Package ‘eks’

May 2, 2024

Version 1.0.5

Date 2024-05-02

Title Tidy and Geospatial Kernel Smoothing

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

Depends R (>= 2.10.0)

Imports dplyr, ggplot2 (>= 3.0.0), isoband, ks, mapsf, sf

Suggests colorspace, ggquiver, ggspatial, ggthemes, knitr, MASS, rmarkdown

VignetteBuilder knitr

Description Extensions of the kernel smoothing functions from the 'ks' package for compatibility with the tidyverse and geospatial ecosystems <doi:10.48550/arXiv.2203.01686>.

License GPL-2 | GPL-3

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

RoxygenNote 7.3.1

NeedsCompilation no

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

Repository CRAN

Date/Publication 2024-05-01 23:24:46 UTC

R topics documented:

`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. (2022) *Statistical visualisation for tidy and geospatial data in R via kernel smoothing methods in the eks package*. Submitted. DOI:10.48550/arXiv.2203.01686

`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), group=FALSE, ...)
## S3 method for class 'sf_ks'
contourLevels(x, cont=c(25,50,75), group=FALSE, ...)
contour_breaks(data, cont=c(25,50,75), group=FALSE)
label_percent(y)

st_get_contour(x, cont=c(25,50,75), breaks, which_deriv_ind, disjoint=TRUE,
    as_point=FALSE)
```

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).
`group`	flag to compute contour levels per group. Default is FALSE.
`breaks`	tibble or vector of contour levels (e.g. output from `contour_breaks`)
`which_deriv_ind`	derivative index (only required for `st_kdde` objects)
`disjoint`	flag to compute disjoint contours. Default is TRUE.
`as_point`	flag to return polygons as point coordinates in tidy format. Default is TRUE.
`y`	factor variable
`...`	other parameters (not implemented)

Details

By default, the 1% to 99% contours are computed for an `st_k*` output, though a plot of all 99 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. The output from `st_get_contour` is an sf object of the contours as multipolygons.

See Also

`geom_contour_ks`

Examples

```
library(ggplot2)
data(crabs, package="MASS")
crabs2 <- dplyr::select(crabs, FL, CW, sex)
crabs2 <- dplyr::group_by(crabs2, sex)
t1 <- tidy_kde(crabs2)
b <- contour_breaks(t1)
ggplot(t1, aes(x=FL, y=CW)) +
    geom_contour_filled_ks(colour=1, breaks=b) + facet_wrap(~sex)

crabs3 <- dplyr::select(crabs, FL, CW)
t2 <- tidy_kde(crabs3)
ggplot(t2, aes(x=FL, y=CW)) +
    geom_contour_filled_ks(colour=1, cont=c(50,75,97.5))

## extract contour polygons
crabs2s <- sf::st_as_sf(crabs2, coords=c("FL","CW"))
t2 <- st_kde(crabs2s)
t2 <- st_get_contour(t2, breaks=b, as_point=TRUE)
t2 <- dplyr::rename(t2, FL=X, CW=Y)
ggplot(t2, aes(x=FL, y=CW)) +
    geom_polygon(aes(fill=contlabel, subgroup=contlabel_group), col=1) +
    scale_fill_viridis_d() + guides(fill=guide_legend(reverse=TRUE)) +
    facet_wrap(~sex)
```

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), label_percent=NULL,
    breaks=NULL, show.legend=NA, inherit.aes=TRUE)
stat_contour_ks(mapping=NULL, data=NULL, geom="contour_ks",
    position="identity", ..., cont=c(25,50,75), label_percent=NULL,
    breaks=NULL, 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), label_percent=NULL,
    breaks=NULL, show.legend=NA, inherit.aes=TRUE)
stat_contour_filled_ks(mapping=NULL, data=NULL, geom="contour_filled_ks",
    position="identity", ..., cont=c(25,50,75), label_percent=NULL,
    breaks=NULL, 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 `cont=c(25,50,75)`.
`label_percent`	Flag for legend label as percentage. Default is TRUE.

breaks	Numeric vector to set the contour breaks e.g. output from `contour_breaks`. Overrides `cont`.
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. If `percent_label=TRUE`, then the legend labels are given as these percentage in `cont`. Otherwise, the labels are the contour levels themselves.

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, sp)
crabs2 <- dplyr::group_by(crabs2, sp)
tt <- tidy_kde(crabs2)
gt <- ggplot(tt, aes(x=FL, y=CW))
gt + geom_contour_ks() + facet_wrap(~sp)
gt + geom_contour(stat="contour_ks") + facet_wrap(~sp) ## same output
gt + geom_contour_filled_ks(colour=1) + facet_wrap(~sp)
gt + geom_contour_filled(stat="contour_filled_ks", colour=1) +
    facet_wrap(~sp) ## same output
```

---

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)
g1 + geom_rug(stat="rug_ks", colour=4) ## same output

## 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(colour=1) + geom_point_ks(colour=4)
g2 + geom_contour(stat="contour_ks", colour=1) +
    geom_point(stat="point_ks", colour=4) ## same output
```

`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.

`scale_transparent` *Change individual colours in discrete colour scale to transparent*

Description

Change individual colours in discrete colour scale to transparent.

Usage

```
scale_transparent(x, ind=NULL)
```

Arguments

`x`	discrete colour scale
`ind`	index of colour scale to change to transparent. Default is median.

Value

The output is the same colour scale, except that the colours at the indices enumerated by `ind` are changed to transparent.

Examples

```
## see ? tidy_kdde
```

`st_add_coordinates` *Add coordinates as attributes to geospatial data*

Description

Add coordinates as attributes to geospatial data.

Usage

```
st_add_coordinates(x, as_sf=FALSE, as_tibble=FALSE, rename=TRUE)
```

Arguments

`x`	`sf` object with point geometry
`as_sf`	flag for output as `sf` object. Default is TRUE.
`as_tibble`	flag for output as tibble. Default is TRUE.
`rename`	flag to rename output from `X,Y` to `lon,lat`. Default is TRUE.

Details

The `sf::st_coordinates` is applied to obtain the longitude and latitude coordinates.

Value

The longitude and latitude of the point geometry are added as attributes.

Examples

```
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
hakeoides_coord <- st_add_coordinates(hakeoides)
```

`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 per species
crabs1 <- dplyr::select(crabs, FL, sp)
crabs1 <- dplyr::group_by(crabs1, sp)
t1 <- tidy_kcde(crabs1)
gt1 <- ggplot(t1, aes(x=FL))
gt1 + geom_line(colour=1) + geom_rug_ks(colour=4) + facet_wrap(~sp)

## 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_crop <- sf::st_crop(paradoxa, xmin=4e5, xmax=8e5, ymin=6.4e6, ymax=6.65e6)
paradoxa_bbox <- sf::st_as_sfc(sf::st_bbox(paradoxa_crop))
xminb <- sf::st_bbox(paradoxa_crop)[1:2]
xmaxb <- sf::st_bbox(paradoxa_crop)[3:4]
s1 <- st_kcde(paradoxa_crop, xmin=xminb, xmax=xmaxb)

## base R filled contour 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 filled contour 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=label_percent(contlabel))) +
    scale_fill_viridis_d() + 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)
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) + scale_fill_brewer(palette="Oranges")

## 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=label_percent(contlabel))) +
    colorspace::scale_fill_discrete_sequential(h1=30,c1=360,c2=30) +
    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 1% to 99% 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(cardio, package="ks")
cardio <- dplyr::as_tibble(cardio[,c("ASTV","Mean","NSP")])
cardio <- dplyr::mutate(cardio, NSP=ordered(NSP))
cardio <- dplyr::group_by(cardio, NSP)
set.seed(8192)
cardio.train.ind <- sample(1:nrow(cardio), round(nrow(cardio)/4,0))
cardio.train <- cardio[cardio.train.ind,]
cardio.train1 <- dplyr::select(cardio.train, ASTV, NSP)
cardio.train2 <- dplyr::select(cardio.train, ASTV, Mean, NSP)

## tidy 1-d classification
t1 <- tidy_kda(cardio.train1)
gt1 <- ggplot(t1, aes(x=ASTV))
gt1 + geom_line(aes(colour=NSP)) +
    geom_rug(aes(colour=label), sides="b", linewidth=1.5) +
    scale_colour_brewer(palette="Dark2", na.translate=FALSE)

## tidy 2-d classification
t2 <- tidy_kda(cardio.train2)
gt2 <- ggplot(t2, aes(x=ASTV, y=Mean)) + theme_bw()
gt2 + geom_contour_ks(aes(colour=NSP)) +
    geom_tile(aes(fill=label), alpha=0.2) +
    scale_fill_brewer(palette="Dark2", na.translate=FALSE) +
    scale_colour_brewer(palette="Dark2")

## 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_union(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, border=NA, legend=FALSE)
plot(s1, add=TRUE, lwd=2, border=rep(colorspace::qualitative_hcl(
    palette="Dark2", n=2, alpha=0.5), each=3))

## geom_sf plot
gs1 <- ggplot(s1) + geom_sf(data=wa, fill=NA) +
    geom_sf(data=dplyr::mutate(s1$grid, species=label), aes(fill=species),
    alpha=0.1, colour=NA) + ggthemes::theme_map()
gs1 + geom_sf(data=st_get_contour(s1), aes(colour=species), fill=NA) +
    colorspace::scale_colour_discrete_qualitative(palette="Dark2") +
    colorspace::scale_fill_discrete_qualitative(palette="Dark2") +
    facet_wrap(~species) + 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 <- dplyr::bind_rows(dplyr::mutate(t1, group=1L), dplyr::mutate(t2, group=2L))
t3$group <- factor(t3$group, label=c("Standard KDE","Deconvolved KDE"))
t3 <- as_tidy_ks(t3)

## deconvolved estimate is more clearly bimodal than standard KDE
gt <- ggplot(t3, aes(x=co2, y=pm10))
gt + geom_contour_filled_ks(colour=1) +
    colorspace::scale_fill_discrete_sequential() + 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`.

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 1% to 99% contour regions of the kernel density derivative estimate, as multipolygons, with attributes `contlabel` derived from the contour level; and `estimate`, `deriv_ind`, `group`, `deriv_group` copied from the `tidy_ks` object.

Examples

```
library(ggplot2)
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) + geom_rug_ks(colour=4)

## 2-d density gradient estimate
crabs2 <- dplyr::select(crabs, FL, CW)
t2 <- tidy_kdde(crabs2, deriv_order=1)
gt2 <- ggplot(t2, aes(x=FL, y=CW)) +
    scale_transparent(colorspace::scale_fill_discrete_diverging())
gt2 + geom_contour_ks(aes(group=deriv_group, colour=after_stat(level))) +
    colorspace::scale_colour_discrete_diverging() + facet_wrap(~deriv_group)
gt2 + geom_contour_filled_ks(colour=1) + facet_wrap(~deriv_group)
## second partial derivative f^(0,1) only
gt2 + geom_contour_filled_ks(data=dplyr::filter(t2, deriv_ind==2), colour=1)
```

```
## geospatial density derivative estimate
data(wa)
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
s1 <- st_kdde(hakeoides, deriv_order=1)
s1_cont <- st_get_contour(s1, which_deriv_ind=1)
s1_cont2 <- st_get_contour(s1, which_deriv_ind=2, cont=c(25,50,75, 97.5))
s1_cont3 <- st_get_contour(s1, breaks=contour_breaks(s1))

## base R filled contour plot
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=1)

## geom_sf filled contour plot
gs <- ggplot(s1) + geom_sf(data=wa, fill=NA) +
    colorspace::scale_fill_discrete_diverging() + ggthemes::theme_map()
gs + geom_sf(data=s1_cont, aes(fill=label_percent(contlabel)))  +
    coord_sf(xlim=xlim, ylim=ylim)
gs + geom_sf(data=s1_cont2, aes(fill=label_percent(contlabel))) +
    coord_sf(xlim=xlim, ylim=ylim)

## facet wrapped geom_sf filled contour plot
## each facet = each partial derivative
gs + geom_sf(data=s1_cont3, aes(fill=contlabel)) +
    coord_sf(xlim=xlim, ylim=ylim) + 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 1% to 99% 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.

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 1% to 99% contour regions of kernel density estimate, as multipolygons, with attributes contlabel derived from the contour level; and estimate, 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))
gt2 + geom_contour_filled_ks(colour=1) +
    colorspace::scale_fill_discrete_sequential()

## default smoothing matrix 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() +
    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))
b <- contour_breaks(t3)
gt3 <- ggplot(t3, aes(x=FL, y=CW))
gt3 + geom_contour_filled_ks(colour=1, breaks=b) +
    colorspace::scale_fill_discrete_sequential() + facet_wrap(~sex)

## 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
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() +
    colorspace::scale_fill_discrete_sequential(palette="Heat2")
gs1 + geom_sf(data=st_get_contour(s1), aes(fill=label_percent(contlabel))) +
    coord_sf(xlim=xlim, ylim=ylim)
```

```
## default smoothing matrix in geom_density_2d_filled() is oversmoothed
gs2 <- ggplot(hakeoides_coord) + geom_sf(data=wa, fill=NA) +
    ggthemes::theme_map()
gs2 + geom_density_2d_filled(aes(x=X, y=Y), bins=4, colour=1) +
    colorspace::scale_fill_discrete_sequential(palette="Heat2") +
    guides(fill=guide_legend(title="Density", reverse=TRUE)) +
    coord_sf(xlim=xlim, ylim=ylim)

## 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)
tt <- c(t1, t2, labels=c("Standard KDE","Sample point KDE"))

## fixed contour levels for all three plots
b <- contour_breaks(tt)
gt <- ggplot(tt, aes(x=GDP.growth, y=inflation))
gt + geom_contour_filled_ks(breaks=b, colour=1) +
    colorspace::scale_fill_discrete_sequential() + facet_wrap(~group)

## 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))
tt <- c(t1, t2, t3, labels=c("Standard KDE","Sample point KDE","Balloon KDE"))
b <- contour_breaks(tt, cont=seq(10,90,by=10))
gt + geom_contour_filled_ks(data=tt, breaks=b, colour=1) +
    colorspace::scale_fill_discrete_sequential() + 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"))
b <- contour_breaks(s3)
bcols <- colorspace::sequential_hcl(nrow(b), palette="Heat2", rev=TRUE)

## 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, col=bcols[1:2], breaks=b)
plot(wa, xlim=xlim, ylim=ylim)
plot(s2, add=TRUE, col=bcols, breaks=b)

## geom_sf plot
gs <- ggplot(s3) + geom_sf(data=wa, fill=NA) + ggthemes::theme_map()
gs + geom_sf(data=st_get_contour(s3, breaks=b), aes(fill=contlabel)) +
    colorspace::scale_fill_discrete_sequential(palette="Heat2") +
    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)
## beta boundary density estimate
t2 <- tidy_kde_boundary(wb2, boundary.kernel="beta", xmin=xmin, xmax=xmax)
## linear boundary density estimate
t3 <- tidy_kde_boundary(wb2, boundary.kernel="linear", xmin=xmin, xmax=xmax)
## tidy truncated density estimate
t4 <- tidy_kde_truncate(wb2, boundary=rectb)

t5 <- c(t1, t2, t3, t4, labels=c("Standard KDE","Beta bd KDE", "Linear bd KDE",
    "Truncated KDE"))

## standard estimate exceeds boundary but not boundary or truncated estimates
gr <- geom_polygon(data=rectb, inherit.aes=FALSE, aes(x=x,y=y),
    fill=NA, colour=1, linetype="dashed")
gt <- ggplot(t5, aes(x=internet,y=ag.value))
gt + geom_contour_ks(aes(colour=group)) + gr + facet_wrap(~group)

## geospatial boundary density estimates
data(wa)
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
hakeoides_crop <- sf::st_crop(hakeoides, xmin=4e5, xmax=5.7e5, ymin=6.47e6, ymax=7e6)
hakeoides_bbox <- sf::st_as_sfc(sf::st_bbox(hakeoides_crop))
xminb <- sf::st_bbox(hakeoides_crop)[1:2]
xmaxb <- sf::st_bbox(hakeoides_crop)[3:4]
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 <- c(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)
plot(wa, xlim=xlim, ylim=ylim)
plot(hakeoides_bbox, add=TRUE, lty=3, lwd=2)
plot(s1, add=TRUE, border=1, col="transparent", legend=FALSE)
plot(s2, add=TRUE, border=2, col="transparent", legend=FALSE)
```

```
mapsf::mf_legend(type="symb", val=c("Standard KDE","Beta bd KDE"), pal=c(1,2),
    cex=rep(3,2), pch=rep("-",2), title="Density", pos="bottomleft")

plot(wa, xlim=xlim, ylim=ylim)
plot(hakeoides_bbox, add=TRUE, lty=3, lwd=2)
plot(s1, add=TRUE, border=1, col="transparent", legend=FALSE)
plot(s3, add=TRUE, border=3, col="transparent", legend=FALSE)
mapsf::mf_legend(type="symb", val=c("Standard KDE","Linear bd KDE"), pal=c(1,3),
    cex=rep(3,2), pch=rep("-",2), title="Density", pos="bottomleft")

plot(wa, xlim=xlim, ylim=ylim)
plot(hakeoides_bbox, add=TRUE, lty=3, lwd=2)
plot(s1, add=TRUE, border=1, col="transparent", legend=FALSE)
plot(s4, add=TRUE, border=4, col="transparent", legend=FALSE)
mapsf::mf_legend(type="symb", val=c("Standard KDE","Truncated KDE"), pal=c(1,4),
    cex=rep(3,2), pch=rep("-",2), title="Density", pos="bottomleft")

## geom_sf plots
gs <- ggplot(s1) + geom_sf(data=wa, fill=NA) +
    geom_sf(data=hakeoides_bbox, linewidth=1.2, linetype="dotted", fill=NA) +
    geom_sf(data=dplyr::mutate(st_get_contour(s1), gr="Standard KDE"),
        aes(colour=gr), fill=NA) + ggthemes::theme_map()
gs + geom_sf(data=st_get_contour(s5), aes(colour=group), fill=NA) +
    coord_sf(xlim=xlim, ylim=ylim) + 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))
gt + geom_contour_filled_ks() +
    scale_transparent(colorspace::scale_fill_discrete_qualitative(
     palette="Dark2", rev=TRUE, breaks=c("hsct6<hsct12","hsct6>hsct12"), order=c(2,1,3)))

## 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)

## 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=label)) +
    colorspace::scale_fill_discrete_qualitative(palette="Dark2", rev=TRUE) +
    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, alpha=0.8) +
    geom_path(aes(colour=label, group=segment), linewidth=1.2, alpha=0.8) +
    scale_colour_manual(values=6)

## 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)
plot(s1, add=TRUE, col=6, lwd=3, alpha=0.8)

## 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=label), linewidth=1.2, alpha=0.8) +
    scale_colour_manual(values=6) + 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 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_kfs(hsct12)
gt <- ggplot(t1, aes(x=PE.Ly65Mac1, y=APC.CD45.2))
gt + geom_contour_filled_ks(aes(colour=label), colour=1) +
    scale_fill_manual(values=7)

## 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, cont=50), aes(fill=label)) +
    scale_fill_manual(values=7) + 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, estimate)
t2 <- dplyr::slice(t2, chull(FL,CW))

gt <- ggplot(t1, aes(x=FL, y=CW))
gt + geom_point(aes(colour=estimate)) +
    geom_polygon(data=t2, aes(fill=estimate), alpha=0.1)
```

```
## 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, estimate)
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, pal=function(.){
    colorspace::qualitative_hcl(n=., palette="Set2")})
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=estimate), alpha=0.5) +
    geom_sf(data=s2, aes(fill=estimate), linetype="dotted", alpha=0.15) +
    colorspace::scale_colour_discrete_qualitative(palette="Set2") +
    colorspace::scale_fill_discrete_qualitative(palette="Set2") +
    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(colour="grey50", cont=seq(10,90,by=10)) +
    colorspace::scale_fill_discrete_sequential(alpha=0.5) +
    ggquiver::geom_quiver(data=t3, aes(u=u, v=v), colour=6)

## geospatial kernel `quiver' estimate
data(wa)
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
hakeoides_coord <- st_add_coordinates(hakeoides)
s1 <- st_kde(hakeoides)
s2 <- st_kdde(hakeoides, deriv_order=1)
s3 <- st_kquiver(s2, thin=9)

## 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]], add=TRUE, display="quiver")

## geom_sf plot - ggquiver::geom_quiver not compatible with ggplot2::geom_sf layers
## use instead geom_segment
gs <- ggplot(s1) + geom_sf(data=wa, fill=NA) + ggthemes::theme_map()
gs + geom_sf(data=st_get_contour(s1), aes(fill=label_percent(contlabel)), alpha=0.5) +
    geom_segment(data=s3$sf, aes(x=lon, xend=lon_end, y=lat, yend=lat_end),
    arrow=grid::arrow(length=0.05*s3$sf$len)) +
    colorspace::scale_fill_discrete_sequential("Heat2") +
    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_kdr`, 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) +
    colorspace::scale_fill_discrete_sequential() +
    scale_linetype_manual(values="dashed", name="Support\nconvex hull")

ggplot(t2, aes(x=FL, y=CW)) +
    geom_polygon(data=t2, aes(colour=label), fill=NA, linetype="dashed") +
    geom_point_ks(data=t1, colour=3) + scale_colour_manual(values=1)

## geospatial density support estimate
data(wa)
data(grevilleasf)
hakeoides <- dplyr::filter(grevilleasf, species=="hakeoides")
s1 <- st_kde(hakeoides)
s2 <- st_ksupp(s1)
s3 <- 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(25,50,75,95,97.5), add=TRUE, border="grey50")
plot(sf::st_geometry(hakeoides), add=TRUE, pch=16, col=1)
plot(s2, add=TRUE, lty=2, lwd=2)
plot(s3, add=TRUE, lty=3, lwd=2)

## geom_sf plot
gs1 <- ggplot(s1) + geom_sf(data=wa, fill=NA) + ggthemes::theme_map()
gs1 + geom_sf(data=st_get_contour(s1, cont=c(25,50,75,95,97.5)),
    aes(fill=label_percent(contlabel)), col="grey50") +
    geom_sf(data=st_get_contour(s2),
    aes(linetype=contlabel), fill=NA, colour=1) +
    geom_sf(data=st_get_contour(s3),
    aes(linetype=contlabel), fill=NA, colour=1) +
    geom_sf(data=hakeoides, aes(colour=species)) +
    colorspace::scale_fill_discrete_sequential(palette="Heat2") +
    scale_colour_manual(values=c(1,1)) +
    guides(colour=guide_legend(title="Locations")) +
    scale_linetype_manual(values=c("dashed","dotted"),
    name="Support\nconvex hull") +
    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 `c("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", percent_label=TRUE, cont=c(25,50,75),
  abs_cont, which_deriv_ind=1, main="", legend=TRUE, ...)
```

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` vector of values for contour heights

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

`main` main plot label. Default is "".

`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`.

... 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