Package 'scdensity'

July 23, 2025

Title Shape-Constrained Kernel Density Estimation

Version 1.0.3

Description Implements methods for obtaining kernel density estimates subject to a variety of shape constraints (unimodality, bimodality, symmetry, tail monotonicity, bounds, and constraints on the number of inflection points). Enforcing constraints can eliminate unwanted waves or kinks in the estimate, which improves its subjective appearance and can also improve statistical performance. The main function scdensity() is very similar to the density() function in 'stats', allowing shape-restricted estimates to be obtained with little effort. The methods implemented in this package are described in Wolters and Braun (2017) <doi:10.1080/03610918.2017.1288247>, Wolters (2012) <doi:10.18637/jss.v047.i06>, and Hall and Huang (2002) <https://www3.stat.sinica.edu.tw/statistica/j12n4/j12n41/j12n41.htm>. See the scdensity() help for for full citations.

Depends R (>= 3.3.0)

License GPL-2

Encoding UTF-8

Suggests testthat

Imports quadprog, lpSolve

RoxygenNote 7.3.2

NeedsCompilation no

Author Mark A. Wolters [aut, cre] (ORCID: <https://orcid.org/0000-0001-7638-8222>)

Maintainer Mark A. Wolters <mark@mwolters.com>

Repository CRAN

Date/Publication 2024-08-27 13:50:02 UTC

Contents

`improve` *Move points closer to a target while maintaining a constraint.*

Description

`improve(startValue, x, confun)` uses a greedy algorithm to move the elements of a user-supplied vector `startValue` closer to their target values `x`, while continually satisfying the constraint-checking function `confun`.

Usage

```
improve(
  startValue,
  x,
  confun,
  verbose = FALSE,
  maxpasses = 500,
  tol = diff(range(c(startValue, x))/1e+05)
)
```

Arguments

`startValue`	The vector of starting values for the search. Must satisfy `confun(startValue) == TRUE`
`x`	The target values.
`confun`	The constraint-checking function. `confun(y)` must return a Boolean value that is invariant to permutations of its vector argument `y`.
`verbose`	A logical value indicating whether or not information about iteration progress should be printed to the console.
`maxpasses`	The maximum allowable number of sweeps through the data points. At each pass, every point that is not pinned at the constraint boundary is moved toward its target point in a stepping-out procedure.
`tol`	Numerical tolerance for constraint checking. A point is considered to be at the constraint boundary if adding `tol` to it causes the constraint to be violated. If `tol` is too large, the algorithm will terminate prematurely. If it is too small, run time will be increased with no discernible benefit in the result.

Details

The algorithm implemented here is the one in Wolters (2012), "A Greedy Algorithm for Unimodal Kernel Density Estimation by Data Sharpening," *Journal of Statistical Software*, 47(6). It could conceivably be useful as a part of other gradient-free optimization schemes where we have an infeasible point and a feasible one, and we seek a point that is on the constraint boundary near the infeasible one.

Value

A vector of the same length as `startValue`, with elements closer to `x`.

Examples

```
#Constrain points to be inside the hypercube with vertices at -1 and +1.
#The target point is a vector of independent random standard normal variates.
#Start at rep(0,n) and "improve" the solution toward the target.
n <- 20
incube <- function(x) all(x <= 1 & x >= -1)
x0 <- rep(0,n)
target <- sort(rnorm(n))
xstar <- improve(x0, target, incube, verbose=TRUE)
dist <- abs(target - xstar)
zapsmall(cbind(target, xstar, dist), 4)
```

---

plot.scdensity *Plot method for class* `scdensity`.

Description

Creates a plot of a shape-constrained kernel density estimate. The amount of information in the plot is controlled by `detail`.

Usage

```
## S3 method for class 'scdensity'
plot(
  x,
  detail = 4,
  main = c("Density Estimate", "Q-Q Plot"),
  xlab = c(x$data.name, "Constrained KDE Quantiles"),
  ylab = c("Density", "Sample Quantiles"),
  type = c("l", "l", "p"),
  lty = c(1, 2, 0),
  pch = c(-1, -1, 1),
  col = c("black", gray(0.4), "black"),
  lwd = c(2, 1, 0),
  zero.line = TRUE,
  ...
)
```

Arguments

x	An object of S3 class scdensity.
detail	An integer from 1 to 4, indicating the level of information to include in the plot. 1: plot only the constrained estimate. 2: draw both the constrained and unconstrained estimates on the same plot. 3: add a rug showing the data points. 4: additionally plot a Q-Q plot of the observed data versus the constrained estimate in a second panel (for qualitative assessment of goodness-of-fit).
main	A string passed on to the main argument of the plot command. If detail == 4, pass a vector of two strings to specify titles for both subfigures.
xlab	A string passed on to the xlab argument of the plot command. If detail == 4, pass a vector of two strings to specify x labels for both subfigures.
ylab	A string passed on to the ylab argument of the plot command. If detail == 4, pass a vector of two strings to specify y labels for both subfigures.
type	A vector of up to 3 strings specifying the type of plot used for 1) the constrained estimate, 2) the unconstrained estimate, and 3) the Q-Q plot.
lty	A vector of up to length 3, specifying the lty arguments passed to the plot commands for 1) the constrained estimate, 2) the unconstrained estimate, and 3) the Q-Q plot. See the description of lty in graphics::par().
pch	A vector of up to 3 integers specifying the pch argument passed to the plot commands for 1) the constrained estimate, 2) the unconstrained estimate, and 3) the Q-Q plot. See graphics::points() for the integer codes.
col	A vector of up to 3 strings specifying the col argument passed to the plot commands for 1) the constrained estimate, 2) the unconstrained estimate, and 3) the Q-Q plot.
lwd	A vector of up to length 3 specifying the lwd argument passed to the plot commands for 1) the constrained estimate, 2) the unconstrained estimate, and 3) the Q-Q plot.
zero.line	A logical value indicating whether or not a horizontal line should be drawn through zero to aid visualization.
...	Extra parameters passed to the initial plot command for each subfigure.

Examples

```
# Basic usage:
x <- rlnorm(30)
scKDE <- scdensity(x)
plot(scKDE)

# Show only the constrained estimate
plot(scKDE, detail=1)

# Show the constrained and unconstrained estimates.  Change line color and width.
plot(scKDE, detail=2, col=c("red","blue"), lwd=c(3,2))

# Show the Q-Q plot, but change that plot's symbol and its size.
plot(scKDE, detail=4, pch=c(-1, -1, 3), cex=0.5)
```

print.scdensity *Print method for class* scdensity.

Description

Displays the names of the elements of the scdensity list object and their sizes and types. Includes minimal comments about the most important elements.

Usage

```
## S3 method for class 'scdensity'
print(x, ...)
```

Arguments

x	An object of S3 class scensity.
...	Included for consistency with generic functions.

print.summary.scdensity *Prints the information in a* summary.scdensity *object to the console.*

Description

Prints the information in a summary.scdensity object to the console.

Usage

```
## S3 method for class 'summary.scdensity'
print(x, ...)
```

Arguments

x	An object of S3 class summary.scdensity.
...	Included for consistency with generic functions.

scdensity *Shape-constrained kernel density estimation.*

Description

scdensity computes kernel density estimates that satisfy specified shape restrictions. It is used in the same way as stats::density(), and takes most of that function's arguments. Its default behavior is to compute a unimodal estimate. Use argument constraint to choose different shape constraints, method to choose a different estimation method, and opts to specify method- and constraint-specific options. The result is a list of S3 class scdensity, which may be inspected via print, summary, and plot methods.

Usage

```
scdensity(
  x,
  bw = "nrd0",
 constraint = c("unimodal", "monotoneRightTail", "monotoneLeftTail", "twoInflections",
    "twoInflections+", "boundedLeft", "boundedRight", "symmetric", "bimodal"),
  method = c("adjustedKDE", "weightedKDE", "greedySharpenedKDE"),
  opts = NULL,
  adjust = 1,
  n = 512,
  na.rm = FALSE
)
```

Arguments

x	A vector of data from which the estimate is to be computed.
bw	The bandwidth. It is specified as either a numerical value or as one of the character strings "nrd0", "nrd", "ucv", "bcv", or "SJ", exactly as in stats::density().
constraint	A vector of strings giving the operative shape constraints. Elements must partially match different alternatives among "unimodal", "monotoneRightTail","monotoneLeftTail", "twoInflections", "twoInflections+", "boundedLeft", "boundedRight", "symmetric", and "bimodal".
method	A string giving the method of enforcing shape constraints. It must paritally match one of "adjustedKDE", "weightedKDE", or "greedySharpenedKDE".
opts	A list giving options specific to the chosen constraints and/or method. E.g. use opts = list(modeLocation = 0) to force the mode to be at zero when the constraint is unimodal. See below for lists of available options.
adjust	A scaling factor for the bandwidth, just as in stats::density().
n	The number of points returned in the density estimate. Same as in stats::density().
na.rm	Logical indicating whether or not to remove missing values from x. Same as in stats::density().

Details

All density estimates in this package use the Gaussian kernel. It is the only common kernel function with three continuous derivatives everywhere. The `adjustedKDE` and `weightedKDE` methods require continuous derivatives to ensure numerical stability.

The default estimation method, `adjustedKDE`, can handle all of the available constraints. The `weightedKDE` method can handle every constraint except `symmetric`, while the `greedySharpenedKDE` method can handle only `unimodal`, `monotoneRightTail`, `monotoneLeftTail`, `boundedLeft`, and `boundedRight`. The `opts` list can also be used to supply method-specific control parameters. See the "Method details" section for more.

Each constraint has a corresponding control parameter that can be supplied as an element of `opts`. The control parameters are described in the following table. See the "Constraint details" section for definitions of each constraint.

Constraint	Compatible methods	Relevant `opts`	Notes

`unimodal`	`adjustedKDE`
	`weightedKDE`
	`greedySharpenedKDE`
		`modeLocation`
		(optional, scalar)
		(default: NULL – do a search)
			• Implies `monotoneRightTail` and `monotoneLeftTail`
			• `modeLocation` not used for `greedySharpenedKDE`

`monotoneRightTail`	`adjustedKDE`
	`weightedKDE`
	`greedySharpenedKDE`
		`rightTail`
		(optional, scalar in [0, 100])
		(default: 90)
			• `rightTail` is a percent point of the unconstrained KDE

`monotoneLeftTail`	`adjustedKDE`
	`weightedKDE`
	`greedySharpenedKDE`
		`leftTail`
		(optional, scalar in [0, 100])
		(default: 10)
			• `leftTail` is a percent point of the unconstrained KDE

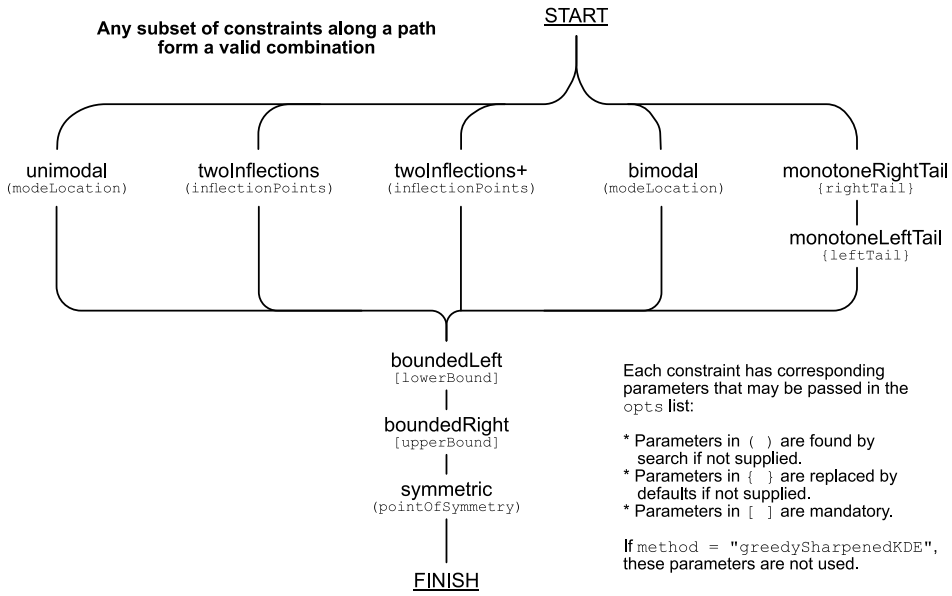
`twoInflections`	`adjustedKDE` `weightedKDE`	`inflectionPoints` (optional, vector, length 2) (default: NULL – do a search)	• Implies `unimodal`, `monotoneRightTail`, and `monotoneLeftTail`
`twoInflections+`	`adjustedKDE` `weightedKDE`	`inflectionPoints` (optional, vector, length 3) (default: NULL – do a search)	• Implies `twoInflections`, `unimodal`, `monotoneRightTail`, and `monotoneLeftTail`

`boundedLeft`	`adjustedKDE`
	`weightedKDE`
	`greedySharpenedKDE`
		`lowerBound`
		(required, scalar)	

`boundedRight`	`adjustedKDE`
	`weightedKDE`
	`greedySharpenedKDE`
		`upperBound`
		(required, scalar)	

`symmetric`	`adjustedKDE`	`pointOfSymmetry` (optional, scalar) (default: NULL – do a search)	
`bimodal`	`adjustedKDE` `weightedKDE`	`modeLocation` (optional, vector, length 3) (default: NULL – do a search)	• `modeLocation` holds the locations of the left mode, antimode, and right mode • `Not compatible with unimodal, monotoneRightTail, monotoneLeftTail, twoInflections, or twoInflections+`

More than one shape constraint can be specified simultaneously. Certain combinations of constraints (e.g., `unimodal` and `monotoneRightTail`) are redundant, and will cause a warning. Other combinations (e.g., `unimodal` and `bimodal`) are incompatible and will cause an error. The figure below summarizes the valid constraint combinations.



Value

A list with the following elements:

- `constraint` The constraint(s) used for estimation. Might differ from the constraints supplied to the function if they included redundant constraints.
- `method` The estimation method used.
- `f0` A function. Use `f0(v)` to evaluate the unconstrained KDE at the points in `v`.
- `fhat` A function. Use `fhat(v)` to evaluate the constrained KDE at the points in `v`.
- `data` The data used to generate the estimate.
- `bw` The bandwidth used.
- `extra` A list holding additional outputs that are specific to the chosen method. See the "method details" section.
- `x` A vector of abscissa values for plotting the estimate. Same as in `stats::density()`.
- `y` A vector of ordinate values for plotting the estimate. Same as in `stats::density()`.
- `n` The sample size, not including missing values. Note, this `n` has no relation to the `n` provided in the arguments.
- `data.name` Deparsed name of the `x` argument, used in plotting.
- `call` The call to the function.
- `has.na` Always `FALSE`. Included for consistency with `stats::density()`.

Constraint details

All of the constraints other than `symmetric` are restrictions on the sign of the estimate, or its derviatives, over certain intervals. The boundaries of the intervals may be called *important points*. If

method="greedySharpenedKDE", the important points are determined implicitly during estimation. For the other methods, the locations of the important points may be supplied in opts; in most cases they are optional. If they are not provided, estimation will be run iteratively inside a search routine (SequentialLineMin) to find good values, and these values will be returned in the extra list.

Here is a list of the constraints with their definitions and any relevant comments about their usage.

- unimodal: The estimate is nondecreasing to the left of opts$modeLocation, and nonincreasing to the right. If modeLocation is not supplied, it is found by search.
- monotoneRightTail: The estimate is nonincreasing to the right of the opts$rightTail percentile of the unconstrained estimate. rightTail is a numeric value between 0 and 100. If it is not supplied, it is set to its default value, 90.
- monotoneLeftTail: The estimate is nondecreasing to the left of the opts$leftTail percentile of the unconstrained estimate. leftTail is a numeric value between 0 and 100. If it is not supplied, it is set to its default value, 10.
- twoInflections: The estimate has two inflection points, found at opts$inflectionPoints[1] and opts$inflectionPoints[2]. This constraint implies unimodality, but provides greater smoothness than unimodal. If inflectionPoints is not supplied, it is found by search.
- twoInflections+: The *derivative* of the estimate has three inflection points, located at opts$inflectionPoints[1], opts$inflectionPoints[2], and opts$inflectionPoints[3]. This constraint implies twoInflections but is even smoother. Most parametric densities with two tails satisfy this constraint. If inflectionPoints is not supplied, it is found by search.
- boundedLeft: The estimate is zero to the left of opts$lowerBound. The value of lowerBound must be specified in opts. This constraint is implemented only up to a numerical tolerance. Consequently it is still possible to use it with the Gaussian kernel.
- boundedRight: The estimate is zero to the right of opts$upperBound. The value of upperBound must be specified in opts. This constraint is implemented only up to a numerical tolerance. Consequently it is still possible to use it with the Gaussian kernel.
- symmetric: The estimate is symmetric around opts$pointOfSymmetry. If pointOfSymmetry is not provided, it is found by search.
- bimodal: The estimate has modes at opts$modeLocation[1] and opts$modeLocation[3], with an antimode (local minimum) at opts$modeLocation[2]. If modeLocation is not specified, it is found by search.

Method details

The adjustedKDE and weightedKDE methods are implemented using a common framework where the standard KDE is first approximated by a binning step, after which the constrained estimate is obtained. The greedySharpenedKDE method uses a different approach.

adjustedKDE and weightedKDE:

The adjustedKDE method is based on the method of Wolters and Braun (2017). The method uses the usual unconstrained kernel density estimate as a pilot estimate, and adjusts the shape of this estimate by adding a function to it. The function is selected to minimally change the shape of the pilot estimate while ensuring the constraints are satisfied. Any of the constraints can be used with this method.

The weightedKDE method is based on the method of Hall and Huang (2002). The method uses a weighted kernel density estimator, with the weights minimally perturbed such that the constraint is satisfied. Any of the constraints except symmetric may be used with this method.

For either of these methods, the following optional arguments can be provided as elements of opts:

- ncheck: The number of abscissa points used for constraint checking. By default, this is set to max(100, ceiling((diff(range(x)) + 6*h) / h)), where h is the bandwidth. With this default it should be rare to encounter constraint violations large enough to be visible in a plot. In the event that constraint violations are observed, re-run the estimation with a larger value of ncheck.
- verbose: If TRUE, progress information will be displayed in the console. The main use of this is to track the progress of the search for important points. Default is FALSE.

When either of these methods are used, the output list extra contains elements giving the locations of the important points used in the final estimate (e.g., modeLocation if the estimate is unimodal or bimodal). Additionally, it containts the following elements:

- conCheckGrid: A vector giving the abscissa values at which the constraints were enforced.
- binnedCenters: A vector giving the locations of the kernel centers determined in the binning step.
- binnedWeights: The weights corresponding to the binned centers.
- finalCenters: The kernel centers used for the final estimate.
- finalWeights: The weights used for the final estimate.

greedySharpenedKDE:

The greedySharpenedKDE method is described in Wolters (2012a, 2012b). It uses a data sharpening (shifting the data points) approach. Starting from an initial solution that satisfies the constraints, a greedy algorithm (implemented in the function improve) is used to move the points as close as possible to the observed data while maintaining feasibility.

The following optional arguments can be provided as elements of opts:

- startValue — A vector of the same length as x, giving the feasible initial solution from which the algorithm is started. If not specified, a vector with all data points at the location of the unconstrained estimate's highest mode will be used. Note, it is not guaranteed that the default will satisfy every constraint for every data set.
- verbose: If TRUE, information about iteration progress will be printed to the console. Default is FALSE.
- maxpasses: Each "pass" through the data points moves each point one-by-one in a greedy fasion. This option limits the maximum number of passes. Default is 500.
- tol: A numerical tolerance for constraint checking. See improve.
- ILS: An integer greater than zero. If supplied, the greedy algorithm is run inside an iterated local search metaheuristic, as described in Wolters (2012b, sec. 3.4). This can improve solution quality, but requires the greedy search to be run 2*ILS extra times.

When this method is used, the output list extra contains the following elements:

- xstar: The final vector of "sharpened" data points used to generate the estimate.

References

Hall and Huang (2002), Unimodal Density Estimation Using Kernel Methods, *Statistica Sinica*, 12, 965-990.

Wolters and Braun (2017), Enforcing Shape Constraints on a Probability Density Estimate Using an Additive Adjustment curve, *Communications in Statistics - Simulation and Computation*, 47(3), 672-691.

Wolters (2012a), A Greedy Algorithm for Unimodal Kernel Density Estimation by Data Sharpening, *Journal of Statistical Software*, 46(6), 1–26.

Wolters (2012b), Methods for Shape-Constrained Kernel Density Estimation. Ph.D. Thesis, University of Western Ontario.

See Also

`plot.scdensity` plot method, `print.scdensity` print method, and `summary.scdensity` summary method.

Examples

```
# Default method gives a unimodal estimate using adjustment curve method.
x <- rlnorm(30)
scKDE <- scdensity(x)
scKDE
summary(scKDE)
plot(scKDE, detail=2)
plot(scKDE, detail=4)

# Constrain the first and fourth quartiles to be monotone, using greedy sharpening method.
x <- rt(50, df=3)
scKDE <- scdensity(x, bw="SJ", adjust=0.5, constraint=c("monotoneL", "monotoneR"),
                   opts=list(verbose=TRUE, leftTail=25, rightTail=75), method="greedy")
plot(scKDE)

# Compare unimodal, twoInflections, and twoInflections+ constraints
x <- rnorm(100)
h <- 0.5 * bw.SJ(x)
fhat1 <- scdensity(x, bw=h, constraint="unimodal")
fhat2 <- scdensity(x, bw=h, constraint="twoInflections")
fhat3 <- scdensity(x, bw=h, constraint="twoInflections+")
plot(density(x, bw=h))
lines(fhat1$x, fhat1$y, col="red")
lines(fhat2$x, fhat2$y, col="blue")
lines(fhat3$x, fhat3$y, col="green", lwd=2)
```

SequentialLineMin *Minimize a function of r variables by sequential univariate searches.*

Description

The function seeks to minimize fcn, a scalar function of r variables. v0 is a starting solution and bounds is a 2-vector giving upper and lower limits for elements of the solution.

Usage

```
SequentialLineMin(fcn, bounds, v0, tol = .Machine$double.eps^0.25)
```

Arguments

fcn	A function with taking an r-vector as its first argument: call as fcn(v,...).
bounds	A 2-vector giving the upper and lower limits for elements of a solution.
v0	A starting solution, with increasing elements. An r-vector. Not used if r == 1.
tol	Tolerance passed to optimize.

Details

This algorithm is designed to search for solutions of the form $v = [v_1 v_2 \dots v_r]$, where bounds(1) $< v_1 < v_2 < \dots < v_r <$ bounds(2). It loops through the solution vector one variable at a time, and does a 1-D line search using optimize() for an improving value of that variable. So when optimizing v_i, it searches the interval (v_{i-1}, v_{i+1}) to maintain the increasing nature of v. The overall search terminates once a pass through all r elements of v fails to produce any changes to v.

Value

a list with elements:

minimizer	An r-vector containing the solution.
minimum	The objective function value at the solution.

Examples

```
fcn <- function(v) (v[1]+1)^2 + (v[2]-1)^2
SequentialLineMin(fcn, c(-5,5), c(-3,3))
```

summary.scdensity *Summary method for class* scdensity.

Description

Collects high-level information about the scdensity object and some descriptive statistics.

Usage

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

Arguments

object	An object of S3 class scensity.
...	Included for consistency with generic functions.

Index