Package ‘multiScaleR’

May 9, 2026

Type Package

Title Methods for Optimizing Scales of Effect

Version 0.6.13

Description
A tool for optimizing scales of effect when modeling ecological processes in space. Specifically, the scale parameter of a distance-weighted kernel distribution is identified for all environmental layers included in the model. Includes functions to assist in model selection, model evaluation, efficient transformation of raster surfaces using fast Fourier transformation, and projecting models. For more details see Peterman (2026) <doi:10.1007/s10980-025-02267-x>.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Imports Rcpp, Matrix, cowplot, dplyr, fields, ggplot2, insight, stats, utils, unmarked, exactextractr, crayon, parallel, optimParallel, AICcmodavg, methods, pscl

LinkingTo Rcpp, RcppArmadillo

Depends R (>= 4.3), terra, sf

Suggests knitr, rmarkdown, MASS, nlme, pkgload, survival, testthat (>= 3.0.0)

Config/testthat/edition 3

URL `https://github.com/wpeterman/multiScaleR`

BugReports `https://github.com/wpeterman/multiScaleR/issues`

BuildVignettes true

VignetteBuilder knitr

Author Bill Peterman [aut, cre] (ORCID: <`https://orcid.org/0000-0001-5229-9268`>)

Maintainer Bill Peterman <`Peterman.73@osu.edu`>

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-04-18 16:40:18 UTC

Contents

`multiScaleR-package` *multiScaleR*

Description

This package is for optimizing scales of effect when modeling ecological processes in space. Specifically, the scale parameter of a distance-weighted kernel distribution is identified for all environmental layers included in the model.

Details

Author(s)

Maintainer: Bill Peterman <`Peterman.73@osu.edu`> (ORCID)

See Also

Useful links:

- `https://github.com/wpeterman/multiScaleR`
- Report bugs at `https://github.com/wpeterman/multiScaleR/issues`

`aic_tab` *multiScaleR model selection*

Description

Function to create AIC(c) table of fitted models

Usage

```
aic_tab(mod_list,
        AICc = TRUE,
        mod_names = NULL,
        verbose = FALSE,
        ...)
```

Arguments

`mod_list`	List containing fitted ‘multiScaleR‘ objects
`AICc`	Use second order AIC in ranking models (Default = TRUE). See Details
`mod_names`	Optional. Specify names for fitted model objects. By default, the right hand side of the fitted ‘multiScaleR‘ model, in combination with the kernel, will be used as the model name.
`verbose`	(Default = FALSE) Should the table be printed to the console
`...`	Additional arguments (Not used)

Details

aic_tab creates a model selection table using `aictabCustom` from the ‘AICcmodavg‘ package

Value

Data frame of class ‘aictab‘ with AIC summary table for provided models

Author(s)

Bill Peterman

Examples

```
## Simulate data
set.seed(555)

points <- vect(cbind(c(5,7,9,11,13),
                     c(13,11,9,7,5)))

mat_list <- list(r1 = rast(matrix(rnorm(20^2),
                                  nrow = 20)),
                 r2 = rast(matrix(rnorm(20^2),
                                  nrow = 20)))
rast_stack <- rast(mat_list)
kernel_inputs <- kernel_prep(pts = points,
                             raster_stack = rast_stack,
                             max_D = 5,
                             kernel = 'gaussian',
                             sigma = NULL)
## Example response data
y <- rnorm(5)

## Create data frame with raster variables
dat <- data.frame(y = y,
                  kernel_inputs$kernel_dat)
mod1 <- glm(y ~ r1,
            data = dat)
mod2 <- glm(y ~ r2,
            data = dat)
mod3 <- glm(y ~ r1 + r2,
            data = dat)

## NOTE: This code is only for demonstration
## Optimization results will have no meaning

opt_mod1 <- multiScale_optim(fitted_mod = mod1,
                             kernel_inputs = kernel_inputs,
                             par = NULL,
                             n_cores = NULL)

opt_mod2 <- multiScale_optim(fitted_mod = mod2,
                             kernel_inputs = kernel_inputs,
                             par = NULL,
                             n_cores = NULL)
opt_mod3 <- multiScale_optim(fitted_mod = mod3,
                             kernel_inputs = kernel_inputs,
                             par = NULL,
                             n_cores = NULL)

 ## AIC table
 mod_list <- list(opt_mod1, opt_mod2, opt_mod3)

 aic_tab(mod_list = mod_list,
         AICc = FALSE)
```

```
## AICc table with specified names
aic_tab(mod_list = mod_list,
        AICc = TRUE,
        mod_names = c('mod1', 'mod2', 'mod3'))
```

`bic_tab` *multiScaleR model selection*

Description

Function to create BIC table of fitted models

Usage

```
bic_tab(mod_list,
        mod_names = NULL,
        verbose = FALSE,
        ...)
```

Arguments

`mod_list`	List containing fitted ‘multiScaleR‘ objects
`mod_names`	Optional. Specify names for fitted model objects. By default, the right hand side of the fitted ‘multiScaleR‘ model, in combination with the kernel, will be used as the model name.
`verbose`	(Default = FALSE) Should the table be printed to the console
`...`	Additional arguments (Not used)

Details

bic_tab creates a model selection table using `bictabCustom` from the ‘AICcmodavg‘ package

Value

Data frame of class ‘bictab‘ with BIC summary table for provided models

Author(s)

Bill Peterman

Examples

```
## Simulate data
set.seed(555)

points <- vect(cbind(c(5,7,9,11,13),
                     c(13,11,9,7,5)))

mat_list <- list(r1 = rast(matrix(rnorm(20^2),
                                  nrow = 20)),
                 r2 = rast(matrix(rnorm(20^2),
                                  nrow = 20)))
rast_stack <- rast(mat_list)
kernel_inputs <- kernel_prep(pts = points,
                             raster_stack = rast_stack,
                             max_D = 5,
                             kernel = 'gaussian',
                             sigma = NULL)
## Example response data
y <- rnorm(5)

## Create data frame with raster variables
dat <- data.frame(y = y,
                  kernel_inputs$kernel_dat)
mod1 <- glm(y ~ r1,
            data = dat)
mod2 <- glm(y ~ r2,
            data = dat)
mod3 <- glm(y ~ r1 + r2,
            data = dat)

## NOTE: This code is only for demonstration
## Optimization results will have no meaning

opt_mod1 <- multiScale_optim(fitted_mod = mod1,
                             kernel_inputs = kernel_inputs,
                             par = NULL,
                             n_cores = NULL)

opt_mod2 <- multiScale_optim(fitted_mod = mod2,
                             kernel_inputs = kernel_inputs,
                             par = NULL,
                             n_cores = NULL)
opt_mod3 <- multiScale_optim(fitted_mod = mod3,
                             kernel_inputs = kernel_inputs,
                             par = NULL,
                             n_cores = NULL)

 ## BIC table
 mod_list <- list(opt_mod1, opt_mod2, opt_mod3)

 bic_tab(mod_list = mod_list)
```

```
## BIC table with specified names
bic_tab(mod_list = mod_list,
        mod_names = c('mod1', 'mod2', 'mod3'))
```

`count_data` *Example data frame*

Description

Example count data to be used for optimizing scales of effect

Usage

```
data(count_data)
```

Format

A data frame with 75 rows and 2 columns. Data were simulated from a Poisson distribution with an intercept of 0.5, a ‘hab‘ effect of 0.75, and scale of effect (sigma) of 75.

y –> Simulated counts at spatial locations

hab –> Scaled and centered weighted mean values from the ‘hab‘ raster at each of the ‘pts‘

`diagnostics` *Retrieve diagnostics from multiScaleR objects*

Description

Returns structured warning/diagnostic information stored on fitted `multiScaleR` objects.

Usage

```
diagnostics(object, ...)

## S3 method for class 'multiScaleR'
diagnostics(object, ...)
```

Arguments

`object`	An object to inspect.
`...`	Additional arguments passed to methods.

Value

For `multiScaleR` objects, a named list of diagnostics.

`hab` *Example raster*

Description

Example habitat raster for optimizing scales of effect

Format

A binary SpatRaster object

hab –> A binary raster

Examples

```
hab <- terra::rast(system.file("extdata",
                   "hab.tif", package = 'multiScaleR'))
```

`kernel_dist` *Scale Distance*

Description

Function to estimate the effective distance encompassing a specified cumulative probability density of the kernel function

Usage

```
kernel_dist(model, prob = 0.9, ...)
```

Arguments

`model`	`multiScale_optim` object of class 'multiScaleR'
`prob`	Density probability cutoff for calculating distance, Default: 0.9
`...`	Parameters to be used if not providing a 'multiScaleR' fitted object. See Details

Details

This function is used to determine the distance at which kernel density distributions have influence. If not providing a fitted model, you can plot kernel distributions by specifying (1) sigma, (2) beta (if using exponential power), and (3) the kernel transformation ('exp' = negative exponential, 'gaussian', 'fixed' = fixed buffer, and 'expow' = exponential power)

Value

Numeric. Distance at which the cumulative kernel density reaches the specified proportion.

See Also

plot.multiScaleR

Examples

```
## Using package data
data('pts')
data('count_data')
hab <- terra::rast(system.file('extdata',
                   'hab.tif', package = 'multiScaleR'))

kernel_inputs <- kernel_prep(pts = pts,
                             raster_stack = hab,
                             max_D = 250,
                             kernel = 'gaussian')

mod <- glm(y ~ hab,
           family = poisson,
           data = count_data)

## Optimize scale
opt <- multiScale_optim(fitted_mod = mod,
                        kernel_inputs = kernel_inputs)

## Uses of `kernel_dist`
kernel_dist(model = opt)
kernel_dist(model = opt, prob = 0.95)
kernel_dist(sigma = 500, kernel = 'gaussian', prob = 0.95)
kernel_dist(sigma = 100, prob = 0.975, kernel = "exp")
kernel_dist(sigma = 100, prob = 0.95, kernel = "expow", beta = 1.5)
kernel_dist(sigma = 100, kernel = "fixed")
```

`kernel_prep` *Kernel Scale Preparation*

Description

Function to prepare data inputs for kernel scale analysis

Usage

```
kernel_prep(
  pts,
  raster_stack,
  max_D,
  kernel = c("gaussian", "exp", "expow", "fixed"),
  sigma = NULL,
```

```
  shape = NULL,
  projected = TRUE,
  progress = FALSE,
  verbose = TRUE
)
```

Arguments

`pts`	Point locations provided as ‘SpatVector‘ or ‘sf‘ objects
`raster_stack`	Raster layer(s) of class ‘SpatRaster‘
`max_D`	The maximum distance to consider during the scale optimization
`kernel`	Kernel function to be used (’gaussian’, ’exp’, ’fixed’, ’expow’; Default: ’gaussian’)
`sigma`	Initial values for optimizing the scale parameter. Default: NULL, initial values will be automatically generated. This is recommended.
`shape`	Initial values for optimizing the shape parameter if using exponential power kernel. Default: NULL, starting values will be automatically generated. This is recommended.
`projected`	Logical. Are ‘pts‘ and ‘raster_stack‘ projected. Function currently requires that both are projected. Default: TRUE
`progress`	Should progress bars be printed to console. Default: FALSE
`verbose`	Logical. Print preparation information to the console. Default: TRUE

Details

Spatial point locations and raster layers should have a defined projection and be the same CRS. If providing starting values for ‘sigma‘ or ‘shape‘, it must be a vector of length equal to the number of raster layers for which scale is being assessed and should be provided in the unit of the used projection. When specifying ‘max_D‘, ensure that your raster layers adequately extend beyond the points provided so that the surrounding landscape can be meaningfully sampled during scale optimization. Row names from ‘pts‘ are preserved in the returned kernel data, distance list, and raw covariate list so downstream model data can be aligned to the original point order.

Value

A list of class ‘multiscaleR‘ with necessary elements to conduct scale optimization using the ‘multiScale_optim‘ function

Examples

```
library(terra)
pts <- vect(cbind(c(3,5,7),
                  c(7,5,3)))

mat_list <- list(r1 = rast(matrix(rnorm(100),
                                  nrow = 10)),
                 r2 = rast(matrix(rnorm(100),
                                  nrow = 10)))
```

```
rast_stack <- rast(mat_list)
kernel_inputs <- kernel_prep(pts = pts,
                             raster_stack = rast_stack,
                             max_D = 2,
                             kernel = 'gaussian',
                             sigma = NULL)
```

`kernel_scale.raster` *Create scaled rasters*

Description

Function to create scaled rasters

Usage

```
kernel_scale.raster(
  raster_stack,
  sigma = NULL,
  multiScaleR = NULL,
  shape = NULL,
  kernel = c("gaussian", "exp", "expow", "fixed"),
  pct_wt = 0.975,
  fft = TRUE,
  scale_center = FALSE,
  clamp = FALSE,
  pct_mx = 0,
  na.rm = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`raster_stack`	Stack of combined ‘SpatRaster‘ layers
`sigma`	Vector of parameters listed in order to scale each raster
`multiScaleR`	If scale optimization with ‘multiScale_optim‘ has been completed, provide the ‘multiScaleR‘ object here. You can also pass an object of class ‘"multiScaleR_data"‘ created using ‘kernel_prep‘. Default: NULL
`shape`	Vector of parameters listed in order to scale each raster if using ‘’expow’‘ kernel. Default: NULL
`kernel`	Kernel function to be used (’gaussian’, ’exp’, ’fixed’, ’expow’; Default: ’gaussian’)
`pct_wt`	The percentage of the weighted density to include when applying the kernel smoothing function, Default: 0.975

`fft`	Logical. If TRUE (Default), a fast Fourier transformation will be used to smooth the raster surface. See details.
`scale_center`	Logical. If ‘TRUE‘, raster values are scaled and centered according to the data used to fit the model. Necessary when predicting model results across the landscape.
`clamp`	Logical. If ‘TRUE‘, scaled values are clamped to the covariate range in the model data.
`pct_mx`	Numeric. If ‘clamp‘ is ‘TRUE‘, this value specifies the amount (percentage; positive or negative) by which to expand or contract the min/max range when clamping. Can range from -0.99–0.99 (Default = 0).
`na.rm`	Logical. If TRUE (Default), NA values are removed from the weighted mean calculation.
`verbose`	Logical. Print status of raster scaling to the console. Default: TRUE
`...`	Not used

Details

The fast Fourier transformation is substantially faster when scaling large raster surfaces with large kernel areas. There will be some edge effects on the outer boundaries.

When a fitted ‘multiScaleR‘ object is supplied, ‘kernel_scale.raster‘ inspects the underlying model to identify all predictors used during model fitting. If the model includes site-level covariates (i.e., predictors not associated with raster layers), these variables are not spatially explicit and therefore cannot be directly projected.

In such cases, the function automatically creates constant raster layers for these covariates. By default, these layers are filled with a value of 0, which corresponds to the reference value for centered and scaled predictors. This ensures compatibility with the fitted model during prediction (e.g., when using ‘terra::predict‘).

Users should be aware that these dummy rasters do not represent spatial variation in the covariate. Instead, they define a fixed projection scenario (e.g., predictions conditional on the covariate being equal to 0). For non-centered variables or alternative projection scenarios, users should manually modify or replace these layers.

Categorical (factor) covariates are not automatically converted to raster layers. If present in the model, a warning is issued and no dummy raster is created.

Value

A ‘SpatRaster‘ object containing scaled rasters. If a ‘multiScaleR‘ object is provided and the fitted model includes additional site-level covariates that are not represented in ‘raster_stack‘, constant ("dummy") raster layers will be added for those covariates to facilitate spatial prediction.

Examples

```
## Not Run
r1 <- rast(matrix(rnorm(25^2),
                  nrow = 25))
```

```
r1_s <- kernel_scale.raster(r1,
                            sigma = 4,
                            kernel = 'gaussian')
plot(c(r1, r1_s))
```

`landscape` *Simulated raster*

Description

Raster data for use with vignette example

Format

'landscape_rast'

A spatRaster object with three surfaces:

land1 –> A binary landscape surface with low autocorrelation

land2 –> A continuous landscape surface with low autocorrelation

land3 –> A continuous landscape surface with high autocorrelation

Examples

```
land_rast <- terra::rast(system.file("extdata",
                         "landscape.tif", package = 'multiScaleR'))
```

`landscape_counts` *Example data frame*

Description

Example count data to be used vignette document example

Usage

```
data(landscape_counts)
```

Format

A data frame with 100 rows and 2 columns. Data were simulated from a Poisson distribution with an intercept of 0.25; land1 effect = -0.5; site effect = 0.3; land2 effect = 0.7. True simulated Gaussian scale effects (sigma): land1 = 250; land2 = 500. For use with package vignette.

counts –> Simulated counts at spatial locations

site –> A habitat variable measured at the site

`multiScale_optim` *Multiscale optimization*

Description

Function to conduct multiscale optimization

Usage

```
multiScale_optim(
  fitted_mod,
  kernel_inputs,
  join_by = NULL,
  par = NULL,
  n_cores = NULL,
  PSOCK = FALSE,
  verbose = TRUE,
  refit_fn = NULL
)
```

Arguments

`fitted_mod`	Model object of class glm, lm, gls, or unmarked
`kernel_inputs`	Object created from running `kernel_prep`
`join_by`	Default: NULL. A data frame containing the variable used to join spatial point data with observation data (see Details)
`par`	Optional starting values for parameter estimation. If provided, should be divided by the ‘max_D‘ value to be appropriately scaled. Default: NULL
`n_cores`	If attempting to optimize in parallel, the number of cores to use. Default: NULL
`PSOCK`	Logical. If attempting to optimize in parallel on a Windows machine, a PSOCK cluster will be created. If using a Unix OS a FORK cluster will be created. You can force a Unix system to create a PSOCK cluster by setting to TRUE. Default: FALSE
`verbose`	Logical. Print status of optimization to the console. Default: TRUE
`refit_fn`	Optional function used to refit ‘fitted_mod‘ during optimization. If provided, it must accept named arguments ‘model‘, ‘data‘, and ‘context‘, and return a fitted model object with a usable log-likelihood. See Details.

Details

Identifies the kernel scale, and uncertainty of that scale, for each raster within the context of the fitted model provided. Summary methods use profile-likelihood confidence intervals for ‘sigma‘ when feasible, while reported standard errors remain Hessian-based approximations from the outer optimization.

To ensure that fitted model function calls are properly parallelized, fit models directly from the packages. For example, fit a negative binomial distribution from the MASS package as ‘fitted_mod <- MASS::glm.nb(y ~ x, data = df)‘

There may situations when using ‘unmarked‘ where sites are sampled across multiple years, but spatial environmental values are relevant for all years. In this situation, you want to join the scaled landscape variables from each site to each observation at a site. This can be achieved by providing a data frame object containing the values (e.g. site names) that will be used to join spatial data to sites. The name of the column in the ‘join_by‘ data frame must match a column name in the data used to fit your ‘unmarked‘ model.

During optimization, ‘multiScale_optim()‘ repeatedly replaces the scaled raster covariates in the original model data and refits the model. For most supported model classes, this is done internally with ‘stats::update()‘ for standard model objects or ‘unmarked::update()‘ for ‘unmarked‘ models. If a model class cannot be refit correctly by the default path, pass ‘refit_fn‘. This function must have the form ‘function(model, data, context)‘ and return a fitted model object. The refitted object must work with ‘stats::logLik()‘ or ‘insight::get_loglikelihood()‘, unless it is an ‘unmarked‘ model with a ‘negLogLike‘ slot.

A minimal custom refit function is:

```
refit_fn <- function(model, data, context) {
  stats::update(model, data = data)
}
```

For models that need to be rebuilt from their original call, use namespace-qualified model-fitting calls inside ‘refit_fn‘ and make sure any required objects are available to the function:

```
refit_fn <- function(model, data, context) {
  call <- model$call
  call$data <- quote(data)
  eval(call, envir = list(data = data), enclos = parent.frame())
}
```

When using ‘n_cores‘ with a PSOCK cluster, ‘refit_fn‘ must be serializable and should avoid hidden dependencies on local workspace objects. Prefer namespace-qualified calls such as ‘stats::update()‘ or ‘survival::coxph()‘. If the function closes over helper objects, those objects must also serialize cleanly to worker processes.

Value

Returns a list of class ‘multiScaleR‘ containing scale estimates, shape estimates (if using kernel = ’expow’), optimization results, and the final optimized model.

See Also

`kernel_dist`

Examples

```
set.seed(555)

points <- vect(cbind(c(5,7,9,11,13),
                     c(13,11,9,7,5)))

mat_list <- list(r1 = rast(matrix(rnorm(20^2),
                                  nrow = 20)),
                 r2 = rast(matrix(rnorm(20^2),
                                  nrow = 20)))
rast_stack <- rast(mat_list)
kernel_inputs <- kernel_prep(pts = points,
                             raster_stack = rast_stack,
                             max_D = 5,
                             kernel = 'gaussian',
                             sigma = NULL)
## Example response data
y <- rnorm(5)

## Create data frame with raster variables
dat <- data.frame(y = y,
                  kernel_inputs$kernel_dat)
mod1 <- glm(y ~ r1 + r2,
            data = dat)

## NOTE: This code is only for demonstration
## Optimization results will have no meaning
opt_mod <- multiScale_optim(fitted_mod = mod1,
                            kernel_inputs = kernel_inputs,
                            par = NULL,
                            n_cores = NULL)

## Using package data
data('pts')
data('count_data')
hab <- terra::rast(system.file('extdata',
                   'hab.tif', package = 'multiScaleR'))

kernel_inputs <- kernel_prep(pts = pts,
                             raster_stack = hab,
                             max_D = 250,
                             kernel = 'gaussian')

mod <- glm(y ~ hab,
           family = poisson,
           data = count_data)

## Optimize scale
opt <- multiScale_optim(fitted_mod = mod,
                        kernel_inputs = kernel_inputs)

## Summary of fitted model
```

```
summary(opt)

## 'True' parameter values data were simulated from:
# hab scale = 75
# Intercept = 0.5,
# hab slope estimate = 0.75

## Plot and visualize kernel density
plot(opt)


## Apply optimized kernel to the environmental raster
opt_hab <- kernel_scale.raster(hab, multiScaleR = opt)

plot(c(hab, opt_hab))

## Project model; scale & center
opt_hab.s_c <- kernel_scale.raster(raster_stack = hab,
                                   multiScaleR = opt,
                                   scale_center = TRUE)

mod_pred <- predict(opt_hab.s_c, opt$opt_mod, type = 'response')
plot(mod_pred)

## Custom refit hook for model classes that need explicit control.
## This example still uses glm(), but the same pattern can be used for
## classes whose default update path is not sufficient.
refit_glm <- function(model, data, context) {
  stats::update(model, data = data)
}

opt_custom <- multiScale_optim(fitted_mod = mod,
                               kernel_inputs = kernel_inputs,
                               refit_fn = refit_glm)
```

`plot.multiScaleR` *Plot method for multiScaleR objects*

Description

Plot kernel weight distributions from optimized `multiScaleR` objects.

Usage

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

Arguments

`x`	An object of class `multiScaleR`.
`...`	Arguments to modify the plot. See Details.

Details

Supported arguments include:

- `prob`: Cumulative weight cutoff for distance scale (default = 0.9).
- `scale_dist`: Logical; add vertical line for distance scale (default = TRUE).
- `add_label`: Logical; annotate scale distance and CI (default = TRUE).

Value

A list of ggplot2 objects.

See Also

`plot_kernel`

Examples

```
## Using package data
data('pts')
data('count_data')
hab <- terra::rast(system.file('extdata',
                   'hab.tif', package = 'multiScaleR'))

kernel_inputs <- kernel_prep(pts = pts,
                             raster_stack = hab,
                             max_D = 250,
                             kernel = 'gaussian')

mod <- glm(y ~ hab,
           family = poisson,
           data = count_data)

## Optimize scale
opt <- multiScale_optim(fitted_mod = mod,
                        kernel_inputs = kernel_inputs)

plot(opt)

plot(opt, prob = 0.95)

plot(opt, scale_dist = FALSE)

plot(opt, scale_dist = TRUE, add_label = FALSE)
```

`plot.sigma_profile` *Plot Sigma Profile*

Description

Plots the profiled log-likelihood or AICc across sigma values for each spatial covariate. The optimized sigma is marked with a vertical red line.

Usage

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

Arguments

`x`	An object of class `sigma_profile` from `profile_sigma`.
`...`	Additional arguments (not currently used).

Value

A list of `ggplot` objects (one per covariate), returned invisibly. Plots are printed as a side effect.

See Also

`profile_sigma`

`plot_kernel` *Plot kernel densities*

Description

Generic function to plot kernels

Usage

```
plot_kernel(
  prob = 0.9,
  sigma,
  beta = NULL,
  kernel,
  scale_dist = TRUE,
  add_label = TRUE,
  ...
)
```

Arguments

`prob`	Cumulative kernel density to identify scale of effect distance, Default: 0.9
`sigma`	Value of scaling parameter, sigma
`beta`	Numeric. Shape parameter for exponential power kernel. Ignored unless kernel = "expow". Values between 1-50 are typically valid. (Default = NULL)
`kernel`	Kernel function to use. Valid functions are c('exp', 'gaussian', fixed', 'expow'). See details
`scale_dist`	Logical. If TRUE (Default), the distance at which the specified density probability is achieved is added to the plot along with 95% confidence interval
`add_label`	Logical. If TRUE (Default), the distance value calculated for 'scale_dist' is added as an annotation to the plot.
`...`	Not used

Details

This function is used to visualize kernel density distributions without having a fitted multiScaleR optimized object. Requires (1) sigma, (2) beta (if using exponential power), and (3) the kernel transformation ('exp' = negative exponential, 'gaussian', 'fixed' = fixed buffer, and 'expow' = exponential power)

Value

ggplot2 objects of kernel density distributions

Examples

```
#' ## General use of plot method
plot_kernel(prob = 0.95,
            sigma = 100,
            kernel = 'gaussian')
plot_kernel(prob = 0.95,
            sigma = 100,
            kernel = 'exp')
plot_kernel(prob = 0.95,
            sigma = 100,
            kernel = 'fixed')
plot_kernel(prob = 0.95,
            sigma = 100,
            beta = 2.5,
            kernel = 'expow')
```

`plot_marginal_effects` *Plot Marginal Effects from a Fitted Model*

Description

Generates marginal effect plots with 95 in a fitted model stored within a ‘multiScaleR‘ object.

Usage

```
plot_marginal_effects(
  x,
  ylab = "Estimated response",
  length.out = 100,
  type = "state",
  link = FALSE
)
```

Arguments

`x`	A ‘multiScaleR‘ object containing at least the elements ‘opt_mod‘ (the fitted model) and ‘scl_params‘ (a list with ‘mean‘ and ‘sd‘ for each covariate used for scaling).
`ylab`	Character. Y-axis label for the marginal effect plots. Default is ‘"Estimated response"‘.
`length.out`	Integer. Number of points at which to evaluate the marginal effect curve. Default is 100.
`type`	For ‘unmarked‘ models, Default is ‘"state"‘
`link`	Logical. An optional switch to predict values on the response scale. Default = ‘FALSE‘. If predicted values seem incorrect, try switching to ‘TRUE‘

Details

For ‘unmarked‘ models, predictions are made using ‘type = "state"‘ and the ‘predict‘ method for state variables. For other models (e.g., ‘lm‘, ‘glm‘), predictions are made using the standard ‘predict(..., se.fit = TRUE)‘ call and transformed by the model’s inverse link function .

Value

A named list of ‘ggplot‘ objects, one for each covariate, showing the predicted response and 95 while holding other covariates at their mean values.

`print.multiScaleR` *Print method for multiScaleR*

Description

Print method for objects of class `multiScaleR`.

Usage

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

Arguments

`x`	A `multiScaleR` object
`...`	Ignored

Value

Invisibly returns the input `multiScaleR` object

`print.multiScaleR_data` *Print method for multiScaleR_data*

Description

Print method for objects of class `multiScaleR_data`.

Usage

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

Arguments

`x`	A `multiScaleR_data` object
`...`	Ignored

Value

Invisibly returns the input `multiScaleR_data` object

`print.summary_multiScaleR` *Print method for summary_multiScaleR*

Description

Print method for objects of class `summary_multiScaleR`.

Usage

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

Arguments

`x`	A `summary_multiScaleR` object
`...`	Ignored

Value

Invisibly returns the input `summary_multiScaleR` object

`profile_sigma` *Profile Model Fit Across Sigma Parameter Space*

Description

Evaluates the model log-likelihood and AICc at a series of sigma values spanning the optimization range for each spatial covariate. This provides a diagnostic view of the likelihood surface and helps assess whether the optimized sigma is at a clear minimum or on a flat plateau.

Usage

```
profile_sigma(
  x,
  n_pts = 10,
  metric = c("AICc", "LL"),
  verbose = TRUE,
  spacing = c("log", "linear"),
  sigma_values = NULL,
  sigma_range = NULL
)
```

Arguments

`x`	A fitted `multiScaleR` object.
`n_pts`	Integer. Number of sigma values to evaluate for each covariate. Default is 10. Ignored when `sigma_values` is supplied.
`metric`	Character. Which metric to profile: `"AICc"` (default) or `"LL"` (log-likelihood).
`verbose`	Logical. Print progress messages. Default is `TRUE`.
`spacing`	Character. How to space automatically generated sigma values: `"log"` (default) or `"linear"`.
`sigma_values`	Optional numeric vector of sigma values to evaluate directly. When supplied, `spacing`, `sigma_range`, and `n_pts` are ignored.
`sigma_range`	Optional numeric vector of length 2 giving the minimum and maximum sigma values for the generated profile grid. Defaults to the optimization range stored in `x`.

Details

For each spatial covariate, sigma is varied across a sequence of candidate values, while all other sigma values are held at their optimized values. By default this is a log-spaced sequence from the minimum to maximum distance considered during optimization. Users can instead request linear spacing with `spacing = "linear"` or supply exact values with `sigma_values`. At each evaluation point the model is refit and the log-likelihood extracted. AICc is computed from the log-likelihood, the number of regression parameters (including sigma), and the number of observations.

Log-spacing concentrates evaluation points at smaller sigma values where the likelihood surface often changes more rapidly, and spaces them out at larger sigma values where the surface tends to be flatter.

Linear spacing can be useful when the profile needs equal resolution across a specific sigma interval. User-supplied `sigma_values` are sorted and duplicate values are removed before profiling to avoid redundant refits.

Value

A list of class `sigma_profile` containing:

profiles A data frame with columns `variable`, `sigma`, `LL`, and `AICc`.

opt_sigma A named numeric vector of the optimized sigma for each covariate.

metric The metric used for profiling.

spacing The profile grid type: `"log"`, `"linear"`, or `"custom"`.

sigma_grid The sigma values evaluated for each covariate.

See Also

`plot.sigma_profile`, `multiScale_optim`

Examples

```
data('pts')
data('count_data')
hab <- terra::rast(system.file('extdata',
                   'hab.tif', package = 'multiScaleR'))

kernel_inputs <- kernel_prep(pts = pts,
                             raster_stack = hab,
                             max_D = 250,
                             kernel = 'gaussian')

mod <- glm(y ~ hab,
           family = poisson,
           data = count_data)

opt <- multiScale_optim(fitted_mod = mod,
                        kernel_inputs = kernel_inputs)

## Profile sigma
prof <- profile_sigma(opt)
plot(prof)

## More evaluation points
prof <- profile_sigma(opt, n_pts = 20)
plot(prof)

## Linearly spaced values between explicit limits
prof <- profile_sigma(opt, n_pts = 8, spacing = "linear",
                      sigma_range = c(25, 250))
plot(prof)

## User-supplied sigma values
prof <- profile_sigma(opt, sigma_values = c(25, 50, 100, 150, 250))
plot(prof)

## Profile log-likelihood instead of AICc
prof <- profile_sigma(opt, metric = "LL")
plot(prof)
```

`pts` *Spatial sample points*

Description

Example point file for optimizing scales of effect

Usage

```
data(pts)
```

Format

An sf class point object:

pts –> spatial location of points

`sim_dat` *Simulate data for optimizing scales of effect*

Description

Function to simulate data with known scales of effect from spatial spatRaster variables

Usage

```
sim_dat(
  alpha = 1,
  beta = NULL,
  kernel = c("gaussian", "exp", "expow", "fixed"),
  type = c("count", "count_nb", "occ", "gaussian"),
  StDev = 0.5,
  n_points = 50,
  min_D = NULL,
  raster_stack = NULL,
  sigma = NULL,
  shape = NULL,
  max_D = NULL,
  user_seed = NULL,
  ...
)
```

Arguments

`alpha`	Intercept term for GLM (Default = 1)
`beta`	Slope term(s) for GLM. Should be vector equal in length to number of spatRaster surfaces provided
`kernel`	Type of kernel transformation. Valid options are 'gaussian', 'exp' (negative exponential), 'expow' (exponential power), and 'fixed' fixed width buffer. (Default = 'gaussian')
`type`	Type of response data to simulate. Valid options are 'count' for Poisson distributed count; 'count_nb' for negative binomial counts; 'occ' for binomial response; and 'gaussian' for normally distributed response. 'count' for normally distributed response (Default = 'count')
`StDev`	If specifying 'count_nb' or 'gaus' for type, this is the dispersion term for those respective processes (Default = 0.5)
`n_points`	Number of spatial sample points (Default = 50). Alternatively, provide a spatVector point file.

`min_D`	Minimum distance between points. Function will attempt to create the number of sample points specified while honoring this minimum distance.
`raster_stack`	A spatRaster object
`sigma`	The scale term dictating the rate of decay with distance
`shape`	If using an exponential power function, the shape parameter must also be specified. Values between 1-50 are generally valid
`max_D`	The maximum distance surrounding spatial points to consider. This typically needs to be >= 2.5x greater than sigma
`user_seed`	Optional seed to reproduce simulation
`...`	Additional arguments. Not currently used

Details

This function distributes sample points across the landscape on a hexagonal grid, then subsamples to the specified number. The weighted values of each landscape are determined according to the simulation parameters, then the specified response is generated.

Value

Returns a list containing:

* obs –> The simulated response variable
* df –> A data frame with the simulated response (obs) as well as the true kernel weighted mean values for each raster surfa
* pts –> An ‘sf‘ object with the simulated spatial point locations

Examples

```
rs <- sim_rast()
rs <- terra::subset(rs, c(1,3))
s_dat <- sim_dat(alpha = 0.5,
                 beta = c(0.75,-0.75),
                 kernel = 'gaussian',
                 sigma = c(75, 150),
                 type = 'count',
                 raster_stack = rs,
                 max_D = 400)

plot(s_dat$df$y ~ s_dat$df$bin1)
plot(s_dat$df$y ~ s_dat$df$cont1)
```

`sim_dat_unmarked` *Simulate data for optimizing scales of effect with ‘unmarked‘*

Description

Function to simulate data with known scales of effect from spatial spatRaster variables for analysis with the R package ‘unmarked‘

Usage

```
sim_dat_unmarked(
  alpha = 1,
  beta = NULL,
  kernel = c("gaussian", "exp", "expow", "fixed"),
  type = c("count", "count_nb", "occ"),
  StDev = 0.5,
  n_points = 50,
  n_surv = 3,
  det = 0.5,
  min_D = NULL,
  raster_stack = NULL,
  sigma = NULL,
  shape = NULL,
  max_D = NULL,
  user_seed = NULL,
  ...
)
```

Arguments

`alpha`	Intercept term for GLM (Default = 1)
`beta`	Slope term(s) for GLM. Should be vector equal in length to number of spatRaster surfaces provided
`kernel`	Type of kernel transformation. Valid options are ’gaussian’, ’exp’ (negative exponential), ’expow’ (exponential power), and ’fixed’ fixed width buffer. (Default = ’gaussian’)
`type`	Type of response data to simulate in ‘unmarked‘. Valid options are ’count’ for Poisson distributed count; ’count_nb’ for negative binomial counts; and ’occ’ for binomial response.(Default = ’count’)
`StDev`	If specifying ’count_nb’ or ’gaus’ for type, this is the dispersion term for those respective processes (Default = 0.5)
`n_points`	Number of spatial sample points (Default = 50).
`n_surv`	Number of surveys to simulate in ‘unmarked‘ (Default = 3).
`det`	The probability of detection. (Default = 0.5)

`min_D`	Minimum distance between points. Function will attempt to create the number of sample points specified while honoring this minimum distance.
`raster_stack`	A spatRaster object
`sigma`	The scale term dictating the rate of decay with distance
`shape`	If using an exponential power function, the shape parameter must also be specified. Values between 1-50 are generally valid
`max_D`	The maximum distance surrounding spatial points to consider. This typically needs to be >= 2.5x greater than sigma
`user_seed`	Optional seed to reproduce simulation
`...`	Additional arguments. Not currently used

Details

This function distributes sample points across the landscape on a hexagonal grid, then subsamples to the specified number. The weighted values of each landscape are determined according to the simulation parameters, then the specified response is generated.

Value

Returns a list containing:

* y –> The simulated observation matrix for use in an unmarkedFrame
* df –> A data frame with the simulated response (obs) as well as the true kernel weighted mean values for each raster surfa
* pts –> An ‘sf‘ object with the simulated spatial point locations

Examples

```
rs <- sim_rast(user_seed = 123)
rs <- terra::subset(rs, c(1,3))
s_dat <- sim_dat_unmarked(alpha = 1,
                          beta = c(0.75,-0.75),
                          kernel = 'gaussian',
                          sigma = c(75, 150),
                          n_points = 75,
                          n_surv = 5,
                          det = 0.5,
                          type = 'count',
                          raster_stack = rs,
                          max_D = 550,
                          user_seed = 123)
plot(s_dat$df$y ~ s_dat$df$bin1)
plot(s_dat$df$y ~ s_dat$df$cont1)
## unmarked analysis
library(unmarked)
kernel_inputs <- kernel_prep(pts = s_dat$pts,
                             raster_stack = rs,
                             max_D = 550,
                             kernel = 'gaus')
```

```
umf <- unmarkedFramePCount(y = s_dat$y,
                           siteCovs = kernel_inputs$kernel_dat)

## Base unmarked model
mod0 <- pcount(~1 ~bin1 + cont1,
               data = umf,
               K = 100)

## `multiscale_optim`
opt1 <- multiScale_optim(fitted_mod = mod0,
                         kernel_inputs = kernel_inputs)

summary(opt1)
```

`sim_rast` *Function to simulate raster surfaces*

Description

Function to create four spatRaster surfaces

Usage

```
sim_rast(
  dim = 100,
  resolution = 10,
  autocorr_range1 = NULL,
  autocorr_range2 = NULL,
  sill = 10,
  plot = FALSE,
  user_seed = NULL,
  ...
)
```

Arguments

`dim`	Dimension (number of cells) on a side a square raster (Default = 100)
`resolution`	Resolution of raster cells (Default = 10)
`autocorr_range1`	Optional, Numeric. Spatial correlation range in map cells. Controls the decay of the exponential covariance. If NULL (default), autocorrelation range will be 5% of specified dimension.
`autocorr_range2`	Optional, Numeric. Spatial correlation range in map cells. Controls the decay of the exponential covariance. If NULL (default), autocorrelation range will be 25% of specified dimension.

`sill`	Numeric. Variance (partial sill) of the random field (default = 10).
`plot`	Logical. If TRUE, the spatRaster stack will be plotted following the simulation
`user_seed`	Optional seed to replicate simulated surfaces
`...`	Additional arguments. Not currently used

Details

This is a simple wrapper to create four different raster surfaces. Surfaces differ in the range of autocorrelation. Binary surfaces are created by thresholding continuous values of the Gaussian random surface.

Value

Four spatRaster surfaces. Two 1/0 binary surfaces and two continuous surfaces.

Examples

```
sim1 <- sim_rast()

sim2 <- sim_rast(dim = 150,
                 resolution = 25)
```

`summary.multiScaleR` *Summarize multiScaleR objects*

Description

Summarizes output from `multiScale_optim`.

Usage

```
## S3 method for class 'multiScaleR'
summary(object, profile = FALSE, ...)
```

Arguments

`object`	An object of class `multiScaleR`.
`profile`	Logical. If `TRUE`, use profile-likelihood confidence limits for ‘sigma‘ when feasible. Defaults to `FALSE` so summaries remain fast; profile results are cached for repeated calls on the same fitted object during the current R session.
`...`	Optional arguments passed to the method (e.g., `prob` for cumulative kernel weight threshold).

Value

An object of class `summary_multiScaleR`. Confidence limits for ‘sigma‘ default to the package’s existing Wald-style limits. If `profile = TRUE`, profile likelihood is used when feasible; if profiling fails, the summary falls back to Wald-style limits.

`surv_pts` *Spatial sample points*

Description

Example point file for use with vignette document example

Usage

```
data(surv_pts)
```

Format

An sf class point object:

pts –> 100 spatial point locations

Index