# Package 'LICORS'

February 19, 2015

**Type** Package

**Title** Light Cone Reconstruction of States - Predictive State Estimation From Spatio-Temporal Data

**Version** 0.2.0

**Date** 2013-11-20

**Author** Georg M. Goerg <gmg@stat.cmu.edu>

**Maintainer** Georg M. Goerg <gmg@stat.cmu.edu>

**Description** Estimates predictive states from spatio-temporal data and consequently can provide provably optimal forecasts.
Currently this implementation
supports an N-dimensional spatial grid observed over equally spaced time intervals. E.g. a video is a 2D spatial systems observed over time. This package implements mixed LICORS, has plotting tools (for (1+1)D and (2+1)D systems), and methods for optimal forecasting. Due to memory limitations it is recommend to only analyze (1+1)D systems.

**License** GPL-2

**Depends** R (>= 2.12.1)

**Imports** RColorBrewer, mvtnorm, zoo, FNN, fields, locfit, Matrix

**Suggests** huge, RANN, yaImpute, itertools

**URL** http://www.stat.cmu.edu/~gmg

**Collate** 'compute_LICORS_loglik.R' 'compute_mixture_penalty.R' 'compute_NEC.R' 'contCA00.R' 'data2LCs.R' 'estimate_LC_pdfs.R' 'estimate_state_adj_matrix.R' 'estimate_state_probs.R' 'get_LC_config.R' 'image2.R' 'initialize_states.R' 'kmeanspp.R' 'LC-utils.R' 'LICORS-package.R' 'merge_states.R' 'mixed_LICORS.R' 'normalize.R' 'predict_FLC_given_PLC.R' 'rdensity.R' 'relabel_vector.R' 'remove_small_sample_states.R' 'search_knn.R' 'setup_LC_geometry.R' 'sparsify_weights.R' 'states2weight_matrix.R' 'threshold.R' 'weight_matrix2states.R' 'wKDE.R' 'mixed_LICORS-utils.R' 'compute_LC_coordinates.R' 'compute_margin_coordinates.R' 'get_spacetime_grid.R' 'embed2.R'

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2013-11-26 07:39:36

## R topics documented:

`LICORS-package` *Light Cone Reconstruction of States - Predictive State Estimation From Spatio-Temporal Data*

## Description

A package for predictive state estimation from spatio-temporal data. The main function is `mixed_LICORS`, which implements an EM algorithm for predictive state recovery (see References).

This is an early release: some function names and arguments might/will (slightly) change in the future, so regularly check with new package updates.

## Details on Methodology - Predictive State Model for Spatio-temporal Processes

*For details and additional references please consult Goerg and Shalizi (2012, 2013).*

Let $\mathcal{D} = \{X(\mathbf{r}, t) \mid \mathbf{r} \in \mathbf{S}, t = 1, \ldots, T\} = (X_1, \ldots, X_{\tilde{N}})$ be a sample from a spatio-temporal process, observed over an $N$-dimensional spatial grid $\mathbf{S}$ and for $T$ time steps. We want to find a model that is optimal for forecasting a new $X(\mathbf{s}, u)$ given the data $\mathcal{D}$. To do this we need to know

$$P(X(\mathbf{s}, u) \mid \mathcal{D})$$

In general this is too complicated/time-intensive since $\mathcal{D}$ is very high-dimensional. But we know that in any physical system, information can only propagate at a finite speed, and thus we can restrict the search for optimal predictors to a subset $\ell^-(\mathbf{r}, t) \subset \mathcal{D}$; this is the **past light cone (PLC)** at $(\mathbf{r}, t)$.

There exists a mapping $\epsilon : \ell^- \rightarrow \mathcal{S}$, where $\mathcal{S} = \{s_1, \ldots, s_K\}$ is the predictive state space. This mapping is such that

$$P(X_i \mid \ell_i^-) = P(X_i \mid s_j),$$

where $s_j = \epsilon(\ell_i^-)$ is the predictive state of PLC $i$. Furthermore, the future is independent of the past given the predictive state:

$$P(X_i \mid \ell_i^-, s_j) = P(X_i \mid s_j).$$

The likelihood of the joint process factorizes as a product of predictive conditional distributions

$$P(X_1, \ldots, X_N) \propto \prod_{i=1}^{N} P(X_i \mid \ell_i^-) = \prod_{i=1}^{N} P(X_i \mid \epsilon(\ell_i^-)).$$

Since $s_j$ is unknown this can be seen as the complete data likelihood of a nonparametric finite mixture model over predictive states:

$$P(X_1, \ldots, X_N) \propto \prod_{i=1}^{N} \sum_{j=1}^{K} \mathbf{1}(\epsilon(\ell_i^-) = s_j) \times P(X_i \mid s_j).$$

This predictive state model is a provably optimal finite mixture model, where the "parameter" $\epsilon$ is chosen to provide optimal forecasts.

The **LICORS** R package implements methods to estimate this optimal mapping $\epsilon$.

## Acronyms and common function arguments

The R package uses a lot of acronyms and terminology from the References, which are provided here for the sake of clarity/easier function navigation:

**LCs** light cones

**PLC** past light cone; notation: $\ell^-$

**FLC** future light cone; notation: $\ell^+$

**LICORS** LIght COne Reconstruction of States

Many functions use these acryonyms as part of their name. Function arguments that repeat over and over again are:

`weight.matrix` an $N \times K$ matrix, where $N$ are the samples and $K$ are the states. That is, each row contains a vector of length $K$ that adds up to one (the mixture weights).

`states` a vector of length $N$ with entry $i$ being the label $k = 1, \ldots, K$ of PLC $i$

## Author(s)

Georg M. Goerg <gmg@stat.cmu.edu>

## References

Goerg and Shalizi (2013), JMLR W\&CP 31:289-297. Also available at arxiv.org/abs/1211.3760.

Goerg and Shalizi (2012). Available at arxiv.org/abs/1206.2398.

## See Also

The main function in this package: `mixed_LICORS`

## Examples

```
## Not run:
# setup the light cone geometry
LC_geom <- setup_LC_geometry(speed = 1, horizon = list(PLC = 2, FLC = 0),
    shape = "cone")
# load the field
data(contCA00)
# get LC configurations from field
contCA_LCs <- data2LCs(contCA00$observed, LC.coordinates = LC_geom$coordinates)
# run mixed LICORS

mod <- mixed_LICORS(contCA_LCs, num.states_start = 10, initialization = "KmeansPLC",
    max_iter = 20)

plot(mod)

## End(Not run)
```

`compute_LC_coordinates` *Computes coordinates of PLC and FLC relative to origin*

## Description

Computes the space-time coordinates of PLC and FLC given control settings relative to the origin $(\mathbf{r}, t) = (\mathbf{0}, 0)$.

Since these coordinates do not change for different space-time positions, they can be computed once before getting the LC configurations for the entire field and then used in each call by array maskexing in `get_LC_config`.

## Usage

```
compute_LC_coordinates(horizon = 1, speed = 1, space.dim = 1, type = c("PLC", "FLC"),
    shape = c("cone", "tube", "revcone"))
```

## Arguments

| | |
|---|---|
| `horizon` | integer; horizon for the PLC or FLC |
| `speed` | speed of propagation |
| `space.dim` | maximum value |
| `type` | `"PLC"` or `"FLC"` |
| `shape` | shape of light cone: `'cone'`, `'tube'`, or `'revcone'`. |

## See Also

`get_LC_config` `setup_LC_geometry` `summary.LC` `plot.LC`

## Examples

```
plot(compute_LC_coordinates(speed = 1, horizon = 4), xlim = c(-4, 2), pch = "-",
    cex = 2, col = 2, xlab = "Time", ylab = "Space")
points(compute_LC_coordinates(speed = 1, horizon = 2, type = "FLC"), pch = "+", cex = 2,
    col = "blue")

plot(compute_LC_coordinates(speed = 1, horizon = 4, shape = "tube", type = "FLC"))
plot(compute_LC_coordinates(speed = 1, horizon = 4, shape = "revcone", type = "PLC"))
```

## `compute_LICORS_loglik` *Log-likelihood of LICORS model*

### Description

Computes the *average* log-likelihood $\frac{1}{N}\ell(\mathbf{W}; \mathcal{D})$ as a function of the weight matrix $\mathbf{W}$ and the predictive state distributions $P(X = x \mid S = s_j) \approx P(X = x \mid \mathbf{W}_j)$ for all $j = 1, \ldots, K$. See References.

### Usage

```
compute_LICORS_loglik(weight.matrix, pdfs.FLC, lambda = 0, penalty = "entropy", q = 2,
    base = exp(1))
```

### Arguments

| | |
|---|---|
| `weight.matrix` | $N \times K$ weight matrix |
| `pdfs.FLC` | an $N \times K$ matrix containing the estimates of all $K$ FLC densities evaluated at all $N$ sample FLCs. |
| `lambda` | regularization parameter. Default: `lambda=0` (`penalty` and `q` will be ignored in this case). |
| `penalty` | type of penalty: `c("entropy", "1-Lq", "lognorm")`. Default: `"entropy"` |
| `base` | logarithm base for the `"entropy"` penalty. Default: `base = 2`. Any other real number is allowed; if `base = "num.states"` then it will internally assign it `base = ncol(weight.matrix)`. |
| `q` | exponent for $L_q$ norm. |

## `compute_margin_coordinates` *Get LC configuration from a (N+1)D field*

### Description

`compute_margin_coordinates` computes the coordinates (boundary) of the margin of the field.

### Usage

```
compute_margin_coordinates(dim, LC.coordinates)
```

### Arguments

| | |
|---|---|
| `dim` | a vector with the dimensions of the field (time, space1, space2, ..., spaceN) |
| `LC.coordinates` | template of the LC coordinates |

### See Also

`compute_LC_coordinates`

### Examples

```
LC_geom <- setup_LC_geometry(speed = 1, horizon = list(PLC = 3, FLC = 0), shape = "cone")

data(contCA00)

aa <- compute_margin_coordinates(dim(contCA00$observed), LC_geom$coordinates)
aa
```

---

## compute_mixture_penalty

*Penalty of mixture weights*

---

### Description

Computes the penalty $\Omega(\mathbf{W})$ of the weight matrix (or vector) for a mixture model.

### Usage

```
compute_mixture_penalty(weigh.matrix, type = c("entropy", "Lq", "lognorm", "MDL"),
    q = 2, row.average = TRUE, base = 2)
```

### Arguments

| | |
|---|---|
| `weigh.matrix` | $N \times K$ weight matrix |
| `type` | type of penalty: `c("entropy", "1-Lq", "lognorm")`. Default: `"entropy"` |
| `q` | exponent for $L_q$ norm. |
| `row.average` | logical; if `TRUE` (default) then an average penalty over all rows will be returned (one single number); if `FALSE` a vector of length $N$ will be returned. |
| `base` | logarithm base for the `"entropy"` penalty. Default: `base = 2`. Any other real number is allowed; if `base = "num.states"` then it will internally assign it `base = ncol(weigh.matrix)`. |

### See Also

`compute_LICORS_loglik` `compute_NEC`

### Examples

```
WW <- matrix(c(rexp(10, 1/10), runif(10), 1/10), ncol = 3, byrow = FALSE)
WW[1, 1] <- 0
WW <- normalize(WW)
compute_mixture_penalty(WW, row.average = FALSE)
compute_mixture_penalty(WW, row.average = TRUE)  # default: average penalty
```

## compute_NEC *Compute Negative Entropy Criterion (NEC)*

### Description

Computes the negative entropy criterion (NEC) to assess the number of clusters in a mixture model. See References for details.

### Usage

```
compute_NEC(weight.matrix, loglik.1 = NULL, loglik.k = NULL)
```

### Arguments

| | |
|---|---|
| weight.matrix | $N \times K$ weight matrix |
| loglik.1 | baseline log-likelihood for $K = 1$ cluster model |
| loglik.k | log-likelihood for $K$ cluster model |

### References

Christophe Biernacki, Gilles Celeux, and G\'erand Govaert(1999). “An improvement of the NEC criterion for assessing the number of clusters in a mixture model”. Non-Linear Anal. 20, 3, 267-272.

### See Also

compute_mixture_penalty

### Examples

```
WW <- matrix(c(rexp(10, 1/10), runif(10)), ncol = 5, byrow = FALSE)
WW <- normalize(WW)
compute_NEC(WW, -2, -1)
```

## contCA00 *Simulated 7 state (1+1)D field*

### Description

Simulated 7 state (1+1)D field

## Format

Contains the running example dataset used in hard LICORS & mixed LICORS.

A list with three $(1+1)D$ fields, each one extending over $N = 100$ pixels in space, and $T = 200$ over time:

- `observed`
- `states`
- `predictive_states`

## References

arxiv.org/abs/1206.2398

## Examples

```
# set original par parameters
op <- par(no.readonly = TRUE)

data(contCA00)
par(mfrow = c(2, 2), mar = c(3, 3, 2, 1))
for (ii in 1:3) {
    image2(contCA00[[ii]], legend = FALSE, col = "RdBu", main = attr(summary(contCA00),
        "dimnames")[[1]][ii])
    mtext("Time", 1, 1)
    mtext("Space", 2, 1)
}
par(op)
## Not run:
LC_geom <- setup_LC_geometry(speed = 1, horizon = list(PLC = 2, FLC = 0),
    shape = "cone")
bb <- data2LCs(contCA00$observed, LC.coordinates = LC_geom$coordinates)
image2(bb$PLC)
image2(cor(bb$PLC), zlim = c(-1, 1), col = "RdBu")
mod_kk <- kmeanspp(bb$PLC, k = 10)
plot(bb$FLC, col = mod_kk$cluster, pch = ".", cex = 3)

ff <- estimate_LC_pdfs(bb$FLC, states = mod_kk$cluster, method = "nonparametric")
matplot(bb$FLC, ff, pch = ".", cex = 2)

## End(Not run)
```

---

## `data2LCs` *Iterate over (N+1)D field and get all LC configurations*

## Description

data2LCs gets all PLC or FLC configuration from a $(N + 1)D$ field given the LC template. The shape and dimension of this LC template depends on coordinates passed on by setup_LC_geometry.

**User-defined LC template:**

Since data2LCs passes the LC.coordinates array to get_LC_config to iterate over the entire dataset, this functional programming approach allows user-defined light cone shapes (independent of the shapes implemented by setup_LC_geometry).

Just replace the $coordinates from the "LC" class with a user-specified LC template.

## Usage

```
data2LCs(field, LC.coordinates = list(PLC = NULL, FLC = NULL))
```

## Arguments

| | |
|---|---|
| field | spatio-temporal field; either a matrix or a 3-dimensional array with time $t$ as the first dimension, and the spatial coordinates as subsequent dimensions. Make sure to check compute_LC_coordinates for correct formatting. |
| LC.coordinates | coordinates for LC shape and dimension (usually the $coordinates value from the "LC" class; but also user-defined coordinates are possible here). |

## See Also

compute_LC_coordinates, setup_LC_geometry

## Examples

```
set.seed(1)
AA <- matrix(rnorm(200), ncol = 10)
LC_geom <- setup_LC_geometry(speed = 1, horizon = list(PLC = 2, FLC = 0), shape = "cone")
bb <- data2LCs(t(AA), LC.coordinates = LC_geom$coordinates)
image2(bb$PLC)
plot(density(bb$FLC))

# a time series example
data(nottem)
xx <- nottem
LC_geom <- setup_LC_geometry(speed = 1, horizon = list(PLC = 24, FLC = 3), space.dim = 0)
bb <- data2LCs(xx, LC.coordinates = LC_geom$coordinates)
image2(bb$PLC)
plot(density(bb$FLC))
```

## `embed2` *Improved embed() function*

### Description

Improved version of the `embed` function in the `stats` package. First it allows embeddings in past and future observation space (backward and forward shift). Secondly, it adds 'NA' to the beginning (or end) of the embedding matrix, depending on the dimension of the embedding. Optionally, they can be removed.

### Usage

```
embed2(x, max.lag = 1, na.omit = FALSE)
```

### Arguments

| | |
|---|---|
| `x` | a numeric vector, matrix, or time series. |
| `max.lag` | a scalar representing the embedding dimension in past or future. Note that contrary to 'dimension = 1' in `embed`, here 'max.lag = 1' will return a 2 column matrix (0 lag, 1 lag), and not just a 1 column matrix. Similarly, for negative shift; e.g., 'max.lag = -2' returns 3 column matrix with (0 lag, -1 lag, -2 lag). |
| `na.omit` | logical; if TRUE, it removes NA values automatically from embedded matrix. |

### See Also

`embed`

### Examples

```
data(nottem)
aa <- embed2(nottem, 12)
```

## `estimate_LC_pdfs` *Estimate PLC/FLC distributions for all states*

### Description

`estimate_LC_pdfs` estimates the PLC and FLC distributions for each state $k = 1, \ldots, K$. It iteratively applies `estimate_LC.pdf.state`.

`estimate_LC.pdf.state` estimates the PLC and FLC distributions using weighted maximum likelihood (`cov.wt`) and nonparametric kernel density estimation (`wKDE`) for one (!) state.

## Usage

```
estimate_LC_pdfs(LCs, weight.matrix = NULL, method = c("nonparametric", "normal",
    "huge"), eval.LCs = NULL)

estimate_LC_pdf_state(state, states = NULL, weights = NULL, LCs = NULL, eval.LCs = NULL,
    method = c("nonparametric", "normal", "huge"))
```

## Arguments

| | |
|---|---|
| `LCs` | matrix of PLCs/FLCs. This matrix has $N$ rows and $n_p$ or $n_f$ columns (depending on the PLC/FLC dimensionality) |
| `weight.matrix` | $N \times K$ weight matrix |
| `states` | vector of length $N$ with entry $i$ being the label $k = 1, \ldots, K$ of PLC $i$ |
| `method` | type of estimation: either a (multivariate) Normal distribution (`"normal"`) or nonparametric with a kernel density estimator (`method = "nonparametric"`). For multivariate distributions (as usual for PLCs) only `'normal'` should be used due to computational efficiency and statistical accuracy. |
| `eval.LCs` | on what LCs should the estimate be evaluated? If `NULL` then densities will be evaluated on the training data `LCs` |
| `state` | integer; which state-conditional density should be estimated |
| `weights` | weights of the samples. Either a i) length $N$ vector with the weights for each observation; ii) $N \times K$ matrix, where the `state` column of that matrix is used as a weight-vector. |

## Value

`estimate_LC_pdfs` returns an $N \times K$ matrix.

`estimate_LC.pdf.state` returns a vector of length $N$ with the state-conditional density evaluated at `eval.LCs`.

## Examples

```
set.seed(10)
WW <- matrix(runif(10000), ncol = 10)
WW <- normalize(WW)
temp_flcs <- cbind(sort(rnorm(nrow(WW))))
temp_flc_pdfs <- estimate_LC_pdfs(temp_flcs, WW)
matplot(temp_flcs, temp_flc_pdfs, col = 1:ncol(WW), type = "l", xlab = "FLCs",
    ylab = "pdf", lty = 1)
###################### one state only ###
temp_flcs <- temp_flcs[order(temp_flcs)]
temp_flc_pdf <- estimate_LC_pdf_state(state = 3, LCs = temp_flcs, weights = WW)

plot(temp_flcs, temp_flc_pdf, type = "l", xlab = "FLC", ylab = "pdf")
```

## estimate_state_adj_matrix

*Estimate adjacency matrix for equivalent FLC distributions based on states*

### Description

This function estimates the adjacency matrix $\mathbf{A}$ of all pairwise equivalent FLC distributions given the states $s_1, \ldots, s_K$. See Details below.

### Usage

```
estimate_state_adj_matrix(states = NULL, FLCs = NULL, pdfs.FLC = NULL, alpha = NULL,
    distance = function(f, g) return(mean(abs(f - g))))
```

### Arguments

| | |
|---|---|
| `states` | vector of length $N$ with entry $i$ being the label $k = 1, \ldots, K$ of PLC $i$ |
| `FLCs` | $N \times n_f$ matrix of FLCs (only necessary if `distance= "KS"`) |
| `pdfs.FLC` | $N \times K$ matrix of all $K$ state-conditional FLC densities evaluated at each FLC $\ell_i^+$, $i = 1, \ldots, N$ (only necessary if `distance = function(f, g) return(...)`). |
| `alpha` | significance level for testing. Default: `alpha=NULL` (this will return a p-value matrix if `method == "KS"`) |
| `distance` | either a Kolmogorov-Smirnov test (`distance = "KS"`) or a function metric (e.g. $L_q$ distance). For a distance function, `distance` requires as input a function of $f$ and $g$ that returns one value. Default: `distance = function(f, g)   return(mean(abs(f-g)))` $\rightarrow L_1$ distance. |

### Value

A $K \times K$ adjacency matrix with a trimmed version of exp(-distance) or p-values. If `alpha!=NULL` then it returns the thresholded $0/1$ matrix. However, here 1 stands for equivalent, i.e. not rejecting. The matrix is obtained by checking for `pval>alpha` (rather than the usual `pval<alpha`).

### Details and user-defined distance function

The $(i, j)$th element of the adjacency matrix is defined as

$$\mathbf{A}_{ij} = distance(P(X \mid s_i), P(X \mid s_j)) = distance(f, g),$$

where `distance` is either

**a metric** in the function space of pdfs $f$ and $g$, or

**a two sample test** for $H_0 : f = g$, e.g. a Kolmogorov-Smirnov test (`distance="KS"`).

Again we use a functional programming approach and allow the user to specify any valid distance/similarity function `distance = function(f, g) return(...)`.

If `distance="KS"` the adjacency matrix contains p-values of a Kolmogorov-Smirnov test or the thresholded versions (if `alpha!=NULL`) - see Return for details.

Otherwise `distance` is an R function that takes as an input two vectors `f` and `g` (e.g. the `wKDE` estimates for two states), and returns a non-negative, real number to estimate their distance. Default is the $L_1$ distance `distance =  function(f, g) return(mean(abs(f-g)))`.

## Examples

```
WW <- matrix(runif(10000), ncol = 10)
WW <- normalize(WW)
temp_flcs <- cbind(rnorm(nrow(WW)))
temp_pdfs.FLC <- estimate_LC_pdfs(temp_flcs, WW)
AA_ks <- estimate_state_adj_matrix(states = weight_matrix2states(WW), FLCs = temp_flcs,
    distance = "KS")
AA_L1 <- estimate_state_adj_matrix(pdfs.FLC = temp_pdfs.FLC)

par(mfrow = c(1, 2), mar = c(1, 1, 2, 1))
image2(AA_ks, zlim = c(0, 1), legend = FALSE, main = "Kolmogorov-Smirnov")
image2(AA_L1, legend = FALSE, main = "L1 distance")
```

---

# `estimate_state_probs` *Estimate conditional/marginal state probabilities*

---

## Description

Estimates $P(S = s_k; \mathbf{W})$, $k = 1, \ldots, K$, the probability of being in state $s_k$ using the weight matrix $\mathbf{W}$.

These probabilites can be marginal ($P(S = s_k; \mathbf{W})$) or conditional ($P(S = s_k \mid \ell^-, \ell^+; \mathbf{W})$), depending on the provided information (`pdfs$PLC` and `pdfs$FLC`).

- If both are `NULL` then `estimate_state_probs` returns a vector of length $K$ with marginal probabilities.
- If either of them is not `NULL` then it returns an $N \times K$ matrix, where row $i$ is the probability mass function of PLC $i$ being in state $s_k$, $k = 1, \ldots, K$.

## Usage

```
estimate_state_probs(weight.matrix = NULL, states = NULL, pdfs = list(FLC = NULL,
    PLC = NULL), num.states = NULL)
```

## Arguments

| | |
|---|---|
| `weight.matrix` | $N \times K$ weight matrix |
| `states` | vector of length $N$ with entry $i$ being the label $k = 1, \ldots, K$ of PLC $i$ |
| `pdfs` | a list with estimated pdfs for PLC and/or FLC evaluated at each PLC, $i = 1, \ldots, N$ and/or FLC, $i = 1, \ldots, N$ |
| `num.states` | number of states in total. If `NULL` (default) then it sets it to `max(states)` or `ncol(weight.matrix)` - depending on which one is provided. |

## Value

A vector of length $K$ or a $N \times K$ matrix.

## Examples

```
WW <- matrix(runif(10000), ncol = 10)
WW <- normalize(WW)
estimate_state_probs(WW)
```

---

## `get_LC_config` *Get configuration of a light cone (LC)*

---

### Description

`get_LC_config` obtains the PLC or FLC at a particular $(\mathbf{r}, t)$ from a $(N + 1)D$ field based on the LC template from `compute_LC_coordinates` (or `setup_LC_geometry`).

### Usage

```
get_LC_config(coord, field, LC.coordinates)
```

### Arguments

| | |
|---|---|
| `coord` | space-time coordinate $(\mathbf{r}, t)$ |
| `field` | spatio-temporal field; either a matrix or a 3-dimensional array with time $t$ as the first coord, and the spatial coords in order. Make sure to see also `compute_LC_coordinates` for correct formatting. |
| `LC.coordinates` | template coords for the LC |

### See Also

`compute_LC_coordinates`

### Examples

```
AA <- matrix(rnorm(40), ncol = 5)
image2(AA)
LCind <- compute_LC_coordinates(speed = 1, horizon = 1, shape = "cone")
AA
get_LC_config(cbind(5, 2), AA, LCind)
# a time series example
data(nhtemp)
xx <- c(nhtemp)
LCind <- compute_LC_coordinates(speed = 1, horizon = 4, shape = "cone", space.dim = 0)
cc <- get_LC_config(6, xx, LCind)
```

---

## `get_spacetime_grid` *Get an iterator over the space-time coordinates of the field.*

---

### Description

This function returns a matrix of space-time coordinates of the field. Both for the whole field as well as the truncated field (without the margin)

### Usage

```
get_spacetime_grid(dim, LC.coordinates)
```

### Arguments

| | |
|---|---|
| `dim` | dimension of the original field (first dimension is time; rest is space) |
| `LC.coordinates` | template of the LC coordinates |

### See Also

`compute_LC_coordinates`, `setup_LC_geometry`

### Examples

```
AA <- matrix(rnorm(200), ncol = 10)
LC.geom <- setup_LC_geometry(speed = 1, horizon = list(PLC = 3, FLC = 0), shape = "cone")
bb <- get_spacetime_grid(dim(AA), LC.geom$coordinates)
```

image2 *Improved image() function*

## Description

Improved version of the image function in the graphics package. In particular, it displays matrices the way they are shown in the R console, not transposed/rearranged/... For example, a covariance matrix has the diagonal in from top-left to bottom-right as it should be, and not from bottom-left to top-right.

The function make_legend also provides a better color scale legend handling.

Optionally image2 displays a color histogram below the image, which can be used to refine the display of a matrix by trimming outliers (as they can often distort the color representation).

## Usage

```
image2(x = NULL, y = NULL, z = NULL, col = NULL, axes = FALSE, legend = TRUE,
    xlab = "", ylab = "", zlim = NULL, density = FALSE, max.height = NULL,
    zlim.label = "color scale", ...)

make_legend(data = NULL, col = NULL, side = 1, zlim = NULL, col.ticks = NULL,
    cex.axis = 2, max.height = 1, col.label = "")
```

## Arguments

| | |
|---|---|
| x,y | locations of grid lines at which the values in z are measured. These must be finite, non-missing and in (strictly) ascending order. By default, equally spaced values from 0 to 1 are used. If x is a list, its components x$x and x$y are used for x and y, respectively. If the list has component z this is used for z. |
| z | a matrix containing the values to be plotted (NAs are allowed). Note that x can be used instead of z for convenience. |
| col | colors: either a string decribing a pallette from the RColorBrewer package (see also http://colorbrewer2.org/), or a list of colors (see image for suggestions). |
| axes | a logical value indicating whether both axes should be drawn on the plot. |
| xlab | a label for the x axis |
| ylab | a label for the y axis |
| legend | logical; if TRUE a color legend for will be plotted |
| zlim | minimum and maximum z values for which colors should be plotted, defaulting to the range of the finite values of z. |
| zlim.label | character string (default: "color scale") to write next to the color legend |
| density | logical; if TRUE a color histogram (density) will be plotted. Default: FALSE. |
| max.height | height of the density plot (typically not modified by user) |
| ... | optional arguments passed to image |

| | |
|---|---|
| `data` | data for which the legend should be plotted |
| `side` | on which side of the plot (1=bottom, 2=left, 3=top, 4=right) |
| `col.ticks` | color tick marks |
| `cex.axis` | The magnification to be used for axis annotation relative to the current setting of `cex`. |
| `col.label` | same as `zlim.label` |

## See Also

`image`, `image.plot`

## Examples

```
## Not run:
# Correlation matrix
data(iris)  # make sure its from 'datasets' package, not from 'locfit'
image(cor(as.matrix(iris[, names(iris) != "Species"])))

# Correlation matrix has diagonal from top left to bottom right
par(mar = c(1, 3, 1, 2))
image2(cor(as.matrix(iris[, names(iris) != "Species"])), col = "RdBu", axes = FALSE)

## End(Not run)
# Color histogram
nn <- 10
set.seed(nn)
AA <- matrix(sample(c(rnorm(nn^2, -1, 0.1), rexp(nn^2/2, 0.5))), ncol = nn)

image2(AA, col = "Spectral")
image2(y = 1:15 + 2, x = 1:10, AA, col = "Spectral", axes = TRUE)
image2(y = 1:15 + 2, x = 1:10, AA, col = "Spectral", density = TRUE, axes = TRUE)

image2(AA, col = "Spectral", density = TRUE, zlim = c(min(AA), 3))
```

---

`initialize_states` *State initialization for iterative algorithms (randomly or variants of kmeans)*

---

## Description

Initializes the state/cluster assignment either uniformly at random from $K$ classes, or using initial *kmeans++* (`kmeanspp`) clustering (in several variations on PLCs and/or FLCs).

## Usage

```
initialize_states(num.states = NULL, num.samples = NULL, method = c("random",
   "KmeansPLC", "KmeansFLC", "KmeansPLCFLC", "KmeansFLCPLC"), LCs = list(PLC = NULL,
    FLC = NULL))
```

## Arguments

| | |
|---|---|
| `num.states` | number of states |
| `num.samples` | number of samples. |
| `method` | how to choose the labels: either uniformly at random from $\{1, \ldots, K\}$ or using K-means on PLCs and FLCs or a combination. Default: `method = "random"`. Other options are `c("KmeansPLC","KmeansFLC","KmeansPLCFLC","KmeansFLCPLC")` |
| `LCs` | (optional) a list of `PLC` ($N \times n_p$ array) and `FLC` ($N \times n_f$ array) |

## Examples

```
x1 <- rnorm(1000)
x2 <- rnorm(200, mean = 2)
yy <- c(x1, x2)
ss <- initialize_states(num.states = 2, num.samples = length(yy), method = "KmeansFLC",
    LCs = list(FLCs = yy))
plot(yy, col = ss, pch = 19)
points(x1, col = "blue")
```

---

`kmeanspp` *Kmeans++*

---

## Description

*kmeans++* clustering (see References) using R's built-in function `kmeans`.

## Usage

```
kmeanspp(data, k = 2, start = "random", iter.max = 100, nstart = 10, ...)
```

## Arguments

| | |
|---|---|
| `data` | an $N \times d$ matrix, where $N$ are the samples and $d$ is the dimension of space. |
| `k` | number of clusters. |
| `start` | first cluster center to start with |
| `iter.max` | the maximum number of iterations allowed |
| `nstart` | how many random sets should be chosen? |
| `...` | additional arguments passed to `kmeans` |

## References

Arthur, D. and S. Vassilvitskii (2007). "k-means++: The advantages of careful seeding." In H. Gabow (Ed.), Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms [SODA07], Philadelphia, pp. 1027-1035. Society for Industrial and Applied Mathematics.

## See Also

kmeans

## Examples

```
set.seed(1984)
nn <- 100
XX <- matrix(rnorm(nn), ncol = 2)
YY <- matrix(runif(length(XX) * 2, -1, 1), ncol = ncol(XX))
ZZ <- rbind(XX, YY)

cluster_ZZ <- kmeanspp(ZZ, k = 5, start = "random")

plot(ZZ, col = cluster_ZZ$cluster + 1, pch = 19)
```

---

LC-utils *Utilities for LC class*

---

## Description

The "LC" class is the core property of the LICORS model as it specifies the spatio-temporal neighborhood of the past and future light cone. The function setup_LC_geometry generates an instance of the "LC" class.

plot.LC plots LCs of $(1+1)D$ and $(2+1)D$ systems with respect to the origin $(\mathbf{r}, t) = (\mathbf{0}, 0)$. This is especially useful for a quick check if the LC geometry specified by setup_LC_geometry is really the intended one.

summary.LC prints a summary of the LC geometry. Returns (invisible) the summary matrix.

LC_coordinates2control_setting computes auxiliary measures given the LC geometry such as horizon, spatial/temporal extension, etc. This function should not be called by the user directly; only by get_spacetime_grid.

## Usage

```
## S3 method for class 'LC'
plot(x, cex.axis = 2, cex.lab = 2, ...)

## S3 method for class 'LC'
summary(object, verbose = TRUE, ...)

LC_coordinates2control_settings(LC.coordinates)
```

## Arguments

| | |
|---|---|
| x | an object of class "LC" (see setup_LC_geometry) |
| cex.axis | The magnification to be used for axis annotation relative to the current setting of cex. |

| | |
|---|---|
| `cex.lab` | The magnification to be used for x and y labels relative to the current setting of `cex`. |
| `...` | optional arguments passed to `plot`. |
| `object` | an object of class `"LC"` |
| `verbose` | logical; if `TRUE` LC information is printed in the console |
| `LC.coordinates` | template of a light cone (with respect to origin) |

## See Also

`compute_LC_coordinates`

## Examples

```
aa <- setup_LC_geometry(horizon = list(PLC = 2, FLC = 1), speed = 1, space.dim = 1,
    shape = "cone")
plot(aa)
bb <- setup_LC_geometry(horizon = list(PLC = 2, FLC = 1), speed = 1, space.dim = 1,
    shape = "revcone")
plot(bb)
aa <- setup_LC_geometry(horizon = list(PLC = 2, FLC = 0), speed = 1, space.dim = 1,
    shape = "cone")
summary(aa)
aa <- setup_LC_geometry(horizon = list(PLC = 2, FLC = 0), speed = 1, space.dim = 1,
    shape = "cone")
LC_coordinates2control_settings(aa$coordinates)
```

---

`merge_states` *Merge several states into one*

---

## Description

This function merges states $i_1, \ldots, i_j$ into a new, single state $i_1$ by adding corresponding columns of the weight matrix ($\mathbf{W}_{i_1} = \mathbf{W}_{i_1} + \ldots + \mathbf{W}_{i_j}$) and removing columns $i_2, \ldots, i_j$.

## Usage

```
merge_states(states, weight.matrix)
```

## Arguments

| | |
|---|---|
| `states` | vector of length $1 \leq j \leq K$ with the states $i_1, \ldots, i_j \subset \{1, \ldots, K\}$ that should be merged; no repeating state labels allowed. |
| `weight.matrix` | $N \times K$ weight matrix |

## Examples

```
set.seed(10)
WW <- matrix(c(rexp(1000, 1/10), runif(1000)), ncol = 5, byrow = FALSE)
WW <- normalize(WW)
image2(WW, density = TRUE)
## Not run:
merge_states(c(1, 1, 5), WW)  # error since states were repeated

## End(Not run)
WW_new <- merge_states(c(1, 3, 5), WW)

par(mfrow = c(1, 2), mar = c(1, 1, 2, 1))
image2(WW, main = paste(ncol(WW), "states"), legend = FALSE)
image2(WW_new, main = paste(ncol(WW_new), "states"), legend = FALSE)
```

---

`mixed_LICORS` *Mixed LICORS: An EM-like Algorithm for Predictive State Space Estimation*

---

## Description

`mixed_LICORS` is the core function of this package as it estimates the "parameters" in the model for the spatio-temporal process.

$$P(X_1, \ldots, X_{\tilde{N}}) \propto \prod_{i=1}^{N} P(X_i \mid \ell_i^-) = \prod_{i=1}^{N} P(X_i \mid \epsilon(\ell_i^-)).$$

## Usage

```
mixed_LICORS(LCs = list(PLC = NULL, FLC = NULL, dim = list(original = NULL,
    truncated = NULL)), num.states.init = NULL, initialization = NULL,
    control = list(max.iter = 500, alpha = 0.01, trace = 0, lambda = 0,
       sparsity = "stochastic", CV.split.random = FALSE, CV.train.ratio = 0.75,
        seed = NULL, loss = function(x, xhat) mean((x - xhat)^2),
        estimation.method = list(PLC = "normal", FLC = "nonparametric")))
```

## Arguments

| | |
|---|---|
| `LCs` | list of PLCs and FLCs matrices (see output of `data2LCs` for details and formatting). |
| `num.states.init` | number of states to start the EM algorithm |
| `initialization` | a a) character string, b) vector, or c) matrix. a) results `num.states.init` many states initialized by passing the character string as `method` argument of `initialize_states`; if b) the vector will be taken as initial state labels; if c) the matrix will be taken as initial weights. Note that for both b) and c) `num.states.init` will be ignored. $k = 1, \ldots, K$ of PLC $i$ |
| `control` | a list of control settings for the EM algorithm. See `complete_LICORS_control` for details. |

## Value

An object of class "LICORS".

## See Also

plot.mixed_LICORS, summary.mixed_LICORS

## Examples

```
## Not run:
data(contCA00)

LC_geom <- setup_LC_geometry(speed = 1, horizon = list(PLC = 2, FLC = 0),
    shape = "cone")
bb <- data2LCs(t(contCA00$observed), LC.coordinates = LC_geom$coordinates)

mm <- mixed_LICORS(bb, num.states.init = 15, init = "KmeansPLC",
    control = list(max.iter = 50, lambda = 0.001))
plot(mm)
ff_new <- estimate_LC_pdfs(bb$FLC, weight.matrix = mm$conditional_state_probs,
    method = "nonparametric")
matplot(bb$FLC, ff_new, pch = ".", cex = 2)

## End(Not run)
```

---

`mixed_LICORS-utils` *Utilities for “LICORS” class*

---

## Description

The "mixed_LICORS" class is the objectput from the mixed_LICORS estimator.

plot.mixed_LICORS gives a visual summary of the estimates such as marginal state probabilities, conditional state probabilities (= weight matrix), predictive state densities, trace plots for log-likelihood/loss/penalty.

summary.mixed_LICORS prints object a summary of the estimated LICORS model.

predict.mixed_LICORS predicts FLCs based on PLCs given a fitted mixed LICORS model. This can be done on an iterative basis, or for a selection of future PLCs.

complete_LICORS_control completes the controls for the mixed LICORS estimator. Entries of the list are:

’loss’ an R function specifying the loss for cross-validation (CV). Default: mean squared error (MSE), i.e. loss = function(x, xhat) mean((x-xhat)^2)

’method’ a list of length 2 with arguments PLC and FLC for the method of density estimation in each (either "normal" or "nonparametric").

’max.iter’ maximum number of iterations in the EM

’trace’ if > 0 it prints output in the console as the EM is running 'sparsity' what type of sparsity (currently not implemented)

'lambda' penalization parameter; larger lambda gives sparser weights

'alpha' significance level to stop testing. Default: `alpha = 0.01`

'seed' set seed for reproducibility. Default: `NULL`. If `NULL` it sets a random seed and then returns this seed in the output.

'CV.train.ratio' how much of the data should be training data. Default: `0.75`, i.e., 75% of data is for training

'CV.split.random' logical; if `TRUE` training and test data are split randomly; if `FALSE` (default) it uses the first part (in time) as training, rest as test.

'estimation' a list of length 2 with arguments `PLC` and `FLC` for the method of density estimation in each (either `"normal"` or `"nonparametric"`).

## Usage

```
## S3 method for class 'mixed_LICORS'
plot(x, type = "both", cex.axis = 1.5, cex.lab = 1.5,
    cex.main = 2, line = 1.5, ...)

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

## S3 method for class 'mixed_LICORS'
predict(object, new.LCs = list(PLC = NULL), ...)

complete_LICORS_control(control = list(alpha = 0.01, CV.split.random = FALSE,
    CV.train.ratio = 0.75, lambda = 0, max.iter = 500, seed = NULL,
    sparsity = "stochastic", trace = 0, loss = function(x, xhat) mean((x -
      xhat)^2), estimation.method = list(PLC = "normal", FLC = "nonparametric")))
```

## Arguments

| | |
|---|---|
| `x` | object of class `"mixed_LICORS"` |
| `type` | should only `"training"`, `"test"`, or `"both"` be plotted. Default: `"both"`. |
| `cex.axis` | The magnification to be used for axis annotation relative to the current setting of `cex`. |
| `cex.lab` | The magnification to be used for x and y labels relative to the current setting of `cex`. |
| `cex.main` | The magnification to be used for main titles relative to the current setting of `cex`. |
| `line` | on which margin line should the labels be ploted, starting at 0 counting objectwards (see also `mtext`). |
| `...` | optional arguments passed to `plot`, `summary`, or `predict` |
| `object` | object of class `"mixed_LICORS"` |
| `new.LCs` | a list with PLC configurations to predict FLCs given these PLCs |
| `control` | a list of controls for `"mixed_LICORS"`. |

## Examples

```
# see examples of LICORS-package see examples in LICORS-package see examples in
# LICORS-package see examples in LICORS-package
```

---

## normalize — *Normalize a matrix/vector to sum to one (probability simplex)*

---

### Description

`normalize` projects a vector or matrix onto the probability simplex.

If all entries (per row or column) get thresholded to $0$ (since they are all negative to start with), then it sets the position of the maximum of `x` to $1$ and leaves all other entries at $0$.

### Usage

```
normalize(x, byrow = TRUE, tol = 1e-06)
```

### Arguments

| | |
|---|---|
| `x` | a numeric matrix(like object). |
| `byrow` | logical; if `TRUE` rows are normalized; otherwise columns. |
| `tol` | a tolerance level to set values $< tol$ to $0$ (after an initial normalization). Default: `tol=1e-6` |

### Value

If `x` is a vector it returns the thresholded vector (see `threshold`) and normalized by its sum. If `x` is a matrix it works by column of by row (argument `byrow`).

### See Also

`threshold`

### Examples

```
print(normalize(c(1, 4, 2, 2, 10)))
print(normalize(c(-1, -2, -1)))
AA <- matrix(rnorm(12), ncol = 3)
print(normalize(AA, byrow = TRUE))
print(normalize(AA, byrow = FALSE))
```

## `predict_FLC_given_PLC` *Predict FLCs given new PLCs*

### Description

This function predicts FLCs given new PLCs based on the estimated $\epsilon$ mappings and estimated conditional distributions.

### Usage

```
predict_FLC_given_PLC(train = list(data = list(FLC = NULL, PLC = NULL),
   weight.matrix = NULL, pdfs = list(FLC = NULL, PLC = NULL)), test = list(PLC = NULL,
    weight.matrix = NULL), type = c("weighted.mean", "mean", "median", "mode"),
    method = list(FLC = "nonparametric", PLC = "normal"))
```

### Arguments

| | |
|---|---|
| `train` | a list of training examples with LC observations (a list of `PLC` and `FLC`), `weight.matrix`, and `pdfs` |
| `test` | a list of test examples with PLC observations and/or the `weight.matrix` associated with the PLC observations. |
| `method` | estimation method for estimating PLC and FLC distributions |
| `type` | prediction: `'mean'`, `'median'`, `'weightedmean'`, or `'mode'`. |

### Value

$N \times K$ matrix

## `rdensity` *Generate random sample from density() or wKDE*

### Description

This function draws random samples given `data` and a `density` estimate (or just providing the correct bandwidth `bw`).

### Usage

```
rdensity(n = 100, data = NULL, fhat = NULL, bw = fhat$bw, weights = NULL,
    kernel = "Gaussian")
```

### Arguments

| | |
|---|---|
| `n` | number of samples |
| `fhat` | an object of class `'density'` for bandwidth selection (if `bw` is not explicitly provided as argument) |
| `weights` | vector of weights. Same length as `data`. Default `weights=NULL` - in this case equal weights for each point |
| `data` | underlying sample of `fhat` |
| `kernel` | kernel choice for `fhat`. Default: `kernel='Gaussian'`. See `density` for other options. |
| `bw` | choice of bandwidth. Default: `bw=fhat$bw`. Again see `density` for other options. |

### Examples

```
set.seed(1923)
xx <- c(rnorm(100, mean = 2), runif(100))
aa <- density(xx)
plot(aa)
xx_sample <- rdensity(n = 1000, fhat = aa, data = xx)
lines(density(xx_sample), col = 2)
```

---

## `relabel_vector` *Relabels a vector to consecutive labels*

---

### Description

This function relabels a vector to have consecutive - no missing in between - labels. Labels always start at 1 and increase by one.

For example, `c(2, 2, 5)` gets relabeled to `c(1, 1, 2)`.

### Usage

```
relabel_vector(vec, order = FALSE)
```

### Arguments

| | |
|---|---|
| `vec` | vector with labels |
| `order` | logical; if `TRUE` then new state labels are assigned by decreasing number of points in that state. That is, state “1” has the most points in the state, followed by state “2” etc. |

### Examples

```
TempVec <- c(10, 2, 1, 2, 2, 2, 10)
print(relabel_vector(TempVec))

print(relabel_vector(c(2, 2, 5)))
```

## remove_small_sample_states — *Reassign low sample states to close states*

### Description

This function removes small sample states by reassigning points in those state to nearby states.

This can become necessary when in an iterative algorithm (like `mixed_LICORS`) the weights start moving away from e.g. state $j$. At some point the effective sample size of state $j$ (sum of column $\mathbf{W}_j$) is so small that state-conditional estimates (mean, variance, kernel density estimate, etc.) can not be obtained accurately anymore. Then it is good to remove state $j$ and reassign its samples to other (close) states.

### Usage

```
remove_small_sample_states(weight.matrix, min)
```

### Arguments

| | |
|---|---|
| `weight.matrix` | $N \times K$ weight matrix |
| `min` | minimum effective sample size to stay in the weight matrix |

### Examples

```
set.seed(10)
WW <- matrix(c(rexp(1000, 1/10), runif(1000)), ncol = 5, byrow = FALSE)
WW <- normalize(WW)
colSums(WW)
remove_small_sample_states(WW, 20)
```

## search_knn — *K nearest neighbor (KNN) search*

### Description

This is a wrapper for several k nearest neighbors (KNNs) algorithms in R. Currently wrapped functions are from the `FNN`, `RANN`, and `yaImpute` package.

It searches for KNN in a $N \times d$ data matrix `data` where $N$ are the number of samples, and $d$ is the dimension of space.

Either knn search in itself `query=NULL` or to query new data points wrt to training dataset.

### Usage

```
search_knn(data, k = 1, query = NULL, method = c("FNN", "RANN", "yaImpute"), ...)
```

## Arguments

| | |
|---|---|
| `data` | an $N \times d$ matrix, where $N$ are the samples and $d$ is the dimension of space. For large $d$ knn search can be very slow. |
| `k` | number of nearest neighbors (excluding point itself). Default: `k=1`. |
| `query` | (optional) an $\tilde{N} \times d$ matrix to find KNN in the training data for. Must have the same $d$ as `data`; can have lower or larger $\tilde{N}$ though. Default: `query=NULL` meaning that nearest neighbors should be looked for in the training data itself. |
| `method` | what method should be used: `'FNN'`, `'RANN'`, or `'yaImpute'`. |
| `...` | other parameters passed to the knn functions in each package. |

## See Also

Packages **FNN**, **RANN**, and **yaImpute** for other options (`...`).

## Examples

```
set.seed(1984)
XX <- matrix(rnorm(40), ncol = 2)
YY <- matrix(runif(length(XX) * 2), ncol = ncol(XX))
knns_of_XX_in_XX <- search_knn(XX, 1)
knns_of_YY_in_XX <- search_knn(XX, 1, query = YY)
plot(rbind(XX, YY), type = "n", xlab = "", ylab = "")
points(XX, pch = 19, cex = 2, xlab = "", ylab = "")
arrows(XX[, 1], XX[, 2], XX[knns_of_XX_in_XX, 1], XX[knns_of_XX_in_XX, 2], lwd = 2)
points(YY, pch = 15, col = 2)
arrows(YY[, 1], YY[, 2], XX[knns_of_YY_in_XX, 1], XX[knns_of_YY_in_XX, 2], col = 2)
legend("left", c("X", "Y"), lty = 1, pch = c(19, 15), cex = c(2, 1), col = c(1, 2))
```

---

# `setup_LC_geometry` *Setup light cone geometry*

---

## Description

`setup_LC_geometry` sets up the light cone geometry for LICORS.

## Usage

```
setup_LC_geometry(horizon = list(PLC = 1, FLC = 0), speed = 1, space.dim = 1,
    shape = "cone")
```

## Arguments

| | |
|---|---|
| `horizon` | a list with `PLC` and `FLC` horizon |
| `speed` | speed of propagation |
| `space.dim` | dimension of the spatial grid. Eg. `2` if the data is a video ( = image sequences). |
| `shape` | shape of light cone: `'cone'`, `'tube'`, or `'revcone'`. |

### Value

A list of class "LC".

### See Also

LC-utils, compute_LC_coordinates

### Examples

```
aa <- setup_LC_geometry(horizon = list(PLC = 3, FLC = 1), speed = 1, space.dim = 1,
    shape = "cone")
aa
plot(aa)
summary(aa)
```

---

## sparsify_weights *Sparsify weights*

---

### Description

This function makes weights of a mixture model more sparse using gradient based penalty methods.

### Usage

```
sparsify_weights(weight.matrix.proposed, weight.matrix.current = NULL,
    penalty = "entropy", lambda = 0)
```

### Arguments

| | |
|---|---|
| weight.matrix.proposed | $N \times K$ weight matrix |
| weight.matrix.current | $N \times K$ weight matrix |
| penalty | type of penalty: c("entropy", "1-Lq", "lognorm"). Default: "entropy" |
| lambda | penalization parameter: larger lambda gives sparser mixture weights |

### See Also

compute_mixture_penalty, mixed_LICORS

### Examples

```
WW <- matrix(c(rexp(10, 1/10), runif(10)), ncol = 5, byrow = FALSE)
WW <- normalize(WW)
WW_sparse <- sparsify_weights(WW, lambda = 0.1)
WW_more_sparse <- sparsify_weights(WW, lambda = 0.5)
compute_mixture_penalty(WW)
compute_mixture_penalty(WW_sparse)
compute_mixture_penalty(WW_more_sparse)
```

## `states2weight_matrix` *Converts label vector to 0/1 mixture weight matrix*

### Description

Converts unique cluster assignment stored in a length $N$ label vector into a $N \times K$ Boolean matrix of mixture weights.

### Usage

```
states2weight_matrix(states, num.states.total = NULL)
```

### Arguments

| | |
|---|---|
| `states` | a vector of length $N$ with the state labels |
| `num.states.total` | total number of states. If `NULL`, then the maximum of `states` is chosen |

### See Also

`weight_matrix2states`

### Examples

```
ss <- sample.int(5, 10, replace = TRUE)
WW <- states2weight_matrix(ss)

image2(WW, col = "RdBu", xlab = "States", ylab = "Samples", axes = FALSE)
```

## `threshold` *Threshold a matrix/vector below and above*

### Description

`threshold` sets values of a vector/matrix below `min` to `min`; values above `max` are set to `max`.

`threshold` is mainly used to project sparsified weight vectors (`sparsify_weights`) back onto the probability simplex (thus `min = 0` and then `normalize`).

### Usage

```
threshold(x, min = -Inf, max = Inf)
```

### Arguments

| | |
|---|---|
| `x` | a numeric matrix(like object) |
| `min` | minimum value |
| `max` | maximum value |

## See Also

`normalize`

## Examples

```
print(threshold(c(1, 4, 2, -1, 10), min = 0))
```

---

`weight_matrix2states` *Returns unique state assignment from a (row-wise) weight matrix*

---

## Description

Converts a probabilistic cluster assignment to a unique cluster assignment using the

`'argmax'` **rule:** state of row $i$ is assigned as the position of the maximum in that row (ties are broken at random).

`'sample'` **rule** state of row $i$ is sampled from the discrete distribution where probabilities equal the weight vector in row $i$

## Usage

```
weight_matrix2states(weight.matrix, rule = c("argmax", "sample"))
```

## Arguments

| | |
|---|---|
| `weight.matrix` | an $N \times K$ matrix |
| `rule` | how do we choose the state given the weight matrix. `c("argmax", "sample")`. |

## See Also

`states2weight_matrix`

## Examples

```
WW <- matrix(runif(12), ncol = 3)
WW <- normalize(WW)
WW
weight_matrix2states(WW)
weight_matrix2states(WW, "sample")
# another 'sample' is in general different from previous conversion unless WW is
# a 0/1 matrix
weight_matrix2states(WW, "sample")
```

wKDE *Weighted kernel density estimator (wKDE)*

## Description

wKDE gives a (weighted) kernel density estimate (KDE) for univariate data.

If weights are not provided, all samples count equally. It evaluates on new data point by interpolation (using approx).

mv_KDE uses the locfit.raw function in the **locfit** package to estimate KDEs for multivariate data. Note: Use this only for small dimensions, very slow otherwise.

## Usage

```
wKDE(x, eval.points = x, weights = NULL, kernel = "gaussian", bw = "nrd0")

mv_wKDE(x, eval.points = x, weights = NULL, kernel = "gaussian")
```

## Arguments

| | |
|---|---|
| x | data vector |
| eval.points | points where the density should be evaluated. Default: eval.points = x. |
| weights | vector of weights. Same length as x. Default: weights=NULL - equal weight for each sample. |
| kernel | type of kernel. Default: kernel='Gaussian'. See density and locfit.raw for additional options. |
| bw | bandwidth. Either a character string indicating the method to use or a real number. Default: bw="nrd0". Again see density for other options. |

## Value

A vector of length length(eval.points) (or nrow(eval.points)) with the probabilities of each point given the nonparametric fit on x.

## Examples

```
### Univariate example ###
xx <- sort(c(rnorm(100, mean = 1), runif(100)))
plot(xx, wKDE(xx), type = "l")
yy <- sort(runif(50, -1, 4) - 1)
lines(yy, wKDE(xx, yy), col = 2)
### Multivariate example ###
XX <- matrix(rnorm(100), ncol = 2)
YY <- matrix(runif(40), ncol = 2)
dens.object <- mv_wKDE(XX)

plot(dens.object)
points(mv_wKDE(XX, YY), col = 2, ylab = "")
```

# Index