Package 'eks'

July 8, 2025

Version 1.1.1

Date 2025-07-08

Title Tidy and Geospatial Kernel Smoothing

Maintainer Tarn Duong <tarn.duong@gmail.com>

Depends R (>= 2.10.0)

Imports colorspace, dplyr, geos, ggplot2 (>= 3.0.0), isoband, ks (>= 1.15.0), lwgeom, mapsf, sf

Suggests ggquiver, ggthemes, knitr, MASS, rmarkdown

VignetteBuilder knitr

Encoding UTF-8

Description Extensions of the kernel smoothing functions from the 'ks' package for compatibility with the tidyverse and geospatial ecosystems <doi:10.1007/s00180-024-01543-9>.

License GPL-2 | GPL-3

URL https://www.mvstat.net/mvksa/

NeedsCompilation no

Author Tarn Duong [aut, cre] (ORCID: <https://orcid.org/0000-0002-1198-3482>)

Repository CRAN

Date/Publication 2025-07-08 11:40:01 UTC

Contents

`eks-package` *eks*

Description

This package extends the functionality of the kernel smoothing functions from the **ks** package in base R to the tidyverse and to GIS (Geographical Information Systems) ecosystems.

Details

As the kernel smoothers from the **ks** package are prefixed as `k*`, their equivalents in **eks** are systematically named as follows:

- `tidy_k*` for 1- and 2-d tidy data
- `st_k*` for 2-d geospatial data.

The output data tibbles (tidy data frames provided by the **tibble** package) from `tidy_k*` can be visualised within the **ggplot2** graphical interface, using the usual layer functions and the custom ones supplied in this package. These `tidy_k*` functions are analogous to those in the **broom** and related packages, though the latter tend to focus on tidying the summary diagnostic output from model fitting (and not on tidying the underlying estimates themselves), whereas `tidy_k*` are more substantive since they do compute tidy estimates.

The output simple feature geometries (provided by the **sf** package) from `st_k*` can be visualised in the (i) **ggplot2** graphical interface using primarily the `geom_sf` layer function, or (ii) in the base R graphical interface using the `plot` method supplied in this package. These simple feature geometries can also be exported as standard geospatial formats (e.g. shapefile, GEOS geometry) for use in external GIS software such as ArcGIS and QGIS.

Author(s)

Tarn Duong

References

Chacon, J.E. & Duong, T. (2018) *Multivariate Kernel Smoothing and Its Applications*. Chapman & Hall/CRC, Boca Raton.

Duong, T. (2024) Statistical visualisation for tidy and geospatial data in R via kernel smoothing methods in the eks package. *Computational Statistics*. DOI: 10.1007/s00180-024-01543-9.

`ales_grid` *Gridded census data in Alès, France*

Description

The `ales_grid` data set contains census data of 806 1 km x 1 km grid cells which cover the city of Alès and its surrounding regions in southern France. The grid cell size was chosen to be a compromise between ensuring privacy and providing detailed public statistics from the 2019 census.

Usage

```
data(ales_grid)
```

Format

`ales_grid` is an sf object, whose geometry are 1 km x 1 km square polygons in the European-wide standard EPSG:3035 (ETRS89-extended / LAEA Europe) projection. There are 31 attributes collected by the French national statistical agency (Insee), e.g. `ind` is the number of individuals. See the URL below for more details.

Source

Insee (2023). Revenus, pauvreté et niveau de vie en 2019 - Données carroyées [In French]. https://www.insee.fr/fr/statistiques/7655464?sommaire=7655515. Accessed on 2025-01-16.

`contour` *Contour functions for tidy and geospatial kernel estimates*

Description

Contour functions for tidy and geospatial kernel estimates.

Usage

```
## S3 method for class 'tidy_ks'
contourLevels(x, ..., cont=c(25,50,75))
## S3 method for class 'sf_ks'
contourLevels(x, ..., cont=c(25,50,75))
contour_breaks(data, cont=c(25,50,75), n=3, group=FALSE, type="density")
st_get_contour(x, cont=c(25,50,75), breaks, disjoint=TRUE, digits)
```

Arguments

x, data	tidy kernel estimate (output from tidy_k*) or geospatial kernel estimate (output from st_k*)
cont	vector of contour levels. Default is c(25,50,75). Used only for level="density", "quantile".
n	number of contour levels. Default is 3. Used only for level="length", "natural".
type	type of contour levels: one of "density", "length", "quantile", "natural". Default is "density".
group	flag to compute contour levels per group. Default is FALSE.
breaks	tibble or vector of contour levels (e.g. output from contour_breaks)
disjoint	flag to compute disjoint contours. Default is TRUE.
digits	number of significant digits in output labels. If missing, default is 4.
...	other parameters (not implemented)

Details

By default, the 5%, 10%, ..., 95% contours are computed for an st_k* output, though a plot of 19 of them would be too crowded. st_get_contour selects a subset of these, as specified by cont. If a contour level in cont does not already exist or if absolute contour levels are specified in breaks, then the corresponding contours are computed. If disjoint=TRUE (default) then the contours are computed as a set of disjoint multipolygons: this allows for plotting without overlapping transparent colours. If disjoint=FALSE then the contours are overlapping and so their colours alpha-mixed, but they strictly satisfy the probabilistic definition, e.g. a 25% contour region is the smallest region that contains 25% of the probability mass defined by the kernel estimate, see geom_contour_ks.

Since these default probability contours are relative contour levels, they aren't suitable for producing a contour plot with fixed contour levels across all groups. It may require trial and error to obtain a single set of contour levels which is appropriate for all groups: one possible choice is provided by contour_breaks.

Value

–The output from contour_breaks is a tibble of the values of the contour breaks. If group=FALSE then a single set of contour breaks is returned. If group=TRUE then a set of contour breaks for each group is returned.

–The output from st_get_contour is an sf object of the contours as multipolygons, with added attributes

estimate	density value or factor with digits s.f.
contlabel	factor of contour probability level
contperc	factor of contour probability level with added %
contregion	factor of contour probability level with added $\leq$ or $\geq$

These are used to render the contours in the base R base graphics, and via the aesthetics in ggplot.

See Also

geom_contour_ks

Examples

```
library(ggplot2)
data(crabs, package="MASS")
## tidy data
crabs2 <- dplyr::select(crabs, FL, CW)
t1 <- tidy_kde(crabs2)
ggplot(t1, aes(x=FL, y=CW)) + geom_contour_filled_ks()

## geospatial data
crabs2s <- sf::st_as_sf(crabs2, coords=c("FL","CW"))
s1 <- st_kde(crabs2s)
s1_cont <- st_get_contour(s1, cont=seq(10,90, by=20))

## base R plot
vc <- function(.) colorspace::sequential_hcl(palette="viridis", n=.)
plot(s1_cont[,"contperc"], pal=vc)

## ggplot
ggplot() + scale_fill_viridis_d() +
    geom_sf(data=s1_cont, aes(fill=contperc))
```

---

geom_contour_ks *Contour and filled contour plots for tidy kernel estimates*

Description

Contour and filled contour plots for tidy kernel estimates for 2-dimensional data.

Usage

```
geom_contour_ks(mapping=NULL, data=NULL, stat="contour_ks",
    position="identity", ..., cont=c(25,50,75), contperc=TRUE,
    breaks=NULL, digits=NULL, show.legend=NA, inherit.aes=TRUE)
stat_contour_ks(mapping=NULL, data=NULL, geom="contour_ks",
    position="identity", ..., show.legend=NA, inherit.aes=TRUE)
geom_contour_filled_ks(mapping=NULL, data=NULL, stat="contour_filled_ks",
    position="identity", ..., cont=c(25,50,75), contperc=TRUE,
    breaks=NULL, transp_neutral=NULL, digits=NULL, show.legend=NA,
    inherit.aes=TRUE)
stat_contour_filled_ks(mapping=NULL, data=NULL, geom="contour_filled_ks",
    position="identity", ..., show.legend=NA, inherit.aes=TRUE)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer, as a string.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour="red" or size=3. They may also be parameters to the paired geom/stat.
cont	Vector of contour probabilities. Default value is c(25,50,75).
contperc	Deprecated.
transp_neutral	Flag to set neutral colour in diverging fill scale to be transparent.
breaks	Numeric vector to set the contour breaks e.g. output from contour_breaks. Overrides cont.
digits	Number of significant digits in legend keys. Default is 4.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders().
geom	The geometric object to use display the data.

Details

These layer functions are modifications of the standard layer functions ggplot2::geom_contour, geom_contour_filled and ggplot2::stat_contour, stat_contour_filled. Their usage and output are similar, except that they require a tidy kernel estimate as the input, rather than the observations themselves, and that the underlying choice of the contour levels is different. For most cases, geom_contour_ks is equivalent to geom_contour(stat="contour_ks"), and likewise for geom_contour_filled_ks.

The choice of the contour levels are based on probability contours. A 25% contour region is the smallest region that contains 25% of the probability mass defined by the kernel estimate. Probability contours offer a more intuitive approach to selecting the contour levels that reveal the pertinent

characteristics of the kernel estimates. See Chacon & Duong (2018, Chapter 2.2). They are specified by the `cont` parameter: the default value is `cont=c(25,50,75)`, which computes the upper quartile, median and lower quartile probability contours. To display legend labels with the percent symbol, `contperc=TRUE` is deprecated. The preferred syntax is `after_stat(contperc)` in the aesthetic, though for `geom_contour_filled_ks`, it is already the default, and can be omitted. See examples below.

Since these probability contours are computed for each group of the grouping variable in `data`, then these relative contour levels are different for each group. To produce a contour plot with fixed contour levels across all groups, then these can be supplied in `breaks`: a possible choice is provided by `contour_breaks`.

Value

Similar output as the standard layer functions `ggplot2::geom_contour`, `geom_contour_filled` and `ggplot2::stat_contour`, `stat_contour_filled`.

References

Chacon, J.E. & Duong, T. (2018) *Multivariate Kernel Smoothing and Its Applications*. Chapman & Hall/CRC, Boca Raton.

See Also

`contour_breaks`

Examples

```
library(ggplot2)
data(crabs, package="MASS")
crabs2 <- dplyr::select(crabs, FL, CW)
t1 <- tidy_kde(crabs2)
gt <- ggplot(t1, aes(x=FL, y=CW))
gt + geom_contour_filled_ks()
gt + geom_contour_ks(aes(colour=after_stat(contperc)))
```

---

geom_point_ks *Rug and scatter plots for tidy kernel estimates*

Description

Rug and scatter plots for tidy kernel estimates for 1- and 2-dimensional data.

Usage

```
geom_point_ks(mapping=NULL, data=NULL, stat="point_ks", position="identity",
    ..., na.rm=FALSE, jitter=FALSE, show.legend=NA, inherit.aes=TRUE)
stat_point_ks(mapping=NULL, data=NULL, geom="point_ks", position="identity",
    ..., na.rm=FALSE, show.legend=NA, inherit.aes=TRUE)
geom_rug_ks(mapping=NULL, data=NULL, stat="rug_ks", position="identity",
    ..., outside=FALSE, sides="bl", length=unit(0.03, "npc"), na.rm=FALSE,
    jitter=FALSE, show.legend=NA, inherit.aes=TRUE)
stat_rug_ks(mapping=NULL, data=NULL, geom="rug_ks", position="identity",
    ..., na.rm=FALSE, show.legend=NA, inherit.aes=TRUE)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer, as a string.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour="red" or size=3. They may also be parameters to the paired geom/stat.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
jitter	Flag to jitter data before plot. Default value is FALSE.
outside	logical that controls whether to move the rug tassels outside of the plot area. Default is off (FALSE). You will also need to use coord_cartesian(clip = "off"). When set to TRUE, also consider changing the sides argument to "tr". See examples.
sides	A string that controls which sides of the plot the rugs appear on. It can be set to a string containing any of "trbl", for top, right, bottom, and left.
length	A grid::unit() object that sets the length of the rug lines. Use scale expansion to avoid overplotting of data.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders().
geom	The geometric object to use display the data

Details

These layer functions are modifications of the standard layer functions ggplot2::geom_point, ggplot2::geom_rug and ggplot2::stat_point. Their usage and output are similar, except that they require a tidy kernel estimate as the input, rather than the observations themselves. For most cases, geom_rug_ks is equivalent to geom_rug(stat="rug_ks"), and likewise for geom_point_ks.

Value

Similar output as the standard layer functions ggplot2::geom_point, ggplot2::geom_rug and ggplot2::stat_point.

Examples

```
library(ggplot2)
data(crabs, package="MASS")

## rug plot for tidy 1-d kernel density estimate
crabs1 <- dplyr::select(crabs, FL)
t1 <- tidy_kde(crabs1)
g1 <- ggplot(t1, aes(x=FL)) + geom_line()
g1 + geom_rug_ks(colour=4)

## scatter plot for tidy 2-d kernel density estimate
crabs2 <- dplyr::select(crabs, FL, CW)
t2 <- tidy_kde(crabs2)
g2 <- ggplot(t2, aes(x=FL, y=CW))
g2 + geom_contour_ks(aes(colour=after_stat(estimate))) +
    geom_point_ks(colour=1)
```

---

grevilleasf *Geographical locations of Grevillea plants in Western Australia*

Description

The wa data set contains the polygon of the administrative boundary of Western Australia (excluding islands). The grevillea data set contains the locations of 22303 grevillea plants in Western Australia.

Usage

```
data(wa)
data(grevilleasf)
```

Format

`wa` is an sf object, whose geometry is the polygon in the EPSG:7850 (GDA2020/MGA zone 50) projection.

`grevilleasf` is an sf object with 22303 rows and 2 attributes. Each row corresponds to an observed plant. The first column is the full scientific name, the second is the species name. The geometry is the point location of the plant in the EPSG:7850 (GDA2020/MGA zone 50) projection. This is a superset of the `grevillea` dataset in the **ks** package.

Source

Atlas of Living Australia (2021). Grevillea occurrence. https://www.ala.org.au. Accessed on 2021-08-18.

Geoscape Australia (2021). WA State Boundary – Geoscape Administrative Boundaries. https://data.gov.au/data/dataset/wa-state-boundary-geoscape-administrative-boundaries. Accessed on 2021-08-18.

---

tidyst_as_kde *Tidy and geospatial quasi kernel density estimates from gridded data*

Description

Tidy and geospatial versions of quasi kernel density estimates for 1- and 2-dimensional gridded data.

Usage

```
tidy_as_kde(data, density, ...)
st_as_kde(x, attrib=1, density, cont, ...)
```

Arguments

`data`	data frame/tibble, where first d columns are evaluation points of an estimation grid, and last column is estimate value
`x`	sf object, where `attrib` is estimate value and estimation grid is rectangle polygon geometry
`attrib`	name or position of estimate variable. Default is 1.
`density`	flag to compute bona fide density. Default is TRUE.
`cont`	vector of contour probabilities. If missing, the default is seq(5, 95, by=5). Usually not required to be set explicitly by the user.
`...`	other parameters

Details

If `density=TRUE` then all the estimate values are non-negative. If `density=FALSE` then the estimate values can be positive or negative.

Value

The input gridded data should be a tidy form of a complete Cartesian product that forms a regular rectangular grid with no missing values or grid points/polygons. The input is converted to a quasi density estimate so that it can be subsequently treated like an output from `tidy_kde` or `st_kde`.

Examples

```
## tidy quasi density estimate
library(ggplot2)
data(faithfuld, package="ggplot2")
t1 <- tidy_as_kde(faithfuld)
## probabilistic density contour levels
ggplot(t1, aes(x=eruptions, y=waiting)) +
    geom_contour_filled_ks(colour=1, aes(fill=after_stat(contperc)))

## non-probabilistic contour levels in ggplot2::geom_contour_filled
ggplot(faithfuld, aes(x=eruptions, y=waiting, z=density)) +
    geom_contour_filled(bins=4) +
    colorspace::scale_fill_discrete_sequential(palette="Heat")

## geospatial quasi density estimate
## see example in ? st_intergrid
```

`tidyst_intergrid` *Tidy and geospatial interpolated grids*

Description

Tidy and geospatial interpolated grids for 2-dimensional gridded data.

Usage

```
tidy_intergrid(data, attrib, cellsize, verbose=FALSE)
st_intergrid(x, attrib, cellsize, verbose=FALSE)
```

Arguments

`data`, `x`	sf object, where attribute is estimate value and incomplete/irregular estimation grid is polygon geometry
`attrib`	name or position of estimate variable. Default is 1.
`cellsize`	cell size. If missing then automatically calculated from data.
`verbose`	flag for verbose output. Default is FALSE.

Details

Any missing grid cells are inferred so there are no gaps in the output, and the attribute value to set to 0. For any other grid cells with missing attribute values, the attributate is also set to 0. `tidy_overgrid`/`st_overgrid` is usually deployed on gridded data from third parties, where geometries are excluded/varying to reduce storage requirements, but `tidy_as_kde`/`st_as_kde` require complete regular rectangular grids.

Value

The input gridded data is interpolated to a complete regular rectangular grid.

Examples

```
## geospatial quasi density estimate
library(ggplot2)
data(ales_grid, package="eks")

## incomplete 1 km x 1 km grid
## ind = #individuals in grid cells
gs <- ggplot() + ggthemes::theme_map() +
   colorspace::scale_fill_continuous_sequential(palette="Heat", breaks=seq(0,6000,by=1000))
gs + geom_sf(data=ales_grid, aes(fill=ind))

## complete regular interpolated 1 km x 1 km grid
ales_sgrid <- st_intergrid(ales_grid, attrib="ind", cellsize=c(1000,1000))
gs + geom_sf(data=ales_sgrid, aes(fill=ind))

## geom_sf KDE plot
ales_skde <- st_as_kde(ales_sgrid)
ggplot(ales_skde) + ggthemes::theme_map() +
    geom_sf(data=st_get_contour(ales_skde), aes(fill=contperc))
```

---

tidyst_kcde *Tidy and geospatial kernel cumulative distribution and copula estimates*

Description

Tidy and geospatial versions of kernel cumulative distribution estimates for 1- and 2-dimensional data, and kernel copula estimates for 2-dimensional data.

Usage

```
tidy_kcde(data, ...)
tidy_kcopula(data, ...)
st_kcde(x, ...)
```

Arguments

`data`	data frame/tibble of data values
`x`	sf object with point geometry
`...`	other parameters in `ks::kcde`, `ks::kcopula` functions

Details

For details of the computation of the kernel distribution and copula estimates, and of the bandwidth selector procedures, see `?ks::kcde`, `?ks::kcopula`.

Value

The outputs from `*_kcde` have the same structure as the kernel density estimate from `*_kde`, except that `estimate` indicates the cumulative distribution rather than the density values. Likewise for `tidy_kcopula`.

Examples

```
library(ggplot2)
data(crabs, package="MASS")
## tidy 1-d distribution estimate
crabs1 <- dplyr::select(crabs, FL)
t1 <- tidy_kcde(crabs1)
gt2 <- ggplot(t1, aes(x=FL))
gt2 + geom_line(colour=1) + geom_rug_ks(colour=4)

## tidy 2-d copula estimate
crabs2 <- dplyr::select(crabs, FL, CW)
t2 <- tidy_kcopula(crabs2)
gt2 <- ggplot(t2, aes(x=FL, y=CW))
gt2 + geom_contour_filled_ks(colour=1, cont=seq(10,90,by=10))

## geospatial distribution estimate
data(wa)
data(grevilleasf)
paradoxa <- dplyr::filter(grevilleasf, species=="paradoxa")
paradoxa_bbox <- sf::st_bbox(c(xmin=4e5, xmax=8e5, ymin=6.4e6, ymax=6.65e6),
    crs=sf::st_crs(paradoxa))
xminb <- paradoxa_bbox[1:2]; xmaxb <- paradoxa_bbox[3:4]
paradoxa_bbox <- sf::st_as_sfc(paradoxa_bbox)
paradoxa_crop <- sf::st_filter(paradoxa, paradoxa_bbox)
s1 <- st_kcde(paradoxa_crop, xmin=xminb, xmax=xmaxb)

## base R plot
xlim <- c(1.2e5, 1.1e6); ylim <- c(6.1e6, 7.2e6)
plot(wa, xlim=xlim, ylim=ylim)
plot(paradoxa_bbox, add=TRUE, lty=3, lwd=2)
plot(s1, add=TRUE)

## geom_sf plot
gs1 <- ggplot(s1) + geom_sf(data=wa, fill=NA) + ggthemes::theme_map()
```

```
gs1 + geom_sf(data=paradoxa_bbox, linewidth=1.2, linetype="dotted", fill=NA) +
    geom_sf(data=st_get_contour(s1), aes(fill=contperc)) +
    coord_sf(xlim=xlim, ylim=ylim)
```

`tidyst_kcurv` *Tidy and geospatial kernel summary density curvature estimates*

Description

Tidy and geospatial versions of kernel summary density curvature estimates for 2-dimensional data.

Usage

```
tidy_kcurv(data, ...)
st_kcurv(x, ...)
```

Arguments

`data`	tidy kernel density curvature estimate (output from `tidy_kdde(deriv_order=2)`)
`x`	geospatial density curvature estimate (output from `st_kdde(deriv_order=2)`)
`...`	other parameters in `ks::kcurv` function

Details

A kernel density summary curvature estimate is a modification of a kernel density curvature estimate where the matrix of second order partial derivative values is summarised as a scalar value. For details of the computation of the kernel density summary curvature estimate, see `?ks::kcurv`. The bandwidth matrix of smoothing parameters is computed as in `ks::kdde(deriv_order=2)`.

Value

The output from `*_kcurv` have the same structure as the input kernel density curvature estimate from `*_kdde`, except that `estimate` indicates the summary curvature values rather than the density curvature values, and that `deriv_group` for each of the partial derivatives is collapsed into a single grouping.

Examples

```
## tidy kernel summary density curvature estimate
library(ggplot2)
theme_set(theme_bw())
data(crabs, package="MASS")
crabs2 <- dplyr::select(crabs, FL, CW)
t1 <- tidy_kdde(crabs2, deriv_order=2)
t2 <- tidy_kcurv(t1)
gt1 <- ggplot(t2, aes(x=FL, y=CW))
gt1 + geom_contour_filled_ks(colour=1)
gt1 + geom_contour_ks(aes(colour=after_stat(contperc)))
```

```
## geospatial kernel summary density curvature estimate
data(wa)
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
s1 <- st_kdde(hakeoides, deriv_order=2)
s2 <- st_kcurv(s1)

## base R plot
xlim <- c(1.2e5, 1.1e6); ylim <- c(6.1e6, 7.2e6)
plot(wa, xlim=xlim, ylim=ylim)
plot(s2, add=TRUE)

## geom_sf plot
gs1 <- ggplot(s2) + geom_sf(data=wa, fill=NA) + ggthemes::theme_map()
gs1 + geom_sf(data=st_get_contour(s2), aes(fill=contperc)) +
    coord_sf(xlim=xlim, ylim=ylim)
```

`tidyst_kda` *Tidy and geospatial kernel discrimination analysis (classification)*

Description

Tidy and geospatial versions of kernel discrimination analysis (classification) for 1- and 2-dimensional data.

Usage

```
tidy_kda(data, ...)
st_kda(x, ...)
```

Arguments

`data`	grouped tibble of data values
`x`	sf object with grouping attribute and with point geometry
`...`	other parameters in `ks::kda` function

Details

A kernel discriminant analysis (aka classification or supervised learning) assigns each grid point to the group with the highest density value, weighted by the prior probabilities.

The output from `*_kda` have the same structure as the kernel density estimate from `*_kde`, except that `estimate` is the weighted kernel density values at the grid points (weighted by `prior_prob`), and `label` becomes the KDA grouping variable that indicates to which of the groups the grid points belong. The output is a grouped tibble, grouped by the input grouping variable.

For details of the computation of the kernel discriminant analysis and the bandwidth selector procedure, see `?ks::kda`. The bandwidth matrix of smoothing parameters is computed as in `ks::kde` per group.

Value

–For tidy_kda, the output is an object of class tidy_ks, which is a tibble with columns:

x	evaluation points in x-axis (name is taken from 1st input variable in data)
y	evaluation points in y-axis (2-d) (name is taken from 2nd input variable in data)
estimate	weighted kernel density estimate values
prior_prob	prior probabilities for each group
ks	first row (within each group) contains the untidy kernel estimate from ks::kda
tks	short object class label derived from the ks object class
label	estimated KDA group label at (x,y)
group	grouping variable (same as input).

–For st_kda, the output is an object of class st_ks, which is a list with fields:

tidy_ks	tibble of simplified output (ks, tks, label, group) from tidy_kda
grid	sf object of grid of weighted kernel density estimate values, as polygons, with attributes estimate, label, group copied from the tidy_ks object
sf	sf object of 5%, 10% ..., 95% contour regions of weighted kernel density estimate, as multipolygons, with attributes contlabel derived from the contour level; and estimate, group copied from the tidy_ks object.

Examples

```
## tidy discriminant analysis (classification)
library(ggplot2)
data(hsct, package="ks")
hsct2 <- dplyr::as_tibble(hsct)
hsct2 <- dplyr::filter(hsct2, PE.Ly65Mac1 >0 & APC.CD45.2>0 &
    (subject==6 | subject==12))
hsct2 <- dplyr::select(hsct2, PE.Ly65Mac1, APC.CD45.2, subject)
hsct2 <- dplyr::mutate(hsct2, subject=factor(paste0("Subject #", subject),
    levels=c("Subject #6", "Subject #12")))
hsct2 <- dplyr::group_by(hsct2, subject)
hsct1 <-dplyr::select(hsct2, PE.Ly65Mac1, subject)

## tidy 1-d classification
t1 <- tidy_kda(hsct1)
gt1 <- ggplot(t1, aes(x=PE.Ly65Mac1))
gt1 + geom_line(aes(colour=subject)) +
    geom_rug(aes(colour=label), sides="b", linewidth=1.5)

## tidy 2-d classification
t2 <- tidy_kda(hsct2)
gt2 <- ggplot(t2, aes(x=PE.Ly65Mac1, y=APC.CD45.2))
gt2 + geom_contour_ks(aes(colour=subject)) +
    geom_tile(aes(fill=label), alpha=0.2) + coord_fixed()

## geospatial classification
data(wa)
```

```
data(grevilleasf)
grevillea_gr <- dplyr::filter(grevilleasf, species=="hakeoides" |
    species=="paradoxa")
grevillea_gr <- dplyr::mutate(grevillea_gr, species=factor(species))
grevillea_gr <- dplyr::group_by(grevillea_gr, species)
s1 <- st_kda(grevillea_gr)
s2 <- st_ksupp(st_kde(grevillea_gr))
s1$grid <- sf::st_filter(s1$grid, sf::st_convex_hull(sf::st_combine(s2$sf)))

## base R plot
xlim <- c(1.2e5, 1.1e6); ylim <- c(6.1e6, 7.2e6)
plot(wa, xlim=xlim, ylim=ylim)
plot(s1, which_geometry="grid", add=TRUE, alpha=0.1, border=NA, legend=FALSE)
plot(s1, add=TRUE, lwd=2)

## geom_sf plot
gs1 <- ggplot(s1) + geom_sf(data=wa, fill=NA) + ggthemes::theme_map() +
    geom_sf(data=s1$grid, aes(fill=label), alpha=0.1, colour=NA)
gs1 + geom_sf(data=st_get_contour(s1), aes(colour=species), fill=NA) +
    coord_sf(xlim=xlim, ylim=ylim)
```

`tidyst_kdcde` *Tidy and geospatial kernel deconvolved density estimates*

Description

Tidy and geospatial versions of kernel deconvolved density estimates for 1- and 2-dimensional data.

Usage

```
tidy_kdcde(data, ...)
st_kdcde(x,...)
```

Arguments

`data`	data frame/tibble of data values
`x`	sf object with point geometry
`...`	other parameters in `ks::kdcde` function

Details

A deconvolved kernel density estimate is a modification of the standard density estimate for data observed with error. This version is based on a weighted kernel density estimate. For details of the computation of the kernel deconvolved density estimate and the bandwidth selector procedure, see `?ks::kdcde`.

Value

The output from `*_kdcde` have the same structure as the standard kernel density estimate from `*_kde`.

Examples

```
## tidy 2-d deconvolved density estimate
library(ggplot2)
data(air, package="ks")
air <- na.omit(air[, c("time","co2","pm10")])
air <- dplyr::filter(air, time=="20:00")
air <- dplyr::select(air, co2, pm10)
## for details on computation of Sigma.air, see ?ks::kdcde
Sigma.air <- diag(c(6705.765, 957.664))

t1 <- tidy_kde(air)
t2 <- tidy_kdcde(air, Sigma=Sigma.air, reg=0.00021)
t3 <- rbind(t1, t2, labels=c("Standard KDE","Deconvolved KDE"))
tb <- contour_breaks(t3, cont=seq(10,90,by=20), group=FALSE)

## deconvolved estimate is more clearly bimodal than standard KDE
gt <- ggplot(t3, aes(x=co2, y=pm10))
gt + geom_contour_filled_ks(colour=1) + facet_wrap(~group)
gt + geom_contour_filled_ks(colour=1, breaks=tb) + facet_wrap(~group)
```

`tidyst_kdde` *Tidy and geospatial kernel density derivative estimates*

Description

Tidy and geospatial versions of kernel density derivative estimates for 1- and 2-dimensional data.

Usage

```
tidy_kdde(data, deriv_order=1, ...)
st_kdde(x, deriv_order=1, ...)
```

Arguments

`data`	data frame/tibble of data values
`deriv_order`	derivative order. Default is 1.
`x`	sf object with point geometry
`...`	other parameters in `ks::kdde` function

Details

The output from `*_kdde` have the same structure as the kernel density estimate from `*_kde`, except that `estimate` is the kernel density derivative values at the grid points, and the additional derived grouping variable `deriv_group` is the index of the partial derivative, e.g. "deriv (1,0)" and "deriv (0,1)" for a first order derivative for 2-d data. The output is a grouped tibble, grouped by the input grouping variable (if it exists) and by `deriv_group`.

For details of the computation of the kernel density derivative estimate and the bandwidth selector procedure, see ?ks::kdde.

In geom_contour_filled_ks and st_get_contour, the 100% contour is set to the bounding box of the data, and it is plotted in a transparent (if data is not missing) or neutral colour (if data is missing). For geom_contour_filled_ks, it is difficult to avoid displaying the boundary of this bounding box , though for geom_sf, using aes(linetype=contline) achieves this.

See *_kde for details on the aesthetics based on contperc, contregion, estimate.

Value

–For tidy_kdde, the output is an object of class tidy_ks, which is a tibble with columns:

x	evaluation points in x-axis (name is taken from 1st input variable in data)
y	evaluation points in y-axis (2-d) (name is taken from 2nd input variable in data)
estimate	kernel density derivative estimate values
deriv_order	derivative order (same as input)
deriv_ind	index of partial derivative
ks	first row (within each group) contains the untidy kernel estimate from ks::kde
tks	short object class label derived from the ks object class
label	long object class label
group	grouping variable (if grouped input) (name is taken from grouping variable in data)
deriv_group	additional derived grouping variable on partial derivative indices.

–For st_kdde, the output is an object of class st_ks, which is a list with fields:

tidy_ks	tibble of simplified output (deriv_ind, ks, tks, label, group, deriv_group) from tidy_kdde
grid	sf object of grid of kernel density derivative estimate values, as polygons, with attributes estimate, deriv_ind, group, deriv_group copied from the tidy_ks object
sf	sf object of 5%, 10%, ..., 95% contour regions of the kernel density derivative estimate, as multipolygons, with attributes estimate, contlabel, contperc, contregion, contline derived from the contour level, and deriv_ind, group, deriv_group copied from the tidy_ks object.

Examples

```
library(ggplot2)
theme_set(theme_bw())
data(crabs, package="MASS")
## 1-d density curvature estimate
crabs1 <- dplyr::select(crabs, FL)
t1 <- tidy_kdde(crabs1, deriv_order=2)
gt1 <- ggplot(t1, aes(x=FL))
gt1 + geom_line(colour=1)
```

```
## 2-d density gradient estimate
crabs2 <- dplyr::select(crabs, FL, CW)
t2 <- tidy_kdde(crabs2, deriv_order=1)
tb1 <- contour_breaks(t2, group=FALSE)
tb2 <- contour_breaks(t2, group=TRUE)
gt2 <- ggplot(t2, aes(x=FL, y=CW))

## overall set of contour breaks suitable for both derivatives
## filled contour regions with density derivative labels w/o breaks
## displays bounding box by default
gt2 + geom_contour_filled_ks(data=t2, aes(fill=after_stat(contregion)),
    colour=1, breaks=tb1) + facet_wrap(~deriv_group)
## contour lines with density derivative labels with breaks
gt2 + geom_contour_ks(data=t2, aes(group=deriv_group,
    colour=after_stat(estimate)), breaks=tb1) + facet_wrap(~deriv_group)

## second partial derivative f^(0,1) only
t22 <- dplyr::filter(t2, deriv_ind==2)
tb22 <- dplyr::filter(tb2, deriv_ind==2)
## filled contour regions with density derivative labels with breaks
## aes(linetype=after_stat(contline)) suppresses displaying bounding box
gt2 + geom_contour_filled_ks(data=t22, aes(fill=after_stat(contregion),
    linetype=after_stat(contline)), colour=1, breaks=tb22)
## contour lines with density derivative labels with breaks
gt2 + geom_contour_ks(data=t22, aes(colour=after_stat(estimate)),
    breaks=tb22)

## geospatial density derivative estimate
data(wa)
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
s1 <- st_kdde(hakeoides, deriv_order=1)
## overall set of contour breaks suitable for both derivatives
sb1 <- contour_breaks(s1, group=FALSE)
s1_cont1 <- st_get_contour(s1, breaks=sb1)
## set of contour breaks per derivative
sb2 <- contour_breaks(s1, group=TRUE)
sb2 <- dplyr::filter(sb2, deriv_ind==2)
s1_cont2 <- st_get_contour(s1)
s1_cont2 <- dplyr::filter(s1_cont2, deriv_ind==2)

## base R plot
## filled contour regions with density derivative labels with breaks
xlim <- c(1.2e5, 1.1e6); ylim <- c(6.1e6, 7.2e6)
plot(wa, xlim=xlim, ylim=ylim)
plot(s1, add=TRUE, which_deriv_ind=2, breaks=sb2)

## geom_sf plot
glim <- coord_sf(xlim=xlim, ylim=ylim)
gs <- ggplot(s1) + geom_sf(data=wa, fill=NA) + ggthemes::theme_map()
## filled contours with density derivative "percentage" labels w/o breaks
## aes(linetype=contline) suppresses displaying bounding box
gs + geom_sf(data=s1_cont2, aes(fill=contregion, linetype=contline),
```

```
    colour=1) + scale_linetype_manual(values=c(0,1)) + glim

## facet wrapped geom_sf plot for each partial derivative
## filled contours with density derivative labels with breaks
gs + geom_sf(data=s1_cont1, aes(fill=contregion, linetype=contline)) +
    scale_linetype_manual(values=c(0,1)) + glim + facet_wrap(~deriv_group)
```

`tidyst_kde` *Tidy and geospatial kernel density estimates*

Description

Tidy and geospatial versions of kernel density estimates for 1- and 2-dimensional data.

Usage

```
tidy_kde(data, ...)
st_kde(x, ...)
```

Arguments

`data`	data frame/tibble of data values
`x`	sf object with point geometry
`...`	other parameters in `ks::kde` function

Details

For `tidy_kde`, the first columns of the output tibble are copied from `aes(x)` (1-d) or `aes(x,y)` (2-d). These columns are the evaluation grid points. The `estimate` column is the kernel density values at these grid points. The `group` column is a copy of the grouping variable of the input data. The `ks` column is a copy of the untidy kernel estimate from `ks::kde`, since the calculations for the layer functions `geom_contour_ks`, `geom_contour_filled_ks` require both the observations `data` and the kernel estimate as a `kde` object. For this reason, it is advised to compute a tidy kernel estimate first and then to create a `ggplot` with this tidy kernel estimate as the default `data` in the layer.

For `st_kde`, the output list contains the field `tidy_ks` which is the output from `tidy_ks`. The field `grid` is the kernel estimate values, with rectangular polygons. The field `sf` is the 5%, 10%, ..., 95% probability contour regions as multipolygons, with the derived attribute `contlabel` = `100%-cont`.

The structure of the `tidy_kde` output is inherited from the input, i.e. if the input is a data frame/(grouped) tibble then the output is a data frame/(grouped) tibble. Likewise for the `sf` object outputs for `st_kde`.

The default bandwidth matrix is the unconstrained plug-in selector `ks::Hpi`, which is suitable for a wide range of data sets, since it is not restrained to smoothing along the coordinate axes. This produces a kernel estimate which is more representative of the data than with the default bandwidth in `geom_density_2d` and `geom_density_2d_filled`. For further details of the computation of the kernel density estimate and the bandwidth selector procedure, see `?ks::kde`.

If breaks is missing, then for geom_contour_ks, geom_filled_contour_ks, the density contours at cont percent are displayed, and contperc is the default aesthetic, which shows legend keys with percentages. For **eks** $\geq$ 1.1.0, the aesthetic label_percent(contlabel) is deprecated in favour of contperc. The other derived quantities contregion, estimate computed via stat_filled_contour_ks, stat_contour_ks, are also suitable aesthetics and these display the contour levels and $\geq$ contour levels in the legend keys respectively. For geom_sf, the default aesthetic is not suitable, and should be chosen from contperc, contregion, estimate, to produce analogous plots.

If breaks is not missing, then the density contours at breaks are displayed, and estimate is the default aesthetic. The other derived quantity contregion is also suitable, whereas contperc is NA so is not suitable.

For facetted plots, it is recommended to set the same breaks for all facets, and to use estimate or contregion as aesthetics. The same density contour percentages cont can lead to different contour levels for different densities in different facets which is not suitable for a facetted plot.

Value

–For tidy_kde, the output is an object of class tidy_ks, which is a tibble with columns:

x	evaluation points in x-axis (name is taken from 1st input variable in data)
y	evaluation points in y-axis (2-d) (name is taken from 2nd input variable in data)
estimate	kernel estimate values
ks	first row (within each group) contains the untidy kernel estimate from ks::kde
tks	short object class label derived from the ks object class
label	long object class label
group	grouping variable (if grouped input) (name is taken from grouping variable in data).

–For st_kde, the output is an object of class st_ks, which is a list with fields:

tidy_ks	tibble of simplified output (ks, tks, label, group) from tidy_kde
grid	sf object of grid of kernel density estimate values, as polygons, with attributes estimate, group copied from the tidy_ks object
sf	sf object of 5%, 10%, ..., 95% contour regions of kernel density estimate, as multipolygons, with attributes estimate, contlabel, contperc, contregion derived from the contour level, and group copied from the tidy_ks object.

Examples

```
## tidy density estimates
library(ggplot2)
data(crabs, package="MASS")
## tidy 1-d density estimate per species
crabs1 <- dplyr::select(crabs, FL, sp)
crabs1 <- dplyr::group_by(crabs1, sp)
t1 <- tidy_kde(crabs1)
gt1 <- ggplot(t1, aes(x=FL))
gt1 + geom_line(colour=1) + geom_rug_ks(colour=4) + facet_wrap(~sp)
```

```
## tidy 2-d density estimate
## suitable smoothing matrix gives bimodal estimate
crabs2 <- dplyr::select(crabs, FL, CW)
t2 <- tidy_kde(crabs2)
gt2 <- ggplot(t2, aes(x=FL, y=CW))

## filled contour regions with density percentage labels
gt2 + geom_contour_filled_ks(colour=1)
## filled contour regions with density level labels
gt2 + geom_contour_filled_ks(colour=1, aes(fill=after_stat(estimate)))
gt2 + geom_contour_filled_ks(colour=1, aes(fill=after_stat(contregion)))

## contour lines only with density percentage labels
gt2 + geom_contour_ks(aes(colour=after_stat(contperc)))
## contour lines only with density level labels
gt2 + geom_contour_ks(aes(colour=after_stat(estimate)))
gt2 + geom_contour_ks(aes(colour=after_stat(contregion)))

## default smoothing in geom_density_2d_filled() gives unimodal estimate
gt3 <- ggplot(crabs2, aes(x=FL, y=CW))
gt3 + geom_density_2d_filled(bins=4, colour=1) +
    colorspace::scale_fill_discrete_sequential(palette="Heat") +
    guides(fill=guide_legend(title="Density", reverse=TRUE))

## facet wrapped geom_sf plot with fixed contour levels for all facets
crabs3 <- dplyr::select(crabs, FL, CW, sex)
t3 <- tidy_kde(dplyr::group_by(crabs3, sex))
tb <- contour_breaks(t3, group=FALSE)
gt4 <- ggplot(t3, aes(x=FL, y=CW)) + facet_wrap(~sex)
## filled contours
gt4 + geom_contour_filled_ks(colour=1, breaks=tb, aes(fill=after_stat(estimate)))
gt4 + geom_contour_filled_ks(colour=1, breaks=tb, aes(fill=after_stat(contregion)))
## contour lines
gt4 + geom_contour_ks(breaks=tb, aes(colour=after_stat(estimate)))
gt4 + geom_contour_ks(breaks=tb, aes(colour=after_stat(contregion)))

## geospatial density estimate
data(wa)
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
hakeoides_coord <- data.frame(sf::st_coordinates(hakeoides))
s1 <- st_kde(hakeoides)

## base R plot
## filled contour regions with density percentage labels
xlim <- c(1.2e5, 1.1e6); ylim <- c(6.1e6, 7.2e6)
plot(wa, xlim=xlim, ylim=ylim)
plot(s1, add=TRUE)

## geom_sf plot
## suitable smoothing matrix gives optimally smoothed contours
gs1 <- ggplot(s1) + geom_sf(data=wa, fill=NA) + ggthemes::theme_map()
```

```
glim <- coord_sf(xlim=xlim, ylim=ylim)
## filled contour regions with density percentage labels
gs1 + geom_sf(data=st_get_contour(s1), aes(fill=contperc)) + glim
## filled contour regions with density level labels
gs1 + geom_sf(data=st_get_contour(s1), aes(fill=estimate)) + glim
gs1 + geom_sf(data=st_get_contour(s1), aes(fill=contregion)) + glim

## default smoothing in geom_density_2d_filled() is oversmoothed
ggplot(hakeoides_coord) + geom_sf(data=wa, fill=NA) +
    ggthemes::theme_map() +
    colorspace::scale_fill_discrete_sequential(palette="Heat") +
    geom_density_2d_filled(aes(x=X, y=Y), bins=4, colour=1) +
    guides(fill=guide_legend(title="Density", reverse=TRUE)) + glim

## facet wrapped geom_sf plot with fixed contour levels for all facets
grevillea_gr <- dplyr::filter(grevilleasf, species=="hakeoides" |
    species=="paradoxa")
grevillea_gr <- dplyr::mutate(grevillea_gr, species=factor(species))
grevillea_gr <- dplyr::group_by(grevillea_gr, species)
s2 <- st_kde(grevillea_gr)
sb <- contour_breaks(s2, group=TRUE)

gs3 <- ggplot(s2) + geom_sf(data=wa, fill=NA) + ggthemes::theme_map() +
    facet_wrap(~species)
## filled contour regions with density percentage labels w/o breaks
gs3 + geom_sf(data=st_get_contour(s2), aes(fill=contperc)) +
    glim
## filled contour regions with density level labels with breaks
gs3 + geom_sf(data=st_get_contour(s2, breaks=sb), aes(fill=estimate)) +
    glim
gs3 + geom_sf(data=st_get_contour(s2, breaks=sb), aes(fill=contregion)) +
    glim

## Not run: ## export as geopackage for external GIS software
sf::write_sf(wa, dsn="grevillea.gpkg", layer="wa")
sf::write_sf(hakeoides, dsn="grevillea.gpkg", layer="hakeoides")
sf::write_sf(gs1_cont, dsn="grevillea.gpkg", layer="hakeoides_cont")
sf::write_sf(s1$grid, dsn="grevillea.gpkg", layer="hakeoides_grid")
## End(Not run)
```

`tidyst_kde_balloon` *Tidy and geospatial kernel density estimates with variable kernels*

Description

Tidy and geospatial versions of kernel density estimates with variable kernels for 2-dimensional data.

Usage

```
tidy_kde_balloon(data, ...)
tidy_kde_sp(data, ...)
st_kde_balloon(x, ...)
st_kde_sp(x, ...)
```

Arguments

data	data frame/tibble of data values
x	sf object with point geometry
...	other parameters in ks::kde.balloon, ks::kde.sp functions

Details

A variable kernel density estimate is a modification of the standard density estimate where the bandwidth matrix is variable. There are two main types: balloon kernel estimates (*_kde_balloon) where the bandwidth varies with the grid point, and sample point kernel estimates (*_kde_sp) where the bandwidth varies with the data points. For details of the computation of the variable kernel estimates and of the bandwidth selector procedure, see ks::kde.balloon, ks::kde.sp.

Value

The outputs from *_kde_balloon, *_kde_sp have the same structure as the standard kernel density estimate from *_kde.

Examples

```
## tidy variable density estimates
library(ggplot2)
data(worldbank, package="ks")
worldbank <- dplyr::as_tibble(worldbank)
wb2 <- na.omit(worldbank[,c("GDP.growth", "inflation")])
xmin <- c(-70,-25); xmax <- c(25,70)

## standard density estimate
t1 <- tidy_kde(wb2, xmin=xmin, xmax=xmax)
## sample point variable density estimate
t2 <- tidy_kde_sp(wb2, xmin=xmin, xmax=xmax)
## balloon variable density estimate
## gridsize=c(21,21) only for illustrative purposes
t3 <- tidy_kde_balloon(wb2, xmin=xmin, xmax=xmax, gridsize=c(21,21))
t4 <- rbind(t1, t2, t3, labels=c("Standard KDE","Sample point KDE","Balloon KDE"))
tb <- contour_breaks(t4, group=FALSE)
gt <- ggplot(t4, aes(x=GDP.growth, y=inflation))
gt + geom_contour_filled_ks(breaks=tb, colour=1) + facet_wrap(~group)

## geospatial variable density estimates
data(wa)
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
```

```
## standard density estimate
s1 <- st_kde(hakeoides)
## sample point variable density estimate
s2 <- st_kde_sp(hakeoides)
s3 <- c(s1, s2, labels=c("Standard KDE","Sample point KDE"))
sb <- contour_breaks(s3, group=FALSE)
bcols <- colorspace::sequential_hcl(nrow(sb), palette="Heat", rev=TRUE)

## base R plot
xlim <- c(1.2e5, 1.1e6); ylim <- c(6.1e6, 7.2e6)
layout(matrix(1:2, ncol=2))
par(mar=(c(0,0,0,0)))
plot(wa, xlim=xlim, ylim=ylim)
plot(s1, add=TRUE, col=bcols[1:2], breaks=sb, legend=FALSE)
par(mar=(c(0,0,0,0)))
plot(wa, xlim=xlim, ylim=ylim)
plot(s2, add=TRUE, col=bcols, breaks=sb)
layout(1)

## geom_sf plot
gs <- ggplot(s3) + geom_sf(data=wa, fill=NA) + ggthemes::theme_map()
gs + geom_sf(data=st_get_contour(s3, breaks=sb), aes(fill=estimate)) +
    coord_sf(xlim=xlim, ylim=ylim) + facet_wrap(~group)
```

`tidyst_kde_boundary` *Tidy and geospatial kernel density estimates with boundary and truncated kernels*

Description

Tidy and geospatial versions of kernel density estimates with boundary and truncated kernels for 1- and 2-dimensional data.

Usage

```
tidy_kde_boundary(data, ...)
tidy_kde_truncate(data, boundary, ...)
st_kde_boundary(x, ...)
st_kde_truncate(x, boundary, ...)
```

Arguments

`data`	data frame/tibble of data values
`x`	sf object with point geometry
`boundary`	data frame/sf point geometry of boundary
`...`	other parameters in `ks::kde.boundary` function

Details

A boundary kernel density estimate is a modification of the standard density estimate for bounded data. There are two main types: beta kernels (`boundary.kernel="beta"`) and linear kernels (`boundary.kernel="linear"`). For details of the computation of the boundary kernel estimates and of the bandwidth selector procedure, see `ks::kde.boundary`. Currently only rectangular boundaries are supported, as defined by `xmin` x `xmax`.

A truncated kernel density estimate is a modification of the standard density estimate for bounded data. All the probability mass outside of `boundary` is set to zero and re-assigned over the regions inside in the boundary. The boundary can be any polygon. For further details of the computation of the truncated kernel estimate, see `ks::kde.truncate`.

For details of the computation of the boundary kernel estimates and the truncated kernel density estimates, and of the bandwidth selector procedure, see `ks::kde.boundary`, `ks::kde.truncate`.

Value

The outputs from `*_kde_boundary`, `*_kde_truncate` have the same structure as the standard kernel density estimate from `*_kde`.

Examples

```
## tidy boundary density estimates
library(ggplot2)
data(worldbank, package="ks")
worldbank <- dplyr::as_tibble(worldbank)
## percentage data is bounded on [0,100] x [0,100]
wb2 <- na.omit(worldbank[,c("internet", "ag.value")])
xmin <- c(0,0); xmax <- c(100,100)
rectb <- data.frame(x=c(0,100,100,0,0), y=c(0,0,100,100,0))

## standard density estimate
t1 <- tidy_kde(wb2)
## tidy truncated density estimate
t2 <- tidy_kde_truncate(wb2, boundary=rectb)
tt <- rbind(t1, t2, labels=c("Standard KDE", "Truncated KDE"))
tb <- contour_breaks(tt, group=FALSE)

## standard estimate exceeds boundary but not truncated estimate
gr <- geom_polygon(data=rectb, aes(x=x,y=y), inherit.aes=FALSE, fill=NA,
    colour=1, linetype="dashed")
gt <- ggplot(tt, aes(x=internet, y=ag.value))
gt + geom_contour_filled_ks(colour=1, breaks=tb) + gr + facet_wrap(~group)

## linear boundary density estimate
## beta boundary density estimate
t3 <- tidy_kde_boundary(wb2, boundary.kernel="beta", xmin=xmin, xmax=xmax)
t4 <- tidy_kde_boundary(wb2, boundary.kernel="linear", xmin=xmin, xmax=xmax)
t5 <- rbind(t1, t2, t3, t4, labels=c("Standard KDE", "Truncated KDE",
    "Beta bd KDE", "Linear bd KDE"))
tb <- contour_breaks(t5, group=FALSE)
## standard estimate exceeds boundary but not boundary or truncated estimates
```

```
gt + geom_contour_filled_ks(data=t5, colour=1, breaks=tb) + gr + facet_wrap(~group)

## geospatial boundary density estimates
data(wa)
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
hakeoides_bbox <- sf::st_bbox(c(xmin=4e5, xmax=5.7e5, ymin=6.47e6, ymax=7e6),
    crs=sf::st_crs(hakeoides))
xminb <- hakeoides_bbox[1:2]; xmaxb <- hakeoides_bbox[3:4]
hakeoides_bbox <- sf::st_as_sfc(hakeoides_bbox)
hakeoides_crop <- sf::st_filter(hakeoides, hakeoides_bbox)

s1 <- st_kde(hakeoides_crop)
s2 <- st_kde_boundary(hakeoides_crop, boundary.kernel="beta",
    xmin=xminb, xmax=xmaxb)
s3 <- st_kde_boundary(hakeoides_crop, boundary.kernel="linear",
    xmin=xminb, xmax=xmaxb)
## geospatial truncated density estimate
s4 <- st_kde_truncate(x=hakeoides_crop, boundary=hakeoides_bbox)
s5 <- rbind(s1, s2, s3, s4, labels=c("Standard KDE", "Beta bd KDE",
    "Linear bd KDE", "Truncated KDE"))

## base R plots
xlim <- c(1.2e5, 1.1e6); ylim <- c(6.1e6, 7.2e6)
cols <- colorspace::qualitative_hcl(n=4, palette="Set2")

layout(matrix(c(1,3,2,4), ncol=2))
plot(wa, xlim=xlim, ylim=ylim)
plot(hakeoides_bbox, add=TRUE, lty=3, lwd=2)
plot(s1, add=TRUE, border=cols[1], col=NA, legend=FALSE)

plot(wa, xlim=xlim, ylim=ylim)
plot(hakeoides_bbox, add=TRUE, lty=3, lwd=2)
plot(s2, add=TRUE, border=cols[2], col=NA, legend=FALSE)

plot(wa, xlim=xlim, ylim=ylim)
plot(hakeoides_bbox, add=TRUE, lty=3, lwd=2)
plot(s3, add=TRUE, border=cols[3], col=NA, legend=FALSE)
mapsf::mf_legend(type="symb", val=c("Standard KDE","Beta bd KDE",
    "Linear bd KDE", "Truncated KDE"), pal=cols, cex=rep(3,4),
    pch=rep("-",4), title="Density",  pos="bottomleft")

plot(wa, xlim=xlim, ylim=ylim)
plot(hakeoides_bbox, add=TRUE, lty=3, lwd=2)
plot(s4, add=TRUE, border=cols[4], col=NA, legend=FALSE)
layout(1)

## geom_sf plots
gs <- ggplot() + geom_sf(data=wa, fill=NA) +
    geom_sf(data=hakeoides_bbox, linetype="dotted", fill=NA) +
    ggthemes::theme_map() +
    colorspace::scale_colour_discrete_qualitative(palette="Set2")
gs + geom_sf(data=st_get_contour(s5), aes(colour=group), fill=NA) +
```

```
coord_sf(xlim=xlim, ylim=ylim) +
guides(colour=guide_legend(title="Density")) + facet_wrap(~group)
```

`tidyst_kde_local_test` *Tidy and geospatial kernel density based local two-sample comparison tests*

Description

Tidy and geospatial versions of kernel density based local two-sample comparison tests for 1- and 2-dimensional data.

Usage

```
tidy_kde_local_test(data1, data2, labels, ...)
st_kde_local_test(x1, x2, labels, ...)
```

Arguments

`data1`, `data2`	data frames/tibbles of data values
`x1`, `x2`	sf objects with point geometry
`labels`	flag or vector of strings for legend labels
`...`	other parameters in `ks::kde.local.test` function

Details

A kernel local density based two-sample comparison is a modification of the standard kernel density estimate where the two data samples are compared. A Hochberg procedure is employed to control the significance level for multiple comparison tests.

For details of the computation of the kernel local density based two-sample comparison test and the bandwidth selector procedure, see `?ks::kde.local.test`. The bandwidth matrix of smoothing parameters is computed as in `ks::kde` per data sample.

If `labels` is missing, then the first sample label is taken from `x1`, and the second sample label from `x2`. If `labels="default"` then these are "f1" and "f2". Otherwise, they are assigned to the values of the input vector of strings.

Value

The output has the same structure as the kernel density estimate from `*_kde`, except that `estimate` is the difference between the density values `data1-data2` rather than the density values, and `label` becomes an indicator factor of the local comparison test result: "f1<f2" = `data1` < `data2`, 0 = `data1` = `data2`, "f2>f1" = `data1` > `data2`.

The output from `st_kde_local_test` has two contours, with `contlabel=-50` (for f1<f2) and `contlabel=50` (for f1>f2), as multipolygons which delimit the significant difference regions.

Examples

```
## tidy local test between unsuccessful and successful grafts
library(ggplot2)
data(hsct, package="ks")
hsct <- dplyr::as_tibble(hsct)
hsct <- dplyr::filter(hsct, PE.Ly65Mac1 >0 & APC.CD45.2>0)
hsct6 <- dplyr::filter(hsct, subject==6)   ## unsuccessful graft
hsct6 <- dplyr::select(hsct6, PE.Ly65Mac1, APC.CD45.2)
hsct12 <- dplyr::filter(hsct, subject==12) ## successful graft
hsct12 <- dplyr::select(hsct12, PE.Ly65Mac1, APC.CD45.2)
t1 <- tidy_kde_local_test(data1=hsct6, data2=hsct12)
gt <- ggplot(t1, aes(x=PE.Ly65Mac1, y=APC.CD45.2)) + coord_fixed()
gt + geom_contour_filled_ks(aes(fill=after_stat(contregion)))

t2 <- tidy_kfs(hsct12)
gt + geom_contour_ks(data=t1, aes(colour=after_stat(contregion))) +
    geom_contour_filled_ks(data=t2, aes(fill=after_stat(contregion))) +
    colorspace::scale_fill_discrete_qualitative(palette="Dark2",
        rev=TRUE, nmax=2) +
    guides(fill=guide_legend(title="Signif curv"))

## geospatial local test between Grevillea species
data(wa)
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
paradoxa <- dplyr::filter(grevilleasf, species=="paradoxa")
s1 <- st_kde_local_test(x1=hakeoides, x2=paradoxa)
s2 <- st_kfs(hakeoides)

## base R plot
xlim <- c(1.2e5, 1.1e6); ylim <- c(6.1e6, 7.2e6)
plot(wa, xlim=xlim, ylim=ylim)
plot(s1, add=TRUE)
plot(s2, add=TRUE, pos="bottom")

## geom_sf plot
gs <- ggplot(s1) + geom_sf(data=wa, fill=NA) + ggthemes::theme_map()
gs + geom_sf(data=st_get_contour(s1), aes(colour=contregion), fill=NA) +
    geom_sf(data=st_get_contour(s2), aes(fill=contregion)) +
    guides(fill=guide_legend(title="Signif curv")) +
    scale_fill_manual(values=7) + coord_sf(xlim=xlim, ylim=ylim)
```

`tidyst_kdr` *Tidy and geospatial kernel density ridge estimates*

Description

Tidy and geospatial versions of kernel density ridge estimates for 2-dimensional data.

Usage

```
tidy_kdr(data, dTolerance, ...)
st_kdr(x, dTolerance, ...)
```

Arguments

data	data frame/tibble of data values
x	sf object with point geometry
dTolerance	tolerance parameter in sf::st_simplify for reducing complexity of density ridge
...	other parameters in ks::kdr function

Details

A density ridge can be interpreted as the line connecting the peaks in the kernel density estimate, like for a mountain range. It can also be interpreted as the filament generalisation of 2-d principal components. For details of the computation and the bandwidth selection procedure of the kernel density ridge estimate, see ?ks::kdr. The bandwidth matrix of smoothing parameters is computed as in ks::kdde(deriv_order=2).

To reduce the complexity of the density ridge, a call to sf::st_simplify(,dTolerance) is made. If dTolerance is missing, then it defaults to approximately the mean distance between each pair of consecutive points in each segment of the density ridge. If dTolerance=0 then this step of Ramer-Douglas-Peucker simplification is not carried out.

Value

The output from *_kdr have the same structure as the kernel density estimate from *_kde, except that x,y indicate the points on the density ridge, rather than the grid points themselves, and estimate becomes NA. For st_kdr, the density ridge is stored as a multipoints sf object.

Examples

```
## tidy density ridge estimate
library(ggplot2)
data(cardio, package="ks")
cardio <- dplyr::as_tibble(cardio[,c("ASTV","Mean")])
set.seed(8192)
cardio <- cardio[sample(1:nrow(cardio), round(nrow(cardio)/4,0)),]
## gridsize=c(21,21) is for illustrative purposes only
## remove for more complete KDR
t1 <- tidy_kdr(cardio, gridsize=c(21,21))
gt <- ggplot(t1, aes(x=ASTV, y=Mean))
gt + geom_point_ks(colour=3) +
    geom_path(aes(colour=label, group=segment))

## geospatial density ridge estimate
data(wa)
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
```

```
## gridsize=c(21,21) is for illustrative purposes only
## remove for more complete KDR
s1 <- st_kdr(hakeoides, gridsize=c(21,21))

## base R plot
xlim <- c(1.2e5, 1.1e6); ylim <- c(6.1e6, 7.2e6)
plot(wa, xlim=xlim, ylim=ylim)
plot(sf::st_geometry(hakeoides), add=TRUE, col=3, pch=16, cex=0.5)
plot(s1, add=TRUE)

## geom_sf plot
gs <- ggplot(s1) + geom_sf(data=wa, fill=NA) + ggthemes::theme_map()
gs + geom_sf(data=hakeoides, colour=3, alpha=0.5) +
    geom_sf(data=s1$sf, aes(colour=contregion)) +
    coord_sf(xlim=xlim, ylim=ylim)
```

`tidyst_kfs` *Tidy and geospatial kernel feature significance*

Description

Tidy and geospatial versions of kernel feature significance for 1- and 2-dimensional data.

Usage

```
tidy_kfs(data, ...)
st_kfs(x, ...)
```

Arguments

`data`	data frame/tibble of data values
`x`	sf object with point geometry
`...`	other parameters in `ks::kfs` function

Details

A significant kernel curvature region consist of all points whose density curvature value is significantly different less than zero (i.e. forms a bump surrounding a local maximum). A Hochberg procedure is employed to control the significance level for multiple significance tests.

For details of the computation of the significant kernel curvature regions, see `?ks::kfs`. The bandwidth matrix of smoothing parameters is computed as in `ks::kdde(deriv_order=2)`.

Value

The output from `tidy_kfs` has the same structure as the kernel density estimate from `tidy_kde`, except that all values of `estimate` outside of the significant curvature regions are set to zero, and the `label` indicates whether the corresponding `x,y` point is inside a significant curvature region.

The output from `st_kfs` has a single contour, with `contlabel=50`, as a multipolygon which delimits its significant curvature regions.

Examples

```
## tidy significant curvature regions
library(ggplot2)
data(hsct, package="ks")
hsct <- dplyr::as_tibble(hsct)
hsct <- dplyr::filter(hsct, PE.Ly65Mac1>0 & APC.CD45.2>0)
hsct12 <- dplyr::filter(hsct, subject==12)
hsct12 <- dplyr::select(hsct12, PE.Ly65Mac1, APC.CD45.2)
t1 <- tidy_kde(hsct12)
t2 <- tidy_kfs(hsct12)
gt <- ggplot(t2, aes(x=PE.Ly65Mac1, y=APC.CD45.2)) +
    geom_contour_ks(data=t1, colour="grey50", cont=seq(10,90,by=10))
gt + geom_contour_filled_ks(aes(fill=after_stat(contregion)), colour=1)
gt + geom_contour_ks(aes(colour=after_stat(contregion)))

## geospatial significant curvature regions
data(wa)
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
s1 <- st_kfs(hakeoides)

## base R plot
xlim <- c(1.2e5, 1.1e6); ylim <- c(6.1e6, 7.2e6)
plot(wa, xlim=xlim, ylim=ylim)
plot(s1, add=TRUE)

## geom_sf plot
gs <- ggplot(s1) + geom_sf(data=wa, fill=NA) + ggthemes::theme_map()
gs + geom_sf(data=st_get_contour(s1), aes(fill=contregion)) +
    coord_sf(xlim=xlim, ylim=ylim)
```

`tidyst_kms` *Tidy and geospatial kernel mean shift clustering*

Description

Tidy and geospatial versions of a kernel mean shift clustering for 1- and 2-dimensional data.

Usage

```
tidy_kms(data, ...)
st_kms(x, ...)
```

Arguments

`data`	data frame/tibble of data values
`x`	sf object with point geometry
`...`	other parameters in `ks::kms` function

Details

Mean shift clustering is a generalisation of k-means clustering (aka unsupervised learning) which allows for non-ellipsoidal clusters and does not require the number of clusters to be pre-specified. The mean shift clusters are determined by following the initial observations along the density gradient ascent paths to the cluster centre.

For details of the computation and the bandwidth selection procedure of the kernel mean shift clustering, see `?ks::kms`. The bandwidth matrix of smoothing parameters is computed as in `ks::kdde(deriv_order=1)`.

Value

The output from `*_kms` have the same structure as the kernel density estimate from `*_kde`, except that `x,y` indicate the data points rather than the grid points, and `estimate` indicates the mean shift cluster label of the data points, rather than the density values.

Examples

```
## tidy 2-d mean shift clustering
library(ggplot2)
data(crabs, package="MASS")
crabs2 <- dplyr::select(crabs, FL, CW)
t1 <- tidy_kms(crabs2)
## convex hulls of clusters
t2 <- dplyr::group_by(t1, label)
t2 <- dplyr::slice(t2, chull(FL,CW))

gt <- ggplot(t1, aes(x=FL, y=CW))
gt + geom_point(aes(colour=label)) +
    geom_polygon(data=t2, aes(fill=label), alpha=0.1, col=1, linetype="dotted")

## geospatial mean shift clustering
data(wa)
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
s1 <- st_kms(hakeoides)
## convex hulls of clusters
s2 <- dplyr::group_by(s1$sf, label)
s2 <- dplyr::summarise(s2, geometry=sf::st_combine(geometry))
s2 <- sf::st_convex_hull(s2)

## base R plot
xlim <- c(1.2e5, 1.1e6); ylim <- c(6.1e6, 7.2e6)
plot(wa, xlim=xlim, ylim=ylim)
plot(s1, add=TRUE, pch=16)
plot(s2, add=TRUE, lty=3, pal=function(.){
    colorspace::qualitative_hcl(n=., palette="Set2", alpha=0.15)})

## geom_sf plot
gs <- ggplot(s1) + geom_sf(data=wa, fill=NA) + ggthemes::theme_map()
gs + geom_sf(data=s1$sf, aes(colour=label), alpha=0.5) +
    geom_sf(data=s2, aes(fill=label), linetype="dotted", alpha=0.15) +
```

```
coord_sf(xlim=xlim, ylim=ylim)
```

`tidyst_kquiver` *Tidy and geospatial kernel density quiver estimate*

Description

Tidy and geospatial versions of a kernel density quiver estimate for 2-dimensional data.

Usage

```
tidy_kquiver(data, thin=5, transf=1/4, neg.grad=FALSE)
st_kquiver(x, thin=5, transf=1/4, neg.grad=FALSE, scale=1)
```

Arguments

`data`	tidy kernel density gradient estimate (output from `tidy_kdde(deriv_order=1)`)
`x`	geospatial kernel density gradient estimate (output from `st_kdde(deriv_order=1)`)
`thin`	number to thin out estimation grid. Default is 5.
`transf`	power index in transformation. Default is 1/4.
`neg.grad`	flag to compute arrows in negative gradient direction. Default is FALSE.
`scale`	scale factor to normalise length of arrows. Default is 1.

Details

A kernel quiver estimate is a modification of the standard kernel density gradient estimate in `*_kdde` where the density derivatives are not given in the separate groups as indexed in `deriv_group`, but as extra columns `u` (for `deriv_group=(1,0)`) and `v` (for `deriv_group=(0,1)`).

The bandwidth matrix of smoothing parameters is computed as in `ks::kdde(deriv_order=1)`.

Value

The output from `tidy_kquiver` has the same structure as the input kernel density gradient estimate, with the added columns `u,v` for the density gradient value in the x-, y-axis. This structure is compatible with the `ggquiver::geom_quiver` layer function for quiver plots.

Since `ggquiver::geom_quiver` is not compatible with `geom_sf` layers, the output from `st_kquiver` has added columns `lon`, `lat`, `lon_end`, `lat_end`, `len` which are required in `geom_segment`.

Examples

```
## tidy kernel quiver estimate
library(ggplot2)
data(crabs, package="MASS")
crabs2 <- dplyr::select(crabs, FL, CW)
t1 <- tidy_kde(crabs2)
t2 <- tidy_kdde(crabs2, deriv_order=1)
t3 <- tidy_kquiver(t2, thin=5)
gt <- ggplot(t1, aes(x=FL, y=CW))
gt + geom_contour_filled_ks(alpha=0.5) +
    ggquiver::geom_quiver(data=t3, aes(u=u, v=v), colour=1)

## geospatial kernel `quiver' estimate
data(wa)
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
s1 <- st_kde(hakeoides)
s2 <- st_kdde(hakeoides, deriv_order=1)
s3 <- st_kquiver(s2, thin=5)

## base R plot
xlim <- c(1.2e5, 1.1e6); ylim <- c(6.1e6, 7.2e6)
plot(wa, xlim=xlim, ylim=ylim)
plot(s1, add=TRUE, alpha=0.5, border="grey50")
plot(s3$tidy_ks$ks[[1]], thin=5, add=TRUE, display="quiver")

## geom_sf plot
gs <- ggplot(s1) + geom_sf(data=wa, fill=NA) + ggthemes::theme_map()
gs + geom_sf(data=st_get_contour(s1), aes(fill=contperc), alpha=0.5) +
   ggquiver::geom_quiver(data=s3$sf, aes(x=lon, y=lat, u=u, v=v), colour=1) +
    coord_sf(xlim=xlim, ylim=ylim)
```

`tidyst_kroc` *Tidy and geospatial kernel receiver operating characteristic (ROC) curve*

Description

Tidy and geospatial versions of kernel receiver operating characteristic (ROC) curve for 1- and 2-dimensional data.

Usage

```
tidy_kroc(data1, data2, ...)
st_kroc(x1, x2, ...)
```

Arguments

`data1`, `data2`	data frames/tibbles of data values

x1, x2	sf objects with point geometry
...	other parameters in ks::kroc function

Details

A kernel ROC curve is a modification of the standard kernel distribution estimate where the two data samples are compared. For details of the computation and the bandwidth selection procedure of the kernel density ROC curve, see ?ks::kroc. The bandwidth matrix of smoothing parameters is computed as in ks::kcde per data sample.

Value

The output has the same structure as the 1-d kernel distribution estimate from *_kcde, except that fpr (x-variable) is the false positive rate (complement of specificity) and estimate is the true positive rate (sensitivity), rather than the usual estimation grid points and cdf values.

Examples

```
## 2-d kernel ROC curve between unsuccessful and successful grafts
library(ggplot2)
data(hsct, package="ks")
hsct <- dplyr::as_tibble(hsct)
hsct <- dplyr::filter(hsct, PE.Ly65Mac1 >0 & APC.CD45.2>0)
hsct6 <- dplyr::filter(hsct, subject==6)   ## unsuccessful graft
hsct6 <- dplyr::select(hsct6, PE.Ly65Mac1, APC.CD45.2)
hsct12 <- dplyr::filter(hsct, subject==12) ## successful graft
hsct12 <- dplyr::select(hsct12, PE.Ly65Mac1, APC.CD45.2)
t1 <- tidy_kroc(data1=hsct6, data2=hsct12)
ggplot(t1, aes(x=fpr)) + geom_line(colour=1)

## geospatial ROC curve between Grevillea species
data(wa)
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
paradoxa <- dplyr::filter(grevilleasf, species=="paradoxa")
s1 <- st_kroc(x1=hakeoides, x2=paradoxa)
ggplot(s1, aes(x=fpr)) + geom_line(colour=1)
```

---

tidyst_ksupp *Tidy and geospatial kernel support estimate*

Description

Tidy and geospatial versions of a kernel support estimate for 2-dimensional data.

Usage

```
tidy_ksupp(data, cont=95, convex_hull=TRUE, ...)
st_ksupp(x, cont=95, convex_hull=TRUE, ...)
```

Arguments

data	tidy kernel density estimate (output from tidy_kde)
x	spatial kernel density estimate (output from st_kde)
cont	scalar contour level. Default is 95.
convex_hull	flag to compute convex hull of contour region. Default is TRUE.
...	other parameters in ks::ksupp function

Details

The kernel support estimate is considered to be the cont% probability contour of the kernel density estimate, with an additional convex hull calculation if convex_hull=TRUE. For details of the computation of the kernel support estimate, see ?ks::ksupp.

Value

The output from *_ksupp have the same structure as the kernel density estimate from *_kde, except that x,y indicate the boundary of the density support estimate (if convex_hull=TRUE) or the grid points inside the density support (if convex_hull=FALSE), rather than the complete grid points themselves.

For st_ksupp, the density support estimate is stored as a (multi)polygon sf object.

Examples

```
## tidy density support estimate
library(ggplot2)
data(crabs, package="MASS")
crabs2 <- dplyr::select(crabs, FL, CW)
t1 <- tidy_kde(crabs2)
t2 <- tidy_ksupp(t1)
ggplot(t1, aes(x=FL, y=CW)) +
    geom_contour_filled_ks(cont=c(25,50,75,95), colour="grey50") +
    geom_polygon(data=t2, aes(linetype=label), fill=NA, colour=1) +
    scale_linetype_manual(values="dashed", name="Support\nconvex hull")

## geospatial density support estimate
data(wa)
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
s1 <- st_kde(hakeoides)
s2 <- st_ksupp(s1, cont=97.5)

## base R plot
xlim <- c(1.2e5, 1.1e6); ylim <- c(6.1e6, 7.2e6)
plot(wa, xlim=xlim, ylim=ylim)
plot(s1, cont=c(25,50,75,97.5), add=TRUE, border="grey50")
plot(s2, add=TRUE, lty=2, pos="bottom")

## geom_sf plot
ggplot(s1) + geom_sf(data=wa, fill=NA) + ggthemes::theme_map() +
```

```
scale_linetype_manual(values=c("dashed"), name="Support\nconvex hull") +
geom_sf(data=st_get_contour(s1, cont=c(25,50,75,97.5)), aes(fill=contperc)) +
geom_sf(data=st_get_contour(s2), aes(linetype=contperc), fill=NA, col=1) +
coord_sf(xlim=xlim, ylim=ylim)
```

`tidyst_plot` *Plots for tidy and geospatial kernel estimates*

Description

Plots for tidy and geospatial kernel estimates.

Usage

```
## S3 method for class 'tidy_ks'
ggplot(data=NULL, mapping=aes(), ...)
## S3 method for class 'sf_ks'
ggplot(data=NULL, mapping=aes(), ..., which_geometry="sf")
## S3 method for class 'sf_ks'
plot(x, ...)
```

Arguments

`data`, `x`	object of class `tidy_ks` (output from `tidy_k*`) or object of class `sf_ks` (output from `st_k*`)
`mapping`	default list of aesthetic mappings to use for plot.
`which_geometry`	type of geometry to display: one of `"sf"`,`"grid"`. Default is `"sf"`.
`...`	other graphics parameters. See below.

Details

For `tidy_ks` objects, the `ggplot` method adds some default aesthetics based on derived variables in the computed kernel estimate. These are `aes(y=estimate, weight=ks)` (1-d) and are `aes(z=estimate, weight=ks)` (2-d). These derived variables computed in the tibble output from `tidy_k*` are: `estimate` is the kernel estimate value and `ks` is the untidy version of the kernel estimate, which is required to compute contour levels. The `ggplot` method also adds some default labels for the axes and grouping variable, and some default formatting for the legends. These defaults replicate the appearance of the corresponding plots from the **ks** package.

For `sf_ks` objects, the `ggplot` method is similar to the above method, except no default aesthetics are added. The function header for the `plot` method is

```
plot(x, which_geometry="sf", cont=c(25,50,75), abs_cont=breaks, breaks,
  which_deriv_ind=1, pal, col, legend=TRUE, legend.title, digits=4, ...)
```

where

`which_geometry` type of geometry to display: one of `c("sf", "grid")`. Default is `"sf"`.

`cont` vector of percentages for contour heights

`abs_cont`,`breaks` vector of values for contour heights

`which_deriv_ind` index for partial derivative for density derivative estimate. Default is 1.

`pal` colour palette function

`col` vector of colours

`legend` flag to add legend. Default is TRUE. The output from `mapsf::mf_legend` in base R is not as robust as the legend output in `ggplot2`.

`legend.title` legend title

`digits` number of significant digits in legend key. Default is 4.

... other graphics parameters in the `plot` method for `sf` objects or for `mapsf::mf_legend`.

Value

`ggplot` plot object is created. Base R plot is sent to graphics window.

See Also

`tidy_kde`,`st_kde`

Index