Kernel Density Estimation: bandwidth and sample size

A Course on Data Visualization

Livio Finos, University of Padova

Contents

Learning goals and lesson route **2**

Recommended reading . . . 3

KDE: build the curve before choosing the bandwidth **3**

From a histogram to overlapping contributions . . . 3

Density is not probability at a point . . . 5

Activity 1: units and probability . . . 5

Bandwidth changes the scientific story **5**

Start with real waiting times . . . 5

Activity 2: how many peaks would you report? . . . 6

One sample, several resolutions . . . 6

Histogram and KDE on a compatible scale . . . 8

Sample size: more observations at a fixed bandwidth **8**

Keep the smoothing parameter unchanged . . . 8

Activity 3: distinguish more data from more smoothing . . . 10

What decreases, and what remains? . . . 10

Bias and variance: examine repeated samples **12**

What changes when another sample is collected? . . . 12

Quantify the trade-off . . . 13

Activity 4: read repeated curves before the error table . . . 14

Optional theory: why sample size matters . . . 14

Choosing a bandwidth in practice **14**

Automatic selectors are starting points . . . 14

Activity 5: bandwidth selection is a starting point . . . 17

Optional: least-squares cross-validation for KDE . . . 17

Kernel shape versus bandwidth 18
Activity 6: kernel or bandwidth? 19

Boundary problems: smoothness cannot override support **22**

Activity 7: check support on real measurements 23

Activity 8: build and compare custom KDEs **25**

Step 1: define the raw kernels and their scales 25
Step 2: implement two KDE functions 26
Step 3: apply and plot at a common SD 26
Step 4: sensitivity and discussion 27

References **28**

```
knitr::opts_chunk$set(echo = TRUE, message = FALSE, warning = TRUE,
                      fig.width = 8, fig.height = 5, fig.align = "center",
                      dpi = 120, comment = "#>")
options(width = 90)
blue <- "#0072B2"
orange <- "#D55E00"
green <- "#009E73"
purple <- "#CC79A7"
B <- as.integer(params$simulation_reps)
stopifnot(length(B) == 1L, is.finite(B), B >= 20L)
# Direct Gaussian KDE for modest sample sizes.
kde <- function(x, grid, h) {
  stopifnot(length(x) > 1L, all(is.finite(x)),
            length(h) == 1L, is.finite(h), h > 0)
  rowMeans(dnorm(outer(grid, x, "-") / h)) / h
}
trapz <- function(x, y) sum(diff(x) * (head(y, -1) + tail(y, -1)) / 2)
# Keep local graphics settings local to each multi-panel example.
with_panels <- function(nr, nc, code) {
  old <- par(mfrow = c(nr, nc), mar = c(4, 4, 2.5, 1))
  on.exit(par(old))
  force(code)
}
```

Learning goals and lesson route

This unit develops Kernel Density Estimation from individual observations, then separates the effects of **bandwidth** and **sample size**.

By the end, you should be able to:

- construct a KDE and identify the centre and mode of each Gaussian kernel;
- interpret density as probability per unit, with probability represented by area;
- distinguish changing bandwidth at fixed sample size from increasing sample size at fixed bandwidth;
- explain bias and variability using repeated samples;

- compare automatic bandwidth choices and recognise support problems.
- compare common kernels at a matched smoothing scale and explain why kernel choice is different from bandwidth choice.

Prerequisites: density, probability as area, mean and variance, and basic R. The asymptotic rate and cross-validation derivation are optional extensions.

Unit route: build a density from observations, explore bandwidth on real waiting times, separate sample size from smoothing using known populations, compare kernel shapes, and check whether the estimator respects the support. Short activities appear where each idea is introduced: predict, plot, explain.

Reproducibility. Examples use base R and datasets shipped with R. No data downloads or objects from other units are needed. Simulations have explicit seeds.

Recommended reading

- Wilke, Chapter 7, §7.1: Histograms and density plots.
- R documentation: density and bandwidth selectors.

KDE: build the curve before choosing the bandwidth

From a histogram to overlapping contributions

A histogram aggregates observations in bins. KDE distributes a small amount of density around every observation and adds the contributions.

The **bandwidth**, written h, is the positive number that controls the width of each contribution. For a Gaussian kernel, h is exactly the standard deviation of the normal curve centred at each observation. It has the same units as the data: for waiting times in minutes, $h = 5$ means a kernel standard deviation of five minutes. It is not the standard deviation of the sample, the spacing of the plotting grid, or the width of a histogram bin.

A smaller bandwidth gives narrower, taller contributions and preserves more local detail, including random fluctuations. A larger bandwidth gives wider, lower contributions and smooths over that detail. We first choose h ourselves to see its effect; later we study ways to choose it from the data.

With n observations $X_1, \ldots, X_n$, the Gaussian KDE is

$$\widehat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} \phi\left(\frac{x - X_i}{h}\right), \qquad h > 0,$$

where ϕ is the standard normal density. The Gaussian kernel does **not** assume that the data have a normal distribution: the estimate is a mixture of many shifted kernels.

```
x_small <- c(-1.7, -1.2, -.9, -.5, .4, .8, 1.0, 1.7)
h_small <- .35
g_small <- seq(-3, 3, length.out = 601)
contributions <- sapply(x_small, function(z)
  dnorm(g_small, mean = z, sd = h_small) / length(x_small))
total <- rowSums(contributions)
matplot(g_small, contributions, type = "l", lty = 1, col = "grey65",
        ylim = c(-.025, max(total) * 1.35),
        xlab = "Observed variable", ylab = "Density")
```

```r
abline(h = 0, col = "grey75")
lines(g_small, total, col = blue, lwd = 3)
# Every observed value is plotted at exactly y = 0, not as a rug below the axis.
points(x_small, rep(0, length(x_small)), pch = 16, cex = 1.1)
# Highlight one observation, its contribution, and the common x coordinate.
i <- 5
peak_height <- dnorm(x_small[i], mean = x_small[i], sd = h_small) /
  length(x_small)
lines(g_small, contributions[, i], col = orange, lwd = 2)
segments(x_small[i], 0, x_small[i], peak_height, col = orange, lty = 2)
points(x_small[i], 0, pch = 16, col = orange, cex = 1.3)
points(x_small[i], peak_height, pch = 1, col = orange, cex = 1.3, lwd = 2)
legend("topright", c("Observed values (y = 0)", "Kernel for highlighted value", "Sum = KDE"),
       col = c("black", orange, blue), lty = c(NA, 1, 1),
       pch = c(16, NA, NA), lwd = c(NA, 2, 3), bty = "n", cex = .85)
```

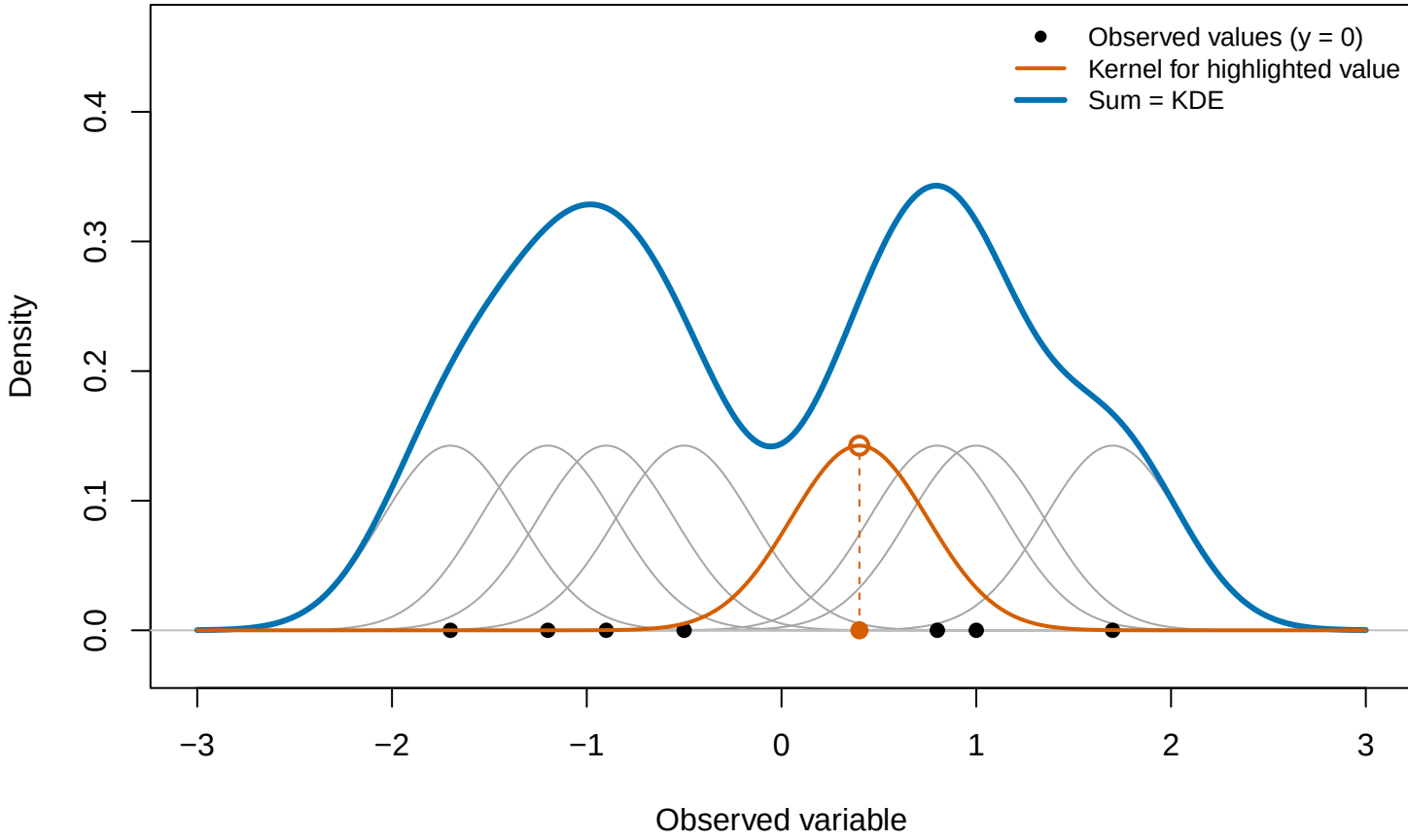


Figure 1: Observed values are shown at y = 0. Each is vertically aligned with the mode of its own normal kernel; the thick blue curve is the sum of the contributions.

Read the alignment. The normal kernel centred at observation X_i has mean, median, and **mode at** $x = X_i$. The point $(X_i, 0)$ lies directly below that kernel's peak. Scaling the kernel by $1/n$ changes its height and area but does not move its mode. The observation is plotted on the baseline to mark its location; its plotted height is not its estimated density.

The highlighted point and peak therefore share the same horizontal coordinate. An observed value need not be a mode of the **sum**: overlapping kernels can shift the modes of the final KDE.

Every contribution has area $1/n$, so the full estimate integrates to one. Larger h means wider and lower individual bumps, not additional observations.

Check understanding. If we double the sample by repeating every observation exactly once and keep h fixed, the KDE does not change. Replication does not add new information about the population.

Density is not probability at a point

For a continuous distribution, probability is area over an interval. A density height can exceed one and changes when measurement units change. For $Z = aX + b$, $a > 0$, the corresponding bandwidth is $h_Z = ah_X$, and

$$\widehat{f}_Z(z) = \frac{1}{a}\widehat{f}_X\left(\frac{z-b}{a}\right).$$

```
# Gaussian-kernel interval probability: integrate each component exactly.
a <- -.5
b <- 1
estimated_probability <- mean(pnorm((b - x_small) / h_small) -
                              pnorm((a - x_small) / h_small))
c(KDE_interval_probability = estimated_probability,
  Empirical_interval_proportion = mean(x_small >= a & x_small <= b))
```

```
#>      KDE_interval_probability Empirical_interval_proportion
#>                     0.3550095                     0.5000000
```

These two numbers need not agree: one uses a smoothed empirical distribution, the other uses the empirical distribution directly.

Activity 1: units and probability

Use these **simulated durations in seconds**, supplied specifically for this activity. They are separate from the dimensionless observations `x_small`.

```
duration_seconds <- c(25, 35, 48, 60, 74, 90, 110, 135)
h_seconds <- 12
grid_seconds <- seq(0, 180, length.out = 501)
density_seconds <- kde(duration_seconds, grid_seconds, h_seconds)
```

Predict the equivalent bandwidth in minutes and the change in density height. Compute a second KDE using `duration_seconds / 60`, `grid_seconds / 60`, and your bandwidth in minutes. Convert that curve back to seconds before overlaying it on `density_seconds`. Submit the plot and check that the estimated probability between 30 and 90 seconds equals the probability between 0.5 and 1.5 minutes. Explain why probability is unchanged even though the numerical density heights change.

Bandwidth changes the scientific story

Start with real waiting times

The built-in `datasets::faithful` dataset records 272 eruptions of the Old Faithful geyser in Yellowstone. Each row is one recorded eruption; `eruptions` is its duration and `waiting` is the waiting time to the next

eruption, both in minutes. These are real observations, not simulated mixture components. See `?faithful` for the source and measurement notes. The data contain rounded measurements; do not interpret fine-scale spikes as precision the observations do not have. Consecutive observations may be dependent, so this descriptive example is not evidence for an independent sampling model.

We need a starting bandwidth before drawing a curve. **`bw.SJ()` is the Sheather-Jones bandwidth selector**: it returns one positive number, in the same units as the observations. Its intuition is to balance two problems: too little smoothing leaves a noisy curve, while too much smoothing hides features. It first uses preliminary smooth estimates to assess how strongly the density bends, then uses that information and the sample size to choose a bandwidth. It estimates the unknown curvature rather than assuming we already know the population density.

This is a starting resolution, not a guarantee that every peak is real. Here we also inspect half and twice that bandwidth. The later section Automatic selectors are starting points explains the rule of thumb and the Sheather-Jones idea in more detail.

```
waiting <- datasets::faithful$waiting
h_wait <- bw.SJ(waiting)
g_wait <- seq(min(waiting) - 15, max(waiting) + 15, length.out = 501)
wait_adjust <- c(1/10,1/2,1,2,10)
wait_curves <- sapply(wait_adjust, function(a)
  kde(waiting, g_wait, a * h_wait))
```

Activity 2: how many peaks would you report?

Before running the next chunk, predict what halving or doubling the bandwidth will do. Use the same data and axes in all three panels. Report one broad feature that persists and one detail that changes. Include the units, sample size, and bandwidth in your explanation. Does a peak prove that there is a distinct latent population group?

```
wait_adjust=c(1/10,1/2,1,2,10)

with_panels(2, 3, {
  for (j in seq_along(wait_adjust)) {
    hist(waiting, breaks = seq(40, 100, by = 5), probability = TRUE,
         xlim = range(g_wait), ylim = c(0, max(wait_curves) * 1.1),
         col = "grey90", border = "white", xlab = "Waiting time (minutes)",
         main = paste0("h = ", round(wait_adjust[j] * h_wait, 2), " minutes"))
    lines(g_wait, wait_curves[, j], col = blue, lwd = 2)
    rug(waiting, col = adjustcolor("black", .2))
  }
})
```

One sample, several resolutions

We simulate a two-component population so that the truth is known. In real data, we do not have this reference curve.

```
r_mix <- function(n) {
  component <- rbinom(n, size = 1, prob = .45)
  rnorm(n, mean = ifelse(component == 0, -1.2, 1.3),
        sd = ifelse(component == 0, .6, .45))
```

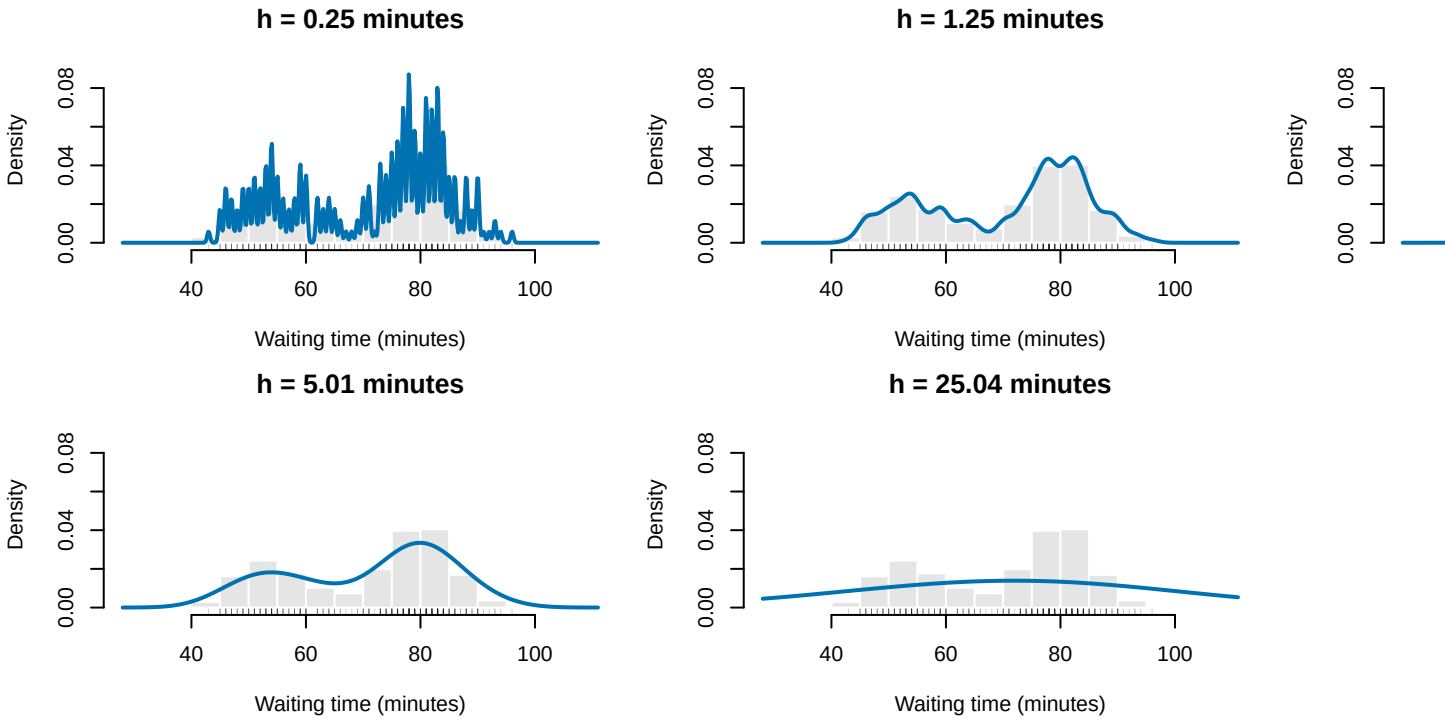


Figure 2: The same real waiting times at three smoothing resolutions. There is no known population density to overlay.

```
}
f_mix <- function(z) .55 * dnorm(z, -1.2, .6) + .45 * dnorm(z, 1.3, .45)
set.seed(1101)
x <- r_mix(220)
grid <- seq(-4.5, 4.5, length.out = 501)
h_values <- c(.10, .35, 1.10)
estimates <- sapply(h_values, function(h) kde(x, grid, h))
```

```
with_panels(1, 3, {
  for (j in seq_along(h_values)) {
    plot(grid, estimates[, j], type = "l", col = blue, lwd = 2,
         ylim = c(0, max(estimates, f_mix(grid)) * 1.05),
         xlab = "x", ylab = "Density", main = paste("h =", h_values[j]))
    lines(grid, f_mix(grid), lty = 2, lwd = 2)
    rug(x, col = adjustcolor("black", .25))
  }
})
```

Count the apparent peaks in each panel and compare them with the generating density. A wiggle is not automatically a population subgroup. Conversely, a smooth unimodal curve can conceal a genuinely multimodal population.

We deliberately show a broad range of bandwidths. The middle value is illustrative, not a universal recommendation.

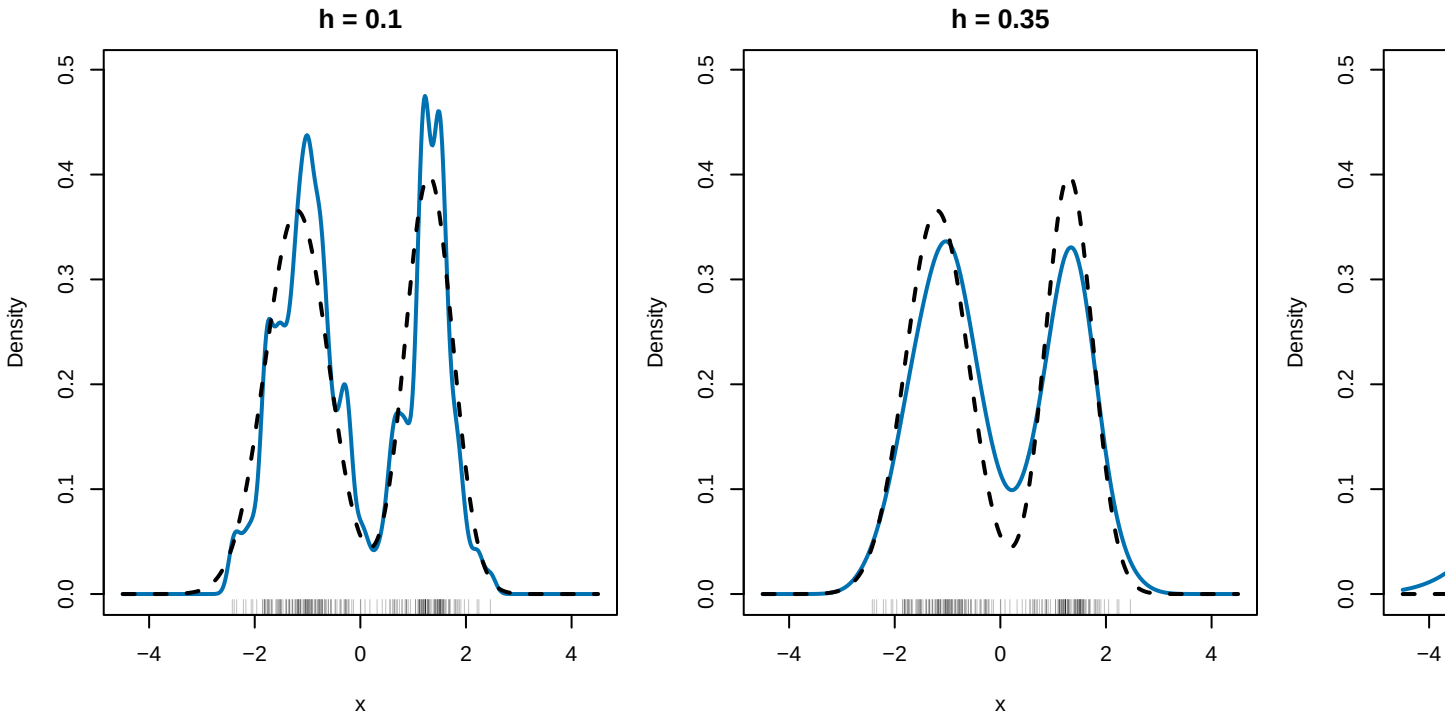


Figure 3: The same sample and axis limits are used in all panels; the dashed curve is the generating density.

Histogram and KDE on a compatible scale

```
hist(x, breaks = seq(-5, 5, by = .4), probability = TRUE,
     col = "grey90", border = "white", xlab = "x", main = "Density, not counts")
lines(grid, kde(x, grid, .35), col = blue, lwd = 2)
rug(x, col = adjustcolor("black", .3))
```

Bin width and bandwidth both control resolution, but they define different estimators. A numerical bin width of 0.4 is not equivalent to a Gaussian bandwidth of 0.4.

Sample size: more observations at a fixed bandwidth

Keep the smoothing parameter unchanged

Now change **only the sample size**. Draw one large independent sample from a standard normal population and use nested prefixes of sizes 30, 150, 750, and 3750. Every panel uses the same Gaussian bandwidth $h = 0.18$, evaluation grid, and axis limits.

With more observations, the random gaps and clusters average out: the estimated curve generally looks **less irregular and more stable**, although we have not widened the kernels. A particular sequence of samples need not become monotonically less wiggly at every step.

```
set.seed(1104)
sample_sizes <- c(30, 150, 750, 3750)
h_fixed <- .18
```

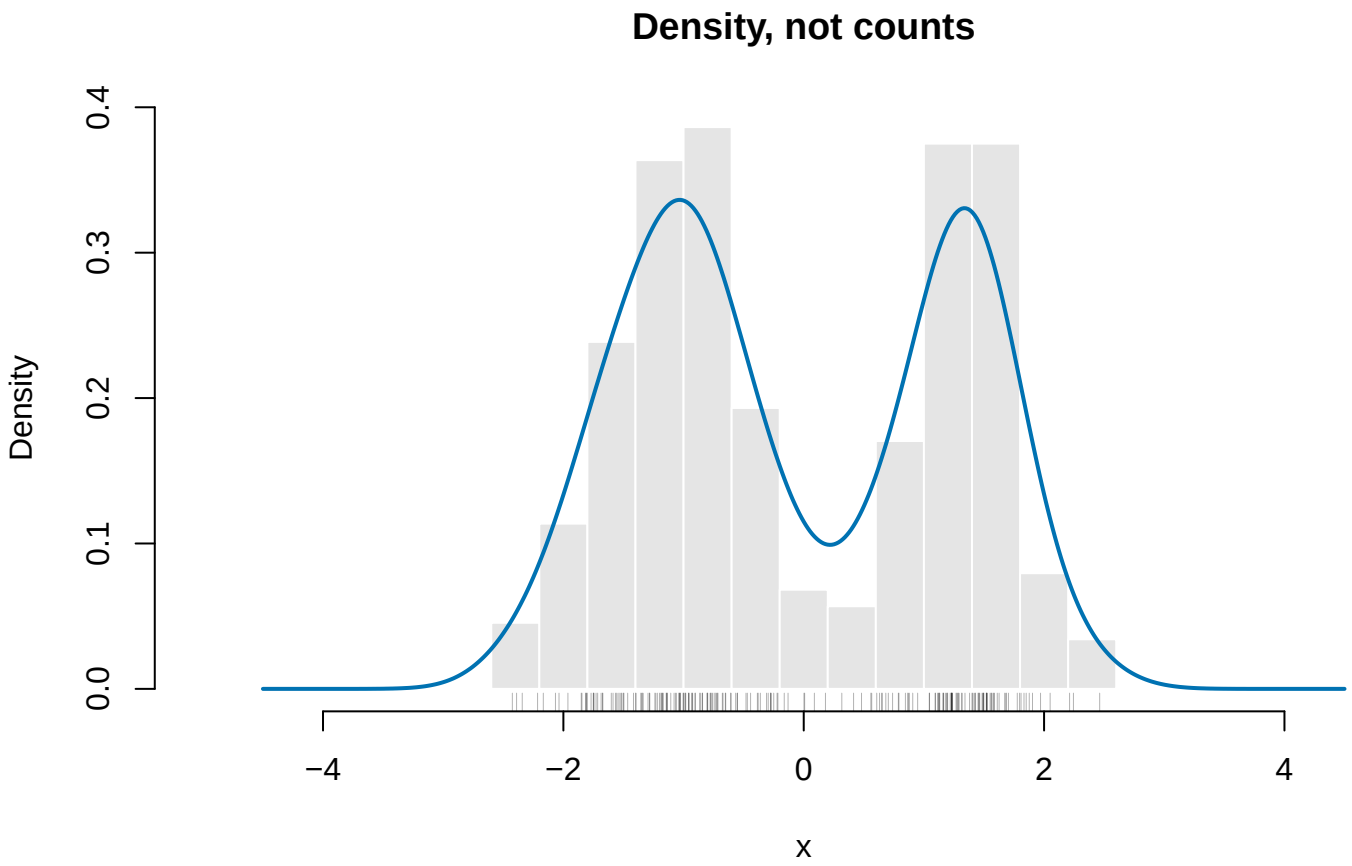


Figure 4: A density-scaled histogram and KDE share the same vertical units.

```r
x_pool <- rnorm(max(sample_sizes))
g_n <- seq(-4, 4, length.out = 601)
curves_n <- sapply(sample_sizes, function(n)
  kde(x_pool[seq_len(n)], g_n, h_fixed))
truth_n <- dnorm(g_n)
# Convolving N(0,1) with N(0,h^2) gives N(0,1+h^2).
expected_n <- dnorm(g_n, sd = sqrt(1 + h_fixed^2))
common_ylim <- c(-.015, 1.18 * max(curves_n, truth_n, expected_n))
with_panels(2, 2, {
  for (j in seq_along(sample_sizes)) {
    n_j <- sample_sizes[j]
    plot(g_n, curves_n[, j], type = "l", col = blue, lwd = 2,
         xlim = range(g_n), ylim = common_ylim,
         xlab = "x", ylab = "Density",
         main = paste0("n = ", n_j, "; h = ", h_fixed))
    lines(g_n, truth_n, lty = 2, lwd = 1.7)
    lines(g_n, expected_n, col = orange, lty = 3, lwd = 1.7)
    abline(h = 0, col = "grey85")
    points(x_pool[seq_len(n_j)], rep(0, n_j), pch = "|",
           col = adjustcolor("black", .18), cex = .55)
    if (j == 1) legend("topright", c("KDE", "Truth", "Expected KDE"),
                       col = c(blue, "black", orange), lty = c(1, 2, 3),
                       lwd = 2, bty = "n", cex = .7)
  }
})
```

Discussion questions. Which peaks persist as new observations are added? Is the kernel width changing? Does the larger sample make the curve more Gaussian because we assumed normal kernels, or because this particular population is normal?

Here the population is normal by construction. Repeating the experiment with a bimodal population would stabilise its smoothed bimodal shape, not force the estimate into one normal curve.

Activity 3: distinguish more data from more smoothing

Generate a nested normal sample of sizes 80, 800, and 8000. Compare bandwidths 0.08, 0.25, and 0.7, holding each bandwidth fixed as sample size changes. Repeat for several seeds, keep identical axes and the true density visible, and report which sample-specific peaks persist. Explain why a bandwidth suited to 80 observations need not be suited to 8000. Replicating the same observations is not the same experiment as collecting new observations.

What decreases, and what remains?

At a fixed evaluation point x, let $Z_i = K_h(x - X_i)$. Independent sampling gives

$$\widehat{f}_h(x) = \frac{1}{n}\sum_i Z_i, \qquad \operatorname{Var}\{\widehat{f}_h(x)\} = \frac{\operatorname{Var}(Z_i)}{n}.$$

At **fixed** h, increasing n by a factor of five reduces pointwise variance by a factor of five, and standard deviation by $\sqrt{5}$. It does not remove smoothing bias: the expectation is the same convolution $f * K_h$ for every n.

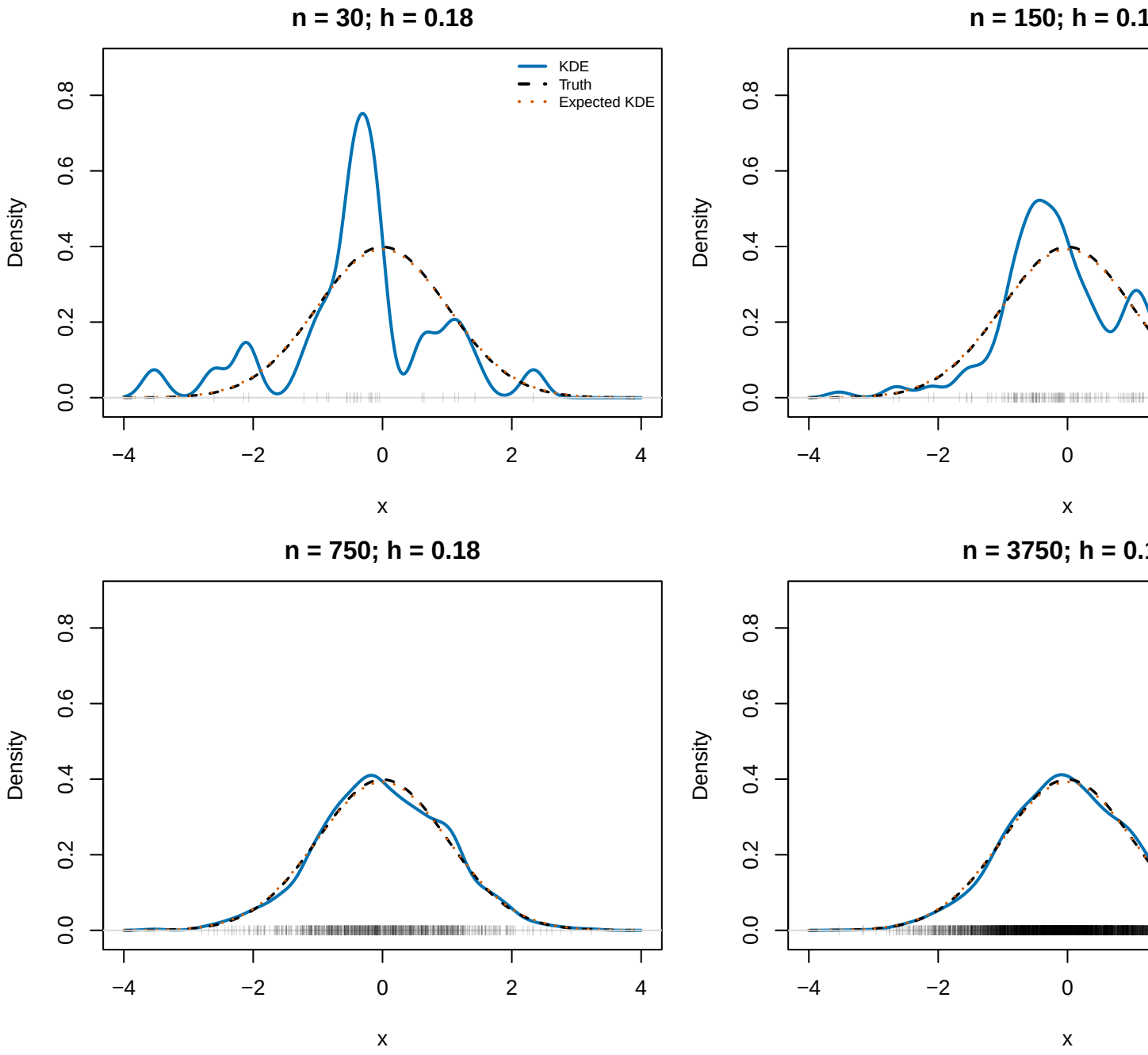


Figure 5: Increasing sample size at the same bandwidth h = 0.18. Blue: estimated KDE; dashed black: true density; dotted orange: the fixed-bandwidth expected density.

For this example, that expectation is a normal density with variance $1+h^2$, slightly wider than the generating $N(0,1)$ density. The orange and black curves nearly overlap because $h = 0.18$ is small; their difference does not shrink just because n increases.

```
# Exact pointwise standard deviation at x = 0 for this normal example.
mean_z <- dnorm(0, sd = sqrt(1 + h_fixed^2))
second_moment_z <- dnorm(0, sd = sqrt(1 + h_fixed^2 / 2)) /
  (2 * sqrt(pi) * h_fixed)
variance_z <- second_moment_z - mean_z^2
knitr::kable(data.frame(
  sample_size = sample_sizes,
  bandwidth = h_fixed,
  expected_density_at_zero = mean_z,
  bias_at_zero = mean_z - dnorm(0),
  standard_deviation_at_zero = sqrt(variance_z / sample_sizes)
), digits = 5)
```

sample_size	bandwidth	expected_density_at_zero	bias_at_zero	standard_deviation_at_zero
30	0.18	0.39263	-0.00631	0.12464
150	0.18	0.39263	-0.00631	0.05574
750	0.18	0.39263	-0.00631	0.02493
3750	0.18	0.39263	-0.00631	0.01115

"More smooth" here refers to reduced sample-specific irregularity. Every Gaussian KDE is already differentiable to all orders, even with very few observations. For consistency for the unsmoothed density, one usually lets h decrease as n grows, with nh increasing; that is a different experiment from holding h fixed.

Bias and variance: examine repeated samples

What changes when another sample is collected?

A small bandwidth makes the estimate sensitive to the particular observations. A large bandwidth averages over a wider neighbourhood, reducing sampling variability but blurring structure. Bias refers to the difference between the **average estimate over repeated samples** and the true density, not to every discrepancy in one curve.

```
set.seed(1102)
mc_grid <- seq(-4.5, 4.5, length.out = 301)
mc_est <- array(NA_real_, dim = c(length(mc_grid), B, length(h_values)))
for (b in seq_len(B)) {
  xb <- r_mix(220)
  for (j in seq_along(h_values)) {
    mc_est[, b, j] <- kde(xb, mc_grid, h_values[j])
  }
}
# Exact mean of the Gaussian KDE for this Gaussian mixture:
# convolution adds the kernel variance h^2 to each component variance.
expected_kde <- function(z, h) {
  .55 * dnorm(z, -1.2, sqrt(.6^2 + h^2)) +
    .45 * dnorm(z, 1.3, sqrt(.45^2 + h^2))
}
```

```r
with_panels(1, 3, {
  for (j in seq_along(h_values)) {
    matplot(mc_grid, mc_est[, seq_len(min(B, 20)), j], type = "l", lty = 1,
            col = adjustcolor("grey40", .25), ylim = c(0, max(mc_est)),
            main = paste("h =", h_values[j]), xlab = "x", ylab = "Density")
    lines(mc_grid, expected_kde(mc_grid, h_values[j]), col = blue, lwd = 3)
    lines(mc_grid, f_mix(mc_grid), lty = 2, lwd = 2)
  }
})
```

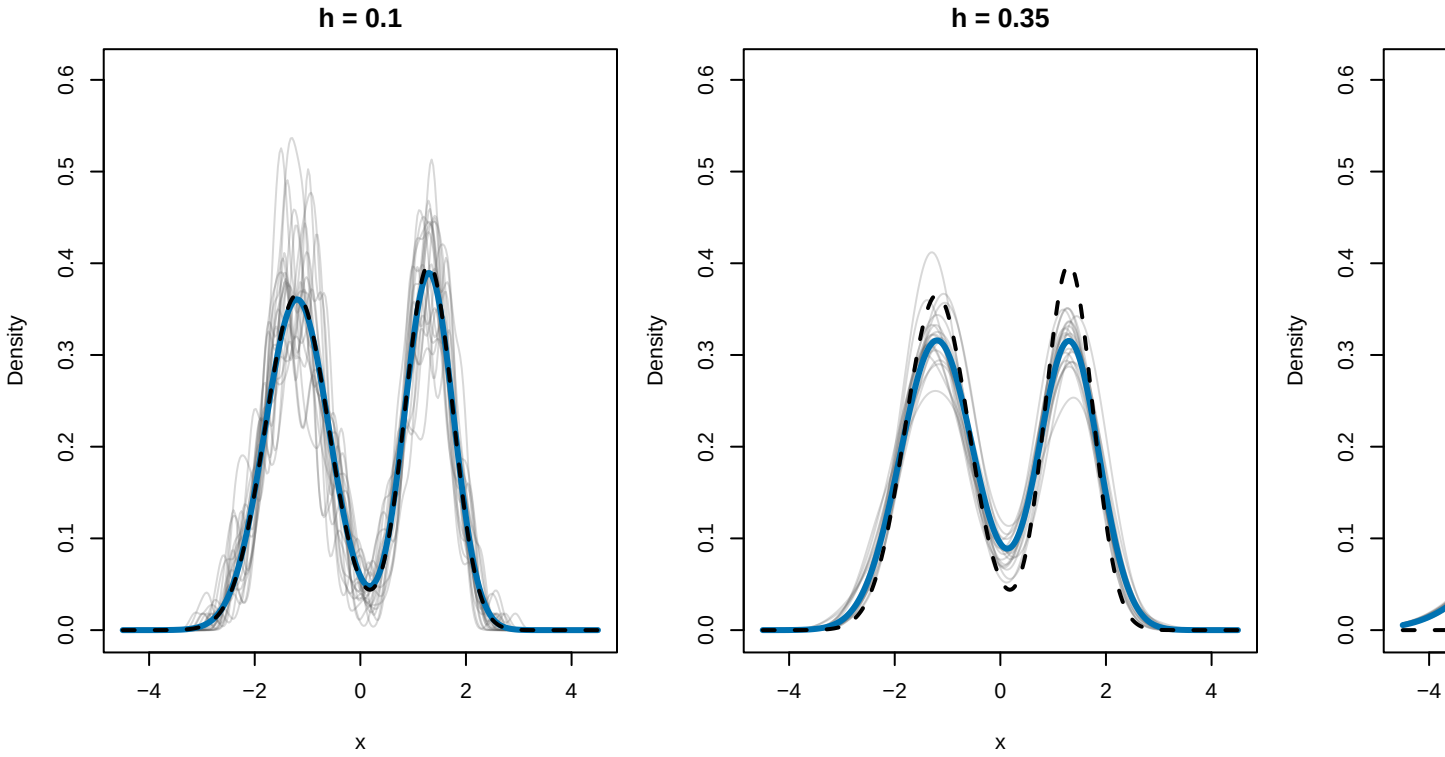


Figure 6: Thin lines are estimates from repeated samples; blue is the exact expected estimate; black dashed is the truth.

Quantify the trade-off

At a fixed x,

$$E\{[\widehat{f}_h(x) - f(x)]^2\} = \mathrm{Var}\{\widehat{f}_h(x)\} + [E\{\widehat{f}_h(x)\} - f(x)]^2.$$

Integrate this identity over x to obtain the corresponding decomposition of mean integrated squared error (MISE). Here we approximate the integrals on a fine grid and estimate the variance by simulation.

```r
tradeoff <- do.call(rbind, lapply(seq_along(h_values), function(j) {
  bias <- expected_kde(mc_grid, h_values[j]) - f_mix(mc_grid)
  integrated_bias2 <- trapz(mc_grid, bias^2)
  integrated_variance <- trapz(mc_grid, apply(mc_est[, , j], 1, var))
  empirical_mise <- mean(apply(mc_est[, , j], 2, function(est)
```

```
    trapz(mc_grid, (est - f_mix(mc_grid))^2)))
  data.frame(h = h_values[j], integrated_bias2, integrated_variance,
             sum_components = integrated_bias2 + integrated_variance,
             empirical_mise)
}))
knitr::kable(tradeoff, digits = 5)
```

h	integrated_bias2	integrated_variance	sum_components	empirical_mise
0.10	0.00008	0.01213	0.01221	0.01221
0.35	0.00742	0.00293	0.01035	0.01011
1.10	0.07469	0.00052	0.07521	0.07515

The last two columns are close, not necessarily identical, because a finite number of simulated samples is used. They also refer to the finite integration interval. Increase `simulation_reps` to reduce Monte Carlo error.

Activity 4: read repeated curves before the error table

Compare the thin curves with the expected KDE and the true density. Which bandwidth gives the largest variability? Which most strongly blurs the two modes? Write two sentences distinguishing sampling variability from smoothing bias, then use the table to support them. Explain why increasing simulation repetitions is different from increasing the size of each sample.

Optional theory: why sample size matters

For independent observations, a sufficiently smooth density, a symmetric second-order kernel, and an interior point, the leading terms behave as

$$\operatorname{Bias}(\widehat{f}_h(x)) \approx \tfrac{1}{2}h^2\mu_2(K)f''(x), \qquad \operatorname{Var}(\widehat{f}_h(x)) \approx \frac{f(x)R(K)}{nh},$$

where $\mu_2(K) = \int u^2K(u)\,du$ and $R(K) = \int K(u)^2\,du$. The integrated approximation has the form $C_1h^4 + C_2/(nh)$ with positive constants under the usual nondegeneracy conditions. Balancing the two terms gives

$$h_{\mathrm{AMISE}} \propto n^{-1/5}.$$

Thus, multiplying the sample size by 32 allows a bandwidth roughly half as large under these assumptions. This is an asymptotic rate, not a universal finite-sample choice, and boundary points require separate treatment. The constants depend on the density and the kernel.

Choosing a bandwidth in practice

Automatic selectors are starting points

For ordinary nondegenerate samples, R's `bw.nrd0` uses

$$h = 0.9\min\{s, \mathrm{IQR}/1.34\}\,n^{-1/5},$$

Here s is the sample standard deviation and n is the sample size. The minimum of s and IQR /1.34 estimates spread while reducing the influence of extreme observations. The $n^{-1/5}$ factor makes the chosen bandwidth smaller as more observations become available. R includes fallback rules when the estimated spread is zero. This is a normal-reference rule, not an optimum for every distribution.

How does `bw.SJ` improve on this? Spread alone does not describe the shape of a density: two distributions with similar spread may have very different peaks and valleys. Sheather-Jones uses preliminary estimates of density derivatives to estimate curvature. The second derivative $f''(x)$ measures how strongly the curve bends: smoothing across a sharp bend can introduce more bias than smoothing a nearly straight part.

The theoretical balance has the form $C_1 h^4 + C_2/(nh)$: the first part grows with excessive smoothing and depends on curvature; the second part reflects sampling variability and decreases with smoothing. A **pilot estimate** is a preliminary smooth estimate used to estimate the unknown quantities in this balance. **Plug-in** means replacing those unknown quantities with estimates from the observations. In R, `bw.SJ()` defaults to a solve-the-equation version (`method = "ste"`); `method = "dpi"` requests the direct plug-in version. You do not need the algorithm's derivation to interpret the returned bandwidth or to check its sensitivity with `adjust`.

`bw.ucv` instead compares bandwidths through a leave-one-out criterion, as explained in the next optional section. All these automatic choices are starting points; they can disagree, especially for small or multimodal samples. These selectors are designed for Gaussian KDEs. In the later kernel comparison we deliberately keep one bandwidth fixed to compare shapes, rather than optimise a separate bandwidth for each kernel. See the R documentation for bandwidth selectors.

```
h_auto <- c(nrd0 = bw.nrd0(waiting), SJ = bw.SJ(waiting))
knitr::kable(data.frame(selector = names(h_auto), bandwidth = unname(h_auto)),
             digits = 4)
```

selector	bandwidth
nrd0	3.9876
SJ	2.5044

```
# density() uses bw = "nrd0" by default.
d_default <- density(waiting)
d_sj <- density(waiting, bw = "SJ")
# adjust multiplies the selected bandwidth:
d_half <- density(waiting, bw = "SJ", adjust = .5)
```

```
plot(d_default, col = blue, lwd = 2, ylim = c(0, max(d_default$y, d_sj$y)),
     xlab = "Waiting time (minutes)", main = "Two starting resolutions")
lines(d_sj, col = orange, lwd = 2)
lines(d_half, col = green, lwd = 2)
rug(waiting, col = adjustcolor("black", .2))
legend("topright", c("nrd0", "SJ","half"),
       col = c(blue, orange), lty = 1, lwd = 2, bty = "n")
```

Practical workflow: select a defensible starting value; show a small range around it; retain raw observations or a histogram; describe features that persist. When comparing groups on the same measurement scale, consider using a common bandwidth to compare at a common resolution. A common bandwidth is not necessarily optimal for each group.

Changing `density(..., n = 2048)` increases the number of evaluation points; it does **not** replace the bandwidth as the smoothing parameter.

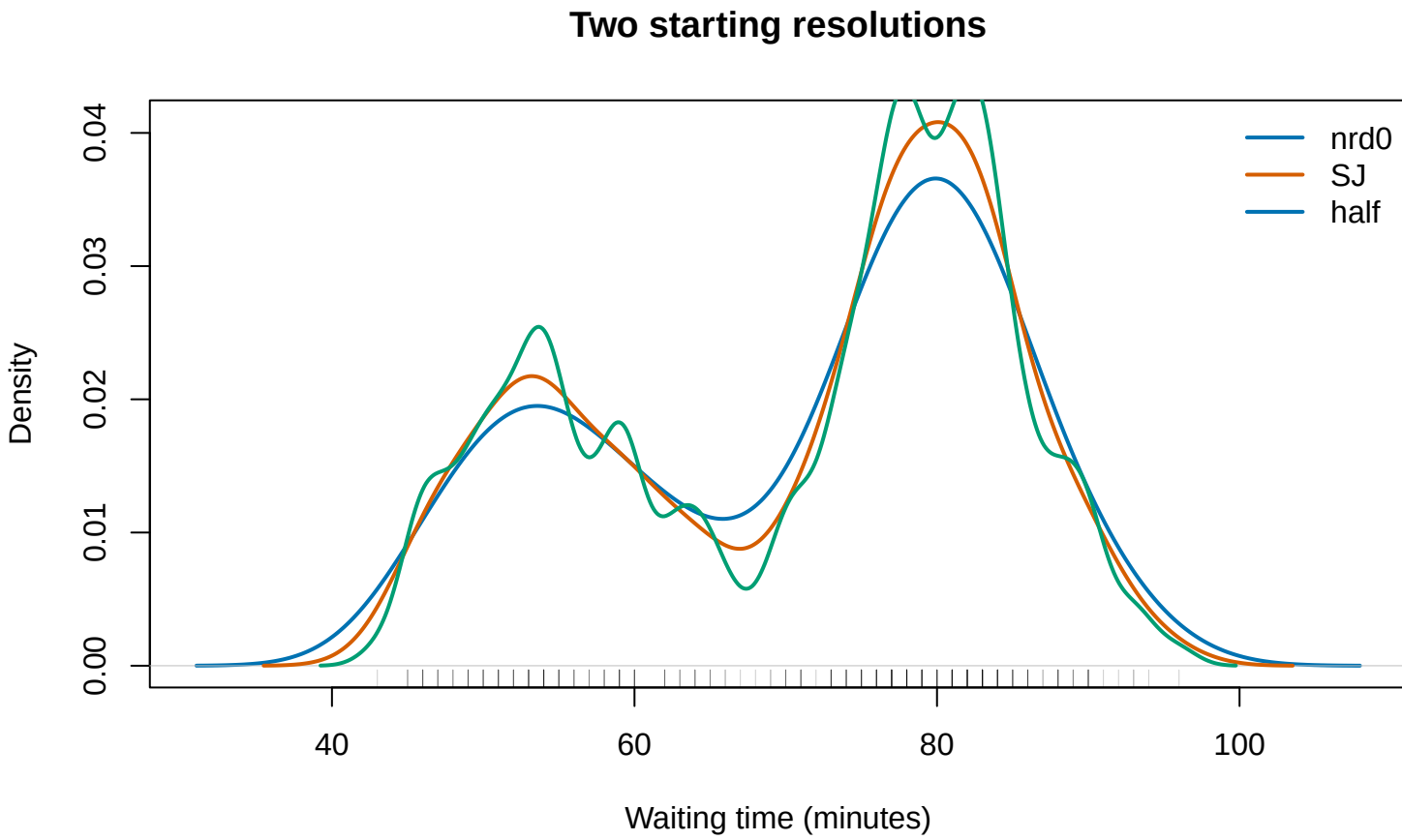


Figure 7: Automatic bandwidth choices for real waiting times; neither is validated by a known population density.

Activity 5: bandwidth selection is a starting point

For `faithful$eruptions`, compare `bw.nrd0` and `bw.SJ`. Plot 0.5, 1, and 2 times the SJ bandwidth on identical axes, with a rug and duration in minutes. Submit the plot and a short account of the stable features. Explain why changing `density(..., n = 2048)` answers a different question from changing `adjust`. Reuse the waiting-time code, replacing the data and axis labels.

Optional: least-squares cross-validation for KDE

For a realised sample, integrated squared error is

$$\mathrm{ISE}(h) = \int \widehat{f}_h^2(x)\,dx - 2\int \widehat{f}_h(x) f(x)\,dx + \int f^2(x)\,dx.$$

The last term does not depend on h. We estimate the second term using leave-one-out densities:

$$\mathrm{CV}(h) = \int \widehat{f}_h^2(x)\,dx - \frac{2}{n}\sum_i \widehat{f}_{h,-i}(X_i).$$

Leaving observation i out prevents its own kernel from inflating the density evaluated at X_i. Under independent sampling this criterion estimates the relevant integrated risk up to an additive constant. It targets integrated density error.

For Gaussian kernels the first integral is available analytically. The next implementation evaluates the criterion on a candidate grid, without numerical integration.

```
gaussian_lscv <- function(x, h) {
  n <- length(x)
  delta <- outer(x, x, "-")
  int_f2 <- mean(dnorm(delta, sd = sqrt(2) * h))
  leave_out <- dnorm(delta, sd = h)
  diag(leave_out) <- 0
  int_ff <- sum(leave_out) / (n * (n - 1))
  int_f2 - 2 * int_ff
}
h_grid <- exp(seq(log(.07), log(1.2), length.out = 65))
cv_kde <- vapply(h_grid, function(h) gaussian_lscv(x, h), numeric(1))
h_cv <- h_grid[which.min(cv_kde)]
# Oracle comparison: possible only because this is a simulation.
ise <- vapply(h_grid, function(h)
  trapz(grid, (kde(x, grid, h) - f_mix(grid))^2), numeric(1))
h_oracle <- h_grid[which.min(ise)]
knitr::kable(data.frame(method = c("CV grid minimum", "Realised-sample oracle"),
                        bandwidth = c(h_cv, h_oracle)), digits = 4)
```

method	bandwidth
CV grid minimum	0.1859
Realised-sample oracle	0.2427

```r
with_panels(1, 2, {
  plot(h_grid, cv_kde, type = "l", log = "x", lwd = 2, col = blue,
       xlab = "Bandwidth (log axis)", ylab = "LSCV criterion")
  abline(v = h_cv, lty = 2)
  plot(h_grid, ise, type = "l", log = "x", lwd = 2, col = orange,
       xlab = "Bandwidth (log axis)", ylab = "Integrated squared error")
  abline(v = h_oracle, lty = 2)
})
```

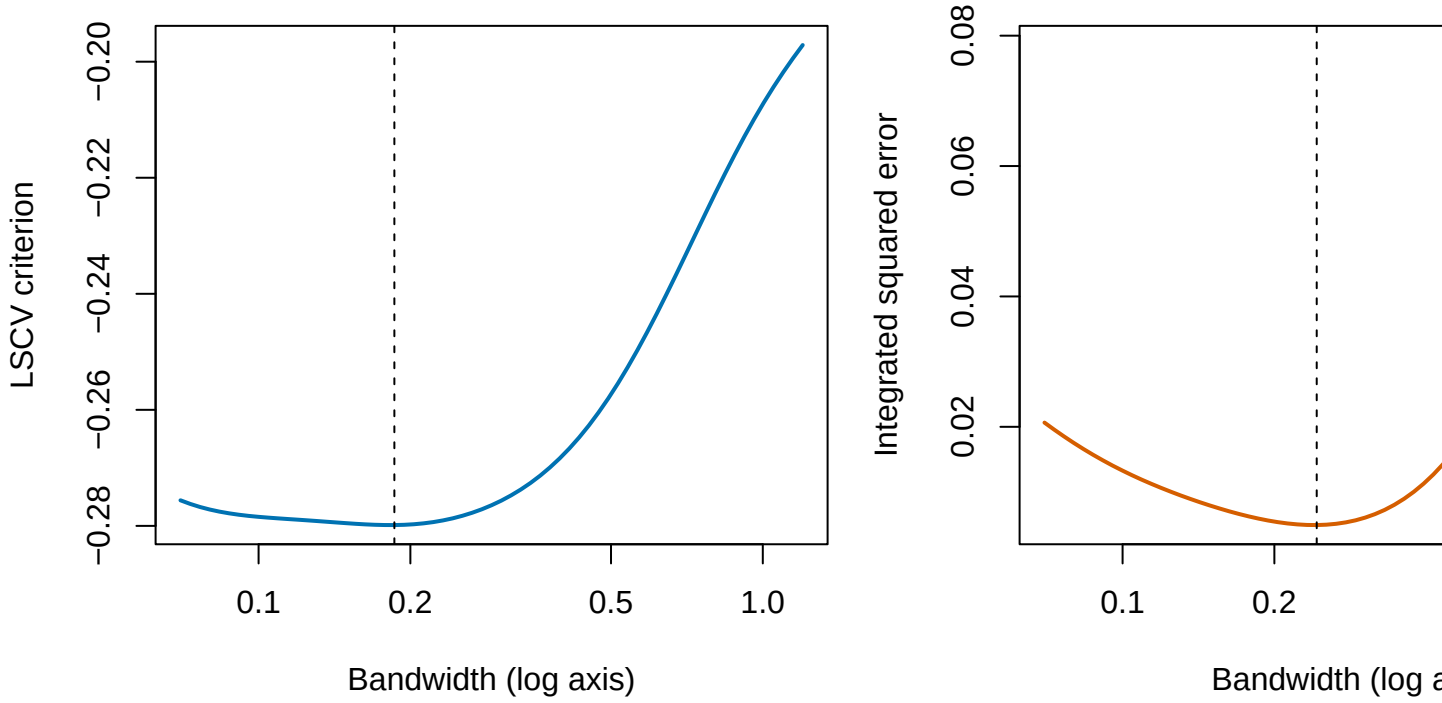


Figure 8: CV is observable; the realised-sample ISE is an oracle quantity available only in simulation.

The selected bandwidths need not coincide. The oracle minimises error for this particular sample, while CV is a noisy risk estimate. The grid is part of the procedure: if its minimum is at an endpoint, expand it and inspect the criterion. Different numerical searches need not return exactly the same result as this grid.

Kernel shape versus bandwidth

The Gaussian is convenient, but it is not the only kernel. In general,

$$\widehat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right).$$

For the nonnegative kernels considered here, each contribution still has area $1/n$. Changing K changes the shape of each contribution; changing h changes its scale. A rectangular kernel has a flat top, so the earlier statement about a unique peak at each observation applies to the Gaussian, not to every kernel.

Kernel	Shape and support	What to notice
Gaussian	Smooth bell; infinite support	Nonzero tails extend beyond every observation.
Epanechnikov	Parabolic bump; compact support	A common efficient second-order kernel; not universally best in finite samples.
Triangular	Piecewise linear bump; compact support	A pointed peak and finite neighbourhood.
Rectangular	Flat bump; compact support	Abrupt edges can produce a visibly less smooth estimate.

R also offers `biweight`, `cosine`, and `optcosine`. Its `density()` function rescales all these kernels to variance one: `bw` is the standard deviation of the smoothing kernel. A textbook Epanechnikov kernel on [-1, 1] uses a different scaling convention. Equal `bw` gives equal kernel standard deviations, not identical support widths or exactly equal risk. See the R documentation.

```
kernel_names <- c("gaussian", "epanechnikov", "triangular", "rectangular")
kernel_colours <- c(blue, orange, green, purple)
kernel_shapes <- lapply(kernel_names, function(k)
  density(0, bw = 1, kernel = k, from = -4, to = 4, n = 1024))
plot(kernel_shapes[[1]], col = blue, lwd = 2,
     ylim = c(0, max(vapply(kernel_shapes, function(d) max(d$y), numeric(1)))),
     xlab = "Distance from the observation", main = "Same standard deviation")
for (j in 2:length(kernel_names)) {
  lines(kernel_shapes[[j]], col = kernel_colours[j], lty = j, lwd = 2)
}
legend("topright", kernel_names, col = kernel_colours,
       lty = seq_along(kernel_names), lwd = 2, bty = "n", cex = .8)
```

Activity 6: kernel or bandwidth?

Use the same waiting times, the same evaluation grid, and `bw = h_wait`. Compare the four kernels below. Then repeat with `bw = 2 * h_wait` while holding the kernel Gaussian. Which change affects the broad interpretation more? Report persistent features and local differences; do not select a kernel solely because it gives the desired number of peaks.

```
kernel_estimates <- lapply(kernel_names, function(k)
  density(waiting, bw = h_wait, kernel = k, from = min(g_wait),
          to = max(g_wait), n = 1024))
plot(kernel_estimates[[1]], col = blue, lwd = 2,
     ylim = c(0, max(vapply(kernel_estimates, function(d) max(d$y), numeric(1)))),
     xlab = "Waiting time (minutes)", main = "Kernel shape at a fixed scale")
for (j in 2:length(kernel_names)) {
  lines(kernel_estimates[[j]], col = kernel_colours[j], lty = 1, lwd = 2)
}
rug(waiting, col = adjustcolor("black", .2))
legend("topright", kernel_names, col = kernel_colours,
       lty = 1, lwd = 2, bty = "n", cex = .8)
```

For many ordinary KDE comparisons, bandwidth changes the broad shape more than switching between common kernels at comparable scales. Kernel choice still affects support and smoothness. A compact-support

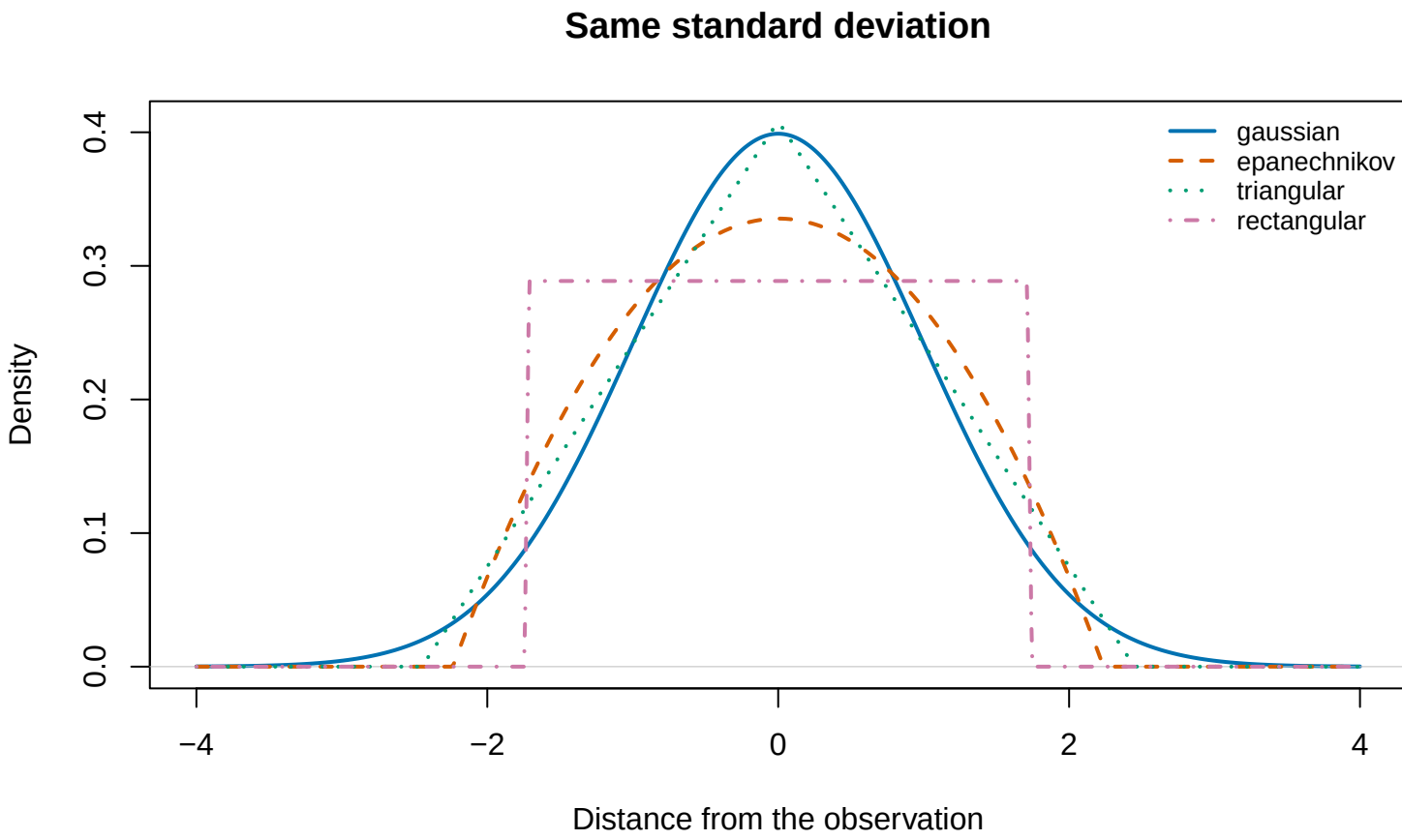


Figure 9: Individual kernels centred at zero, with bandwidth 1 under R's unit-variance convention.

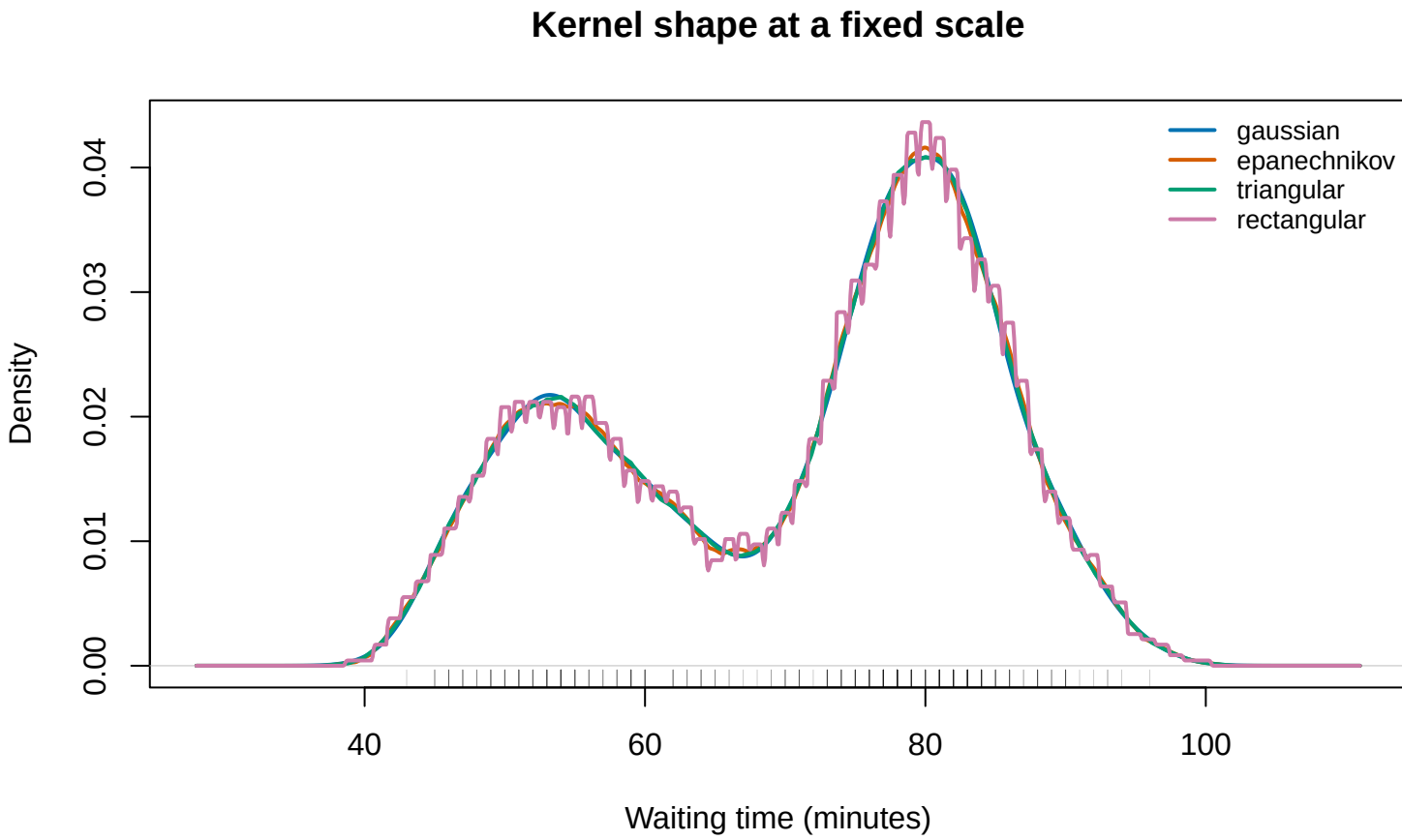


Figure 10: Four kernels applied to real waiting times at a common kernel standard deviation. The bandwidth has been fixed before comparing shapes.

kernel does not automatically solve a boundary problem: its bump can cross zero when an observation is near zero. The earlier exact convolution and cross-validation formulas remain specifically Gaussian; do not reuse them unchanged for other kernels.

Boundary problems: smoothness cannot override support

A Gaussian KDE of a positive variable assigns some probability to negative values. Hiding that part of the curve changes the display but does not correct the estimator.

One alternative is to estimate the density of $Z = \log X$, then express that density in the original units. **Changing the horizontal axis alone is not enough**, because density means probability *per unit of horizontal length*. The logarithm compresses larger values more strongly than smaller values. The height must change as well to preserve probability as area.

For a small interval of width dx near $x > 0$, the corresponding interval on the log scale has width approximately $dz = dx/x$, since $d\log(x)/dx = 1/x$. The same probability can be written on either scale:

$$f_X(x)\,dx \approx f_Z(\log x)\,dz \approx f_Z(\log x)\frac{dx}{x}.$$

Consequently, the change-of-variable formula is

$$\widehat{f}_X(x) = \frac{\widehat{f}_Z(\log x)}{x}, \qquad x > 0.$$

The factor $1/x$ is the absolute derivative of the transformation. In one dimension this is called the **Jacobian factor**. Here it simply converts density per log-unit into density per original unit. For example, an interval of one mile near 100 miles corresponds to about 0.01 log-units: its probability is approximately $\widehat{f}_Z(\log 100) \times 0.01$, so its density per mile is approximately $\widehat{f}_Z(\log 100)/100$.

In code, evaluate the log-scale KDE at `log(grid_original)`, then divide its heights by `grid_original`. Exponentiating the log coordinates without dividing the heights by x would not preserve the area or produce a density in the original units.

```
set.seed(1103)
positive <- rlnorm(250, meanlog = 0, sdlog = .65)
g_pos <- seq(-1, 5, length.out = 601)
h_pos <- bw.SJ(positive)
ordinary <- kde(positive, g_pos, h_pos)
# Fit on log values; division by x converts heights to original-scale density.
g_strict <- seq(.015, 5, length.out = 601)
h_log <- bw.SJ(log(positive))
log_corrected <- kde(log(positive), log(g_strict), h_log) / g_strict
plot(g_pos, ordinary, type = "l", col = blue, lwd = 2,
     ylim = c(0, max(ordinary, log_corrected, dlnorm(g_strict, 0, .65))),
     xlab = "Positive variable", ylab = "Density", main = "Respect the support")
lines(g_strict, log_corrected, col = orange, lwd = 2)
lines(g_strict, dlnorm(g_strict, 0, .65), lty = 2, lwd = 2)
abline(v = 0, col = "grey50")
rug(positive, col = adjustcolor("black", .25))
```

```
#> Warning in rug(positive, col = adjustcolor("black", 0.25)): alcuni valori saranno
#> tagliati
```

```
legend("topright", c("Ordinary KDE", "Log-transform KDE", "Truth"),
       col = c(blue, orange, "black"), lty = c(1, 1, 2), lwd = 2, bty = "n")
```

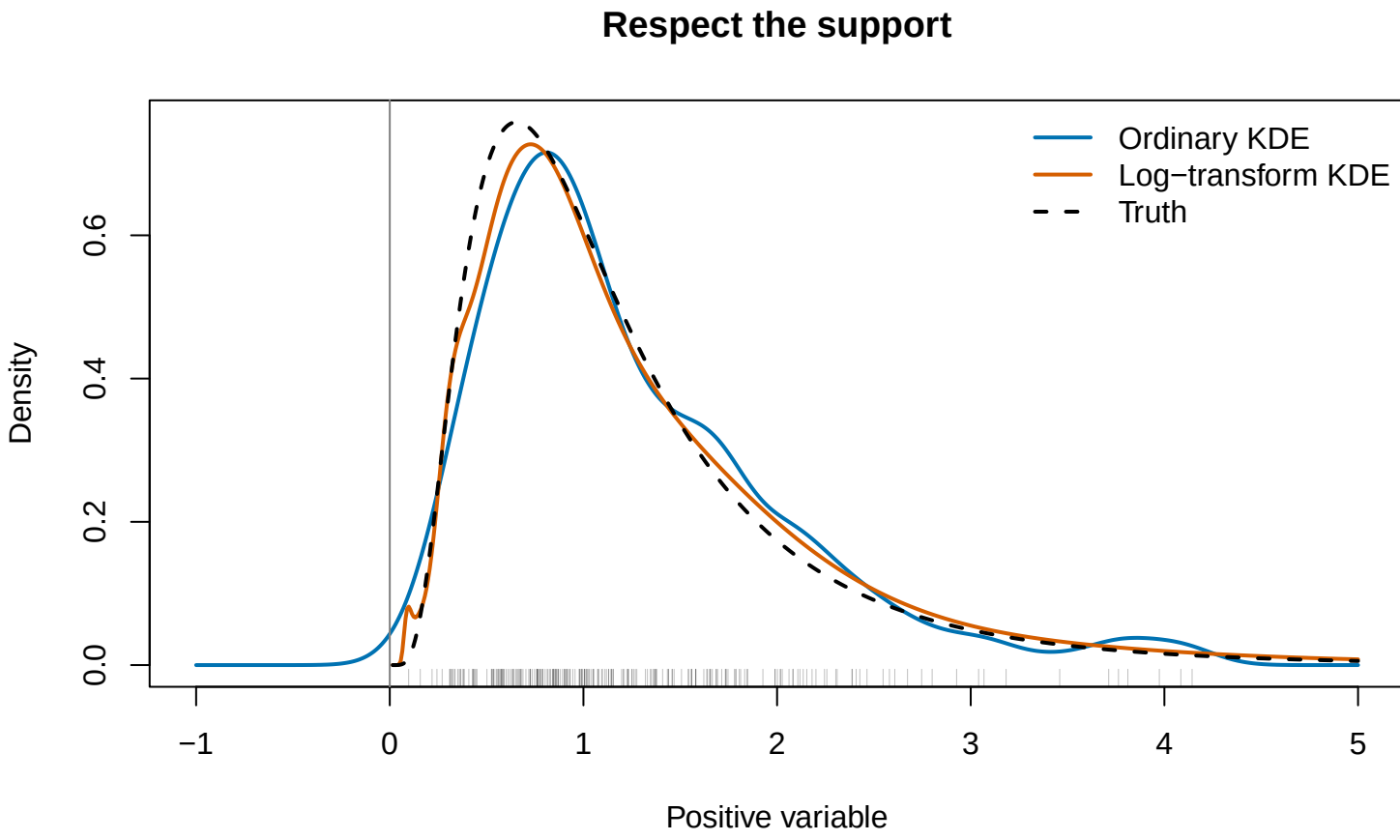


Figure 11: Ordinary KDE leaks below zero. A log-scale estimator respects positivity but still has estimation error.

```
mean(pnorm(-positive / h_pos))  # exact mass below zero for ordinary KDE
```

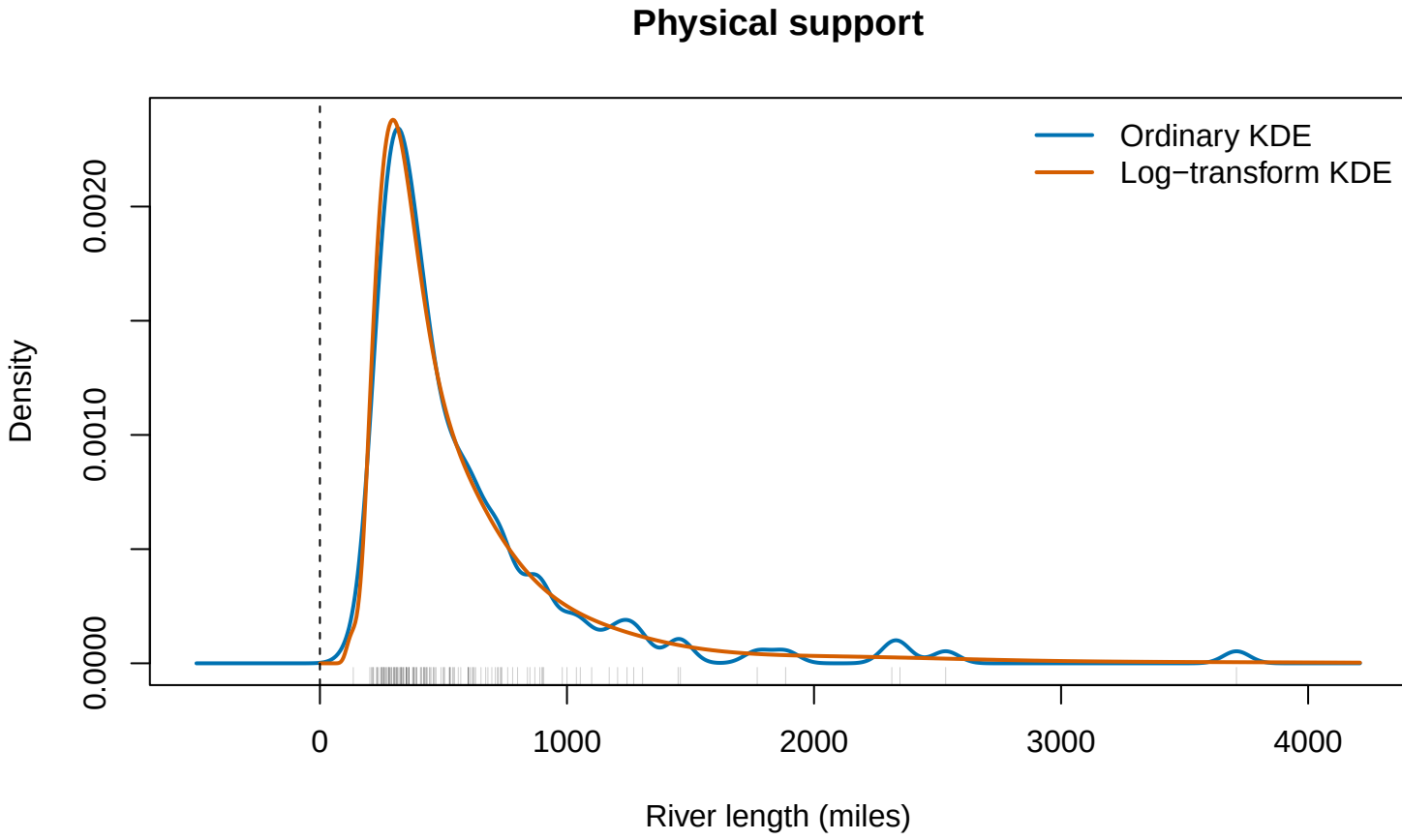
```
#> [1] 0.00395856
```

The $1/x$ factor is essential. This transformation is not available for zero or negative observations without a different model, and it changes the smoothing scale: smoothing on the log scale is multiplicative on the original scale. It is one approach, not a universal boundary correction.

Activity 7: check support on real measurements

The built-in `datasets::rivers` dataset contains lengths in miles of 141 major North American rivers; one observation is one river. Consult `?rivers` for the source. This is a collection of major rivers, not a random sample of all rivers. Compare an ordinary Gaussian KDE with a KDE fitted to log lengths. Express the latter in miles using the formula just derived: evaluate at the log of each positive grid value and divide the density by that grid value. Submit a plot with units and compute the ordinary KDE's mass below zero. Discuss the support problem and the effect of the changed smoothing scale; do not claim that the corrected curve is truth.

```r
river_lengths <- datasets::rivers
h_river <- bw.SJ(river_lengths)
g_river <- seq(-500, max(river_lengths) + 500, length.out = 801)
g_river_positive <- seq(1, max(river_lengths) + 500, length.out = 801)
ordinary_river <- kde(river_lengths, g_river, h_river)
log_river <- kde(log(river_lengths), log(g_river_positive),
                 bw.SJ(log(river_lengths))) / g_river_positive
plot(g_river, ordinary_river, type = "l", col = blue, lwd = 2,
     ylim = c(0, max(ordinary_river, log_river)),
     xlab = "River length (miles)", ylab = "Density", main = "Physical support")
lines(g_river_positive, log_river, col = orange, lwd = 2)
abline(v = 0, lty = 2)
rug(river_lengths, col = adjustcolor("black", .2))
legend("topright", c("Ordinary KDE", "Log-transform KDE"),
       col = c(blue, orange), lty = 1, lwd = 2, bty = "n")
```




```r
mean(pnorm(-river_lengths / h_river))
```

```
#> [1] 4.305145e-05
```

Activity 8: build and compare custom KDEs

Use the real waiting times in `datasets::faithful$waiting`, already introduced in Activity 2. There are 272 observations, measured in minutes. The data have two broad waiting-time regions; there is no known population density to overlay. Your task is to construct two KDE functions and investigate how kernel shape affects their description of these observations.

Alongside a rectangular kernel, design a kernel that you expect to **"do its job poorly"**: give it low values near zero and higher values near the edges of its support. Choose its shape yourself, then investigate whether and why this allocation of density produces an unhelpful summary. The aim is to explain the mechanism, not just to label a curve as good or bad.

Main comparison: equal kernel standard deviations. In this activity the argument `h` of both functions means the standard deviation of the smoothing kernel, in minutes, just as `bw` does in R's `density()`. This is a measure of the width of each contribution, not the standard deviation of the waiting times themselves.

```
x_activity <- datasets::faithful$waiting
h_activity <- bw.SJ(x_activity)
adjust_activity <- c(.5, 1, 2)
# Define the comparison grid after choosing your custom kernel and its SD.
c(sample_size = length(x_activity), kernel_sd_minutes = h_activity)
```

```
#>       sample_size kernel_sd_minutes
#>        272.000000          2.504371
```

The SJ reference value is approximately 2.50 minutes. You will also compare approximately 1.25 and 5.01 minutes (half and twice SJ), keeping the same grid and axis limits. These values are exploratory resolutions, not optimal bandwidths established separately for the custom kernels. Recall the intuition behind bandwidth selection.

Step 1: define the raw kernels and their scales

Use the following rectangular kernel, which is zero outside $[-1, 1]$:

$$K_R(u) = \begin{cases} \frac{1}{2}, & |u| \le 1, \\ 0, & |u| > 1. \end{cases}$$

Design a second kernel K_C that is also zero outside $[-1, 1]$, nonnegative, symmetric, and has area one, but has low values near zero and higher values near the edges. You are not given a formula: choose a function, normalize it, and check these requirements. Define it as a vectorized R function `K_custom(u)`.

Plot both raw shapes with solid lines. Before looking at the KDEs, explain where each kernel places density relative to a single observation. Predict what your custom kernel might do poorly and what evidence would support or challenge your prediction.

Calculate the standard deviation of your chosen raw kernel. Since both kernels are symmetric about zero, their means are zero:

$$\sigma_R = \sqrt{1/3}, \qquad \sigma_C = \sqrt{\int_{-1}^{1} u^2 K_C(u)\, du}.$$

If the scale multiplying a raw kernel is s, the standard deviation of its contribution is $s\sigma_K$. To give both contributions the specified standard deviation h, use

$$s_R = \frac{h}{\sigma_R} = h\sqrt{3}, \qquad s_C = \frac{h}{\sigma_C}.$$

At the SJ reference value the rectangular raw scale is approximately 4.34 minutes. Calculate the custom raw scale from your own kernel's SD. Both contributions will have SD 2.50 minutes, but their compact support widths may differ. Equal SD does not imply equal support, equal shape, or equal statistical error.

Step 2: implement two KDE functions

Construct `kde_rectangular(x, grid, h)` and `kde_custom(x, grid, h, kernel, kernel_sd)`. Pass your raw kernel function and its standard deviation to the second function. Use the appropriate raw scale $s_K = h/\sigma_K$ in

$$\widehat{f}_h(t) = \frac{1}{n s_K} \sum_{i=1}^{n} K\left(\frac{t - X_i}{s_K}\right).$$

For each grid value `t`, calculate `u <- (t - x) / scale`, apply the raw kernel, average the contributions and divide by `scale`. Check finite observations and a single positive finite `h`; also check that `kernel_sd` is positive and finite. Return one height per grid value and do not call `density()` inside your functions.

Complete this chunk before running it; the unfinished handout does not execute it when knitted.

```
K_custom <- function(u) {
  # Choose a shape with low values near zero and higher values near the edges.
  # Return zero outside [-1, 1] and normalize the area to one.
  stop("Design and implement your custom kernel.")
}
sigma_custom <- NA_real_  # Calculate the SD of your normalized raw kernel.

kde_rectangular <- function(x, grid, h) {
  # h is the requested kernel SD in minutes.
  # Use scale <- h / sqrt(1 / 3), then apply K_R.
  stop("Complete the rectangular KDE.")
}
kde_custom <- function(x, grid, h, kernel, kernel_sd) {
  # Use scale <- h / kernel_sd, then apply kernel(u).
  stop("Complete the custom KDE.")
}
```

Step 3: apply and plot at a common SD

Apply both functions to the waiting times. The existing Gaussian helper `kde()` uses a standard-normal kernel, so its `h` is already the kernel SD. All three curves below therefore use the **same kernel SD in minutes**.

```
# Cover both compact kernels at all requested SDs, plus practically all
# Gaussian mass (within four SDs of the most extreme observations).
margin_activity <- max(4, 1 / sqrt(1 / 3), 1 / sigma_custom) *
                   max(adjust_activity) * h_activity
grid_activity <- seq(min(x_activity) - margin_activity,
                     max(x_activity) + margin_activity,
```

```
                    length.out = 8193)

y_rect <- kde_rectangular(x_activity, grid_activity, h_activity)
y_custom <- kde_custom(x_activity, grid_activity, h_activity,
                       kernel = K_custom, kernel_sd = sigma_custom)
y_gauss <- kde(x_activity, grid_activity, h_activity)

plot(grid_activity, y_rect, type = "l", col = blue, lwd = 2,
     ylim = c(0, max(y_rect, y_custom, y_gauss)),
     xlab = "Waiting time (minutes)", ylab = "Density",
     main = paste("Common kernel SD:", round(h_activity, 2), "minutes"))
lines(grid_activity, y_custom, col = orange, lwd = 2)
lines(grid_activity, y_gauss, col = green, lwd = 2)
rug(x_activity)
legend("topright", c("Rectangular", "Custom", "Gaussian"),
       col = c(blue, orange, green), lty = 1, lwd = 2, bty = "n")

c(rectangular_area = trapz(grid_activity, y_rect),
  custom_area = trapz(grid_activity, y_custom),
  gaussian_area_on_grid = trapz(grid_activity, y_gauss))
```

The grid covers the full support of the two compact-kernel estimates at all three requested SDs. Their numerical areas should be close to one; small errors occur at jumps. The Gaussian has infinite support, but the extended grid should also capture practically all its mass.

Step 4: sensitivity and discussion

Submit your functions, the kernel-shape plots, and KDE comparisons at `.5 * h_activity`, `h_activity`, and `2 * h_activity`. Use the same x and y limits across panels and only solid lines.

1. Which broad waiting-time regions persist across kernels and bandwidths? Which local peaks or shoulders change? Compare them with the rug; a peak does not automatically identify a population subgroup.
2. Start with a single observation: where does your custom kernel place its largest contributions, and how much density does it place near the observation itself? Explain how this behaviour carries over to the sum of contributions. Does it produce displaced peaks, extra shoulders, or irregularities you would hesitate to interpret? Connect any unhelpful features to the kernel's allocation of density rather than just its name.
3. Explain why using the same raw scale for the rectangular and custom kernels would compare different SDs. Report the SD and both raw scales in minutes for each panel.
4. Did the kernel "do its job poorly" in the way you predicted? Explain why or why not, and whether changing the bandwidth alters your conclusion. Can a nonnegative estimate with area one still give an unhelpful visual summary? Does finding an unhelpful result here prove the kernel is invalid or fails for every dataset?

Interpretation. A kernel satisfying the stated conditions is mathematically valid, even if you find its visual summary unhelpful. The waiting times already contain broad structure, so do not attribute their two main regions to the kernel alone. Kernel shape can add local irregularity, and its impact depends on bandwidth. Matching SD makes scale explicit; it does not make the kernels equivalent or select their respective optimal bandwidths.

References

See the linked readings above for Wilke and the R documentation.

- Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. Chapman & Hall. Further reading on KDE, bandwidth selection, and asymptotic risk.

The following records the software versions used when the document is knitted.

```
sessionInfo()
```

```
#> R version 4.6.1 (2026-06-24 ucrt)
#> Platform: x86_64-w64-mingw32/x64
#> Running under: Windows 11 x64 (build 26200)
#>
#> Matrix products: default
#>   LAPACK version 3.12.1
#>
#> locale:
#> [1] LC_COLLATE=Italian_Italy.utf8  LC_CTYPE=Italian_Italy.utf8
#> [3] LC_MONETARY=Italian_Italy.utf8 LC_NUMERIC=C
#> [5] LC_TIME=Italian_Italy.utf8
#>
#> time zone: Europe/Rome
#> tzcode source: internal
#>
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets  methods   base
#>
#> loaded via a namespace (and not attached):
#>  [1] compiler_4.6.1    fastmap_1.2.0     cli_3.6.6         tools_4.6.1
#>  [5] htmltools_0.5.9   otel_0.2.0        rstudioapi_0.19.0 yaml_2.3.12
#>  [9] tinytex_0.60      rmarkdown_2.31    knitr_1.51        xfun_0.60
#> [13] digest_0.6.39     rlang_1.3.0       evaluate_1.0.5
```