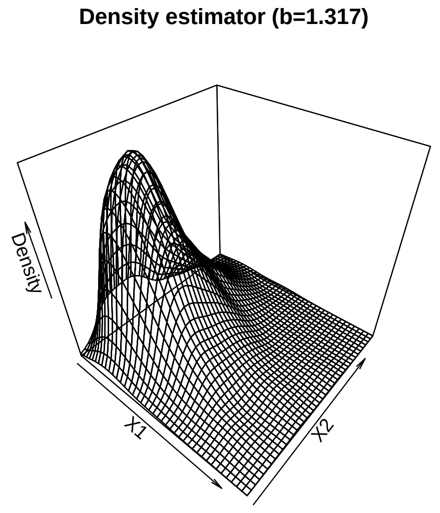
```r
all(diag(DCV.values) == DCV.values.same)
opt.bw.dcv.same.fine <- optimise(function(b) DCV(X, bw = b, kernel = ker), interval = range(b1grid))
if (write.pdf) pdf("24-DCV-3D.pdf", 9, 5)
par(mfrow = c(1, 2))
plot(b1grid, DCV.values.same, type = "l", ylab = "DCV", xlab = "Unique bandwidth", bty = "n", log =
↪  "x", col = "blue", main = "DCV of one bandwidth")
points(b1grid[which.min(DCV.values.same)], min(DCV.values.same), col = "forestgreen", pch = 16, cex =
↪  2)
points(opt.bw.dcv.same.fine$minimum, opt.bw.dcv.same.fine$objective, col = "green", pch = 16, cex = 2)
par(mar = c(1, 1, 3, 1))
p <- plotcomplex()
lines(trans3D(log(b1grid), log(b1grid), DCV.values.same, p), lwd = 2, col = "blue")
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)
dev.off()

# Cross-validating the bandwidths for Nadaraya---Watson estimator via Least Squares
# Method 1, more computationally difficult: each bandwidth separately
# This section takes 80 seconds on 12 cores under Mac/Linux; on Windows, 80*12 seconds
# Here, we are using the density-cross-validated bandwith as the initial values
nbgrid <- 31
b1grid <- opt.bw.dcv.fine$par[1] * exp(seq(-2, 0.3, length.out = nbgrid))
b2grid <- opt.bw.dcv.fine$par[2] * exp(seq(-2, 0.3, length.out = nbgrid))
step.size <- log(b1grid[2] / b1grid[1])
bw.grid <- expand.grid(b1 = b1grid, b2 = b2grid)

system.time({
  if (.Platform$OS.type == "windows") {
    LSCV.values <- apply(bw.grid, 1, function(b) LSCV(X, Y, bw = b, kernel = ker)) # Might take two
    ↪  minutes
  } else {
    LSCV.values <- unlist(mclapply(1:nrow(bw.grid), function(i) LSCV(X, Y, bw = as.numeric(bw.grid[i,
    ↪  ]), kernel = ker, rescale = TRUE), mc.cores = detectCores()))
  }
})

LSCV.values <- matrix(LSCV.values, ncol = nbgrid)
min.lscv.index <- which.min(LSCV.values)
opt.bw.lscv <- as.numeric(bw.grid[min.lscv.index, ])
cat(opt.bw.lscv, "- LSCV =", min(LSCV.values), "\n")
opt.bw.lscv.fine <- optim(opt.bw.lscv, function(b) LSCV(X, Y, bw = b, kernel = ker), control =
↪  list(trace = 2))
cat(opt.bw.lscv.fine$par, "- LSCV =", opt.bw.lscv.fine$value, "\n") # We were really close

plotcomplex <- function() {
  p <- persp(log(b1grid), log(b2grid), LSCV.values, theta = 45, phi = 40, main = "LSCV of two
  ↪  bandwidths", axes = FALSE)
  points(trans3D(log(opt.bw.lscv[1]), log(opt.bw.lscv[2]), min(LSCV.values), p), pch = 16, col =
  ↪  "red", cex = 2)
  points(trans3D(log(opt.bw.lscv.fine$par[1]), log(opt.bw.lscv.fine$par[2]), opt.bw.lscv.fine$value,
  ↪  p), pch = 16, col = "orange", cex = 2)
  textpos1 <- trans3D(log(b1grid[seq(1, nbgrid, 3)]), rep(min(log(b2grid) - 2 * step.size),
  ↪  nbgrid)[seq(1, nbgrid, 3)], rep(min(LSCV.values), nbgrid)[seq(1, nbgrid, 3)], p)
  text(textpos1$x, textpos1$y, labels = as.character(round(b1grid, 2))[seq(1, nbgrid, 3)], cex = 0.7)
  textpos2 <- trans3D(rep(max(log(b1grid)) + 2 * step.size, nbgrid)[seq(1, nbgrid, 3)],
  ↪  log(b2grid)[seq(1, nbgrid, 3)], rep(min(LSCV.values), nbgrid)[seq(1, nbgrid, 3)], p)
  text(textpos2$x, textpos2$y, labels = as.character(round(b2grid, 2))[seq(1, nbgrid, 3)], cex = 0.7)
  textpos3 <- trans3D(c(mean(log(b1grid)), max(log(b1grid)) + 5 * step.size), c(min(log(b2grid)) - 5 *
  ↪  step.size, mean(log(b2grid))), rep(min(LSCV.values), 2), p)
  text(textpos3$x, textpos3$y, labels = c("b1", "b2"))
  return(p)
}

# Method 2, assuming the bandwidth is the same---searching on the diagonal of the plot
LSCV.values.same <- sapply(b2grid, function(b) LSCV(X, Y, bw = b, kernel = ker))
opt.bw.lscv.same.fine <- optimise(function(b) LSCV(X, Y, bw = b, kernel = ker), interval =
↪  range(b1grid))
if (write.pdf) pdf("25-LSCV-3D.pdf", 9, 5)
par(mfrow = c(1, 2))
plot(b2grid, LSCV.values.same, type = "l", ylab = "LSCV", bty = "n", xlab = "Unique bandwidth", log =
↪  "x", col = "blue", main = "LSCV of one bandwidth")
points(b2grid[which.min(LSCV.values.same)], min(LSCV.values.same), col = "forestgreen", pch = 16, cex
↪  = 2)
```

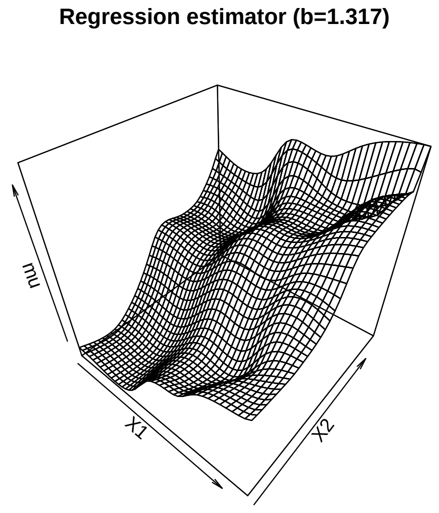
```r
points(opt.bw.lscv.same.fine$minimum, opt.bw.lscv.same.fine$objective, col = "green", pch = 16, cex = 2)
par(mar = c(1, 1, 3, 1))
p <- plotcomplex()
lines(trans3D(log(b2grid), log(b2grid), LSCV.values.same, p), lwd = 2, col = "blue")
points(trans3D(log(opt.bw.lscv.same.fine$minimum), log(opt.bw.lscv.same.fine$minimum), opt.bw.lscv.same.fine$objective, p), pch = 16, col = "forestgreen", cex = 1.5)
dev.off()

# We take the LSCV bandwidth
fhat.opt <- kernelDensity(X, mygrid, opt.bw.lscv.fine$par, kernel = ker)
muhat.opt <- kernelSmooth(X, Y, mygrid, opt.bw.lscv.fine$par, kernel = ker)

end.time <- Sys.time() # Start clock.
seconds.taken <- difftime(end.time, start.time, units = "s")

if (write.pdf) pdf(file = "26-density-and-regression-multi.pdf", width = 9, height = 5)
par(oma = c(0, 0, 3, 0))
par(mfrow = c(1, 2))
par(mar = c(1, 1, 3, 1))

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 = "")
points3D(X1, X2, rep(0, n), pch = 16, col = "red", cex = 0.5, alpha = 0.75, add = TRUE)

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 = "")
points3D(X1, X2, Y, pch = 16, col = "red", cex = 0.5, alpha = 0.75, add = TRUE)

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 = ", "), ")"))
mtext(top.plot.title, outer = TRUE, line = 1:-1)
dev.off()

if (write.pdf) pdf(file = "27-density-and-regression-multi-levels.pdf", width = 9, height = 5)
par(oma = c(0, 0, 3, 0))
par(mfrow = c(1, 2))
par(mar = c(4, 4, 2, 1))
contour(X1grid.fine, X2grid.fine, matrix(kernelDensity(X, mygrid.fine, opt.bw.lscv.fine$par, kernel = ker), nrow = nfine), xlab = expression(X[1]), ylab = expression(X[2]), bty = "n", levels = round(seq(0, max(fhat.opt) * 0.99, length.out = nlev), 5), col = tcol)
contour(X1grid.fine, X2grid.fine, matrix(kernelSmooth(X, Y, mygrid.fine, opt.bw.lscv.fine$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)
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 = ", "), ")"))
mtext(top.plot.title, outer = TRUE, line = 1:-1)
dev.off()
```

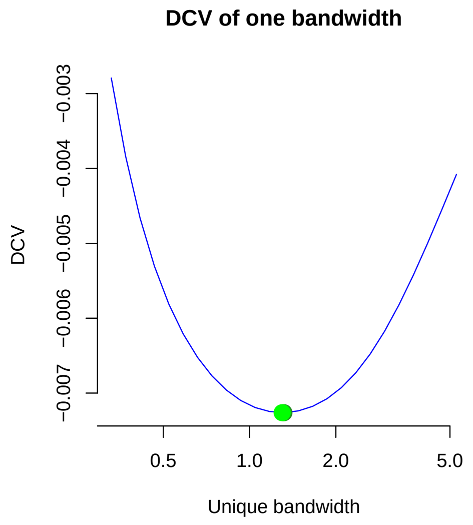


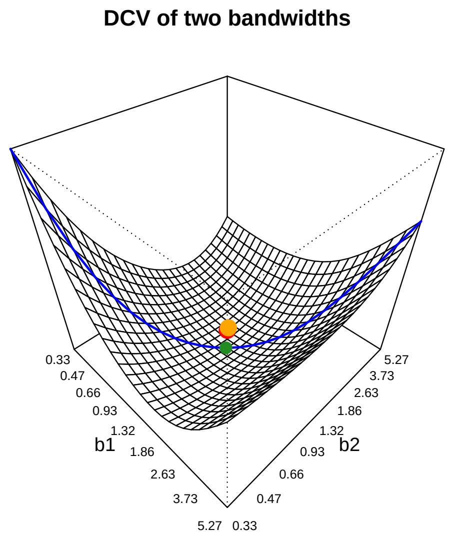

True density levels
0.00142
0.00213
0.00355
0.00426
0.00568
0.00639
0.00781
0.00852
0.00995
0.00071
0.00284
X_1
X_2
0
5
10
15
20


True conditional expectation levels
27.7
26.1
24.49
22.89
34.12
30.91
32.51
29.31
40.54
37.33
35.72
34.12
30.91
21.29
27.7
18.08
14.87
13.26
21.29
19.68
27.7
11.66
10.06
16.47
8.45
6.85
13.26
5.24
19.68
6.85
8.45
14.87
21.29
X_1
X_2
0
5
10
15
20

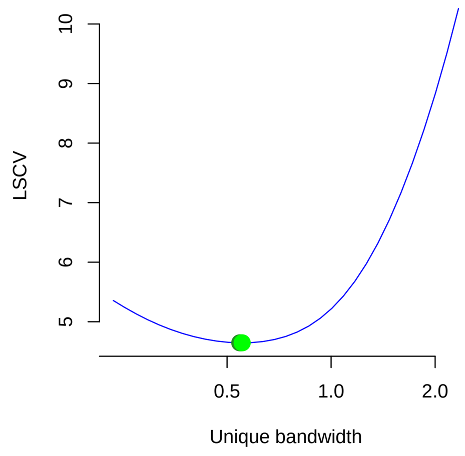

Density estimator (b=1.317)
Density
X1
X2


Regression estimator (b=1.317)
mu
X1
X2

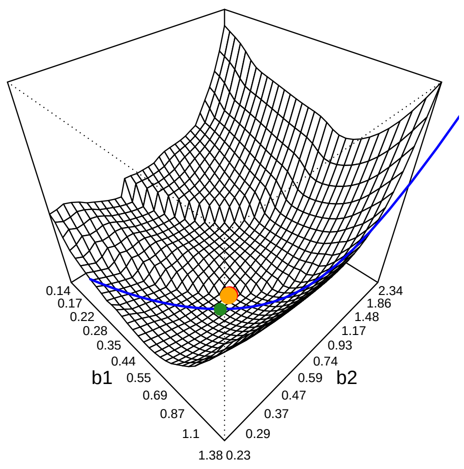

DCV of one bandwidth
DCV
−0.003
−0.004
−0.005
−0.006
−0.007
0.5
1.0
2.0
5.0
Unique bandwidth


DCV of two bandwidths
b1
0.33
0.47
0.66
0.93
1.32
1.86
2.63
3.73
5.27
b2
5.27
3.73
2.63
1.86
1.32
0.93
0.66
0.47
0.33

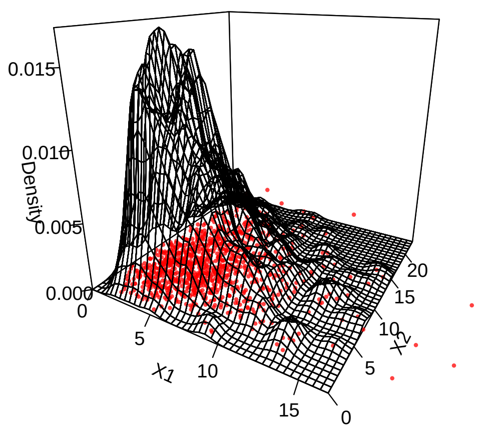
LSCV of one bandwidth

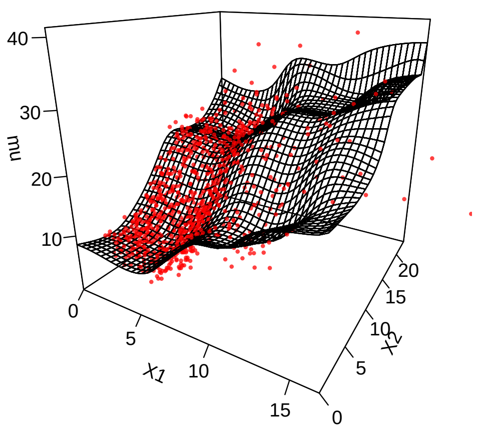


LSCV of two bandwidths

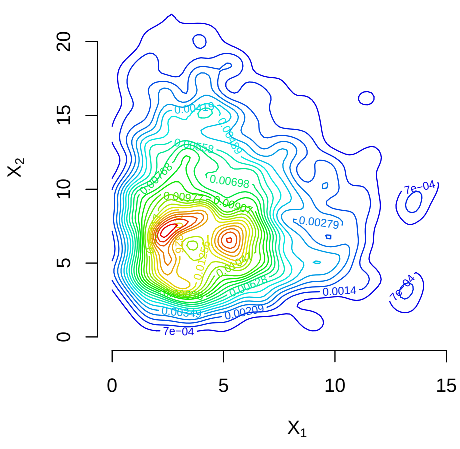


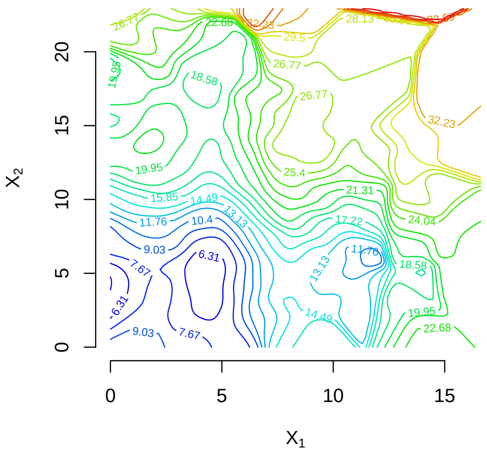
Density and regression estimates
n = 1000, time(sec) = 197
LSCV bandwidth = (0.509, 0.674)





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



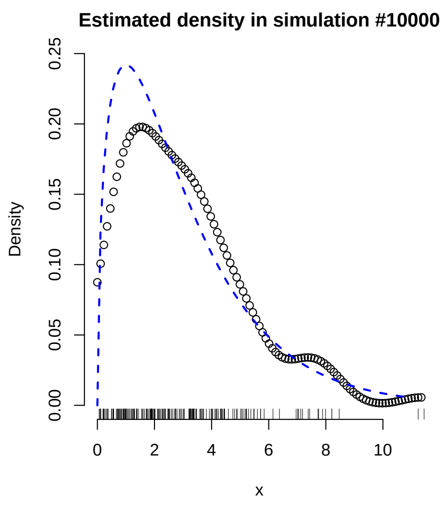


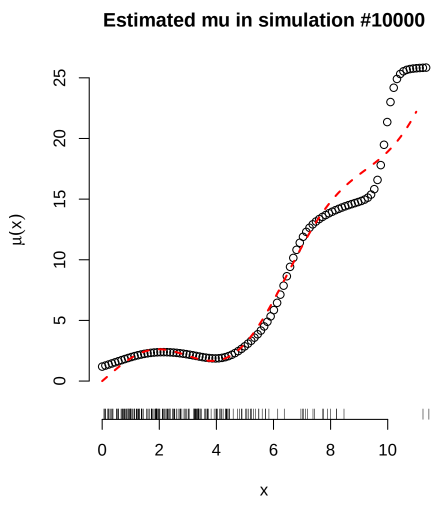
smoothing-simulation-03-distribution.R

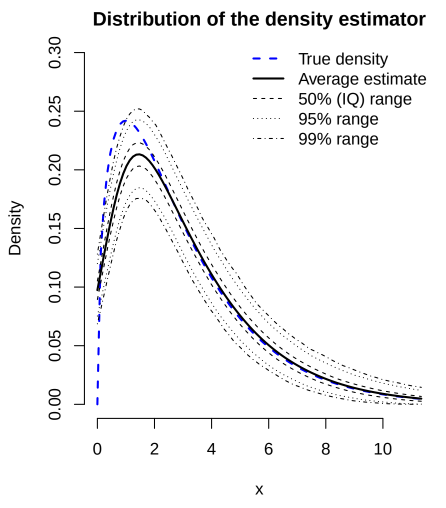
```r
# R code for simulating the conditional distribution of the kernel estimator of pdf_X and E(Y|X).
# Code: Andreï V. Kostyrka
# March 18, 2019
# v1.0: 2019-03-18

# Make sure you have ImageMagick installed on your computer, otherwise the animation magic will not
↪  happen!
# Go to https://www.imagemagick.org/script/download.php and install the latest version.

rm(list = ls()) # Clear workspace.
source("smoothing-functions.R") # Load the functions.
n <- 200 # Number of observations.
MC <- 10000 # Number of Monte-Carlo simulations
Xgrid <- seq(0, qchisq(0.99, 3), length.out = 101)
fhat <- muhat <- matrix(NA, nrow = length(Xgrid), ncol = MC)
mu <- function(x) 0.2 * x^2 + 2 * sin(x)
mybw <- 0.7
write.img <- TRUE # We shall be using PNG and PDF image formats because PDF cannot handle 10000 lines
↪  well

# We are using the following DGP:
# X is chi-squared with 3 df
# U is centred Poisson with lambda 4 (discrete!)
# Y = 0.2*X^2 + 2*sin(X)

# This simulation takes about 20 seconds
system.time({
  for (i in 1:MC) {
    set.seed(i)
    X <- rchisq(n, 3)
    U <- rpois(n, lambda = 4) - 4
    Y <- mu(X) + U
    # plot(X, Y)
    # curve(mu(x), 0, 10, add=TRUE)
    myfhat <- kernelDensity(X, Xgrid, bw = mybw)
    mymuhat <- kernelSmooth(X, Y, Xgrid, bw = mybw)
    fhat[, i] <- myfhat
    muhat[, i] <- mymuhat
    if (i %% 100 == 0) cat(i, "\n")
  }
})

# Substantial productivity gains can be achieved if parallslisation (unavailable on Windows) is used!
library(parallel)
ncores <- if (.Platform$OS.type == "windows") 1 else detectCores()
if (.Platform$OS.type != "windows") {
  system.time({
    a <- mclapply(1:MC, function(i) {
      set.seed(i)
      X <- rchisq(n, 3)
      U <- rpois(n, lambda = 4) - 4
      Y <- mu(X) + U
      myfhat <- kernelDensity(X, Xgrid, bw = mybw)
      mymuhat <- kernelSmooth(X, Y, Xgrid, bw = mybw)
      if (i %% 100 == 0) cat(i, "\n")
      return(list(myfhat, mymuhat))
    }, mc.cores = ncores)
    fhat.parallel <- matrix(unlist(lapply(a, "[[", 1)), ncol = MC)
    muhat.parallel <- matrix(unlist(lapply(a, "[[", 2)), ncol = MC)
  })
  print(all.equal(fhat, fhat.parallel))
  print(all.equal(muhat, muhat.parallel))
}

# Quickly look at the optimal bandwidths
opt.bw.dcv <- optimise(function(b) DCV(X, bw = b), c(0.1, 5))$minimum
opt.bw.lscv <- optimise(function(b) LSCV(X, Y, bw = b), c(0.1, 5))$minimum
cat("Optimal DCV BW is", round(opt.bw.dcv, 3), "and LSCV BW is", round(opt.bw.lscv, 3), ".\n")

# Quickly look at what one density and conditional expectation estimation looked like
if (write.img) pdf("30-estimation-example.pdf", 9, 5)
```

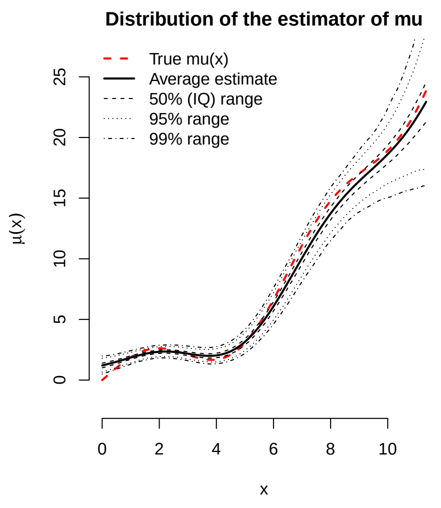
```r
par(mfrow = c(1, 2))
par(oma = c(0, 0, 0, 0))
par(mar = c(4, 4, 2, 1))
plot(Xgrid, myfhat, ylim = c(0, 0.25), main = "Estimated density in simulation #10000", bty = "n", ylab = "Density", xlab = "x")
curve(dchisq(x, 3), 0, 11, add = TRUE, lwd = 2, lty = 2, col = "blue")
rug(X)
plot(Xgrid, mymuhat, ylim = c(-2, 27), main = "Estimated mu in simulation #10000", bty = "n", ylab = expression(mu(x)), xlab = "x")
curve(0.2 * x^2 + 2 * sin(x), 0, 11, add = TRUE, lwd = 2, lty = 2, col = "red")
rug(X)
dev.off()

# Plot the distribution of the density and regression function estimator
if (write.img) pdf("31-estimator-bands.pdf", 9, 5)
par(mfrow = c(1, 2))
par(oma = c(0, 0, 0, 0))
par(mar = c(4, 4, 2, 1))
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")
lines(Xgrid, apply(fhat, 1, mean), lwd = 2)
for (q in c(0.250, 0.750)) lines(Xgrid, apply(fhat, 1, function(x) quantile(x, q)), lwd = 1, lty = 2)
for (q in c(0.025, 0.975)) lines(Xgrid, apply(fhat, 1, function(x) quantile(x, q)), lwd = 1, lty = 3)
for (q in c(0.005, 0.995)) lines(Xgrid, apply(fhat, 1, function(x) quantile(x, q)), lwd = 1, lty = 4)
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")
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")
lines(Xgrid, apply(muhat, 1, mean), lwd = 2)
for (q in c(0.250, 0.750)) lines(Xgrid, apply(muhat, 1, function(x) quantile(x, q)), lwd = 1, lty = 2)
for (q in c(0.025, 0.975)) lines(Xgrid, apply(muhat, 1, function(x) quantile(x, q)), lwd = 1, lty = 3)
for (q in c(0.005, 0.995)) lines(Xgrid, apply(muhat, 1, function(x) quantile(x, q)), lwd = 1, lty = 4)
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")
dev.off()

# We can also plot all lines with semi-transparency at once and look at the shades
if (write.img) png("32-estimation-lines.png", 1200, 600, pointsize = 14, type = "cairo")
par(mfrow = c(1, 2))
par(oma = c(0, 0, 0, 0))
par(mar = c(4, 4, 2, 1))
plot(NULL, NULL, xlim = range(Xgrid), ylim = c(0, 0.3), ylab = "Density", xlab = "x", bty = "n", main = "Distribution of the density estimator")
for (l in 1:MC) lines(Xgrid, fhat[, l], lwd = 1, col = "#00000002")
lines(Xgrid, dchisq(Xgrid, 3), col = "blue", lty = 2, lwd = 2)
lines(Xgrid, apply(fhat, 1, mean), lwd = 2, col = "black")
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")
for (l in 1:MC) lines(Xgrid, muhat[, l], lwd = 1, col = "#00000002")
lines(Xgrid, mu(Xgrid), col = "red", lwd = 2, lty = 2)
lines(Xgrid, apply(muhat, 1, mean), lwd = 2, col = "black")
curve(dchisq(x, 3) * 50, 0, 11, add = TRUE, lty = 3, col = "blue")
dev.off()

# And now---film
den.grid <- seq(0, 0.3, length.out = 401)
if (write.img) png("33-density-estimator-distribution.png", 1200, 600, pointsize = 14, type = "cairo")
par(mfrow = c(1, 2))
par(oma = c(0, 0, 0, 0))
par(mar = c(4, 4, 2, 1))
i <- 1
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)
abline(v = dchisq(Xgrid[i], 3), lwd = 2, col = "blue", lty = 2)
rug(fhat[i, ], col = "#00000011")
legend("topright", "True density value", col = "blue", lwd = 2, lty = 2, bty = "n")
i <- 30
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)
```

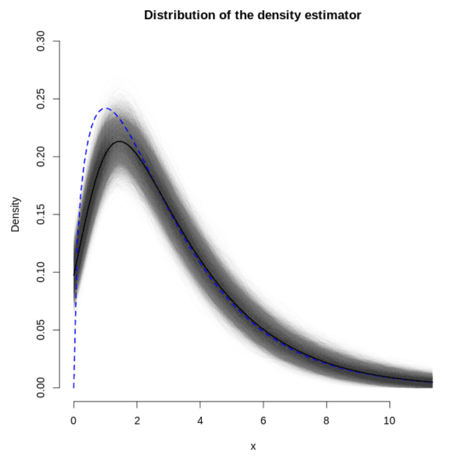
```r
abline(v = dchisq(Xgrid[i], 3), lwd = 2, lty = 2, col = "blue")
rug(fhat[i, ], col = "#00000011")
dev.off()

# Density estimator
if (!("animation" %in% rownames(installed.packages()))) install.packages("animation")
library(animation) # We can make a wireframe plot using this extra library
options(bitmapType = "cairo")

saveGIF({ # Takes roughly 30 seconds on Linux and a whopping 20 minutes on Windows because of the rug
  for (i in 1:101) {
    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")
    abline(v = dchisq(Xgrid[i], 3), lwd = 2, col = "blue", lty = 2)
    rug(fhat[i, ], col = "#00000011")
    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")
    if (i %% 10 == 0) cat(i, "\n")
  }
}, movie.name = "34-density-estimators-the-film.gif", interval = 0.25, ani.width = 640, ani.height =
↪ 480, ani.dev = function(...) png(..., type="cairo"))

# Regression estimator with a slightly growing bandwidth
mu.grid <- seq(0, 27, length.out = 401)
saveGIF({ # Takes roughly 30 seconds
  for (i in 1:101) {
    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))
    abline(v = mu(Xgrid[i]), lwd = 2, col = "red")
    rug(muhat[i, ], col = "#00000011")
    legend("topright", legend = expression(mu(x)), col = "red", lwd = 2, bty = "n")
    if (i %% 10 == 0) cat(i, "\n")
  }
}, movie.name = "35-regression-estimators-the-film.gif", interval = 0.25, ani.width = 640, ani.height
↪ = 480, ani.dev = function(...) png(..., type="cairo"))

# In 3D, we can use a coarser grig, but generating 101 densities still takes a while, so we
↪ parallelise while we can
den.grid <- seq(0, 0.3, length.out = 201)
kdes <- mclapply(1:101, function(i) kernelDensity(fhat[i, ], den.grid, bw.nrd(fhat[i, ]) / 5),
↪ mc.cores = ncores)
kdes <- lapply(kdes, function(x) x / max(x)) # Normalising them to one for nice plotting
saveGIF({ # Takes roughly 30 seconds on Linux but 20 minutes on Windows!
  for (a in 1:360) {
    par(mar = c(2, 2, 2, 2))
    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")
    lines(trans3d(Xgrid, dchisq(Xgrid, 3), rep(0, 101), pmat = p), lwd = 1, col = "blue")
    for (i in 1:101) lines(trans3d(rep(Xgrid[i], length(den.grid)), den.grid, kdes[[i]], pmat = p),
    ↪ lwd = 1, col = "#00000077")
    if (a %% 5 == 0) cat(a, "/ 360\n")
  }
}, movie.name = "36-kde-3d.gif", interval = 1 / 25, ani.width = 480, ani.height = 480, ani.dev =
↪ function(...) png(..., type="cairo"))

mu.grid <- seq(0, 27, length.out = 201)
mus <- mclapply(1:101, function(i) kernelDensity(muhat[i, ], mu.grid, bw.nrd(muhat[i, ]) / 5),
↪ mc.cores = ncores)
mus <- lapply(mus, function(x) x / max(x))
saveGIF({ # Takes roughly 30 seconds on Linux but 20 minutes on Windows!
  for (a in 1:360) {
    par(mar = c(2, 2, 2, 2))
    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")
    lines(trans3d(Xgrid, mu(Xgrid), rep(0, 101), pmat = p), lwd = 2, col = "red")
    for (i in 1:101) lines(trans3d(rep(Xgrid[i], length(mu.grid)), mu.grid, mus[[i]], pmat = p), lwd =
    ↪ 1, col = "#00000077")
    if (a %% 5 == 0) cat(a, "/ 360\n")
  }
}, movie.name = "37-mu-3d.gif", interval = 1 / 25, ani.width = 480, ani.height = 480, ani.dev =
↪ function(...) png(..., type="cairo"))
```

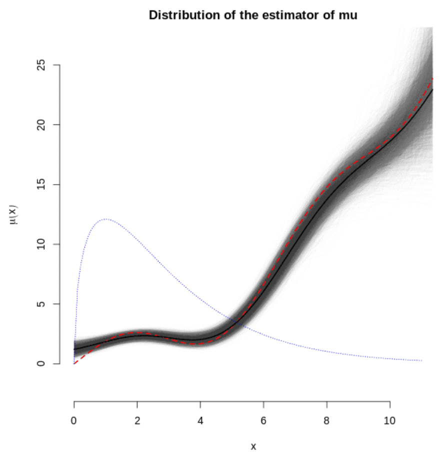

Estimated density in simulation #10000
Density
x

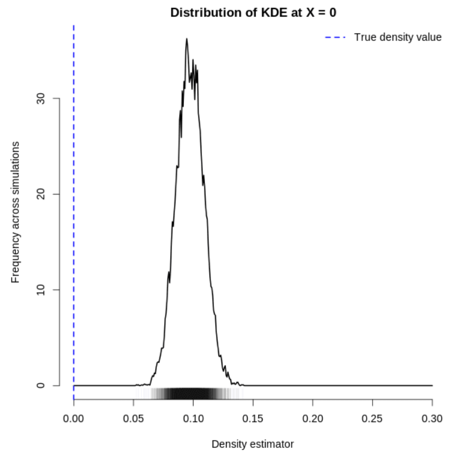

Estimated mu in simulation #10000
μ(x)
x

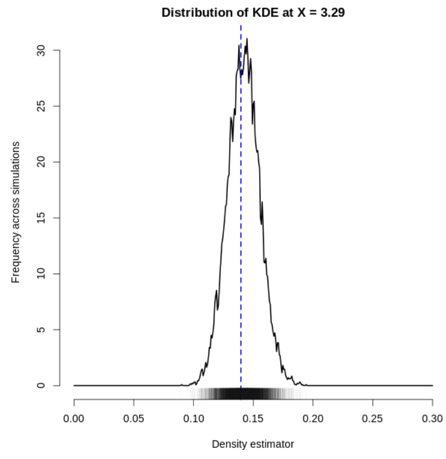

Distribution of the density estimator
True density
Average estimate
50% (IQ) range
95% range
99% range
Density
x


Distribution of the estimator of mu
True mu(x)
Average estimate
50% (IQ) range
95% range
99% range
μ(x)
x


Distribution of the density estimator
Density
x


Distribution of the estimator of mu
μ(x)
x


Distribution of KDE at X = 0
True density value
Frequency across simulations
Density estimator


Distribution of KDE at X = 3.29
Frequency across simulations
Density estimator