Package 'ldmppr'

March 3, 2026

Type Package

Title Estimate and Simulate from Location Dependent Marked Point Processes

Version 1.1.3

Maintainer Lane Drew <laneTdrew@gmail.com>

Description A suite of tools for estimating, assessing model fit, simulating from, and visualizing location dependent marked point processes characterized by regularity in the pattern.
You provide a reference marked point process, a set of raster images containing location specific covariates, and select the estimation algorithm and type of mark model.
'ldmppr' estimates the process and mark models and allows you to check the appropriateness of the model using a variety of diagnostic tools.
Once a satisfactory model fit is obtained, you can simulate from the model and visualize the results.
Documentation for the package 'ldmppr' is available in the form of a vignette.

License GPL (>= 3)

Encoding UTF-8

LazyData true

Imports stats, bundle, Rcpp (>= 1.0.12), terra, doParallel, xgboost, parsnip (>= 1.4.0), dials, recipes, rsample, tune, workflows, magrittr, hardhat, ggplot2, spatstat.geom, spatstat.explore, nloptr, GET, progress, progressr, future, furrr, foreach, yardstick

LinkingTo Rcpp, RcppArmadillo

URL https://github.com/lanedrew/ldmppr, https://lanedrew.github.io/ldmppr/

BugReports https://github.com/lanedrew/ldmppr/issues

RoxygenNote 7.3.3

Suggests knitr, rmarkdown, testthat (>= 3.0.0), dplyr, rstudioapi, parallelly, ranger

VignetteBuilder knitr

Depends R (>= 3.5.0)

Config/testthat/edition 3

NeedsCompilation yes

Author Lane Drew [aut, cre, cph] (ORCID:
<https://orcid.org/0009-0006-5427-4092>),
Andee Kaplan [aut] (ORCID: <https://orcid.org/0000-0002-2940-889X>)

Repository CRAN

Date/Publication 2026-03-03 08:50:09 UTC

Contents

`check_model_fit` *Check the fit of an estimated model using global envelope tests*

Description

Performs global envelope tests for nonparametric L, F, G, J, E, and V summary functions (`spatstat`/`GET`). These tests assess goodness-of-fit of the estimated model relative to a reference marked point pattern. The reference marked point pattern can be supplied directly via `reference_data` (a marked `ppp` object), or derived internally from a `ldmppr_fit` object.

Usage

```
check_model_fit(
  reference_data = NULL,
  t_min = 0,
  t_max = 1,
  process = c("self_correcting"),
  process_fit = NULL,
  anchor_point = NULL,
  raster_list = NULL,
  scaled_rasters = FALSE,
  mark_model = NULL,
  xy_bounds = NULL,
  include_comp_inds = FALSE,
  thinning = TRUE,
  edge_correction = "none",
  competition_radius = 15,
  n_sim = 2500,
  save_sims = TRUE,
  verbose = TRUE,
  seed = 0,
  parallel = FALSE,
  num_cores = max(1L, parallel::detectCores() - 1L),
  set_future_plan = FALSE,
  mark_mode = NULL,
  fg_correction = c("km", "rs"),
  max_attempts = NULL
)
```

Arguments

`reference_data`	(optional) a marked ppp object for the reference dataset. If `NULL`, the reference pattern is derived from `process_fit` when `process_fit` is an `ldmppr_fit` and contains `data_original` (preferred) or `data` with columns `(x,y,size)`.
`t_min`	minimum value for time.
`t_max`	maximum value for time.
`process`	type of process used (currently supports `"self_correcting"`).
`process_fit`	either an `ldmppr_fit` object (from `estimate_process_parameters`) or a numeric vector of length 8 giving the process parameters.
`anchor_point`	(optional) vector of (x,y) coordinates of the point to condition on. If `NULL`, inferred from the reference data (largest mark if available) or from `ldmppr_fit`.
`raster_list`	(optional) list of raster objects used for mark prediction. Required when `mark_mode='mark_model'` unless rasters are stored in `mark_model`.
`scaled_rasters`	`TRUE` or `FALSE` indicating whether rasters are already scaled. Ignored when `mark_mode='time_to_size'`.
`mark_model`	a mark model object used when `mark_mode='mark_model'`. May be an `ldmppr_mark_model`, `model_fit`, or `workflow`.

`xy_bounds` (optional) vector of bounds as `c(a_x, b_x, a_y, b_y)`. If `NULL`, will be inferred from `reference_data`'s window when `reference_data` is provided, otherwise from `ldmppr_fit` with lower bounds assumed to be 0.

`include_comp_inds`
`TRUE` or `FALSE` indicating whether to compute competition indices.

`thinning` `TRUE` or `FALSE` indicating whether to use the thinned simulated values.

`edge_correction`
type of edge correction to apply (`"none"` or `"toroidal"`).

`competition_radius`
positive numeric distance used when `include_comp_inds = TRUE`.

`n_sim` number of simulated datasets to generate.

`save_sims` `TRUE` or `FALSE` indicating whether to save and return the simulated metrics.

`verbose` `TRUE` or `FALSE` indicating whether to show progress of model checking. When `TRUE`, progress is reported via **progressr** (if available) and is compatible with parallel execution.

`seed` integer seed for reproducibility.

`parallel` `TRUE` or `FALSE`. If `TRUE`, simulations run in parallel via **furrr/future**. For small simulation sizes, parallel overhead may outweigh speed gains.

`num_cores` number of workers to use when `parallel=TRUE`. Defaults to one fewer than detected cores.

`set_future_plan`
`TRUE` or `FALSE`. If `TRUE` and `parallel=TRUE`, set a temporary **future** plan internally and restore the previous plan on exit.

`mark_mode` (optional) mark generation mode: `"mark_model"` uses `predict()` on a mark model, while `"time_to_size"` maps simulated times back to sizes via `delta`. If `NULL`, inferred as `"mark_model"` when `mark_model` is provided, otherwise `"time_to_size"`.

`fg_correction` correction used for F/G/J summaries (`"km"` or `"rs"`).

`max_attempts` maximum number of simulation attempts when rejection occurs due to non-finite summaries.

Details

This function relies on the `spatstat` package for the calculation of the point pattern metrics and the `GET` package for the global envelope tests. The L, F, G, J, E, and V functions are a collection of non-parametric summary statistics that describe the spatial distribution of points and marks in a point pattern. See the documentation for `Lest()`, `Fest()`, `Gest()`, `Jest()`, `Emark()`, and `Vmark()` for more information. Also, see the `global_envelope_test()` function for more information on the global envelope tests.

Value

an object of class `"ldmppr_model_check"`.

References

Baddeley, A., Rubak, E., & Turner, R. (2015). *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press, London. ISBN 9781482210200. Available at: https://www.routledge.com/Spatial-Point-Patterns-Methodology-and-Applications-with-R/Baddeley-Rubak-Turner/p/book/9781482210200.

Myllymäki, M., & Mrkvička, T. (2023). GET: Global envelopes in R. *arXiv:1911.06583 [stat.ME]*. doi:10.48550/arXiv.1911.06583.

Examples

```
# Note: The example below is provided for illustrative purposes and may take some time to run.

data(small_example_data)

file_path <- system.file("extdata", "example_mark_model.rds", package = "ldmppr")
mark_model <- load_mark_model(file_path)

raster_paths <- list.files(system.file("extdata", package = "ldmppr"),
                           pattern = "\\.tif$", full.names = TRUE)
raster_paths <- raster_paths[!grepl("_med\\.tif$", raster_paths)]
rasters <- lapply(raster_paths, terra::rast)
scaled_raster_list <- scale_rasters(rasters)

reference_data <- generate_mpp(
  locations = small_example_data[, c("x", "y")],
  marks = small_example_data$size,
  xy_bounds = c(0, 25, 0, 25)
)

estimated_parameters <- c(
  0.05167978, 8.20702166, 0.02199940, 2.63236890,
  1.82729512, 0.65330061, 0.86666748, 0.04681878
)

# Keep parallel=FALSE in examples to avoid setup overhead.
example_model_fit <- check_model_fit(
  reference_data = reference_data,
  t_min = 0,
  t_max = 1,
  process = "self_correcting",
  process_fit = estimated_parameters,
  raster_list = scaled_raster_list,
  scaled_rasters = TRUE,
  mark_model = mark_model,
  xy_bounds = c(0, 25, 0, 25),
  include_comp_inds = TRUE,
  thinning = TRUE,
  edge_correction = "none",
  competition_radius = 10,
  n_sim = 100,
  save_sims = FALSE,
```

```
  verbose = TRUE,
  seed = 90210,
  parallel = FALSE
)

plot(example_model_fit, which = 'combined')
```

`estimate_process_parameters` *Estimate point process parameters using log-likelihood maximization*

Description

Estimate spatio-temporal point process parameters by maximizing the (approximate) full log-likelihood using `nloptr`.

Usage

```
estimate_process_parameters(
  data,
  process = c("self_correcting"),
  grids,
  budgets,
  parameter_inits = NULL,
  delta = NULL,
  parallel = FALSE,
  num_cores = max(1L, parallel::detectCores() - 1L),
  set_future_plan = FALSE,
  strategy = c("local", "global_local", "multires_global_local"),
  global_algorithm = "NLOPT_GN_CRS2_LM",
  local_algorithm = "NLOPT_LN_BOBYQA",
  starts = list(global = 1L, local = 1L, jitter_sd = 0.35, seed = 1L),
  finite_bounds = NULL,
  refine_best_delta = TRUE,
 rescore_control = list(enabled = TRUE, top = 5L, objective_tol = 1e-06, param_tol =
    0.1, avoid_bound_solutions = TRUE, bound_eps = 1e-08),
  verbose = TRUE
)
```

Arguments

`data`	a data.frame or matrix. Must contain either columns `(time, x, y)` or `(x, y, size)`. If a matrix is provided without time, it must have column names `c("x","y","size")`.
`process`	type of process used (currently supports `"self_correcting"`).
`grids`	a `ldmppr_grids` object specifying the integration grid schedule (single-level or multi-resolution). The integration bounds are taken from `grids$upper_bounds`.

budgets — a ldmppr_budgets object controlling optimizer options for the global stage and local stages (first level vs refinement levels).

parameter_inits — (optional) numeric vector of length 8 giving initialization values for the model parameters. If NULL, defaults are derived from data and grids$upper_bounds.

delta — (optional) numeric scalar or vector. Used only when data does not contain time (i.e., data has (x,y,size)).

- If length(delta) == 1, fits the model once using power_law_mapping(size, delta).
- If length(delta) > 1, performs a delta-search by fitting the model for each candidate value and selecting the best objective. If refine_best_delta = TRUE and multiple grid levels are used, the best delta is refined on the remaining (finer) grid levels.

If data already contains time, delta is ignored when length(delta)==1 and is an error when length(delta)>1.

parallel — TRUE or FALSE specifying furrr/future to parallelize either: (a) over candidate delta values (when length(delta) > 1), and/or (b) over local multi-start initializations (when starts$local > 1), and/or (c) over global restarts (when starts$global > 1). For small problems, parallel overhead may outweigh speed gains.

num_cores — number of workers to use when parallel=TRUE and set_future_plan=TRUE.

set_future_plan — TRUE or FALSE. If TRUE and parallel=TRUE, set a temporary **future** plan internally and restore the previous plan on exit.

strategy — character string specifying the estimation strategy:

- "local": local optimization only (single-level or multi-level polish).
- "global_local": global optimization then local polish (single grid level).
- "multires_global_local": multi-resolution (coarsest uses global+local; refinements use local only).

global_algorithm, local_algorithm — NLopt algorithms to use for the global and local optimization stages, respectively.

starts — a list controlling restart and jitter behavior:

- global: integer, number of global restarts at the first/coarsest level (default 1).
- local: integer, number of local multi-starts per level (default 1).
- jitter_sd: numeric SD for jittering (default 0.35).
- seed: integer base seed (default 1).

finite_bounds — (optional) list with components lb and ub giving finite lower and upper bounds for all 8 parameters. If NULL, bounds are derived from parameter_inits. Global algorithms and select local algorithms in NLopt require finite bounds.

refine_best_delta — TRUE or FALSE. If TRUE and length(delta) > 1, performs refinement of the best delta across additional grid levels (if available).

`rescore_control`
: controls candidate rescoring and bound-handling behavior in multi-resolution fitting. Can be either:

  - a single logical value (toggle rescoring on/off while keeping defaults), or
  - a named list with any of: `enabled`, `top`, `objective_tol`, `param_tol`, `avoid_bound_solutions`, `bound_eps`.

  Defaults are: `list(enabled = TRUE, top = 5L, objective_tol = 1e-6, param_tol = 0.10, avoid_bound_solutions = TRUE, bound_eps = 1e-8)`.

`verbose`
: `TRUE` or `FALSE` indicating whether to show progress of model estimation.

Details

For the self-correcting process, arrival times must lie on $(0, 1)$ and can be supplied directly in `data` as `time`, or constructed from `size` via the gentle-decay (power-law) mapping `power_law_mapping` using `delta`. When `delta` is a vector, the model is fit for each candidate value and the best objective is selected.

This function supports multi-resolution estimation via a `ldmppr_grids` schedule. If multiple grid levels are provided, the coarsest level may use a global optimizer followed by local refinement, and subsequent levels run local refinement only.

Value

an object of class `"ldmppr_fit"` containing the best `nloptr` fit and (optionally) stored fits from global restarts and/or a delta search.

References

Møller, J., Ghorbani, M., & Rubak, E. (2016). Mechanistic spatio-temporal point process models for marked point processes, with a view to forest stand data. *Biometrics*, 72(3), 687-696. doi:10.1111/biom.12466.

Examples

```
# Load example data
data(small_example_data)

# Define grids and budgets
ub <- c(1, 25, 25)
g  <- ldmppr_grids(upper_bounds = ub, levels = list(c(10,10,10)))
b  <- ldmppr_budgets(
  global_options = list(maxeval = 150),
  local_budget_first_level = list(maxeval = 50, xtol_rel = 1e-2),
  local_budget_refinement_levels = list(maxeval = 25, xtol_rel = 1e-2)
)

# Estimate parameters using a single delta value
fit <- estimate_process_parameters(
  data = small_example_data,
  grids = g,
```

```
  budgets = b,
  delta = 1,
  strategy = "global_local",
  global_algorithm = "NLOPT_GN_CRS2_LM",
  local_algorithm  = "NLOPT_LN_BOBYQA",
  starts = list(global = 2, local = 2, jitter_sd = 0.25, seed = 1),
  verbose = TRUE
)
coef(fit)
logLik(fit)


# Estimate parameters using multiple delta values (delta search)
g2 <- ldmppr_grids(upper_bounds = ub, levels = list(c(8,8,8), c(12,12,12)))
fit_delta <- estimate_process_parameters(
  data = small_example_data, # x,y,size
  grids = g2,
  budgets = b,
  delta = c(0.35, 0.5, 0.65, 0.9, 1.0),
  parallel = TRUE,
  set_future_plan = TRUE,
  num_cores = 2,
  strategy = "multires_global_local",
  starts = list(local = 1),
  refine_best_delta = FALSE,
  verbose = FALSE
)
plot(fit_delta)
```

`extract_covars` *Extract covariate values from a set of rasters*

Description

Extract covariate values from a set of rasters

Usage

```
extract_covars(locations, raster_list)
```

Arguments

`locations`	a matrix/data.frame of (x,y) locations.
`raster_list`	a list of SpatRaster objects.

Value

a data.frame of covariates (no ID column; unique names).

Examples

```
# Load example raster data
raster_paths <- list.files(system.file("extdata", package = "ldmppr"),
  pattern = "\\.tif$", full.names = TRUE
)
raster_paths <- raster_paths[!grepl("_med\\.tif$", raster_paths)]
rasters <- lapply(raster_paths, terra::rast)

# Scale the rasters
scaled_raster_list <- scale_rasters(rasters)

# Load example locations
locations <- small_example_data %>%
  dplyr::select(x, y) %>%
  as.matrix()

# Extract covariates
example_covars <- extract_covars(locations, scaled_raster_list)
head(example_covars)
```

`generate_mpp` *Generate a marked process given locations and marks*

Description

Creates an object of class "ppp" that represents a marked point pattern in the two-dimensional plane.

Usage

```
generate_mpp(locations, marks = NULL, xy_bounds = NULL)
```

Arguments

`locations`	a data.frame or matrix of (x,y) locations. If a data.frame is supplied, it must contain columns named "x" and "y".
`marks`	a vector of marks.
`xy_bounds`	a vector of domain bounds (2 for x, 2 for y).

Value

a ppp object with marks.

Examples

```
# Load example data
data(small_example_data)

# Generate a marked point process
generate_mpp(
  locations = small_example_data %>% dplyr::select(x, y),
  marks = small_example_data$size,
  xy_bounds = c(0, 25, 0, 25)
)
```

`ldmppr_budgets` *Create an optimization budget specification for estimate_process_parameters()*

Description

`ldmppr_budgets()` defines per-stage optimization options (budgets) used by `estimate_process_parameters` for NLopt via `nloptr`.

Usage

```
ldmppr_budgets(
  global_options = NULL,
  local_budget_first_level = NULL,
  local_budget_refinement_levels = NULL
)
```

Arguments

`global_options`	(optional) list of NLopt options used for the global stage (only relevant when `strategy` uses a global optimizer). Examples: `list(maxeval = 2000, maxtime = 10)`.
`local_budget_first_level`	(optional) list of NLopt options used for the local stage at the first (coarsest) grid level.
`local_budget_refinement_levels`	(optional) list of NLopt options used for local refinement on subsequent (finer) grid levels in multi-resolution strategies. If `NULL`, the estimator will fall back to `local_budget_first_level`.

Details

The returned object is an S3 class. Use `summary()` and `as.data.frame()` methods to inspect.

Value

an object of class "ldmppr_budgets".

See Also

ldmppr_grids-class for methods and details.

Examples

```
b <- ldmppr_budgets(
  global_options = list(maxeval = 150),
  local_budget_first_level = list(maxeval = 300, xtol_rel = 1e-5),
  local_budget_refinement_levels = list(maxeval = 150, xtol_rel = 1e-5)
)
b
```

---

ldmppr_budgets-class *Optimization budget specification object*

Description

Objects of class ldmppr_budgets define optimizer budget/options used by estimate_process_parameters.

Usage

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

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

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

## S3 method for class 'ldmppr_budgets'
as.data.frame(x, ...)

## S3 method for class 'ldmppr_budgets'
length(x)

## S3 method for class 'ldmppr_budgets'
x[i, ...]

## S3 method for class 'ldmppr_budgets'
as.list(x, ...)
```

Arguments

x	an object of class ldmppr_budgets.
...	unused.
object	an object of class ldmppr_budgets.
i	indices of local stages to keep: 1 = first level, 2 = refinement levels.

Details

A ldmppr_budgets is a list with (at minimum):

- global_options: list of NLopt options for the global stage (e.g., maxeval, maxtime).
- local_budget_first_level: list of NLopt options for the local stage at the first/coarsest grid level.
- local_budget_refinement_levels: optional list of NLopt options for local refinement levels (used only when estimate_process_parameters(strategy = "multires_global_local")).

Value

print() prints a brief description of configured budgets.

summary() returns a summary.ldmppr_budgets.

as.data.frame() a compact table of the global/local budget entries (when present).

length() number of available local budget stages (1 or 2).

[ ] subset local budget stages (keeps global options).

as.list() returns the underlying list structure.

Functions

- print(ldmppr_budgets): Print a brief summary of optimization budgets.
- summary(ldmppr_budgets): Summarize an optimization budget specification.
- print(summary.ldmppr_budgets): Print a summary produced by summary.ldmppr_budgets().
- as.data.frame(ldmppr_budgets): Convert budgets to a data.frame.
- length(ldmppr_budgets): Number of available local budget stages.
- [: Subset local budget stages (keeps global options).
- as.list(ldmppr_budgets): Extract the underlying list representation.

`ldmppr_fit` *Fitted point-process model object*

Description

Objects of class `ldmppr_fit` are returned by `estimate_process_parameters`. They contain the best-fitting optimization result (and optionally multiple fits, e.g. from a delta search) along with metadata used to reproduce the fit.

Usage

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

## S3 method for class 'ldmppr_fit'
coef(object, ...)

## S3 method for class 'ldmppr_fit'
logLik(object, ...)

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

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

## S3 method for class 'ldmppr_fit'
plot(x, ...)

as_nloptr(x, ...)

## S3 method for class 'ldmppr_fit'
as_nloptr(x, ...)

## S3 method for class 'ldmppr_fit'
nobs(object, ...)

## S3 method for class 'ldmppr_fit'
as.data.frame(x, ...)
```

Arguments

`x`	an object of class `ldmppr_fit`.
`...`	additional arguments (not used).
`object`	an object of class `ldmppr_fit`.

Details

A `ldmppr_fit` is a list with (at minimum):

- `process`: process name (e.g. `"self_correcting"`)
- `fit`: best optimization result (currently an `nloptr` object)
- `mapping`: mapping information (e.g. chosen `delta`, objectives)
- `grid`: grid definitions used by likelihood approximation

Value

`print()` prints a brief summary of the fit.

`coef()` returns the estimated parameter vector.

`logLik()` returns the log-likelihood at the optimum.

`summary()` returns a `summary.ldmppr_fit`.

`plot()` plots diagnostics for multi-fit runs, if available.

Methods (by generic)

- `print(ldmppr_fit)`: Print a brief summary of a fitted model.
- `coef(ldmppr_fit)`: Extract the estimated parameter vector.
- `logLik(ldmppr_fit)`: Log-likelihood at the optimum.
- `summary(ldmppr_fit)`: Summarize a fitted model.
- `plot(ldmppr_fit)`: Plot diagnostics for a fitted model.
- `as_nloptr(ldmppr_fit)`: Extract the underlying `nloptr` result.
- `nobs(ldmppr_fit)`: Number of observations used in the fitted model.
- `as.data.frame(ldmppr_fit)`: Coerce fit summary to a one-row data frame.

Functions

- `print(summary.ldmppr_fit)`: Print a summary produced by `summary.ldmppr_fit`.
- `as_nloptr()`: Extract the underlying `nloptr` result.

`ldmppr_grids` *Create a grid schedule for estimate_process_parameters()*

Description

`ldmppr_grids()` constructs a multi-resolution grid schedule used by `estimate_process_parameters`. The returned object is an S3 class with helper methods; see `ldmppr_grids-class`.

Usage

```
ldmppr_grids(upper_bounds, levels, labels = NULL, include_endpoints = TRUE)
```

Arguments

upper_bounds	a vector c(b_t, b_x, b_y) giving the maximum bounds for time and the spatial domain. Grids must lie within these.
levels	a list describing the grid schedule. Each entry can be either: • a numeric length-3 vector c(nx, ny, nt) (number of points per dimension), • a list with elements nx, ny, nt, • a list with explicit vectors x, y, t.
labels	(optional) character vector of length equal to levels, used only for printing.
include_endpoints	TRUE or FALSE indicating; if TRUE (default) each generated grid uses seq(0, bound, length.out = n) including endpoints.

Value

an object of class "ldmppr_grids".

See Also

ldmppr_grids-class for methods and details.

Examples

```
# A 3-level coarse-to-fine schedule (counts per dimension)
g <- ldmppr_grids(
  upper_bounds = c(1, 50, 50),
  levels = list(
    c(25, 25, 25),
    c(75, 75, 75),
    c(100, 100, 100)
  )
)
g
length(g)
summary(g)

# Explicit vectors (single level)
g2 <- ldmppr_grids(
  upper_bounds = c(1, 50, 50),
  levels = list(list(
    x = seq(0, 50, by = 2),
    y = seq(0, 50, by = 2),
    t = seq(0, 1,  length.out = 30)
  ))
)
as.data.frame(g2)
```

`ldmppr_grids-class` *Grid schedule object*

Description

Objects of class `ldmppr_grids` define one or more grid "levels" used by `estimate_process_parameters`. Each level contains numeric vectors `x`, `y`, and `t` defining the approximation grid. Levels are typically ordered from coarse to fine.

Usage

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

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

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

## S3 method for class 'ldmppr_grids'
as.data.frame(x, ...)

## S3 method for class 'ldmppr_grids'
length(x)

## S3 method for class 'ldmppr_grids'
x[i, ...]

## S3 method for class 'ldmppr_grids'
as.list(x, ...)
```

Arguments

`x`	an object of class `ldmppr_grids`.
`...`	unused.
`object`	an object of class `ldmppr_grids`.
`i`	indices of levels to keep.

Details

A `ldmppr_grids` is a list with (at minimum):

- `levels`: list of levels; each level is a list with `x`, `y`, `t`
- `upper_bounds`: numeric `c(b_t, b_x, b_y)`
- `labels`: optional labels used only for printing
- `include_endpoints`: logical

Value

`print()` prints a brief description of bounds and grid levels.

`summary()` returns a `summary.ldmppr_grids`.

`as.data.frame()` returns one row per level with dimensions and ranges.

`length()` returns the number of levels.

`[ ]` subsets levels, preserving class.

`as.list()` returns the underlying list structure.

Functions

- `print(ldmppr_grids)`: Print a brief summary of a grid schedule.
- `summary(ldmppr_grids)`: Summarize a grid schedule.
- `print(summary.ldmppr_grids)`: Print a summary produced by `summary.ldmppr_grids()`.
- `as.data.frame(ldmppr_grids)`: Convert a grid schedule to a data.frame.
- `length(ldmppr_grids)`: Number of levels in a grid schedule.
- `[`: Subset grid levels.
- `as.list(ldmppr_grids)`: Extract the underlying list representation.

---

ldmppr_mark_model *Mark model object*

Description

`ldmppr_mark_model` objects store a fitted mark model and preprocessing information used to predict marks at new locations and times. These objects are typically returned by `train_mark_model` and can be saved/loaded with `save_mark_model` and `load_mark_model`.

Usage

```
ldmppr_mark_model(
  engine,
  fit_engine = NULL,
  xgb_raw = NULL,
  recipe = NULL,
  outcome = "size",
  feature_names = NULL,
  rasters = NULL,
  info = list()
)

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

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

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

## S3 method for class 'ldmppr_mark_model'
predict(
  object,
  new_data = NULL,
  sim_realization = NULL,
  raster_list = NULL,
  scaled_rasters = FALSE,
  xy_bounds = NULL,
  include_comp_inds = FALSE,
  competition_radius = 15,
  edge_correction = "none",
  seed = NULL,
  ...
)

save_mark_model(object, path, ...)

## S3 method for class 'ldmppr_mark_model'
save_mark_model(object, path, ...)

load_mark_model(path)
```

Arguments

engine	character string (currently "xgboost" and "ranger").
fit_engine	fitted engine object (e.g. xgb.Booster or a ranger fit).
xgb_raw	raw xgboost payload (e.g. UBJ) used for rehydration.
recipe	a prepped recipes object used for preprocessing new data.
outcome	outcome column name (default "size").
feature_names	(optional) vector of predictor names required at prediction time.
rasters	(optional) list of rasters used for prediction (e.g. for spatial covariates).
info	(optional) list of metadata.
x	an object of class summary.ldmppr_mark_model.
...	passed to methods.
object	a ldmppr_mark_model object.
new_data	a data frame of predictors (and possibly outcome columns). Ignored when sim_realization is supplied.
sim_realization	optional simulation realization containing x, y, and time. When supplied, predictors are built from rasters and optional competition indices.

raster_list	optional list of rasters used when sim_realization is supplied. If omitted, uses rasters stored in object when available.
scaled_rasters	TRUE or FALSE; whether supplied rasters are pre-scaled.
xy_bounds	domain bounds c(a_x, b_x, a_y, b_y) used for competition indices.
include_comp_inds	TRUE or FALSE; include competition-index features.
competition_radius	positive numeric distance used when include_comp_inds = TRUE.
edge_correction	edge correction for competition indices ("none" or "toroidal").
seed	optional nonnegative integer seed.
path	path to an .rds created by save_mark_model (or legacy objects).

Details

The model may be backed by different engines (currently "xgboost" and "ranger"). For "xgboost", the object can store a serialized booster payload to make saving/loading robust across R sessions.

Value

print() prints a brief summary.

predict() returns numeric predictions for new data.

an object of class "ldmppr_mark_model".

Methods (by generic)

- print(ldmppr_mark_model): Print a brief summary of the mark model.
- summary(ldmppr_mark_model): Summarize a mark model.
- predict(ldmppr_mark_model): Predict marks for new data.
- save_mark_model(ldmppr_mark_model): Save method for ldmppr_mark_model.

Functions

- ldmppr_mark_model(): Create a mark model container.
- print(summary.ldmppr_mark_model): Print a summary produced by summary.ldmppr_mark_model.
- save_mark_model(): Save a mark model to disk.
- load_mark_model(): Load a saved mark model from disk.

`ldmppr_model_check` *Model fit diagnostic object*

Description

Objects of class `ldmppr_model_check` are returned by `check_model_fit`. They contain global envelope test results and curve sets for multiple summary functions/statistics.

Usage

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

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

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

## S3 method for class 'ldmppr_model_check'
plot(x, which = c("combined", "L", "F", "G", "J", "E", "V"), ...)
```

Arguments

`x`	an object of class `ldmppr_model_check`.
`...`	additional arguments passed to the underlying `plot()` method (e.g., from **GET**).
`object`	an object of class `ldmppr_model_check`.
`which`	which envelope to plot. `"combined"` plots the global envelope; otherwise one of `"L"`, `"F"`, `"G"`, `"J"`, `"E"`, `"V"`.

Details

An `ldmppr_model_check` is a list with components such as:

- `combined_env`: a global envelope test object (typically from **GET**)
- `envs`: named list of envelope test objects (e.g., `L`, `F`, `G`, `J`, `E`, `V`)
- `curve_sets`: named list of curve set objects
- `settings`: list of settings used when generating envelopes (e.g., `n_sim`, `thinning`)

Value

`print()` prints a brief summary of the diagnostic object.

`summary()` returns a `summary.ldmppr_model_check` object.

`plot()` plots the combined envelope or a selected statistic envelope.

Methods (by generic)

- `print(ldmppr_model_check)`: Print a brief summary of the diagnostic results.
- `summary(ldmppr_model_check)`: Summarize p-values from the combined and individual envelopes.
- `plot(ldmppr_model_check)`: Plot the combined envelope or a selected statistic.

Functions

- `print(summary.ldmppr_model_check)`: Print a summary produced by `summary.ldmppr_model_check`.

`ldmppr_sim` *Simulated marked point process object*

Description

`ldmppr_sim` objects are returned by `simulate_mpp`. They contain the simulated realization, an associated marked point pattern object, and metadata used to reproduce or inspect the simulation.

Usage

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

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

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

## S3 method for class 'ldmppr_sim'
as.data.frame(x, ...)

## S3 method for class 'ldmppr_sim'
nobs(object, ...)

## S3 method for class 'ldmppr_sim'
plot(x, pattern_type = "simulated", ...)

mpp.ldmppr_sim(x, ...)
```

Arguments

`x`	a `ldmppr_sim` object.
`...`	additional arguments (not used).
`object`	a `ldmppr_sim` object.
`pattern_type`	type of pattern to plot `"simulated"` (default).

Details

An `ldmppr_sim` is a list with at least:

- `process`: process name (e.g. `"self_correcting"`)
- `mpp`: a marked point pattern object
- `realization`: data.frame with columns `time`, `x`, `y`, `marks`
- `params`, `bounds`, and other metadata

Value

For methods:

`print()` prints a summary of the simulation.

`summary()` returns a `summary.ldmppr_sim` object.

`plot()` returns a ggplot visualization of the marked point pattern.

`as.data.frame()` returns the simulated realization as a data.frame.

`nobs()` returns the number of points in the realization.

`mpp()` returns the marked point pattern object.

Methods (by generic)

- `print(ldmppr_sim)`: Print a brief summary of the simulation.
- `summary(ldmppr_sim)`: Summarize a simulated realization.
- `as.data.frame(ldmppr_sim)`: Coerce to a data.frame of the simulated realization.
- `nobs(ldmppr_sim)`: Number of simulated points.
- `plot(ldmppr_sim)`: Plot the simulated marked point pattern.

Functions

- `print(summary.ldmppr_sim)`: Print a summary produced by `summary.ldmppr_sim`.
- `mpp.ldmppr_sim()`: Extract the underlying marked point pattern object.

`medium_example_data` *Medium Example Data*

Description

A medium sized example dataset consisting of 111 observations in a (50m x 50m) square domain.

Usage

```
data("medium_example_data")
```

Format

medium_example_data A data frame with 111 rows and 3 columns:

x x coordinate

y y coordinate

size Size ...

Details

The dataset was generated using the Snodgrass dataset available at https://data.ess-dive.lbl.gov/view/doi:10.15485/2476543. The full code to generate this dataset is available in the package's data_raw directory.

Source

Real example dataset. Code to generate it can be found in data_raw/medium_example_data.R.

---

plot_mpp *Plot a marked point process*

Description

Plot a marked point process

Usage

```
plot_mpp(mpp_data, pattern_type = c("reference", "simulated"))
```

Arguments

mpp_data	ppp object with marks or data.frame with columns (x, y, size).
pattern_type	type of pattern to plot ("reference" or "simulated").

Value

a ggplot object of the marked point process.

Examples

```
# Load example data
data(small_example_data)
mpp_data <- generate_mpp(
  locations = small_example_data %>% dplyr::select(x, y),
  marks = small_example_data$size,
  xy_bounds = c(0, 25, 0, 25)
)

# Plot the marked point process
plot_mpp(mpp_data, pattern_type = "reference")
```

power_law_mapping *Gentle decay (power-law) mapping function from sizes to arrival times*

Description

Gentle decay (power-law) mapping function from sizes to arrival times

Usage

```
power_law_mapping(sizes, delta)
```

Arguments

sizes	vector of sizes to be mapped to arrival times.
delta	numeric value (greater than 0) for the exponent in the mapping function.

Value

vector of arrival times.

Examples

```
# Generate a vector of sizes
sizes <- runif(100, 0, 100)

# Map the sizes to arrival times using a power-law mapping with delta = .5
power_law_mapping(sizes, .5)
```

predict_marks *Predict values from the mark distribution*

Description

Prefer using the S3 method `predict()` on an `ldmppr_mark_model`: `predict(mark_model, sim_realization = ..., xy_bounds = ...)`. This wrapper is retained for backward compatibility and is deprecated.

Usage

```
predict_marks(
  sim_realization,
  raster_list = NULL,
  scaled_rasters = FALSE,
  mark_model = NULL,
  xy_bounds = NULL,
  include_comp_inds = FALSE,
```

```
  competition_radius = 15,
  edge_correction = "none",
  seed = NULL
)
```

Arguments

sim_realization	a data.frame containing a thinned or unthinned realization from simulate_mpp (or simulate_sc). Must contain x, y, and time.
raster_list	list of raster objects used for mark prediction.
scaled_rasters	TRUE or FALSE indicating whether rasters are already scaled.
mark_model	a mark model object. May be an ldmppr_mark_model, model_fit, or workflow.
xy_bounds	vector of domain bounds as c(a_x, b_x, a_y, b_y).
include_comp_inds	TRUE or FALSE indicating whether to generate and use competition indices as covariates.
competition_radius	positive numeric distance used when include_comp_inds = TRUE.
edge_correction	type of edge correction to apply ("none" or "toroidal").
seed	optional nonnegative integer seed for reproducibility.

Value

a vector of predicted mark values.

---

scale_rasters *Scale a set of rasters*

Description

Scale a set of rasters

Usage

```
scale_rasters(raster_list, reference_resolution = NULL)
```

Arguments

raster_list	a list of raster objects.
reference_resolution	the resolution to resample the rasters to.

Value

a list of scaled raster objects.

Examples

```
# Create two example rasters
rast_a <- terra::rast(
  ncol = 10, nrow = 10,
  xmin = 0, xmax = 10,
  ymin = 0, ymax = 10,
  vals = runif(100)
)

rast_b <- terra::rast(
  ncol = 10, nrow = 10,
  xmin = 0, xmax = 10,
  ymin = 0, ymax = 10,
  vals = runif(100)
)

# Scale example rasters in a list
rast_list <- list(rast_a, rast_b)
scale_rasters(rast_list)
```

`simulate_mpp` *Simulate a realization of a location dependent marked point process*

Description

Simulate a realization of a location dependent marked point process

Usage

```
simulate_mpp(
  process = c("self_correcting"),
  process_fit = NULL,
  t_min = 0,
  t_max = 1,
  anchor_point = NULL,
  raster_list = NULL,
  scaled_rasters = FALSE,
  mark_model = NULL,
  xy_bounds = NULL,
  include_comp_inds = FALSE,
  competition_radius = 15,
  edge_correction = "none",
  thinning = TRUE,
  seed = NULL,
  mark_mode = NULL,
  size_range = NULL,
  delta = NULL
)
```

Arguments

process	type of process used (currently supports "self_correcting").
process_fit	either (1) a ldmppr_fit object returned by estimate_process_parameters, or (2) a numeric vector of length 8 giving the self-correcting process parameters: $(\alpha_1, \beta_1, \gamma_1, \alpha_2, \beta_2, \alpha_3, \beta_3, \gamma_3)$ (alpha_1, beta_1, gamma_1, alpha_2, beta_2, alpha_3, beta_3, gamma_3).
t_min	minimum value for time.
t_max	maximum value for time.
anchor_point	(optional) vector of (x,y) coordinates of the point to condition on. If NULL, inferred from the reference data (largest mark if available) or from process_fit$data_original (largest size).
raster_list	(optional) list of raster objects used for mark prediction. Required when mark_mode='mark_model' unless rasters are stored in mark_model.
scaled_rasters	TRUE or FALSE indicating whether rasters are already scaled. Ignored when mark_mode='time_to_size'.
mark_model	a mark model object used when mark_mode='mark_model'. May be an ldmppr_mark_model, model_fit, or workflow.
xy_bounds	(optional) vector of bounds as c(a_x, b_x, a_y, b_y). If NULL, bounds are inferred from process_fit when available.
include_comp_inds	TRUE or FALSE indicating whether to compute competition indices.
competition_radius	positive numeric distance used when include_comp_inds = TRUE.
edge_correction	type of edge correction to apply ("none" or "toroidal").
thinning	TRUE or FALSE indicating whether to use the thinned simulated values.
seed	integer seed for reproducibility.
mark_mode	(optional) mark generation mode: "mark_model" uses predict() on a mark model, while "time_to_size" maps simulated times back to sizes via delta. If NULL, inferred as "mark_model" when mark_model is provided, otherwise "time_to_size".
size_range	numeric vector c(smin, smax) used for mark_mode='time_to_size'. If NULL, inferred from process_fit when possible.
delta	positive scalar used for mark_mode='time_to_size'. If NULL, inferred from process_fit when possible.

Value

an object of class "ldmppr_sim".

Examples

```
# Specify the generating parameters of the self-correcting process
generating_parameters <- c(2, 8, .02, 2.5, 3, 1, 2.5, .2)

# Specify an anchor point
M_n <- c(10, 14)

# Load the raster files
raster_paths <- list.files(system.file("extdata", package = "ldmppr"),
  pattern = "\\.tif$", full.names = TRUE
)
raster_paths <- raster_paths[!grepl("_med\\.tif$", raster_paths)]
rasters <- lapply(raster_paths, terra::rast)

# Scale the rasters
scaled_raster_list <- scale_rasters(rasters)

# Load the example mark model
file_path <- system.file("extdata", "example_mark_model.rds", package = "ldmppr")
mark_model <- load_mark_model(file_path)

# Simulate a realization
example_mpp <- simulate_mpp(
  process = "self_correcting",
  process_fit = generating_parameters,
  t_min = 0,
  t_max = 1,
  anchor_point = M_n,
  raster_list = scaled_raster_list,
  scaled_rasters = TRUE,
  mark_model = mark_model,
  xy_bounds = c(0, 25, 0, 25),
  include_comp_inds = TRUE,
  competition_radius = 10,
  edge_correction = "none",
  thinning = TRUE,
  seed = 90210
)

# Plot the realization and provide a summary
plot(example_mpp, pattern_type = "simulated")
summary(example_mpp)
```

`simulate_sc` *Simulate from the self-correcting model*

Description

Allows the user to simulate a realization from the self-correcting model given a set of parameters and a point to condition on.

Usage

```
simulate_sc(
  t_min = 0,
  t_max = 1,
  sc_params = NULL,
  anchor_point = NULL,
  xy_bounds = NULL
)
```

Arguments

t_min	minimum value for time.
t_max	maximum value for time.
sc_params	a vector of parameter values corresponding to $(\alpha_1, \beta_1, \gamma_1, \alpha_2, \beta_2, \alpha_3, \beta_3, \gamma_3)$ (i.e., alpha_1, beta_1, gamma_1, alpha_2, beta_2, alpha_3, beta_3, gamma_3).
anchor_point	a vector of (x,y) coordinates of point to condition on.
xy_bounds	a vector of domain bounds (2 for x, 2 for y).

Value

a list containing the thinned and unthinned simulation realizations.

Examples

```
# Specify the generating parameters of the self-correcting process
generating_parameters <- c(2, 8, .02, 2.5, 3, 1, 2.5, .2)

# Specify an anchor point
M_n <- c(10, 14)

# Simulate the self-correcting process
generated_locs <- simulate_sc(
  t_min = 0,
  t_max = 1,
  sc_params = generating_parameters,
  anchor_point = M_n,
  xy_bounds = c(0, 25, 0, 25)
)
```

---

small_example_data *Small Example Data*

Description

A small example dataset for testing and examples consisting of 121 observations in a (25m x 25m) square domain.

Usage

```
data("small_example_data")
```

Format

small_example_data A data frame with 121 rows and 3 columns:

x x coordinate

y y coordinate

size Size ...

Details

The dataset was generated using the example raster data and an exponential decay size function.

The full code to generate this dataset is available in the package's data_raw directory.

Source

Simulated dataset. Code to generate it can be found in data_raw/small_example_data.R.

---

thin_st_fast *calculates acceptance for thinning mechanism during simulation*

Description

calculates acceptance for thinning mechanism during simulation

Usage

```
thin_st_fast(data, params)
```

Arguments

data	NumericMatrix with columns (time, x, y). Assumed sorted by time ascending.
params	NumericVector length 3: (alpha3, beta3, gamma3

Value

LogicalVector length n of whether to keep each point (true) or thin it (false).

`train_mark_model` *Train a flexible model for the mark distribution*

Description

Trains a predictive model for the mark distribution of a spatio-temporal process. `data` may be either (1) a data.frame containing columns `x`, `y`, `size` and `time`, (2) a data.frame containing `x`, `y`, `size` (time will be derived via `delta`), or (3) a `ldmppr_fit` object returned by `estimate_process_parameters`. Allows the user to incorporate location specific information and competition indices as covariates in the mark model.

Usage

```
train_mark_model(
  data,
  raster_list = NULL,
  scaled_rasters = FALSE,
  model_type = "xgboost",
  xy_bounds = NULL,
  delta = NULL,
  save_model = FALSE,
  save_path = NULL,
  parallel = FALSE,
  num_cores = NULL,
  include_comp_inds = FALSE,
  competition_radius = 15,
  edge_correction = "none",
  selection_metric = "rmse",
  cv_folds = 5,
  tuning_grid_size = 200,
  seed = 0,
  verbose = TRUE
)
```

Arguments

`data`	a data.frame or a `ldmppr_fit` object. See Description.
`raster_list`	list of raster objects used for mark-model training.
`scaled_rasters`	`TRUE` or `FALSE` indicating whether rasters are already scaled.
`model_type`	the machine learning model type (`"xgboost"` or `"random_forest"`).
`xy_bounds`	a vector of domain bounds (2 for x, 2 for y). If `data` is an `ldmppr_fit` and `xy_bounds` is `NULL`, defaults to `c(0, b_x, 0, b_y)` derived from fit.
`delta`	(optional) numeric scalar used only when `data` contains `(x,y,size)` but not `time`. If `data` is an `ldmppr_fit` and time is missing, the function will infer the `delta` value from the fit.

save_model TRUE or FALSE indicating whether to save the generated model.

save_path path for saving the generated model.

parallel TRUE or FALSE. If TRUE, tuning is parallelized over resamples. For small datasets, parallel overhead may outweigh speed gains.

num_cores number of workers to use when parallel=TRUE. Ignored when parallel=FALSE.

include_comp_inds TRUE or FALSE indicating whether to generate and use competition indices as covariates.

competition_radius positive numeric distance used when include_comp_inds = TRUE.

edge_correction type of edge correction to apply ("none", "toroidal", or "truncation").

selection_metric metric to use for identifying the optimal model ("rmse", "mae", or "rsq").

cv_folds number of cross-validation folds to use in model training. If cv_folds <= 1, tuning is skipped and the model is fit once with default hyperparameters.

tuning_grid_size size of the tuning grid for hyperparameter tuning.

seed integer seed for reproducible resampling/tuning/model fitting.

verbose TRUE or FALSE indicating whether to show progress of model training.

Value

an object of class "ldmppr_mark_model" containing the trained mark model.

Examples

```
# Load the small example data
data(small_example_data)

# Load example raster data
raster_paths <- list.files(system.file("extdata", package = "ldmppr"),
  pattern = "\\.tif$", full.names = TRUE
)
raster_paths <- raster_paths[!grepl("_med\\.tif$", raster_paths)]
rasters <- lapply(raster_paths, terra::rast)

# Scale the rasters
scaled_raster_list <- scale_rasters(rasters)


# Train the model
mark_model <- train_mark_model(
  data = small_example_data,
  raster_list = scaled_raster_list,
  scaled_rasters = TRUE,
  model_type = "xgboost",
  xy_bounds = c(0, 25, 0, 25),
```

```
  delta = 1,
  parallel = FALSE,
  include_comp_inds = FALSE,
  competition_radius = 10,
  edge_correction = "none",
  selection_metric = "rmse",
  cv_folds = 3,
  tuning_grid_size = 2,
  verbose = TRUE
)

print(mark_model)
```

Index