Package 'ldmppr'

January 13, 2026

Type Package

Title Estimate and Simulate from Location Dependent Marked Point Processes

Version 1.1.1

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, ranger, parsnip (>= 1.4.0), dials, recipes, rsample, tune, workflows, magrittr, hardhat, ggplot2, spatstat.geom, spatstat.explore, nloptr, GET, progress, future, furrr, foreach, yardstick

LinkingTo Rcpp, RcppArmadillo

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

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

RoxygenNote 7.3.3

Suggests knitr, rmarkdown, testthat (>= 3.0.0), dplyr

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 2026-01-13 08:30:07 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
)
```

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	a list of raster objects used for predicting marks.
scaled_rasters	TRUE or FALSE indicating whether the rasters have already been scaled.
mark_model	a mark model object. May be a ldmppr_mark_model or a legacy model.
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	distance for competition radius if 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.
seed	integer seed for reproducibility.

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.

# Load the small example data
data(small_example_data)

# Load the example mark model that previously was trained on the small example data
file_path <- system.file("extdata", "example_mark_model.rds", package = "ldmppr")
mark_model <- load_mark_model(file_path)

# 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)

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


# Specify the estimated parameters of the self-correcting process
# Note: These would generally be estimated using estimate_process_parameters.
```

```
# These values are estimates from the small_example_data generating script.
estimated_parameters <- c(
  0.05167978, 8.20702166, 0.02199940, 2.63236890,
  1.82729512, 0.65330061, 0.86666748, 0.04681878
)

# Check the model fit
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
)

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`. For the self-correcting process, the arrival times must be on $(0, 1)$ and can either be supplied directly in `data` as `time`, or constructed from `size` via the gentle-decay (power-law) mapping `power_law_mapping` using `delta` (single fit) or `delta_values` (delta search).

Usage

```
estimate_process_parameters(
  data,
  process = c("self_correcting"),
  x_grid = NULL,
  y_grid = NULL,
  t_grid = NULL,
```

```
  upper_bounds = NULL,
  parameter_inits = NULL,
  delta = NULL,
  delta_values = NULL,
  parallel = FALSE,
  num_cores = max(1L, parallel::detectCores() - 1L),
  set_future_plan = FALSE,
  strategy = c("local", "global_local", "multires_global_local"),
  grid_levels = NULL,
  refine_best_delta = TRUE,
  global_algorithm = "NLOPT_GN_CRS2_LM",
  local_algorithm = "NLOPT_LN_BOBYQA",
  global_options = list(maxeval = 150),
  local_options = list(maxeval = 300, xtol_rel = 1e-05, maxtime = NULL),
  n_starts = 1L,
  jitter_sd = 0.35,
  seed = 1L,
  finite_bounds = NULL,
  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 for delta search, it must have column names `c("x","y","size")`.
`process`	character string specifying the process model. Currently supports `"self_correcting"`.
`x_grid, y_grid, t_grid`	numeric vectors defining the integration grid for (x, y, t).
`upper_bounds`	numeric vector of length 3 giving `c(b_t, b_x, b_y)`.
`parameter_inits`	numeric vector of length 8 giving initialization values for the model parameters.
`delta`	(optional) numeric scalar used only when `data` contains `(x,y,size)` but not `time`.
`delta_values`	(optional) numeric vector. If supplied, the function fits the model for each value of `delta_values` (mapping `size -> time` via `power_law_mapping`) and returns the best fit (lowest objective).
`parallel`	logical. If `TRUE`, uses furrr/future to parallelize either (a) over `delta_values` (when provided) or (b) over multi-start initializations (when `delta_values` is `NULL` and `n_starts > 1`).
`num_cores`	Integer number of workers to use when `set_future_plan = TRUE`.
`set_future_plan`	`TRUE` or `FALSE`, if `TRUE`, temporarily sets `future::plan(multisession, workers = num_cores)` and restores the original plan on exit.
`strategy`	Character string specifying the estimation strategy: - `"local"`: single-level local optimization from `parameter_inits`. - `"global_local"`: single-level global

optimization (from parameter_inits) followed by local polish. - "multires_global_local": multi-resolution fitting over grid_levels (coarsest level uses global + local; finer levels use local polish only).

grid_levels (optional) list defining the multi-resolution grid schedule when strategy = "multires_global_local". Each entry can be a numeric vector c(nx, ny, nt) or a list with named entries list(nx=..., ny=..., nt=...). If NULL, uses the supplied (x_grid, y_grid, t_grid) as a single level.

refine_best_delta
TRUE or FALSE, if TRUE and delta_values is supplied, performs a final refinement fit at the best delta found using the full multi-resolution strategy.

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

global_options, local_options
named lists of options to pass to nloptr::nloptr() for the global and local optimization stages, respectively.

n_starts integer number of multi-start initializations to use for the local optimization stage.

jitter_sd numeric standard deviation used to jitter the multi-start initializations.

seed integer random seed used for multi-start initialization jittering.

finite_bounds (optional) list with components lb and ub giving finite lower and upper bounds for all 8 parameters. Used only when the selected optimization algorithms require finite bounds.

verbose TRUE or FALSE, if TRUE, prints progress messages during fitting.

Details

For the self-correcting process, the log-likelihood integral is approximated using the supplied grid (x_grid, y_grid, t_grid) over the bounded domain upper_bounds. When delta_values is supplied, this function performs a grid search over delta values, fitting the model separately for each mapped dataset and selecting the best objective value.

Value

an object of class "ldmppr_fit" containing the best nloptr fit and (optionally) all fits from 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

```
data(small_example_data)

x_grid <- seq(0, 25, length.out = 5)
y_grid <- seq(0, 25, length.out = 5)
t_grid <- seq(0, 1,  length.out = 5)

parameter_inits <- c(1.5, 8.5, .015, 1.5, 3.2, .75, 3, .08)
upper_bounds <- c(1, 25, 25)

fit <- estimate_process_parameters(
  data = small_example_data,
  process = "self_correcting",
  x_grid = x_grid,
  y_grid = y_grid,
  t_grid = t_grid,
  upper_bounds = upper_bounds,
  parameter_inits = parameter_inits,
  delta = 1,
  strategy = "global_local",
  global_algorithm = "NLOPT_GN_CRS2_LM",
  local_algorithm = "NLOPT_LN_BOBYQA",
  global_options = list(maxeval = 150),
  local_options = list(maxeval = 25, xtol_rel = 1e-2),
  verbose = TRUE
)

coef(fit)
logLik(fit)


# Delta-search example (data has x,y,size; time is derived internally for each delta)
fit_delta <- estimate_process_parameters(
  data = small_example_data, # x,y,size
  process = "self_correcting",
  x_grid = x_grid,
  y_grid = y_grid,
  t_grid = t_grid,
  upper_bounds = upper_bounds,
  parameter_inits = parameter_inits,
  delta_values = c(0.35, 0.5, 0.65, 0.9, 1.0),
  parallel = TRUE,
  set_future_plan = TRUE,
  num_cores = 2,
  strategy = "multires_global_local",
  grid_levels = list(
  list(nx = 5, ny = 5, nt = 5),
  list(nx = 8, ny = 8, nt = 8),
  list(nx = 10, ny = 10, nt = 10)
  ),
  global_options = list(maxeval = 100),
  local_options  = list(maxeval = 100, xtol_rel = 1e-3),
```

```
  n_starts = 3,
  refine_best_delta = TRUE,
  verbose = TRUE
)
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 of (x,y) locations with names "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_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, ...)
```

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.

Functions

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

`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,
  info = list()
)

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

predict.ldmppr_mark_model(object, new_data, ...)

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.
info	(optional) list of metadata.
x	a ldmppr_mark_model object.
...	passed to methods.
object	a ldmppr_mark_model object.
new_data	a data frame of predictors (and possibly outcome columns).
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.
- save_mark_model(ldmppr_mark_model): Save method for ldmppr_mark_model.

Functions

- ldmppr_mark_model(): Create a mark model container.
- predict.ldmppr_mark_model(): Predict marks for new data.
- 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'
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.

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.
- 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

- 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

Predict values from the mark distribution

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"
)
```

Arguments

sim_realization	a data.frame containing a thinned or unthinned realization from simulate_mpp (or simulate_sc).
raster_list	a list of raster objects.
scaled_rasters	TRUE or FALSE indicating whether the rasters have been scaled.
mark_model	a mark model object. May be a ldmppr_mark_model or a legacy model.
xy_bounds	a vector of domain bounds (2 for x, 2 for y).
include_comp_inds	TRUE or FALSE indicating whether to generate and use competition indices as covariates.
competition_radius	distance for competition radius if include_comp_inds is TRUE.
edge_correction	type of edge correction to apply ("none" or "toroidal").

Value

a vector of predicted mark values.

Examples

```
# Simulate a realization
generating_parameters <- c(2, 8, .02, 2.5, 3, 1, 2.5, .2)
M_n <- c(10, 14)
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)
)

# 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)

# Predict the mark values
predict_marks(
```

```
  sim_realization = generated_locs$thinned,
  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"
)
```

`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
)
```

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 self-correcting 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`	a list of raster objects used for predicting marks.
`scaled_rasters`	`TRUE` or `FALSE` indicating whether the rasters have already been scaled.
`mark_model`	a mark model object. May be a `ldmppr_mark_model` or a legacy model.

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.

competition_radius
distance for competition radius if 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.

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
)

# 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`	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`	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`.

`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 = TRUE,
  n_cores = NULL,
  include_comp_inds = FALSE,
  competition_radius = 15,
  edge_correction = "none",
  selection_metric = "rmse",
  cv_folds = 5,
  tuning_grid_size = 200,
  verbose = TRUE
)
```

Arguments

`data`	a data.frame or a `ldmppr_fit` object. See Description.
`raster_list`	a list of raster objects.
`scaled_rasters`	`TRUE` or `FALSE` indicating whether the rasters have been 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 indicating whether to use parallelization in model training.

n_cores — number of cores to use in parallel model training (if parallel is TRUE).

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

competition_radius — distance for competition radius if include_comp_inds is 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.

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