Package 'tiltdens'

September 28, 2026

Type Package

Title Tilted and Data-Sharpened Nonparametric Density Estimation

Version 0.2.0

Description High-order nonparametric density estimators built by perturbing a conventional kernel estimator, either by re-weighting the observations (``tilting'') or by moving them (``data sharpening''). The perturbation is chosen so that the estimator inherits the fast convergence rate of an infinite-order kernel estimator, such as the sinc or trapezoidal flat-top estimator, while remaining a proper non-negative density without the oscillatory tails those estimators suffer from. Two criteria are provided: minimising the L2 distance to an infinite-order comparator, following Doosti and Hall (2016) <doi:10.1111/rssb.12112>, and minimising a cross-validation criterion that needs no comparator and is much faster, following Doosti, Hall and Mateu (2018) <doi:10.1016/j.jspi.2017.12.003>.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

Depends R (>= 3.5.0)

Imports graphics, stats, quadprog

Suggests testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.3.1

Config/testthat/edition 3

URL https://github.com/DoostiH/tiltdens

BugReports https://github.com/DoostiH/tiltdens/issues

NeedsCompilation no

Author Hassan Doosti [aut, cre, cph]

Maintainer Hassan Doosti <hassan.doosti@mq.edu.au>

Repository CRAN

Date/Publication 2026-09-28 07:10:03 UTC

Contents

`bandwidth` *Bandwidth selectors*

Description

Three rules, for three different jobs.

Usage

```
bw_comparator_cv(
  x,
  comparator = c("sinc", "trapezoid"),
  q_step = 0.05,
  q_max = NULL,
  q_max_factor = 2,
  q_grid = NULL
)

bw_flattop(x, c_thresh = 2, kn = NULL, t_max = NULL, n_grid = 4000)

bw_nrd_robust(x, kernel = "gaussian")

bw_canonical_factor(kernel = "gaussian")

bw_convert(bw, from, to)
```

Arguments

x	Numeric vector of observations.
comparator	'"sinc"' (default) or '"trapezoid"'.
q_step	Spacing of the frequency grid $q = 1/h$ (default 0.05).
q_max	Largest frequency searched. Defaults to 'q_max_factor' divided by 'bw_flattop(x)'.
q_max_factor	Multiplier for the automatic cutoff (default 2).
q_grid	Supply the frequency grid outright, overriding the above.
c_thresh	Threshold constant in the flat-top rule (default 2).
kn	Length of the stretch that must stay below threshold. Defaults to 'max(5, sqrt(log10(n)))'.
t_max	Largest frequency searched by the flat-top rule.
n_grid	Number of frequencies searched by the flat-top rule.
kernel	Kernel the bandwidth is for; see [tilt_kernels()].
bw	Bandwidth to convert.
from, to	Kernels to convert between, for 'bw_convert()'.

Details

'bw_comparator_cv()' chooses the bandwidth of an infinite-order comparator estimator by minimising least-squares cross-validation,

$$CV(h) = \int \check{f}(x|h)^2 dx - \frac{2}{n} \sum_i \check{f}_{-i}(x_i|h),$$

equation (2.7) of Doosti and Hall (2016). It is the default throughout the package, and it is the rule the bandwidths used for the published simulations were computed with.

'bw_flattop()' implements the empirical rule of Politis (2003): take $\hat{q}$ to be the smallest frequency beyond which the modulus of the empirical characteristic function stays below $c\sqrt{\log_{10}(n)/n}$ over a stretch of length 'kn', and set $h = 1/\hat{q}$. It is also used internally to cap the cross-validation search.

'bw_nrd_robust()' is a rule of thumb for the *conventional* estimator: $(4/3)^{1/5} n^{-1/5}$ times a robust scale, rescaled for the kernel in use (see below).

'bw_canonical_factor()' and 'bw_convert()' handle the fact that bandwidths mean different things for different kernels.

Value

A single positive number. 'bw_comparator_cv()' attaches an attribute '"cv"', a data frame of the frequency grid and criterion, which is worth plotting: the criterion is multimodal, and seeing that is more informative than the single number.

Why the cross-validation search is bounded

Least-squares cross-validation for an infinite-order kernel is multimodal in 'h', and its deepest minimum is often a spurious one at a very small bandwidth. Past the frequency at which the empirical characteristic function decays into sampling noise, the criterion is chasing noise, so its global minimum badly undersmooths. The search is therefore capped at 'q_max_factor' times the frequency identified by 'bw_flattop()'.

Canonical bandwidths

A bandwidth is only meaningful relative to the kernel it scales: ‘bw = 0.5‘ smooths far less with the Epanechnikov kernel, supported on $[-1, 1]$, than with the Gaussian, which has unit variance. Marron and Nolan (1988) resolve this with the canonical factor

$$\delta_K = \left(R(K)/\mu_2(K)^2\right)^{1/5},$$

where $R(K) = \int K^2$ and $\mu_2(K) = \int u^2 K(u)\, du$. Bandwidths in units of δ_K are comparable across kernels: the same number produces the same amount of smoothing, and the asymptotically optimal bandwidths coincide.

‘bw_canonical_factor()‘ returns δ_K. ‘bw_convert()‘ maps a bandwidth from one kernel to the equivalent for another, multiplying by the ratio of their factors. The estimators use this automatically: when you do not supply ‘bw‘, the bandwidth chosen for the comparator is rescaled to the kernel in use, so switching kernels changes the shape of the fit without changing how much it is smoothed.

References

Doosti, H. and Hall, P. (2016). Making a non-parametric density estimator more attractive, and more accurate, by data perturbation. *Journal of the Royal Statistical Society B* **78**, 445-462.

Politis, D. N. (2003). Adaptive bandwidth choice. *Journal of Nonparametric Statistics* **15**, 517-533.

Marron, J. S. and Nolan, D. (1988). Canonical kernels for density estimation. *Statistics and Probability Letters* **7**, 195-199.

Examples

```
set.seed(1)
x <- c(rnorm(50, -1.5), rnorm(50, 1.5))

bw_comparator_cv(x)
bw_flattop(x)
bw_nrd_robust(x)

## The criterion is multimodal; the search is capped to avoid the
## spurious minima at small bandwidths.
cv <- attr(bw_comparator_cv(x), "cv")
plot(cv$q, cv$cv, type = "l", xlab = "frequency 1/h", ylab = "CV")

## Bandwidths are not comparable across kernels until rescaled.
bw_canonical_factor("gaussian")
bw_canonical_factor("epanechnikov")
bw_convert(0.5, from = "gaussian", to = "epanechnikov")
```

`comparator_density` *Infinite-order kernel density estimators*

Description

Evaluate the two comparator estimators used by the tilting and data sharpening methods. Both are built from kernels of unlimited order: they converge very fast when the underlying density is smooth, but they can take negative values and they oscillate in the tails. Removing those defects without giving up the convergence rate is the point of [tilt_density()].

Usage

```
sinc_density(x, bw = NULL, n = 512, from = NULL, to = NULL)

trapezoid_density(x, bw = NULL, n = 512, from = NULL, to = NULL)
```

Arguments

`x`	Numeric vector of observations.
`bw`	Positive bandwidth. Defaults to [bw_comparator_cv()] for the matching kernel.
`n`	Number of grid points at which to evaluate.
`from`, `to`	Range of the grid. Defaults to the data range extended by three bandwidths.

Details

‘sinc_density()‘ uses $L(u) = \sin(u)/(\pi u)$, whose Fourier transform is flat on $[-1, 1]$ and zero outside.

‘trapezoid_density()‘ uses $L(u) = (\cos u - \cos 2u)/(\pi u^2)$, whose Fourier transform is 1 on $[-1, 1]$ and tapers linearly to 0 at $|z| = 2$ (Politis 2003; Politis and Romano 1999).

Value

An object of class ‘"density"‘, so that ‘plot()‘, ‘lines()‘ and ‘print()‘ work as they do for [stats::density()].

References

Politis, D. N. (2003). Adaptive bandwidth choice. *Journal of Nonparametric Statistics* **15**, 517-533.

Examples

```
set.seed(1)
x <- c(rnorm(50, -1.5), rnorm(50, 1.5))
plot(sinc_density(x))
abline(h = 0, col = "grey")   # the estimate dips below zero
```

`conventional_density` *Conventional kernel density estimator*

Description

The ordinary estimator with uniform weights, evaluated with the same kernel code as the tilted estimators. It is provided so that comparisons within the package, and the replication materials of the accompanying article, do not depend on [stats::density()], whose output changed slightly between R versions 4.3 and 4.4.

Usage

```
conventional_density(
  x,
  bw = NULL,
  kernel = "gaussian",
  n = 512,
  from = NULL,
  to = NULL
)
```

Arguments

`x`	Numeric vector of observations.
`bw`	Positive bandwidth. Defaults to [bw_nrd_robust()] for the kernel.
`kernel`	Kernel name or object; see [tilt_kernels()].
`n`, `from`, `to`	Grid on which to evaluate.

Value

An object of class ‘"density"‘.

Examples

```
set.seed(1)
x <- rnorm(100)
plot(conventional_density(x))
lines(conventional_density(x, kernel = "epanechnikov"), col = "red")
```

ise *Integrated squared error against a known density*

Description

Approximates $\int(\hat{f} - f)^2$ by the trapezoidal rule over the grid of the fit. Useful for simulation studies where the truth is known.

Usage

```
ise(object, true_density)
```

Arguments

object	A ‘"tiltdens"‘ or ‘"density"‘ object.
true_density	A function of one numeric vector.

Value

A single number.

Examples

```
set.seed(1)
fit <- tilt_density(rnorm(80), m = 3)
ise(fit, dnorm)
```

kernels *Kernels for the perturbed estimator*

Description

The estimator $\hat{f}(x) = \sum_i p_i K_h(x - x_i)$ needs a kernel ‘K‘. The papers use the standard normal throughout, noting in Section 4.1 of Doosti and Hall (2016) that it "simplified numerical work"; the theory in Section 3.1 is in fact stated for kernels satisfying condition (3.1), which holds when ‘K‘ is a k-fold convolution of a Laplace density. Both families are available here, along with the usual compactly supported kernels.

Usage

```
tilt_kernels()

tilt_kernel(name = "gaussian")
```

Arguments

name	Kernel name, or a list describing a custom kernel (see Details).

Details

A custom kernel may be supplied as a list with elements ‘dens‘, ‘ft‘ and ‘support‘. ‘dens‘ and ‘ft‘ must be vectorised, ‘ft‘ must satisfy ‘ft(0) == 1‘, and ‘support‘ is the half-width of the support or ‘Inf‘. The self-convolution is then computed numerically.

Value

‘tilt_kernel()‘ returns a list with components ‘name‘, ‘dens‘ (the kernel), ‘ft‘ (its Fourier transform), ‘self_conv‘ (the self-convolution at bandwidth 1) and ‘support‘ (‘Inf‘ for the unbounded kernels). ‘tilt_kernels()‘ returns the available names.

Available kernels

‘"gaussian"‘ The standard normal density. The default, and the one used for every numerical result in both papers.

‘"laplace"‘ $\frac{1}{2}e^{-|u|}$. The k = 1 case of condition (3.1) of the 2016 paper.

‘"laplace2"‘ $\frac{1}{4}(1+|u|)e^{-|u|}$, the twofold convolution of a Laplace density and the k = 2 case of condition (3.1). Given as an example in the paper.

‘"laplace3"‘, ‘"laplace4"‘ Higher-order members of the same family, progressively smoother.

‘"epanechnikov"‘ $\frac{3}{4}(1-u^2)$ on $[-1,1]$. The minimum-variance choice under the usual asymptotics.

‘"biweight"‘ $\frac{15}{16}(1-u^2)^2$ on $[-1,1]$.

‘"triweight"‘ $\frac{35}{32}(1-u^2)^3$ on $[-1,1]$.

‘"triangular"‘ $1-|u|$ on $[-1,1]$.

Why any kernel keeps the problem convex

The quadratic form is $A_{ij} = (K * K)(x_i - x_j)$, whose Fourier transform is $\phi_K^2 \geq 0$. A function with a non-negative Fourier transform has a positive semidefinite Gram matrix, so ‘A‘ is positive semidefinite whatever ‘K‘ is, and weight selection remains a convex quadratic program. Equivalently, $p^\top A p = \int \hat{f}^2 \geq 0$ by construction.

A caution about bandwidths

Bandwidths are on each kernel’s own scale, so they are not comparable across kernels: ‘bw = 0.5‘ means something different for the Gaussian, which has unit variance, than for the Epanechnikov, which is supported on $[-1,1]$. If you switch kernels, let the bandwidth be chosen afresh rather than carrying a number over.

Examples

```
tilt_kernels()

set.seed(1)
x <- c(rnorm(50, -1.5), rnorm(50, 1.5))

## The Laplace-convolution kernel of the paper's condition (3.1).
fit <- tilt_density(x, m = 3, kernel = "laplace2")
fit
```

`make_kernel` *Construct and assess a custom kernel*

Description

'make_kernel()' builds a kernel object from a density function alone, computing everything else the package needs – the Fourier transform, the self-convolution, the second moment and the canonical bandwidth factor – by quadrature. 'check_kernel()' assesses whether a kernel, built-in or custom, has the properties the estimators assume, and reports them.

Usage

```
make_kernel(
  dens,
  support = Inf,
  name = "custom",
  ft = NULL,
  self_conv = NULL,
  support_eff = 40
)

check_kernel(kernel, tolerance = 0.001)
```

Arguments

`dens`	A vectorised function giving the kernel density.
`support`	Half-width of the support, or 'Inf'. For an unbounded kernel the numerical integrals are truncated at 'support_eff' (default 40), which is ample for any kernel with tails at least as light as the Laplace.
`name`	A name for the kernel.
`ft`, `self_conv`	Optional closed forms for the Fourier transform and the self-convolution; computed numerically when omitted.
`support_eff`	Truncation point used for unbounded kernels.
`kernel`	A kernel name or object, for 'check_kernel()'.
`tolerance`	Tolerance for the mass and symmetry checks.

Details

A kernel is suitable for tilting if it is a symmetric probability density. Non-negativity is what guarantees that the tilted estimate is a density; symmetry is assumed by the Fourier representation of the cross terms; and unit mass is what keeps the weights on the simplex meaningful.

Value

‘make_kernel()‘ returns a ‘"tilt_kernel"‘ object usable wherever a kernel name is accepted. ‘check_kernel()‘ returns, invisibly, a list of named logicals (‘integrates_to_one‘, ‘non_negative‘, ‘symmetric‘, ‘ft_at_zero_is_one‘) together with the computed ‘mass‘, ‘moment2‘, ‘roughness‘ and ‘canonical‘ factor, and prints a short report.

Examples

```
## The logistic kernel, from its density alone.
logistic <- make_kernel(function(u) exp(-u) / (1 + exp(-u))^2,
                        support = Inf, name = "logistic")
logistic
check_kernel(logistic)

set.seed(1)
x <- c(rnorm(40, -1.5), rnorm(40, 1.5))
tilt_density(x, m = 3, kernel = logistic)$distance2

## A kernel that is not a density is caught.
bad <- make_kernel(function(u) ifelse(abs(u) <= 1, 1 - u^2, 0), support = 1)
check_kernel(bad)$integrates_to_one
```

`plot.tiltdens_md` *Contour plot of a bivariate fit*

Description

Contour plot of a bivariate fit

Usage

```
## S3 method for class 'tiltdens_md'
plot(x, n = 60, points = TRUE, ...)
```

Arguments

`x`	A ‘"tiltdens_md"‘ object with two dimensions.
`n`	Grid points per axis.
`points`	Draw the observations over the contours.
`...`	Passed to [graphics::contour()].

Value

‘x‘, invisibly.

`predict.tiltdens` *Evaluate a fitted density at new points*

Description

Evaluate a fitted density at new points

Usage

```
## S3 method for class 'tiltdens'
predict(object, newdata = NULL, ...)

## S3 method for class 'tiltdens_md'
predict(object, newdata = NULL, ...)
```

Arguments

`object`	A ‘"tiltdens"‘ object.
`newdata`	Numeric vector of points at which to evaluate. Defaults to the grid stored in ‘object‘.
`...`	Ignored.

Value

A numeric vector of density values.

Examples

```
set.seed(1)
fit <- tilt_density(rnorm(60), m = 3)
predict(fit, newdata = c(-1, 0, 1))
```

print.tiltdens *Print a tilted or sharpened density fit*

Description

Print a tilted or sharpened density fit

Usage

```
## S3 method for class 'tiltdens'
print(x, digits = getOption("digits") - 2L, ...)
```

Arguments

x	A ‘"tiltdens"‘ object.
digits	Number of significant digits.
...	Ignored.

Value

‘x‘, invisibly.

print.tiltdens_md *Print a multivariate fit*

Description

Print a multivariate fit

Usage

```
## S3 method for class 'tiltdens_md'
print(x, digits = getOption("digits") - 2L, ...)
```

Arguments

x	A ‘"tiltdens_md"‘ object.
digits	Number of significant digits.
...	Ignored.

Value

‘x‘, invisibly.

`real_ga` *Real-coded genetic algorithm for small box-constrained problems*

Description

Minimises ‘fn‘ over the box ‘[lower, upper]‘. This is a tidied, dependency-free version of the algorithm used for the data sharpening results of Doosti and Hall (2016).

Usage

```
real_ga(
  fn,
  lower,
  upper,
  max_iterations = 130L,
  population = 50L,
  crossover_rate = 0.8,
  mutation_rate = 0.4,
  mutation_step = 0.02,
  blend_gamma = 0.3,
  selection_pressure = 8,
  trace = FALSE
)
```

Arguments

`fn`	Function of a numeric vector, returning a scalar.
`lower`, `upper`	Numeric vectors of bounds; their length sets the dimension.
`max_iterations`	Generations (default 130).
`population`	Population size (default 50).
`crossover_rate`	Fraction of the population replaced by offspring (0.8).
`mutation_rate`	Initial fraction of mutants (0.4, decayed after a third of the run).
`mutation_step`	Mutation scale as a fraction of the box width (0.02).
`blend_gamma`	Blend-crossover expansion factor (0.3).
`selection_pressure`	Exponent of the fitness-proportional selection (8).
`trace`	Print the best cost each generation.

Value

A list with ‘par‘, ‘value‘ and ‘history‘.

`sharpen_density` *Data-sharpened density estimation*

Description

Finds shifts ‘q‘ for the estimator

$$\hat{f}(x \mid h, q) = \frac{1}{nh} \sum_i K\left(\frac{x - x_i - q_i}{h}\right)$$

by minimising its L2 distance to an infinite-order comparator. This is the second form of data perturbation in Doosti and Hall (2016): the observations are moved rather than re-weighted.

Usage

```
sharpen_density(
  x,
  m = Inf,
  comparator = c("sinc", "trapezoid"),
  bw = NULL,
  comparator_bw = NULL,
  kernel = "gaussian",
  breaks = c("equal", "modal"),
  min_block = NULL,
  max_shift = 1,
  optimizer = "auto",
  polish = TRUE,
  control = list(),
  n = 512,
  from = NULL,
  to = NULL
)
```

Arguments

`x`	Numeric vector of observations.
`m`	Number of distinct shift values. ‘Inf‘ (default) or 3.
`comparator`	‘"sinc"‘ (default) or ‘"trapezoid"‘.
`bw`, `comparator_bw`, `breaks`, `min_block`	As in [tilt_density()]. Breakpoint search is not available here, so ‘breaks‘ defaults to ‘"equal"‘; supply ‘"modal"‘ or an explicit numeric vector if you want different boundaries.
`kernel`	Kernel of the estimator; see [tilt_kernels()].
`max_shift`	Half-width of the box searched, as a multiple of the data range (default 1).

`optimizer` How the shifts are searched for: a character string naming a built-in strategy (‘"auto"‘, ‘"multistart"‘ or ‘"ga"‘), or a function; see the section on choosing the optimizer. The default ‘"auto"‘ uses a bounded quasi-Newton search from several starting points when there are few distinct shifts (‘m <= 10‘) and the genetic algorithm otherwise.

`polish` Refine the search result with a bounded quasi-Newton step, [stats::optim()] with method ‘"L-BFGS-B"‘ (default ‘TRUE‘).

`control` A list of options for the built-in optimizer: for ‘"multistart"‘, ‘n_starts‘ (default 5); for ‘"ga"‘, the arguments of [real_ga()].

`n`, `from`, `to` Grid on which to evaluate the fitted density.

Details

The objective is not convex, so the result can depend on the starting points; set a seed before calling if you need reproducibility. In the published simulations tilting was usually at least as accurate and far cheaper, so [tilt_density()] and [tilt_density_cv()] are the better default.

Value

An object of class ‘"tiltdens"‘; see [tilt_density()]. The ‘shifts‘ and ‘sharpened‘ components hold ‘q‘ and ‘x + q‘.

Choosing the optimizer

The objective is smooth and, for a small number of distinct shifts, low dimensional, so a bounded quasi-Newton search from a handful of starting points finds a good minimum in a few hundred function evaluations. That is ‘optimizer = "multistart"‘, and it is what ‘"auto"‘ selects when ‘m <= 10‘. With one shift per observation the problem has ‘length(x)‘ variables and many local minima; ‘"ga"‘, the real-coded genetic algorithm of [real_ga()], is the more robust choice there, at a cost of thousands of evaluations.

To use a different search, pass a function with signature ‘function(fn, lower, upper)‘ returning a list with elements ‘par‘ and ‘value‘. Anything from **stats**, **nloptr**, **GA** or **DEoptim** can be wrapped this way; the example below wraps a single bounded quasi-Newton run.

Sharpening is univariate only. For multivariate data use [tilt_density()], where the weights, unlike the shifts, keep the problem convex.

References

Doosti, H. and Hall, P. (2016). Making a non-parametric density estimator more attractive, and more accurate, by data perturbation. *Journal of the Royal Statistical Society B* **78**, 445-462.

See Also

[tilt_density()], [tilt_density_cv()]

Examples

```
set.seed(1)
x <- c(rnorm(20, -1.5), rnorm(20, 1.5))

fit <- sharpen_density(x, m = 3)
fit
plot(fit)
round(unique(fit$shifts), 3)

## A custom optimizer: one bounded quasi-Newton run from the origin.
from_origin <- function(fn, lower, upper) {
  r <- stats::optim(rep(0, length(lower)), fn, method = "L-BFGS-B",
                    lower = lower, upper = upper)
  list(par = r$par, value = r$value)
}
sharpen_density(x, m = 3, optimizer = from_origin)$distance2
```

`summary.tiltdens` *Summarise a tilted or sharpened density fit*

Description

Summarise a tilted or sharpened density fit

Usage

```
## S3 method for class 'tiltdens'
summary(object, ...)

## S3 method for class 'summary.tiltdens'
print(x, digits = getOption("digits") - 2L, ...)
```

Arguments

`object`	A ‘"tiltdens"‘ object.
`...`	Ignored.
`x`	A ‘"summary.tiltdens"‘ object.
`digits`	Number of significant digits to print.

Value

An object of class ‘"summary.tiltdens"‘: a list with the method, sample size, bandwidth, kernel, the number of distinct weights and their range, the criterion value, and the minimum of the fitted density. It has a ‘print‘ method.

Examples

```
set.seed(1)
summary(tilt_density_cv(c(rnorm(30, -1.5), rnorm(30, 1.5))))
```

`tilt_cv` *Cross-validation criterion for a tilted estimator*

Description

Evaluates

$$CV(h,p) = \int \hat{f}(x \mid h,p)^2 dx - \frac{2}{n}\sum_i \hat{f}_{-i}(x_i \mid h,p),$$

equation (2.1) of Doosti, Hall and Mateu (2018), where $\hat{f}_{-i}$ is the estimator with the term in x_i deleted. As in the paper, ‘p‘ is *not* re-standardised after deletion; that step turns out to be unnecessary and skipping it saves computation.

Usage

```
tilt_cv(x, bw, weights = NULL, kernel = "gaussian")
```

Arguments

`x`	Numeric vector of observations, or a numeric matrix or data frame with one row per observation for multivariate data; see Details.
`bw`	Positive bandwidth.
`weights`	Tilt weights. Defaults to the uniform weights ‘1/n‘, in which case this reduces to ordinary least-squares cross-validation for the bandwidth.
`kernel`	Kernel of the estimator; see [tilt_kernels()].

Details

The first term is $p^\top A p$ and the second is linear in ‘p‘, so this criterion has the same quadratic form as the 2016 distance criterion. The only difference is the linear term: the 2016 method compares against a comparator estimator, the 2018 method against the leave-one-out fit. That is why one solver serves both.

Value

A single number, with attributes ‘"A"‘ and ‘"C"‘ holding the quadratic and linear parts, which [tilt_density_cv()] reuses.

References

Doosti, H., Hall, P. and Mateu, J. (2018). Nonparametric tilted density function estimation: a cross-validation criterion. *Journal of Statistical Planning and Inference* **197**, 51-68.

Examples

```
set.seed(1)
x <- rnorm(60)
tilt_cv(x, bw = 0.4)
```

`tilt_density` *Tilted density estimation by distance to a comparator*

Description

Chooses non-negative weights ‘p‘ summing to one for the estimator

$$\hat{f}(x \mid h, p) = \sum_i p_i K_h(x - x_i)$$

by minimising its L2 distance to an infinite-order comparator estimator. This is the method of Doosti and Hall (2016): it inherits the fast convergence rate of the comparator while staying a proper, non-negative density with no oscillatory tails.

Usage

```
tilt_density(
  x,
  m = Inf,
  comparator = c("sinc", "trapezoid"),
  bw = NULL,
  comparator_bw = NULL,
  bw_grid = NULL,
  kernel = "gaussian",
  breaks = c("optimal", "equal", "modal"),
  min_block = NULL,
  constraint = "none",
  constraint_range = NULL,
  constraint_grid_n = 100L,
  n = 512,
  from = NULL,
  to = NULL
)
```

Arguments

`x`	Numeric vector of observations, or a numeric matrix or data frame with one row per observation for multivariate data; see Details.
`m`	Number of distinct weight values. ‘Inf‘ (default) or 3 are the cases studied in the paper.
`comparator`	‘"sinc"‘ (default) or ‘"trapezoid"‘.

`bw` Bandwidth of the tilted estimator. Defaults to the comparator bandwidth, as in the papers.

`comparator_bw` Bandwidth of the comparator. Defaults to [bw_comparator_cv()].

`bw_grid` Optional grid of candidate bandwidths. When supplied, the weights are re-optimised at each one and the pair ‘(h, p)‘ with the smallest distance is returned, which is the joint minimisation over $\theta = (h, p)$ that the paper defines.

`kernel` Kernel of the perturbed estimator: ‘"gaussian"‘ (default), ‘"laplace"‘, ‘"laplace2"‘, ‘"epanechnikov"‘, ‘"biweight"‘ and others; see [tilt_kernels()] for the full list, or supply a custom kernel as a list. Bandwidths are on each kernel’s own scale and are not comparable across kernels.

`breaks` How the block boundaries are chosen when ‘m‘ is finite. One of:

‘"optimal"‘ (default) The boundaries are chosen jointly with the weights, minimising the same criterion. This is what Section 4.1 of the 2016 paper specifies: the breakpoints r_1, r_2 are part of the optimisation. Exhaustive for ‘m <= 3‘, coordinate descent above.

‘"equal"‘ Blocks of near-equal size, Algorithm A of the 2018 paper. Much cheaper, and often nearly as good.

‘"modal"‘ Boundaries placed at the troughs of a pilot density estimate, following the practical suggestion in Section 2 of the 2018 paper.

A numeric vector of ‘m - 1‘ breakpoints, given as ranks in the sorted sample, may be supplied instead.

`min_block` Smallest number of observations allowed in a block, used by ‘"optimal"‘ and ‘"modal"‘. Defaults to ‘max(2, floor(0.02 * n))‘.

`constraint` Shape constraint imposed on the fitted density: ‘"none"‘ (default), ‘"unimodal"‘, ‘"increasing"‘ or ‘"decreasing"‘. The estimator is linear in the weights, so a shape requirement evaluated on a grid becomes a set of linear inequalities and the problem stays a convex quadratic program. Univariate only. See Details.

`constraint_range` Interval on which the shape constraint is enforced. Defaults to ‘range(x)‘. It cannot usefully extend past the observations: a Gaussian kernel mixture must rise from zero on the left and fall to zero on the right, so a monotonicity requirement outside the data range has no feasible solution.

`constraint_grid_n` Number of points at which the shape constraint is enforced (default 100). The constraint holds exactly on this grid; between grid points a negligible violation is possible, so raise this if you need a tighter guarantee.

`n`, `from`, `to` Grid on which to evaluate the fitted density.

Details

The problem is a convex quadratic program over the probability simplex, so it has a unique solution.

Value

An object of class ‘"tiltdens"‘, which inherits from ‘"density"‘, so ‘plot()‘, ‘lines()‘ and ‘points()‘ work directly. Components beyond those of [stats::density()] include ‘weights‘, ‘distance2‘ (the attained squared L2 distance), ‘comparator‘, ‘comparator_bw‘, ‘n_groups‘ and ‘group_of‘.

Choosing ‘m‘

‘m = Inf‘ lets every observation take its own weight, which is the most flexible option and the one the paper calls T_n. ‘m = 3‘ restricts the weights to three distinct values (T_3); it is faster, and in the paper's simulations it was often more accurate, because fewer free parameters means less overfitting.

When ‘m‘ is finite the block boundaries matter as much as the number of blocks, and ‘breaks‘ controls them. The default ‘"optimal"‘ treats the boundaries as part of the optimisation, which is what Section 4.1 of the 2016 paper specifies. On bimodal data it typically places them at the trough between the modes without being told to, which is the same answer ‘"modal"‘ arrives at by construction from a pilot estimate.

References

Doosti, H. and Hall, P. (2016). Making a non-parametric density estimator more attractive, and more accurate, by data perturbation. *Journal of the Royal Statistical Society B* **78**, 445-462.

See Also

[tilt_density_cv()] for the faster cross-validation variant, [sharpen_density()] for perturbing the observations instead of their weights.

Examples

```
set.seed(1)
x <- c(rnorm(25, -1.5), rnorm(25, 1.5))

fit <- tilt_density(x, m = 3)
fit
plot(fit)
lines(sinc_density(x), col = "red", lty = 2)
abline(h = 0, col = "grey")

## The comparator dips below zero; the tilted estimate cannot.
min(sinc_density(x)$y)
min(fit$y)


## Choosing the block boundaries, rather than fixing them, lowers the
## criterion the method is minimising. Searching them is the slow option:
## it examines every admissible pair, so its cost grows with n^2.
c(equal   = tilt_density(x, m = 3, breaks = "equal")$distance2,
  modal   = tilt_density(x, m = 3, breaks = "modal")$distance2,
  optimal = tilt_density(x, m = 3, breaks = "optimal")$distance2)
```

`tilt_density_cv` *Tilted density estimation by cross-validation*

Description

Chooses the bandwidth and the tilt weights together by minimising [tilt_cv()], rather than by comparing against an infinite-order estimator. This is method (III) of Doosti, Hall and Mateu (2018). It needs no comparator and is much cheaper than [tilt_density()], which is the point: the 2016 construction is accurate but slow, and this makes it practical.

Usage

```
tilt_density_cv(
  x,
  max_groups = 3,
  bw_grid = NULL,
  rho = 0.8,
  kernel = "gaussian",
  breaks = c("equal", "modal", "optimal"),
  min_block = NULL,
  constraint = "none",
  constraint_range = NULL,
  constraint_grid_n = 100L,
  n = 512,
  from = NULL,
  to = NULL
)
```

Arguments

`x`	Numeric vector of observations, or a numeric matrix or data frame with one row per observation for multivariate data; see Details.
`max_groups`	Largest number of distinct weight values (default 3).
`bw_grid`	Candidate bandwidths. Defaults to 40 points spaced logarithmically over $[n^{-1/5-c_1}, n^{-1/5+c_1}]$ times a robust scale, matching the class 'H' of equation (3.1) with $c_1 = 1/30$.
`rho`	Constant $\rho \in (0, 1)$ of equation (2.6) (default 0.8).
`kernel`	Kernel of the estimator; see [tilt_kernels()].
`breaks`	How the block boundaries are chosen. '"equal"' (the default) gives blocks of near-equal size, which is Algorithm A of the paper. '"modal"' places them at the troughs of a pilot density estimate, the refinement the paper suggests for multimodal densities. '"optimal"' chooses them jointly with the weights, which is more expensive but usually gives a smaller criterion. A numeric vector of breakpoints may be supplied instead. See [tilt_density()] for the full description.
`min_block`	Smallest number of observations allowed in a block.

`constraint` Shape constraint, as in [tilt_density()]. Applying a shape constraint to the cross-validation criterion follows the extension suggested in the discussion of the 2018 paper.

`constraint_range`, `constraint_grid_n` As in [tilt_density()].

`n`, `from`, `to` Grid on which to evaluate the fitted density.

Details

The sequential scheme of the paper is used. Start with uniform weights and choose the bandwidth by ordinary least-squares cross-validation. At step ‘r‘, minimise ‘CV(h, p)‘ jointly over ‘h‘ and over weights taking ‘r + 1‘ distinct values, subject to ‘h >= rho * h_previous‘; the constraint reflects the optimal bandwidth growing as the bias shrinks. Stop at ‘max_groups‘ blocks.

Value

An object of class ‘"tiltdens"‘; see [tilt_density()]. The ‘trace‘ component records the number of blocks, bandwidth and criterion value at each step, which is useful for seeing whether extra blocks bought anything.

References

Doosti, H., Hall, P. and Mateu, J. (2018). Nonparametric tilted density function estimation: a cross-validation criterion. *Journal of Statistical Planning and Inference* **197**, 51-68.

See Also

[tilt_density()], [tilt_cv()]

Examples

```
set.seed(1)
x <- c(rnorm(50, -1.5), rnorm(50, 1.5))

fit <- tilt_density_cv(x)
fit
fit$trace
plot(fit)
```

Index