LOESS: nonparametric trends and span selection

A Course on Data Visualization

Livio Finos, University of Padova

Contents

The question we will investigate **2**

Learning goals . . . 2

What you need . . . 2

LOESS: estimate a trend from paired observations **3**

Start with a scatter plot and a global line . . . 3

One local fit at a target point . . . 3

Span is a fraction, not a distance . . . 5

How the LOESS span changes the trend **7**

Optional: degree is a second modelling choice . . . 8

Choose the span with prediction in mind **9**

Why training error is insufficient . . . 9

Robust LOESS: unusual responses can change the curve **12**

Boundaries, sparse regions, and extrapolation **14**

Edge fits have less balanced information . . . 14

Extrapolation is not a smoothing solution . . . 17

Explore and discuss **20**

Exercise 1 — Explain a local fit . . . 20

Exercise 2 — Select and diagnose a LOESS span . . . 20

Exercise 3 — Robustness and the target . . . 20

Exercise 4 — Critique these statements . . . 20

Take-home messages **20**

Sources and further reading **21**

```
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"
# 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)
}
```

The question we will investigate

How can we describe a relationship that bends, without imposing one global straight line?

You already know how a simple linear regression summarizes the association between two variables. LOESS extends that idea by fitting simple relationships locally and moving the neighbourhood across the data. We will investigate how much detail to retain, how to compare smoothing choices, and where a fitted trend becomes unreliable.

Use this unit during our discussion and return to its explanations and worked examples afterwards. Before changing a setting, predict what will happen to the curve. Then compare your prediction with the plot.

Learning goals

By the end of this unit, you should be able to:

- explain how distance weights produce a local straight-line fit;
- distinguish the span from a fixed distance in predictor units;
- compare spans using prediction on observations left out of the fit;
- investigate the effect of unusual responses and sparse regions;
- explain why an estimated trend does not justify extrapolation or a causal claim.

Prerequisites: scatterplots, mean and variance, simple least-squares regression, and basic R. Weighted fitting and cross-validation are introduced through examples. The local quadratic comparison is an optional extension; no multiple-regression course or statistical inference is required.

What you need

All examples use base R and the standard `stats` and `datasets` packages. Only `knitr` and `rmarkdown` are needed to render the document. There are no external data files or downloads. Each simulation has an explicit seed.

Open `DataViz.Rproj` and work from the course folder. You can knit this file to HTML and keep it as a study reference. PDF output also requires a LaTeX installation. The examples and exercises are teaching constructions unless explicitly described as real data.

LOESS: estimate a trend from paired observations

For ordinary local least squares, the mean-regression target is $m(x) = E(Y \mid X = x)$: the average response associated with a given predictor value. The vertical axis remains in response units. This is a trend estimate, not a probability density as in the KDE unit.

"Nonparametric" does not mean assumption-free. LOESS uses local regularity without imposing one global polynomial relationship.

Start with a scatter plot and a global line

We generate a nonlinear mean curve plus independent noise:

$$Y_i = m(X_i) + \varepsilon_i, \qquad m(x) = \sin(x) + 0.15x, \quad \varepsilon_i \sim N(0, 0.35^2).$$

```
set.seed(2101)
reg <- data.frame(x = seq(0, 10, length.out = 180))
true_mean <- function(z) sin(z) + .15 * z
reg$y <- true_mean(reg$x) + rnorm(nrow(reg), sd = .35)
xg <- seq(min(reg$x), max(reg$x), length.out = 401)
```

```
plot(reg$x, reg$y, pch = 16, col = adjustcolor("black", .4),
     xlab = "x", ylab = "Response y")
abline(lm(y ~ x, data = reg), col = orange, lwd = 2)
lines(xg, true_mean(xg), lty = 2, lwd = 2)
legend("topleft", c("Global least-squares line", "True mean"),
       col = c(orange, "black"), lty = c(1, 2), lwd = 2, bty = "n")
```

A global line uses all observations to estimate the same intercept and slope. LOESS lets the local fit change across the predictor range. It describes association, not a causal effect.

One local fit at a target point

At a target x_0, take a neighbourhood containing approximately a fraction α of the observations. This fraction is called the **span**: it is the `span` argument in `loess()`. Let $d(x_0)$ be its radius. Tricube distance weights are

$$w_i(x_0) = \begin{cases} \left[1 - \left|\frac{X_i - x_0}{d(x_0)}\right|^3\right]^3, & |X_i - x_0| < d(x_0), \\ 0, & \text{otherwise.} \end{cases}$$

For a local linear fit, minimise

$$\sum_i w_i(x_0)\{Y_i - \beta_0 - \beta_1(X_i - x_0)\}^2.$$

The prediction at x_0 is $\widehat{\beta}_0$. Move x_0, compute new weights, and fit again. A local quadratic fit adds $\beta_2(X_i - x_0)^2$.

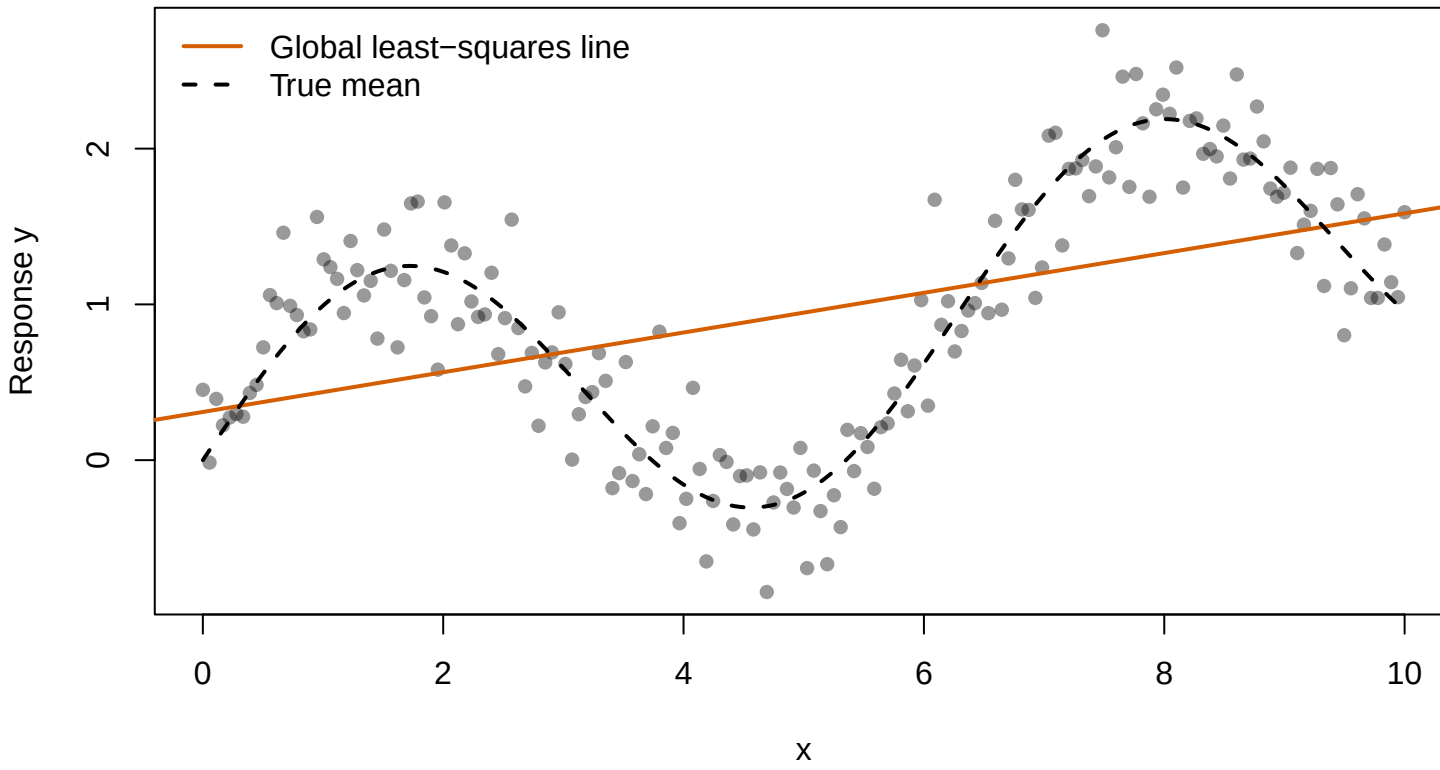

Figure 1: A global linear fit misses the systematic shape in these simulated observations.

```
x0 <- 4
alpha_demo <- .35
q <- ceiling(alpha_demo * nrow(reg))
distance <- abs(reg$x - x0)
radius <- sort(distance)[q]
w <- ifelse(distance < radius, (1 - (distance / radius)^3)^3, 0)
local_data <- transform(reg, centred_x = x - x0)
local_fit <- lm(y ~ centred_x, data = local_data, weights = w)
local_prediction <- unname(coef(local_fit)[1])
with_panels(1, 2, {
  plot(reg$x, w, type = "h", col = blue, lwd = 2,
       xlab = "x", ylab = "Distance weight", main = "Neighbourhood at x0 = 4")
  abline(v = x0, lty = 2)
  plot(reg$x, reg$y, pch = 16, col = "grey75", xlab = "x", ylab = "y")
  points(reg$x[w > 0], reg$y[w > 0], pch = 16,
         col = blue, cex = .5 + 1.2 * w[w > 0])
  local_grid <- seq(x0 - radius, x0 + radius, length.out = 100)
  lines(local_grid, predict(local_fit,
       newdata = data.frame(centred_x = local_grid - x0)), col = orange, lwd = 2)
  points(x0, local_prediction, pch = 23, bg = orange, cex = 1.6)
})
```

This is an explicit illustration of one local fit, not a byte-for-byte reimplementation of R's neighbourhood rounding or interpolation algorithm. Points exactly at the neighbourhood boundary have weight zero.

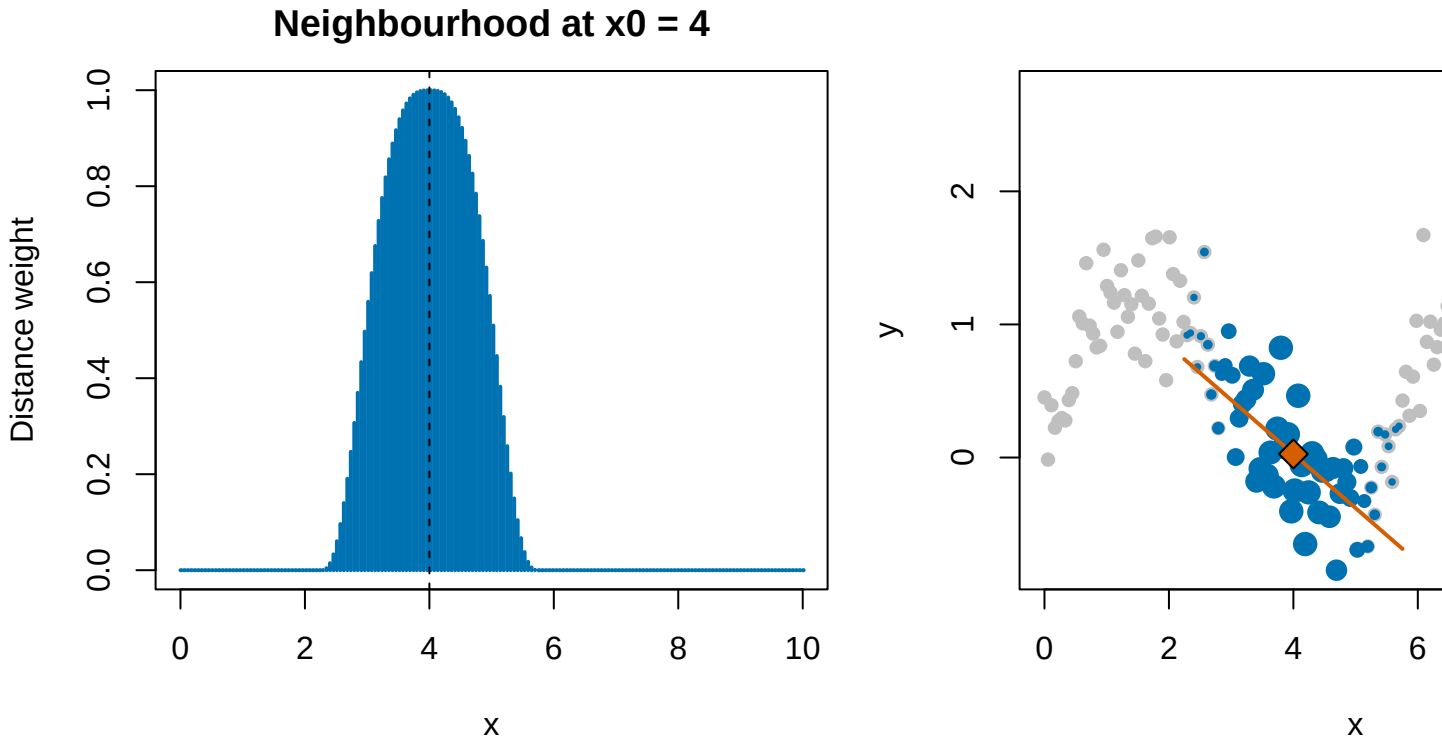

Figure 2: Left: tricube distance weights. Right: the local line, with its prediction at the target.

Span is a fraction, not a distance

For $0 < \alpha \leq 1$, the neighbourhood includes approximately αn observations. With uneven predictor spacing, the same span covers a narrow interval in a dense region and a wider interval in a sparse region. The span therefore does not specify a fixed distance in predictor units.

```
set.seed(2102)
x_uneven <- sort(c(runif(150, 0, 3), runif(35, 3, 10)))
targets <- c(1.5, 7)
a_local <- .3
radii <- vapply(targets, function(z)
  sort(abs(x_uneven - z))[ceiling(a_local * length(x_uneven))], numeric(1))
plot(x_uneven, rep(0, length(x_uneven)), pch = "|", ylim = c(-.2, 2.5),
     xlab = "x", ylab = "", yaxt = "n", main = "A fixed span adapts to data spacing")
for (j in seq_along(targets)) {
  segments(targets[j] - radii[j], j, targets[j] + radii[j], j,
           col = c(blue, orange)[j], lwd = 3)
  points(targets[j], j, pch = 16, col = c(blue, orange)[j])
}
axis(2, at = 1:2, labels = c("Dense region", "Sparse region"), las = 1, cex.axis = .8)
```

R also permits spans larger than one, with a modified distance scaling. For clarity, all span comparisons in this unit use values at most one.

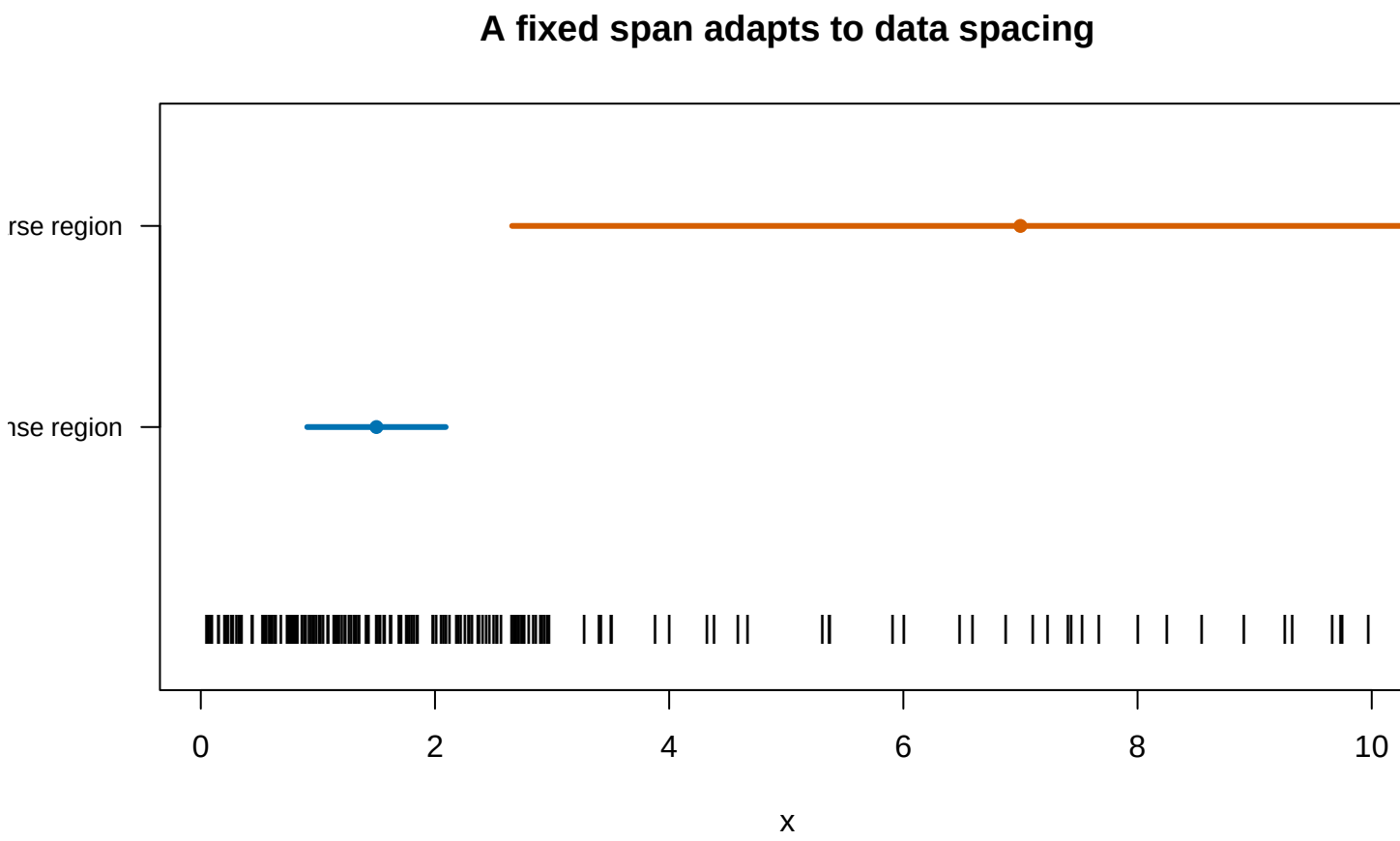

Figure 3: The same fraction of observations requires different physical neighbourhood widths.

How the LOESS span changes the trend

```
# Explicit settings prevent hidden changes in degree, family, or surface.
fit_loess <- function(dat, span, family = "gaussian", degree = 1) {
  loess(y ~ x, data = dat, span = span, degree = degree, family = family,
        control = loess.control(surface = "direct"),)
}
spans <- c(.12, .35, .85, 1,100)
fits <- lapply(spans, function(s) fit_loess(reg, s))
```

```
with_panels(2, length(spans)/2, {
  for (j in seq_along(spans)) {
    plot(reg$x, reg$y, pch = 16, cex = .6, col = adjustcolor("black", .3),
         xlab = "x", ylab = "y", main = paste("Span =", spans[j]))
    lines(xg, predict(fits[[j]], data.frame(x = xg)), col = blue, lwd = 2)
    lines(xg, true_mean(xg), lty = 2, lwd = 2)
  }
})
```

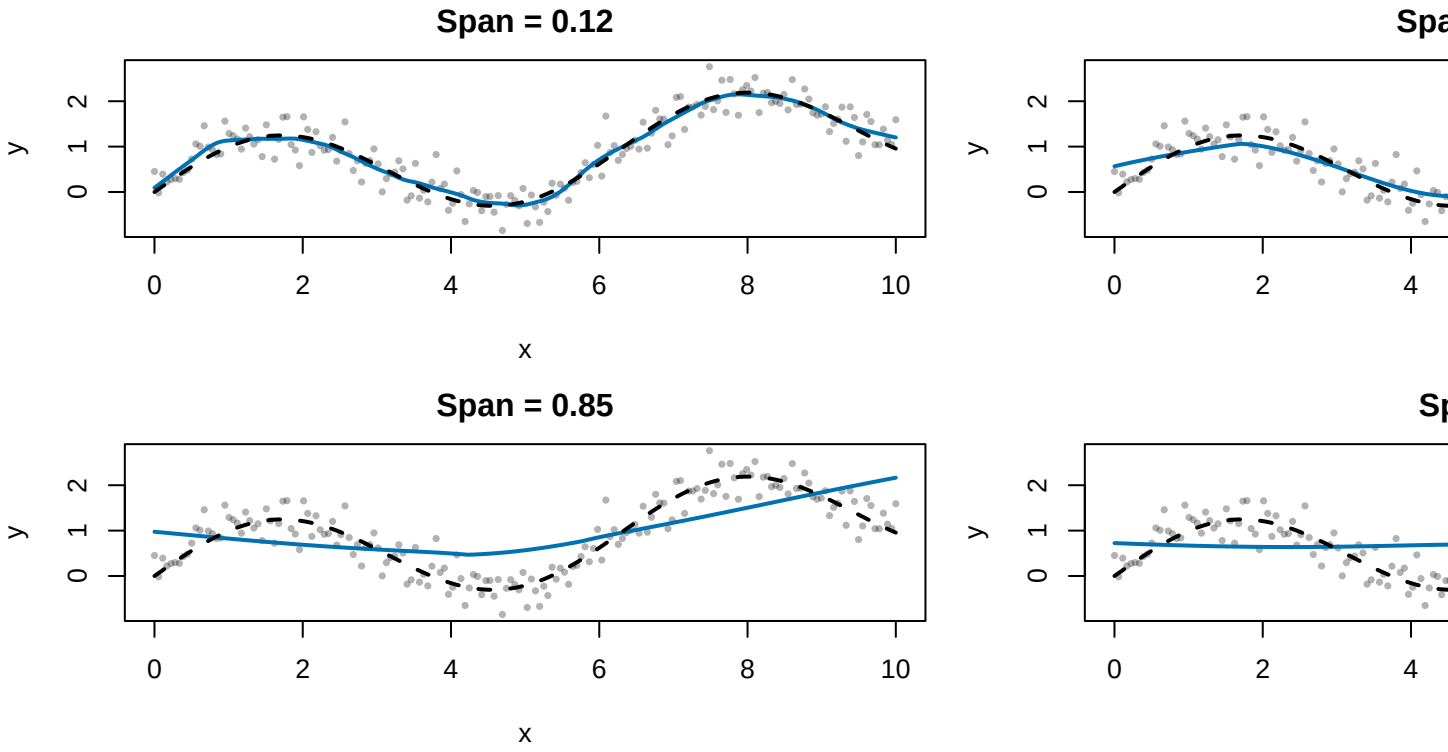

Figure 4: Local linear LOESS with three spans. Points and dashed true mean are identical across panels.

A smaller span generally increases flexibility and sampling variability. A larger span generally increases smoothing bias. Which span is useful depends on sample size, noise, curvature, and the intended task.

The examples use degree 1 to make the local-line explanation transparent. R's `loess()` default is degree 2 and span 0.75; the default is not selected by cross-validation. `family = "gaussian"` requests ordinary local least squares; normal errors are not required merely to compute that fit.

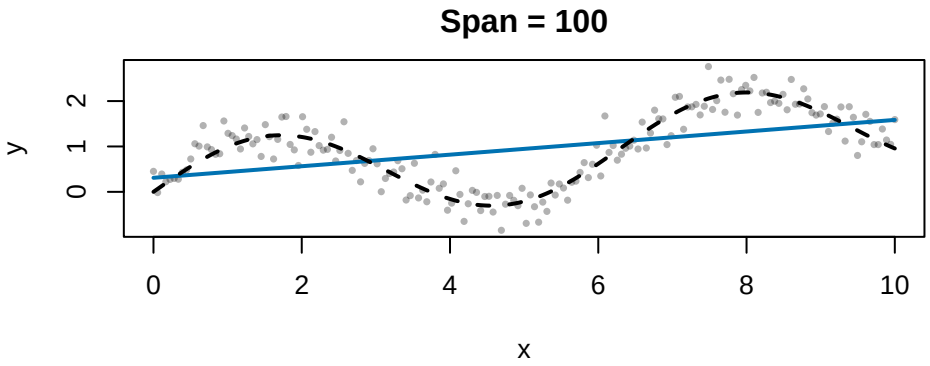

Figure 5: Local linear LOESS with three spans. Points and dashed true mean are identical across panels.

Optional: degree is a second modelling choice

```
fit_linear <- fit_loess(reg, .45, degree = 1)
fit_quadratic <- fit_loess(reg, .45, degree = 2)
plot(reg$x, reg$y, pch = 16, col = adjustcolor("black", .3),
     xlab = "x", ylab = "y")
lines(xg, predict(fit_linear, data.frame(x = xg)), col = blue, lwd = 2)
lines(xg, predict(fit_quadratic, data.frame(x = xg)), col = orange, lwd = 2, lty = 2)
legend("topleft", c("Degree 1", "Degree 2"), col = c(blue, orange),
       lty = c(1, 2), lwd = 2, bty = "n")
```

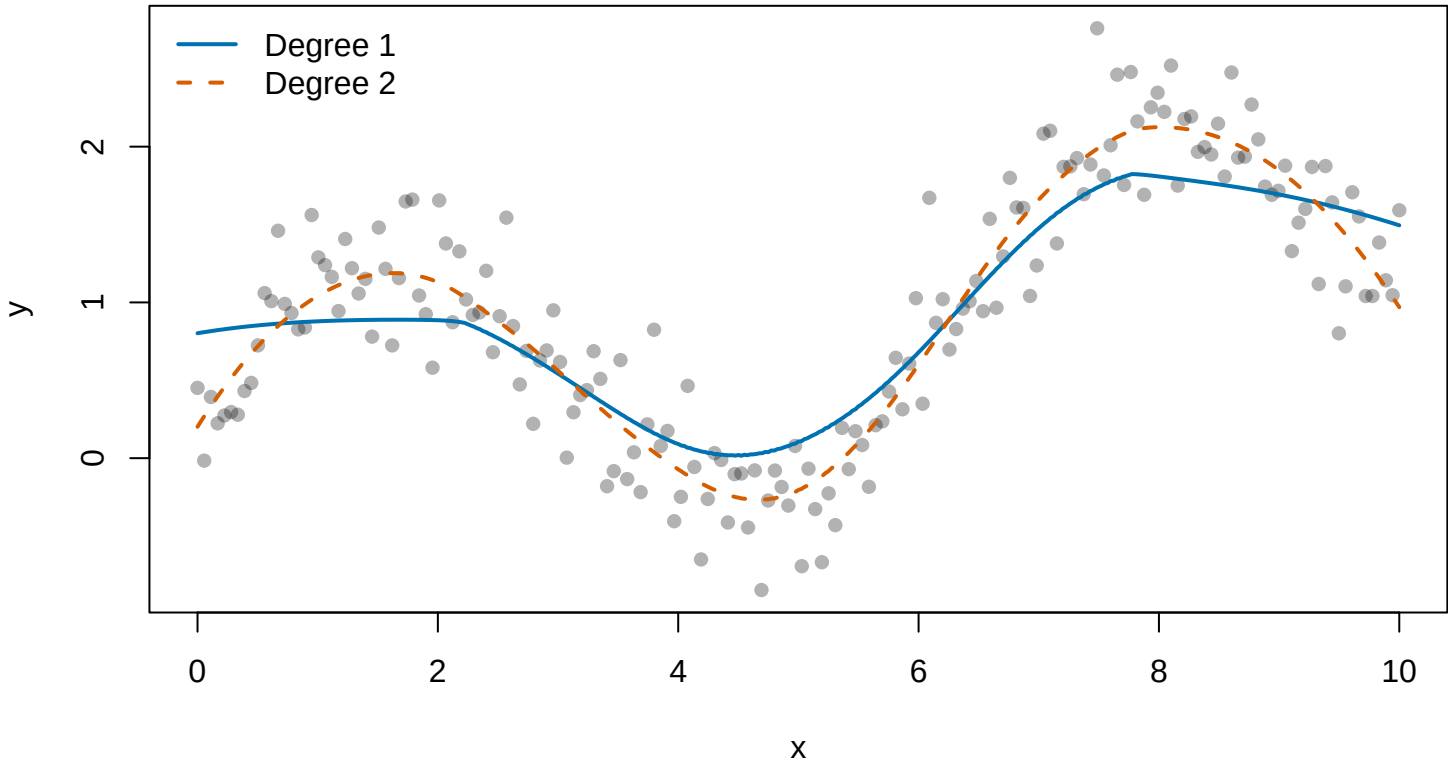

Figure 6: At a fixed span, local linear and local quadratic fits can differ, particularly near the edges.

A quadratic local fit may capture curvature better, but needs enough local information. A span of one still

uses target-dependent distance weights: it is not generally identical to an unweighted global polynomial fit.

Choose the span with prediction in mind

Why training error is insufficient

A flexible curve can follow the training observations closely, including their noise. This creates an unfair advantage when we assess it on those same observations: each Y_i helped determine the fitted value against which we compare it. A very small neighbourhood gives an individual response considerable influence on its own prediction. Following that response more closely does not necessarily help predict a new response nearby.

On a common evaluation set I, training error is

$$\text{Training}(\alpha) = \frac{1}{|I|} \sum_{i \in I} \{Y_i - \widehat{m}_\alpha(X_i)\}^2.$$

Here the fit uses **all** observations, including observation i. In leave-one-out cross-validation, we remove observation i, refit the entire smoother, and only then predict its response. This removes the direct contribution of Y_i to its own prediction. Both criteria measure error in predicting responses, including their noise; neither is the squared error against the noise-free mean curve.

We keep the original simulated observations and extend the candidate grid down to span 0.03. With 180 observations this means only about five neighbours, compared with about 27 at span 0.15. This deliberately includes curves flexible enough to follow short-scale noise. A grid starting at 0.15 can hide this part of the comparison: both criteria may choose its lower endpoint, even though they answer different questions.

For independent pairs and a fixed candidate span, leave-one-out cross-validation uses

$$\text{CV}(\alpha) = \frac{1}{|I|} \sum_{i \in I} \{Y_i - \widehat{m}_{\alpha,-i}(X_i)\}^2.$$

Here I excludes the two extreme predictor values. If either were left out, predicting it would require extrapolation beyond the remaining predictor range. Thus our criterion targets **interior prediction**, not boundary or out-of-range performance. All spans are compared on the same evaluation set.

```
span_grid <- c(.03, .04, .05, .075, .10, seq(.15, .75, by = .05))
interior <- 2:(nrow(reg) - 1)
cv_error <- training_error <- numeric(length(span_grid))
for (j in seq_along(span_grid)) {
  s <- span_grid[j]
  full <- fit_loess(reg, s)
  full_pred <- predict(full, reg[interior, , drop = FALSE])
  training_error[j] <- mean((reg$y[interior] - full_pred)^2)
  held_out <- vapply(interior, function(i) {
    fit_i <- fit_loess(reg[-i, ], s)
    as.numeric(predict(fit_i, reg[i, , drop = FALSE]))
  }, numeric(1))
  # Do not silently compare scores computed on different subsets.
  stopifnot(all(is.finite(held_out)))
  cv_error[j] <- mean((reg$y[interior] - held_out)^2)
}
training_span <- span_grid[which.min(training_error)]
```

```r
best_span <- span_grid[which.min(cv_error)]
knitr::kable(data.frame(span = span_grid, training_MSE = training_error,
                        interior_LOOCV_MSE = cv_error), digits = 4)
```

span	training_MSE	interior_LOOCV_MSE
0.030	0.0425	0.1210
0.040	0.0614	0.1154
0.050	0.0706	0.1151
0.075	0.0773	0.1018
0.100	0.0788	0.0966
0.150	0.0805	0.0924
0.200	0.0840	0.0930
0.250	0.0891	0.0970
0.300	0.0974	0.1046
0.350	0.1078	0.1150
0.400	0.1249	0.1325
0.450	0.1455	0.1537
0.500	0.1729	0.1820
0.550	0.2025	0.2125
0.600	0.2358	0.2468
0.650	0.2649	0.2765
0.700	0.2919	0.3038
0.750	0.3114	0.3234

```r
with_panels(1, 2, {
  for (upper in c(max(span_grid), .30)) {
    keep <- span_grid <= upper
    matplot(span_grid[keep], cbind(training_error, cv_error)[keep, ],
            type = "b", pch = c(1, 16), lty = c(2, 1),
            col = c("grey40", blue), xlab = "Span", ylab = "Mean squared error",
            main = if (upper > .30) "Full grid" else "Small-span detail")
    abline(v = training_span, col = "grey40", lty = 2)
    abline(v = best_span, col = blue, lty = 3)
    points(training_span, min(training_error), pch = 1, cex = 1.8, lwd = 2)
    points(best_span, min(cv_error), pch = 16, cex = 1.5, col = blue)
    legend("topleft", c("Training", "Interior LOOCV"),
           col = c("grey40", blue), lty = c(2, 1), pch = c(1, 16), bty = "n")
  }
})
```

On this grid, the training minimum is at span **0.03**, whereas the interior LOOCV minimum is at span **0.15**. Read the right-hand panel from left to right: initially, allowing more neighbours sacrifices some agreement with the observed responses but improves held-out prediction. At sufficiently large spans, smoothing away the underlying curvature increases both errors. The full-grid panel alone makes the differences among small spans harder to see.

The smallest training error is not a reliable selection rule. Training error need not be strictly monotone in span because these fits are not a sequence of nested least-squares models. The CV minimum is also noisy: similar scores do not support a precise claim that one span is uniquely best.

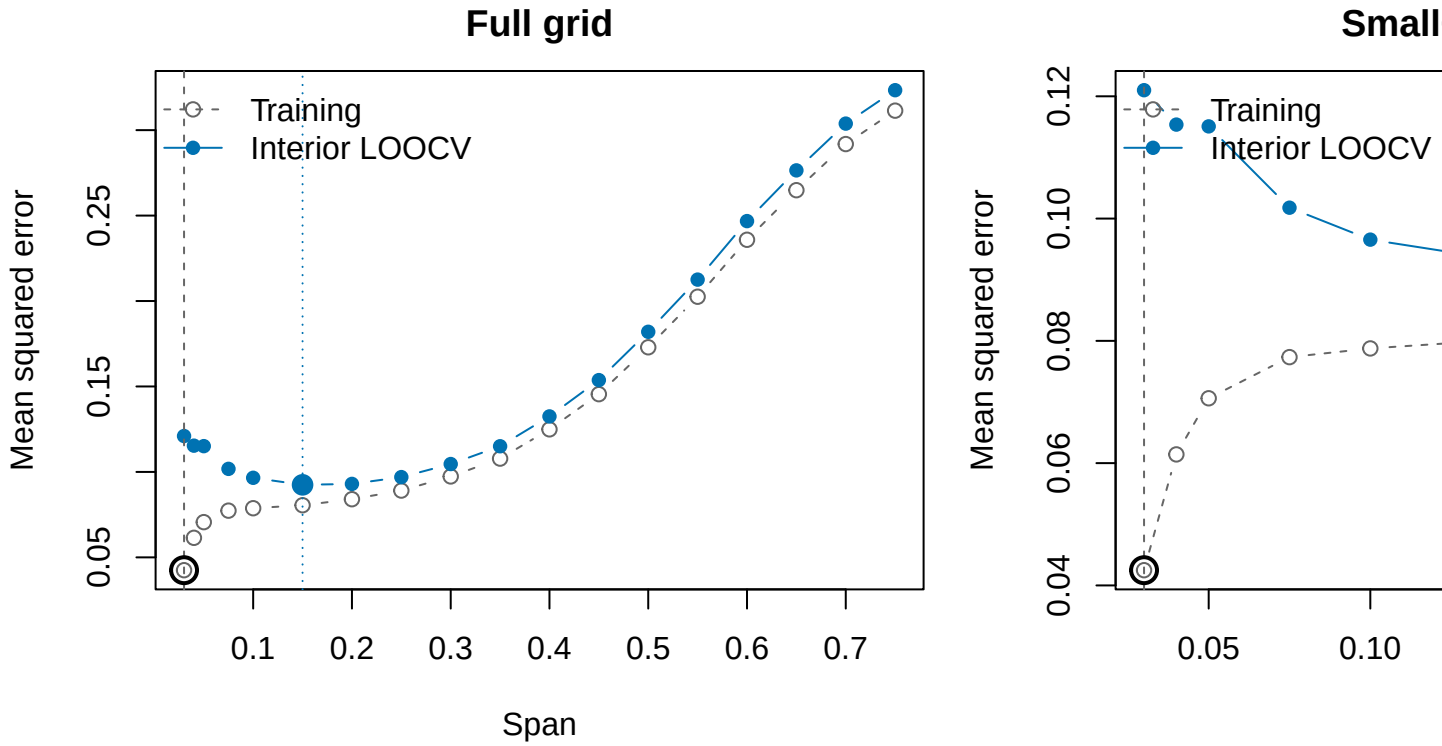

Figure 7: Left: the full candidate grid. Right: a close-up of small spans, with each criterion's minimum marked.

```
best_fit <- fit_loess(reg, best_span)
selection_spans <- c(training_span, best_span)
with_panels(1, 2, {
  for (j in seq_along(selection_spans)) {
    selected_fit <- fit_loess(reg, selection_spans[j])
    plot(reg$x, reg$y, pch = 16, col = adjustcolor("black", .35),
         xlab = "x", ylab = "y",
         main = paste(c("Training choice:", "LOOCV choice:")[j], selection_spans[j]))
    lines(xg, predict(selected_fit, data.frame(x = xg)), col = blue, lwd = 2)
    lines(xg, true_mean(xg), lty = 2, lwd = 2)
    legend("topleft", c("Fitted curve", "True mean"),
           col = c(blue, "black"), lty = c(1, 2), bty = "n")
  }
})
```

Notice which short-scale fluctuations disappear in the second panel. We can compare with the true mean only because these data are simulated; that curve was not used to select either span.

Selection and assessment should be distinguished. The minimum of the CV scores used to select a span is an optimistic performance summary for the selected procedure. An independent test set or outer validation loop is needed for an independent assessment. For exploratory teaching plots, report the selection method and inspect nearby spans.

For time series or clustered observations, random folds or ordinary leave-one-out can leak dependent information. Use a validation design reflecting the prediction task, such as forward evaluation for forecasting or leaving entire subjects out for new-subject prediction. A smooth time curve does not account for serial dependence by itself.

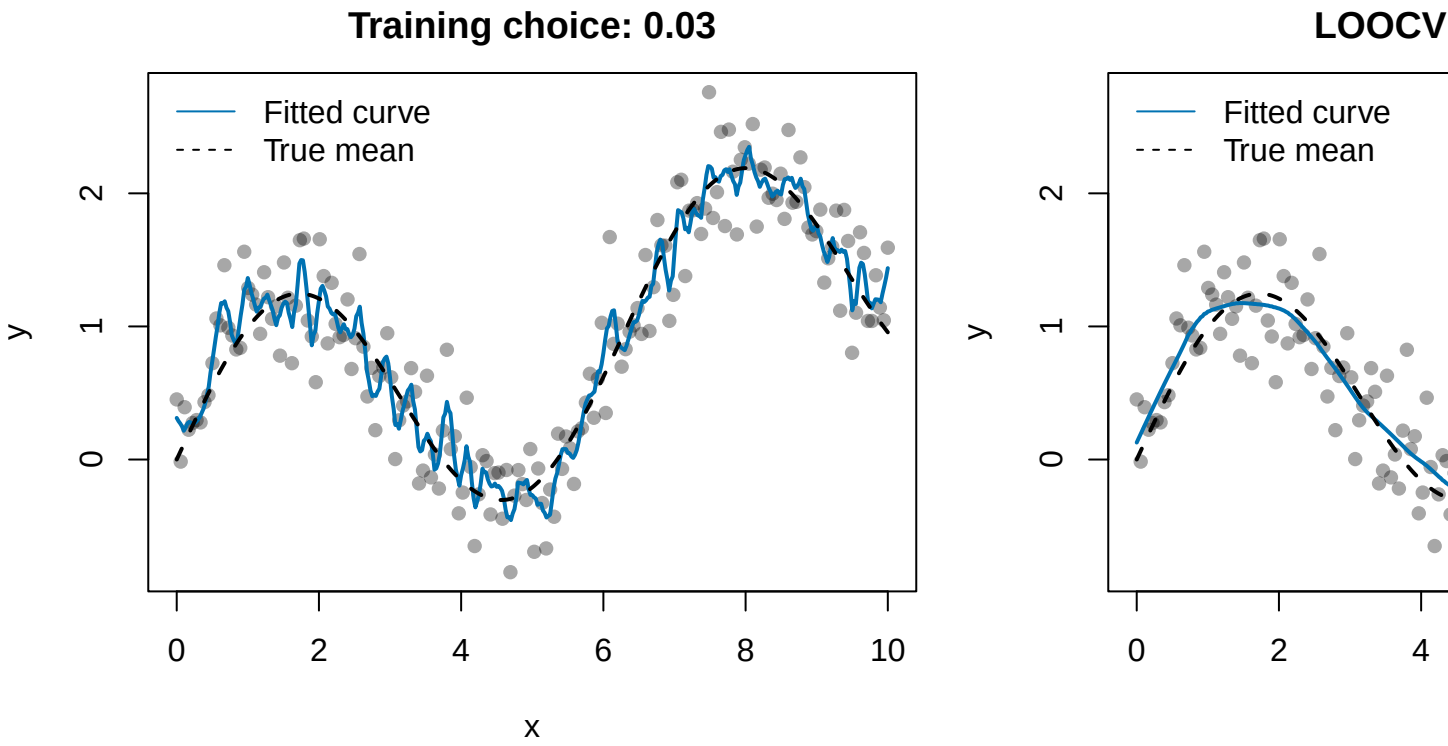

Figure 8: The same observations and true mean, with spans chosen by training error and by interior LOOCV.

Robust LOESS: unusual responses can change the curve

A response outlier can pull a local least-squares fit toward it. Robust LOESS iteratively adds residual-based weights. These are different from the distance weights: a nearby point can receive low robustness weight if its residual is large.

Start with the ordinary local least-squares curve $\widehat{m}^{(0)}$. After iteration t, define the residuals and their scale by

$$r_i^{(t)} = Y_i - \widehat{m}^{(t)}(X_i), \qquad s_t = \text{median}_{i=1,\dots,n}\, |r_i^{(t)}|.$$

For $s_t > 0$, the **Tukey bisquare robustness weights** for the next fit are

$$b_i^{(t+1)} = \begin{cases} \left[1 - \left(\dfrac{r_i^{(t)}}{6s_t}\right)^2\right]^2, & |r_i^{(t)}| < 6s_t, \\ 0, & |r_i^{(t)}| \geq 6s_t. \end{cases}$$

Thus a zero residual receives weight 1, a residual of magnitude $3s_t$ receives weight $9/16$, and one of magnitude at least $6s_t$ receives weight 0. Here s_t is the **median absolute residual**, not R's default `mad()` (which centres its input and includes a scaling constant). The zero-scale case needs a numerical safeguard and is not covered by the division above.

At each target x_0, refit the local line by minimising

$$\sum_i \underbrace{w_i(x_0)}_{\text{predictor distance}} \underbrace{b_i^{(t+1)}}_{\text{response residual}} \{Y_i - \beta_0 - \beta_1(X_i - x_0)\}^2.$$

The distance weights change with the target; the robustness weight for an observation is computed from its residual in the current whole curve and reused across targets. Recompute residuals after refitting and repeat a small number of times. This is the robust local-fitting scheme described in the LOESS reference manual; `family = "symmetric"` in R uses Tukey's biweight iterations. The starting fit can itself be affected by contamination, so robustness is not a guarantee of recovering the underlying trend.

```
contaminated <- reg
bad <- which.min(abs(reg$x - 5)) + (-1:1)
contaminated$y[bad] <- contaminated$y[bad] + 5
ordinary_fit <- fit_loess(contaminated, .3, family = "gaussian")
robust_fit <- fit_loess(contaminated, .3, family = "symmetric")
plot(contaminated$x, contaminated$y, pch = 16, col = "grey65",
     xlab = "x", ylab = "y")
points(contaminated$x[bad], contaminated$y[bad], pch = 4, col = orange, lwd = 2)
lines(xg, predict(ordinary_fit, data.frame(x = xg)), col = blue, lwd = 2)
lines(xg, predict(robust_fit, data.frame(x = xg)), col = green, lwd = 2, lty = 2)
lines(xg, true_mean(xg), col = "black", lty = 3)
legend("topleft", c("Local least squares", "Robust LOESS", "True mean"),
       col = c(blue, green, "black"), lty = c(1, 2, 3), lwd = 2, bty = "n")
```

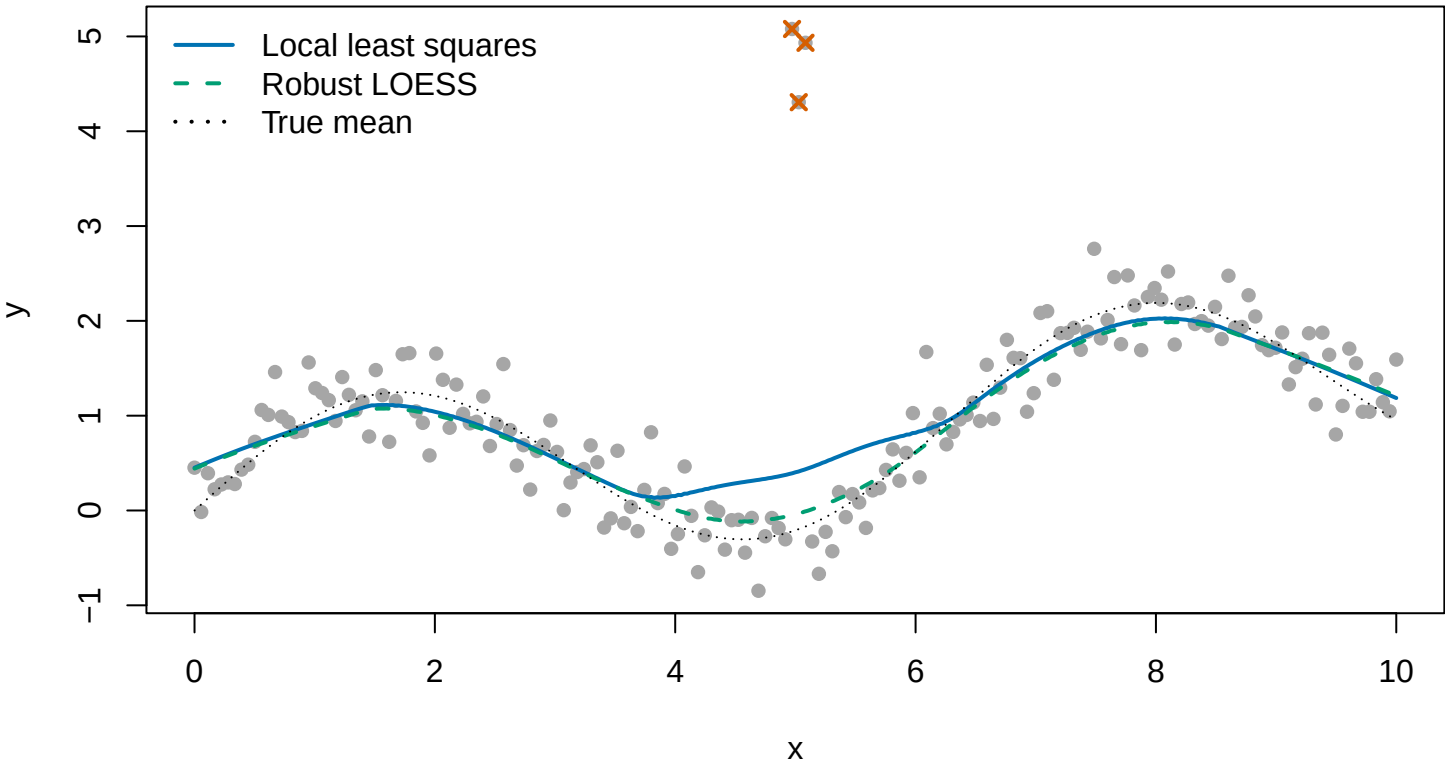

Figure 9: Robustness reduces the influence of a small group of contaminated responses in this example.

The simulation tells us which observations were contaminated; real data do not. A cluster of unusual values may represent a real event or a distinct group. Robust fitting is a sensitivity analysis, not permission to discard inconvenient observations. Under asymmetric response distributions, a robust conditional location need not equal the conditional mean.

Boundaries, sparse regions, and extrapolation

Edge fits have less balanced information

Near the edge of the predictor range, neighbours lie mostly on one side of the target. To see why this matters, compare a local weighted average with a local straight line, using exactly the same distance weights.

Write $u_i = X_i - x_0$ and $S_k = \sum_i w_i(x_0) u_i^k$, for $k = 0, 1, 2$. The local constant estimate is

$$\widehat{m}_{\text{constant}}(x_0) = \frac{\sum_i w_i(x_0) Y_i}{S_0}.$$

Suppose, for a moment, that the responses follow an exact straight line $Y_i = a + bX_i$. Then

$$\widehat{m}_{\text{constant}}(x_0) - (a + bx_0) = b\frac{S_1}{S_0}.$$

The ratio S_1/S_0 is the displacement of the weighted mean predictor from the target. With balanced neighbours it is close to zero. At the left edge it is positive, so a local average overestimates an increasing straight-line trend. This is a geometric effect even without response noise.

The local linear estimate corrects this first-order imbalance. Solving the weighted simple-regression equations gives, when $D = S_0 S_2 - S_1^2 > 0$,

$$\widehat{m}_{\text{linear}}(x_0) = \sum_i \ell_i(x_0) Y_i, \qquad \ell_i(x_0) = \frac{w_i(x_0)(S_2 - S_1 u_i)}{D}.$$

These **equivalent weights** satisfy

$$\sum_i \ell_i(x_0) = 1, \qquad \sum_i \ell_i(x_0) u_i = 0.$$

Consequently the local line reproduces $a + bx_0$ exactly, including at an edge. There is a price: at the left endpoint, all $u_i \geq 0$, so balancing their weighted sum generally requires some **negative equivalent weights**. The original distance weights in the least-squares objective remain nonnegative. Equivalent weights describe how the final prediction combines the responses; they are not probabilities.

A comparison on actual data. The built-in `datasets::cars` data contain 50 pairs of speed (mph) and stopping distance (ft). These are historical measurements, not a time series; see `?cars` for their source. Compare speed 4 (the left endpoint) with speed 15 (an interior target), holding span 0.7 fixed. The function below implements the displayed equations explicitly, with the same neighbourhood convention as our earlier local-fit illustration.

```
local_weights <- function(x, x0, span) {
  u <- x - x0
  radius <- sort(abs(u))[ceiling(span * length(x))]
  stopifnot(radius > 0)
  w <- pmax(0, 1 - (abs(u) / radius)^3)^3
  S0 <- sum(w); S1 <- sum(w * u); S2 <- sum(w * u^2)
  D <- S0 * S2 - S1^2
  stopifnot(D > 0)
  ell <- w * (S2 - S1 * u) / D
  stopifnot(abs(sum(ell) - 1) < 1e-8, abs(sum(ell * u)) < 1e-8)
  list(w = w, average = w / S0, ell = ell, displacement = S1 / S0)
```

```
}
edge_targets <- c(4, 15)
edge_weights <- lapply(edge_targets, function(z) local_weights(cars$speed, z, .7))
```

```
with_panels(2, 2, {
  for (j in seq_along(edge_targets)) {
    z <- edge_targets[j]; ww <- edge_weights[[j]]
    centre <- cars$speed - z
    local_line <- lm(dist ~ centre, data = cars, weights = ww$w)
    active <- ww$w > 0
    local_x <- seq(min(cars$speed[active]), max(cars$speed[active]), length.out = 100)
    avg <- sum(ww$average * cars$dist)
    plot(cars$speed, cars$dist, pch = 16, col = "grey75",
         xlab = "Speed (mph)", ylab = "Stopping distance (ft)",
         main = paste("Target speed =", z))
    points(cars$speed[active], cars$dist[active], pch = 16, col = "grey40")
    segments(min(local_x), avg, max(local_x), avg, col = orange, lwd = 2)
    lines(local_x, predict(local_line, data.frame(centre = local_x - z)),
          col = blue, lwd = 2)
    abline(v = z, lty = 3)
    points(c(z, z), c(avg, sum(ww$ell * cars$dist)),
           pch = c(17, 16), col = c(orange, blue), cex = 1.4)
    legend("topleft", c("Local average", "Local line"),
           col = c(orange, blue), lty = 1, bty = "n", cex = .8)
  }
  weight_ylim <- range(unlist(lapply(edge_weights, function(w) c(w$average, w$ell))))
  for (j in seq_along(edge_targets)) {
    ww <- edge_weights[[j]]
    plot(cars$speed, ww$ell, type = "h", col = blue, ylim = weight_ylim,
         xlab = "Speed (mph)", ylab = "Weight in the prediction",
         main = paste("Weights at speed", edge_targets[j]))
    points(cars$speed, ww$average, pch = 17, col = orange, cex = .7)
    abline(h = 0, col = "grey60"); abline(v = edge_targets[j], lty = 3)
  }
})
```

```
edge_summary <- do.call(rbind, lapply(seq_along(edge_targets), function(j) {
  ww <- edge_weights[[j]]
  # Change just one response at the speed nearest the target by +10 ft.
  # With fixed span and nonrobust weights, its exact effect is 10 * ell[i].
  i <- which.min(abs(cars$speed - edge_targets[j]))
  data.frame(target = edge_targets[j], weighted_x_shift = ww$displacement,
             local_average = sum(ww$average * cars$dist),
             local_line = sum(ww$ell * cars$dist),
             negative_weights = sum(ww$ell < -1e-10),
             prediction_change_for_10ft_shift = 10 * ww$ell[i])
}))
knitr::kable(edge_summary, digits = 3,
             col.names = c("Target speed", "Weighted x shift", "Local average",
                           "Local line", "Negative weights", "Change after +10 ft"))
```

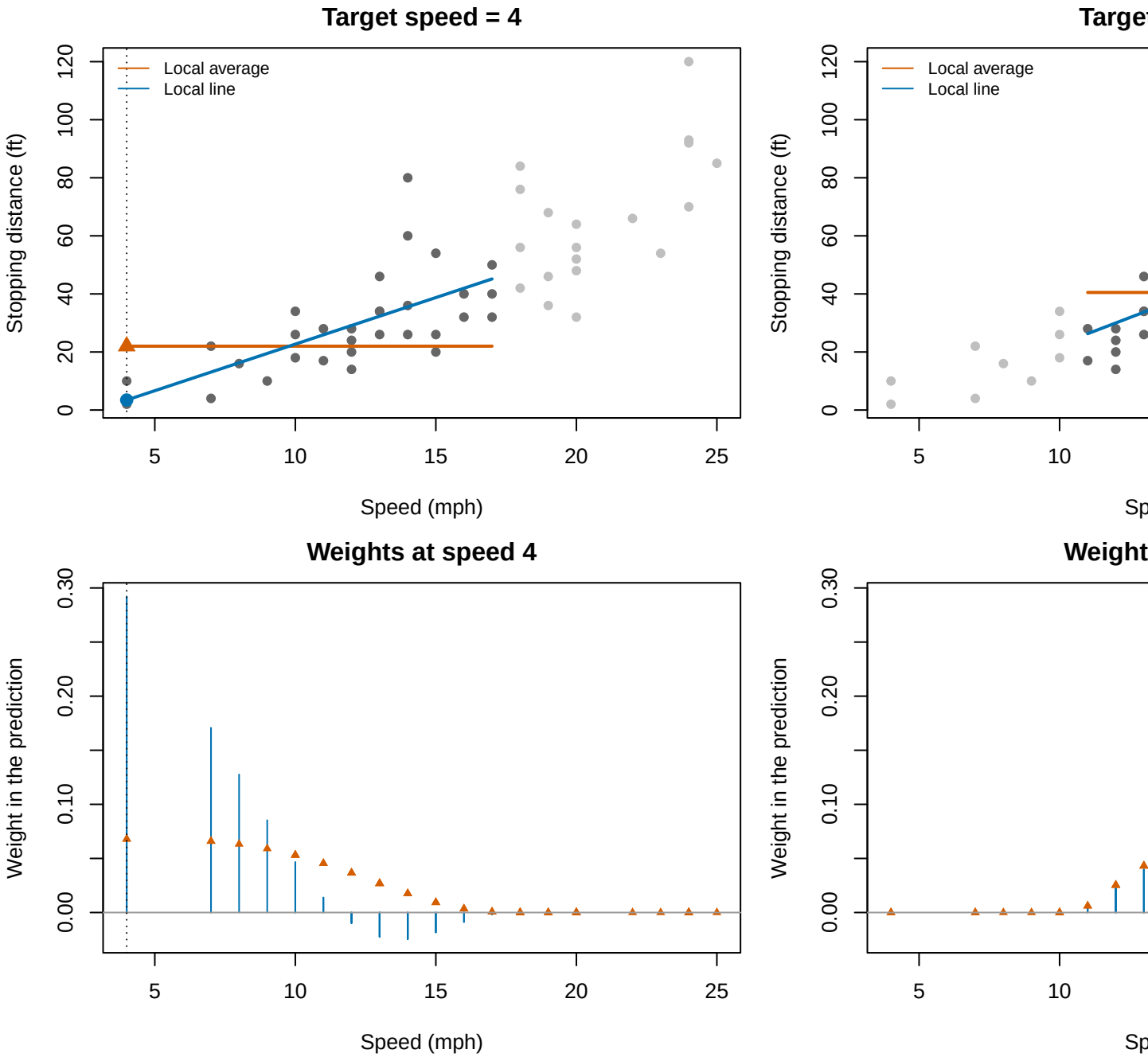

Figure 10: Top: the same observations, local average (orange) and local line (blue), evaluated at each target. Bottom: normalized averaging weights and equivalent local-linear weights. Repeated speeds have overlapping weights.

Target speed	Weighted x shift	Local average	Local line	Negative weights	Change after +10 ft
4	5.781	21.990	3.444	20	2.921
15	-0.164	40.495	41.103	0	0.529

At the left endpoint, the local average is **21.99 ft**, while the local line predicts **3.44 ft**. At the interior target, the corresponding predictions are **40.49 ft** and **41.1 ft**. The weighted predictor displacement is measured in mph, and the predictions and response changes in ft.

Read the left column of plots first. The average pools larger speeds and is pulled upward relative to the local line at speed 4. The negative blue weights show how the local line compensates for the one-sided geometry. At speed 15, compare the smaller predictor displacement and the more balanced weight pattern. The table's last column is a sensitivity experiment: how much would the prediction change if just one nearby stopping distance increased by 10 ft? It holds predictor values and weights fixed and does not involve a significance test.

We do not know the true mean stopping distance in these data, so this comparison does not prove which estimate is closer to it. The exact-line calculation establishes the first-order correction; the data show its mechanism and sensitivity. Curvature and noise remain, and a local line cannot replace the missing observations on the other side. Small spans, few distinct predictor values, or higher polynomial degree can further destabilize an edge fit. LOESS also does not impose nonnegative distances or monotonicity.

Extrapolation is not a smoothing solution

Example 1: a numerical result depends on the fitting option. Keep the same `cars` data, span and degree. Change only `surface`: the default interpolation surface returns `NA` outside the observed speed range, whereas direct evaluation can produce a number there. Check the table before looking at the curves.

```
car_fit <- loess(dist ~ speed, data = cars, degree = 1, span = .7,
                 control = loess.control(surface = "interpolate"))
car_direct <- loess(dist ~ speed, data = cars, degree = 1, span = .7,
                    control = loess.control(surface = "direct"))
speed_limits <- range(cars$speed)
outside <- data.frame(speed = c(0, speed_limits[1], 15, speed_limits[2], 30))
knitr::kable(data.frame(outside,
                        within_observed_range = outside$speed >= speed_limits[1] &
                                                outside$speed <= speed_limits[2],
                        interpolation_surface = as.numeric(predict(car_fit, outside)),
                        direct_surface = as.numeric(predict(car_direct, outside))),
             digits = 2)
```

speed	within_observed_range	interpolation_surface	direct_surface
0	FALSE	NA	-8.43
4	TRUE	3.46	3.44
15	TRUE	41.10	41.10
25	TRUE	89.16	89.13
30	FALSE	NA	121.66

```
car_grid <- data.frame(speed = seq(0, 30, length.out = 301))
car_predictions <- list(predict(car_fit, car_grid), predict(car_direct, car_grid))
in_range <- car_grid$speed >= speed_limits[1] & car_grid$speed <= speed_limits[2]
```

```r
car_ylim <- range(cars$dist, unlist(car_predictions), na.rm = TRUE)
with_panels(1, 2, {
  for (j in 1:2) {
    plot(cars$speed, cars$dist, xlim = c(0, 30), ylim = car_ylim,
         pch = 16, col = "grey60", xlab = "Speed (mph)", ylab = "Stopping distance (ft)",
         main = c("Default: no outside prediction", "Direct: numbers outside too")[j])
    abline(v = speed_limits, lty = 3, col = "grey40")
    lines(car_grid$speed, ifelse(in_range, car_predictions[[j]], NA), col = blue, lwd = 2)
    lines(car_grid$speed, ifelse(!in_range, car_predictions[[j]], NA),
          col = orange, lty = 2, lwd = 2)
  }
})
```

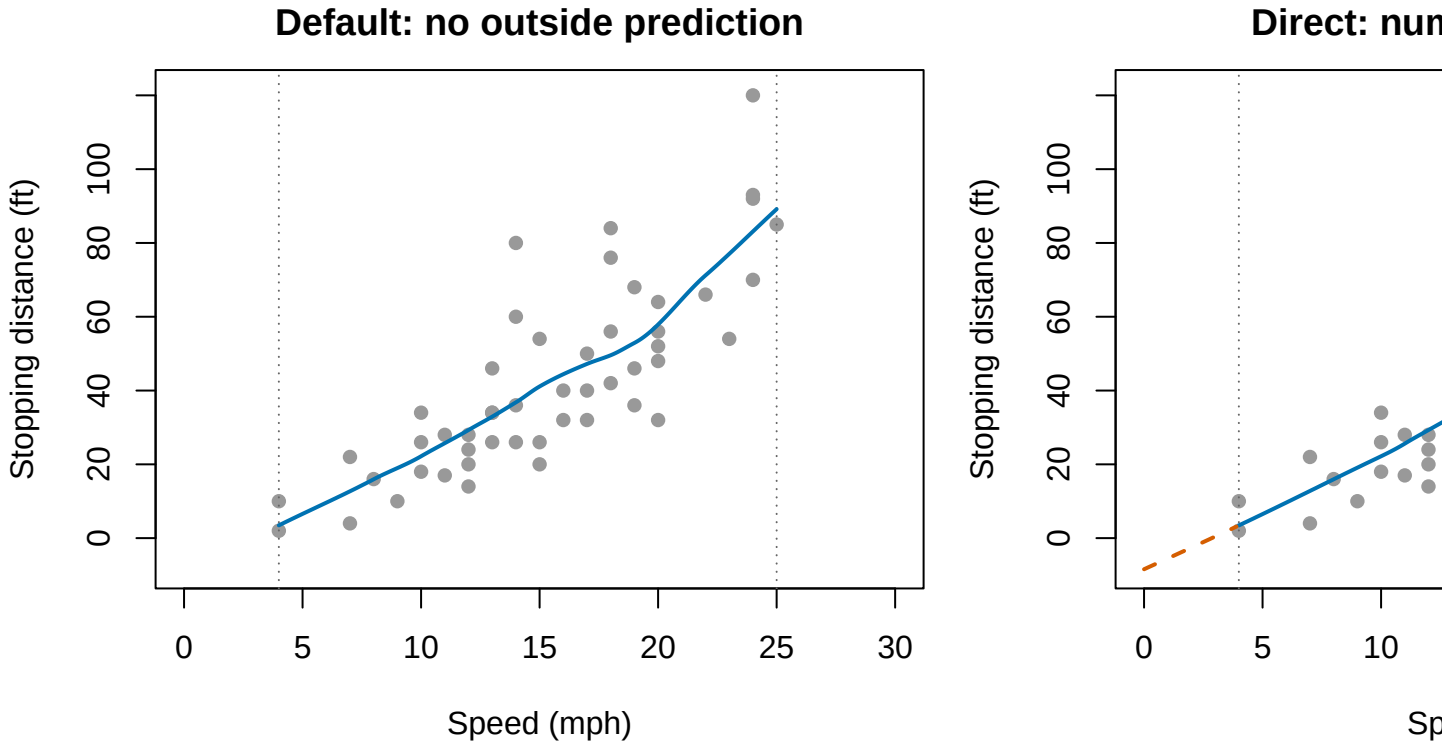

Figure 11: Identical data and smoothing settings; different surface options. Dashed orange portions are direct evaluations outside the observed speed range, not observed evidence.

The dashed portions have no observations at those speeds. Direct evaluation recomputes a local polynomial at an unsupported target; it is not necessarily a straight extension of the last plotted segment. Differences between the two curves *inside* the range can also arise from the interpolation approximation. Neither option adds information outside the range.

Example 2: the observed data cannot choose between two continuations. Return to our simulated observations, all with $0 \le X \le 10$. Consider two possible mean functions:

$$m_A(x) = \sin(x) + 0.15x, \qquad m_B(x) = m_A(x) + 0.25\{\max(x - 10, 0)\}^2.$$

They coincide throughout the observed range and even have the same slope at 10, but separate to its right. With the same noise distribution, either mean function generates exactly the same distribution of our observed data. No span selection within $[0, 10]$ can tell us which continuation applies.

```r
future_x <- seq(7, 14, length.out = 281)
mean_A <- true_mean(future_x)
mean_B <- mean_A + .25 * pmax(future_x - 10, 0)^2
continuation_fit <- fit_loess(reg, .30)
loess_future <- as.numeric(predict(continuation_fit, data.frame(x = future_x)))
visible <- reg$x >= 7
plot(reg$x[visible], reg$y[visible], xlim = c(7, 14),
     ylim = range(reg$y[visible], mean_A, mean_B, loess_future),
     pch = 16, col = "grey60", xlab = "x", ylab = "Response / mean",
     main = "Same observations, different possible futures")
abline(v = 10, lty = 3)
lines(future_x, mean_A, col = "black", lwd = 2)
lines(future_x, mean_B, col = green, lwd = 2, lty = 2)
lines(future_x, ifelse(future_x <= 10, loess_future, NA), col = blue, lwd = 2)
lines(future_x, ifelse(future_x >= 10, loess_future, NA), col = orange, lwd = 2, lty = 2)
legend("topleft", c("Mean A", "Mean B", "LOESS within range", "LOESS outside range"),
       col = c("black", green, blue, orange), lty = c(1, 2, 1, 2), lwd = 2,
       bty = "n", cex = .8)
```

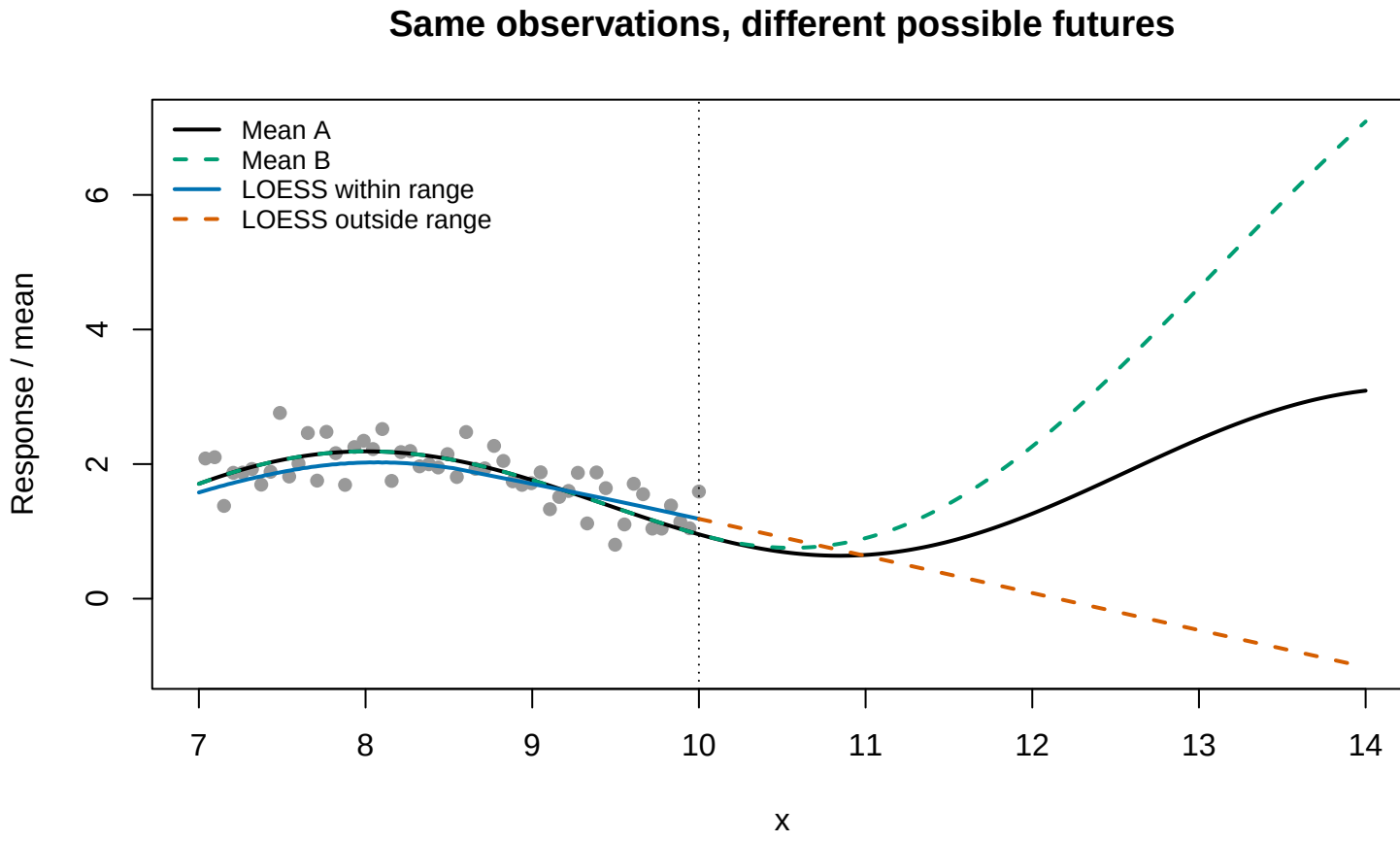

Figure 12: Two mean functions agree on every observed predictor value but diverge beyond 10. The LOESS extrapolation is a third continuation, with no data to choose among them.

Although we know mean A was used in this simulation, a learner seeing only the observations cannot distinguish these two worlds. A smoother's preferred continuation is an assumption, not evidence about unobserved predictor values. Extrapolation requires additional assumptions, subject knowledge, or new data in the region of interest.

Explore and discuss

Use the worked examples above to investigate these questions. For each exercise, show the relevant plot or comparison and explain the result in your own words. Keep the same observations when comparing spans or fitting methods.

Exercise 1 — Explain a local fit

Repeat the one-local-fit example with $x_0 = 0.5$ and $x_0 = 8$. What changes in the weights and fitted line? Compare spans 0.2 and 0.6. Why is the intercept of the centred local regression the prediction at x_0?

Exercise 2 — Select and diagnose a LOESS span

Generate 150 independent observations with $X \sim U(0, 1)$ and $Y = 4(X - 0.5)^2 + \varepsilon$, where $\varepsilon \sim N(0, 0.15^2)$. Compare local linear fits with spans 0.2, 0.4, 0.6, and 0.8. Choose a span by cross-validation, explicitly documenting how you handle extreme predictor values. Compare the selected curve with the known mean. Why might degree 2 with a large span work particularly well in this example?

Exercise 3 — Robustness and the target

Add three large positive response shifts near $x = 0.5$ in Exercise 2. Compare ordinary and robust LOESS at the same span. State what additional information would help determine whether those observations are errors, a distinct group, or a genuine local feature.

Exercise 4 — Critique these statements

1. "A LOESS span of 0.5 always means a window of width 0.5."
2. "The smallest training error identifies the best span."
3. "If software returns a LOESS prediction beyond the observed predictor range, that prediction is supported by the data."
4. "The plotted trend proves the effect is causal."

Take-home messages

- LOESS builds a smooth trend from local weighted fits. The familiar straight-line model is fitted repeatedly, with the neighbourhood centred at different predictor values.
- A span specifies a fraction of observations. Its width in predictor units depends on the spacing of the data.
- A smaller span follows finer variation, including noise; a larger span can remove features of interest. Keep the data fixed when comparing them.
- Training error and held-out prediction error answer different questions. State how you selected the span and which observations were used to evaluate it.
- Robustness weights and distance weights have different roles. Unusual responses require investigation and context, not automatic deletion.
- Sparse and boundary regions contain less information. A numerical prediction outside the observed range does not create empirical support there.
- A fitted association does not establish a causal effect. Keep the observations visible and state the limits of the interpretation.

Sources and further reading

- Wilke, Chapter 14: Visualizing trends.
- NIST LOESS introduction.
- R documentation: loess, predict.loess, and the built-in dataset documentation `?cars`.
- Cleveland, Grosse and Shyu, LOESS reference manual, for the robust local-fitting algorithm.
- Cleveland, W. S. (1979). Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74, 829–836.

Reproducibility information

The source file was structurally checked when prepared, but its R chunks were not executed in the preparation environment because R was unavailable. When you knit it, the following records your actual runtime and package versions.

```
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
```

Note on AI assistance

These teaching materials were prepared with assistance from ChatGPT (OpenAI) in drafting explanations, developing examples, and generating R code. The author retains responsibility for the final content.