Package: multiScaleR (via r-universe)

October 20, 2026

Type Package

Title Methods for Optimizing Scales of Effect

Version 0.7.4

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

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, landscapemetrics, geodiv, 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] (<https://orcid.org/0000-0001-5229-9268>)

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

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev make libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository https://wpeterman.r-universe.dev

Date/Publication 2026-10-19 21:55:07 UTC

RemoteUrl https://github.com/wpeterman/multiscaler

RemoteRef HEAD

RemoteSha 87bf8779fe571921f2fd5d3e45c0f9e478530022

Contents

multiScaleR-package . . . 3
aic_tab . . . 3
bic_tab . . . 5
count_data . . . 7
diagnostics . . . 8
estimate_multiscale_ram . . . 9
hab . . . 12
kernel_dist . . . 12
kernel_prep . . . 15
kernel_prep_by_group . . . 18
kernel_scale.raster . . . 21
landscape . . . 23
landscape_counts . . . 24
msr_vars . . . 25
multiScale_optim . . . 30
plot.multiScaleR . . . 36
plot.sigma_profile . . . 38
plot_kernel . . . 39
plot_marginal_effects . . . 41
print.multiScaleR . . . 43
print.multiScaleR_data . . . 44
print.summary_multiScaleR . . . 44
profile_sigma . . . 45
pts . . . 47
sim_dat . . . 48
sim_dat_unmarked . . . 50
sim_rast . . . 52
summary.multiScaleR . . . 54
surv_pts . . . 55

Index **56**

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)

Authors:

- 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

Creates a model selection table ranked by AIC or AICc for a list of fitted models. Mixed lists containing both `multiScaleR` objects and plain model objects (e.g., `glm`) are supported, allowing comparison of scale-optimized models against fixed-scale alternatives.

Usage

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

Arguments

mod_list	List containing fitted multiScaleR objects (from multiScale_optim) and/or plain fitted model objects (e.g., glm, lm). All models must have been fit to the same set of observations; an error is raised if sample sizes differ.
AICc	Logical. If TRUE (default), the second-order corrected AIC (AICc) is used. Set to FALSE for standard AIC. AICc is recommended when the ratio of observations to parameters is small (n / K < 40 is a common rule of thumb).
mod_names	Optional character vector of model names. Length must equal length(mod_list). By default, names are constructed as [kernel]right-hand-side-formula for multiScaleR objects.
verbose	Logical. If TRUE, the table is also printed to the console. Default: FALSE.
...	Additional arguments (not used).

Details

The table is constructed using aictabCustom from the AICcmodavg package. For multiScaleR objects, the optimized (refitted) model stored in opt_mod is used for log-likelihood and parameter count. The sigma parameter (and shape for "expow") is added to K automatically. For plain model objects in a mixed list, sigma is not added to K. The fitted model's log-likelihood degrees of freedom are used when available, so negative-binomial dispersion contributes one parameter. For unmarkedFit models, native AIC and log-likelihood metadata determine the count, including detection coefficients, factor contrasts, mixture parameters, and estimated random-effect variances. Optimizer and TMB free-parameter metadata are checked for agreement when present. Latent random effects and the integration limit K are not counted. Missing or inconsistent parameter metadata and non-finite log-likelihoods raise an error instead of guessing from predictor names. AICc tables raise an error if any model has n <= K + 1; use AICc = FALSE for an AIC table in that case.

Value

A data frame of class "aictab" (from the AICcmodavg package) sorted by ascending AIC(c), containing:

Modnames Model name (from mod_names or auto-generated).

K Number of fitted likelihood parameters, including ancillary parameters such as negative-binomial dispersion, plus optimized scale and shape parameters for multiScaleR models.

AICc **or** AIC Information criterion value.

Delta_AICc **or** Delta_AIC Difference from the top model.

ModelLik Relative likelihood (exp(-0.5 * Delta)).

AICcWt **or** AICWt Akaike weight.

LL Log-likelihood.

Cum.Wt Cumulative Akaike weight.

Author(s)

Bill Peterman

Examples

```
## Bundled counts simulated with a habitat scale of 75 map units
data("pts")
data("count_data")
hab <- terra::rast(system.file("extdata", "hab.tif", package = "multiScaleR"))
prepared <- kernel_prep(pts, hab, max_D = 300, sigma = 75, verbose = FALSE)
dat <- data.frame(y = count_data$y, prepared$kernel_dat)
fixed_model <- glm(y ~ hab, family = poisson(), data = dat)
null_model <- glm(y ~ 1, family = poisson(), data = dat)
spatial_model <- multiScale_optim(fixed_model, prepared, par = 75 / 300,
                                  n_cores = 1, verbose = FALSE)
models <- list(spatial = spatial_model, fixed = fixed_model, null = null_model)

## The spatial model pays for one extra estimated sigma parameter.
aic_tab(models, AICc = FALSE, mod_names = names(models))
aic_tab(models, AICc = TRUE, mod_names = names(models))
```

`bic_tab` *multiScaleR model selection*

Description

Creates a model selection table ranked by BIC for a list of fitted models. Mixed lists containing both `multiScaleR` objects and plain model objects are supported.

Usage

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

Arguments

`mod_list`	List containing fitted `multiScaleR` objects (from `multiScale_optim`) and/or plain fitted model objects. All models must have been fit to the same number of observations.
`mod_names`	Optional character vector of model names. Length must equal `length(mod_list)`. By default, names are constructed as `[kernel]right-hand-side-formula` for `multiScaleR` objects.
`verbose`	Logical. If `TRUE`, the table is also printed to the console. Default: `FALSE`.
`...`	Additional arguments (not used).

Details

The table is constructed using `bictabCustom` from the `AICcmodavg` package. BIC penalizes model complexity more heavily than AIC as sample size grows, and is often preferred when the goal is identifying the single best-supported model rather than model averaging. Sigma (and shape for `"expow"`) is counted in K. Fitted parameter counts and log-likelihood validation follow `aic_tab`, including all estimated components of unmarked models.

Value

A data frame of class `"bictab"` (from the `AICcmodavg` package) sorted by ascending BIC, containing:

`Modnames` Model name.

`K` Number of fitted likelihood parameters, including ancillary parameters such as negative-binomial dispersion, plus optimized scale and shape parameters for `multiScaleR` models.

`BIC` Bayesian information criterion value.

`Delta_BIC` Difference from the top model.

`BICWt` BIC weight.

`Cum.Wt` Cumulative BIC weight.

`LL` Log-likelihood.

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. Diagnostics are populated automatically during multiScale_optim and flag potential issues with the optimization result.

Usage

```
diagnostics(object, ...)

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

Arguments

object	An object to inspect. Must be of class multiScaleR.
...	Additional arguments passed to methods.

Details

These diagnostics are evaluated automatically at the end of multiScale_optim. Console warnings are printed when any diagnostic is triggered. diagnostics() provides programmatic access to the same information without re-running the model. These stored diagnostics retain the historical 1D 90% distance and threshold, independently of the new 2D reporting default in plots and summaries. Changing dimension when reporting a fit does not recompute its stored diagnostics. Use kernel_dist(object) to inspect the ideal planar radius separately.

When max_distance is triggered, consider re-running kernel_prep with a larger max_D value. The suggested minimum is stored in suggested_max_D.

When sigma_precision is triggered, the variable may not have a meaningful scale of effect, or the data may be insufficient to estimate it precisely. Interpret affected covariates with caution.

Value

A named list with four elements:

max_distance A list describing whether the estimated scale of effect approaches or exceeds max_D. Fields include: triggered (logical), variables (names of affected covariates), effective_distance (estimated 90% one-dimensional kernel distance per covariate; see kernel_dist), max_D (the limit used), ratio (max_D / effective_distance), and suggested_max_D (a recommended minimum max_D value). Triggered when max_D / effective_distance < 2.

sigma_precision A list describing whether sigma estimates are imprecise. Fields include: triggered (logical), variables (names of affected covariates), estimate (sigma means), se (standard errors), and se_to_mean (ratio of SE to mean). Triggered when SE / Mean >= 0.5 for any covariate. NULL when no precision concern was flagged.

shape_precision Identical structure to sigma_precision but for the shape parameter of the exponential power kernel. NULL unless kernel = "expow" was used and a precision concern arose.

sample_size A list with fitted_n, the number of observations in the final model, and prepared_n, the number of observations in the preparation object, or model rows for repeated-site unmarked data. For unmarked fits, prepared_sites is the number of distinct spatial sites in the preparation object. triggered is TRUE when the fitted model uses fewer observations. Candidate scales that change the initial model's sample are rejected during optimization.

See Also

multiScale_optim, kernel_prep

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)

## Access diagnostics
diag <- diagnostics(opt)
diag$max_distance$triggered
diag$sigma_precision
```

---

estimate_multiscale_ram

Estimate Memory Requirements for Multiscale Optimization

Description

Estimate the memory footprint of a kernel_prep object and provide a conservative recommendation for parallel worker counts. The helper inspects the stored kernel inputs, optionally builds the same optimization context used by multiScale_optim, detects the number of physical CPU cores, and estimates a maximum n_cores value while leaving a user-defined number of cores free.

Usage

```
estimate_multiscale_ram(
  kernel_inputs,
  fitted_mod = NULL,
  join_by = NULL,
  refit_fn = NULL,
  n_cores = NULL,
  PSOCK = (.Platform$OS.type != "unix"),
  safety_factor = 1.5,
  ram_fraction = 0.75,
  leave_free = 2L
)
```

Arguments

kernel_inputs	A "multiScaleR_data" object created by kernel_prep.
fitted_mod	Optional fitted model object. When supplied, the helper also measures the optimization context that would be serialized to parallel workers during multiScale_optim.
join_by	Optional data frame passed through to multiScale_optim for unmarked workflows.
refit_fn	Optional custom refit function passed through to multiScale_optim.
n_cores	Optional positive integer. When supplied, the helper also reports the estimated peak memory for that specific worker count. When omitted, the reported peak estimate uses the recommended maximum worker count.
PSOCK	Logical. Should parallel runs be budgeted as PSOCK workers. Default: TRUE on non-Unix platforms and FALSE on Unix-like platforms. Even on systems that support forking, the recommendation uses a conservative per-worker duplication assumption.
safety_factor	Numeric scalar ≥ 1. Multiplier applied to the per-worker payload to allow for temporary allocations during optimization. Default: 1.5.
ram_fraction	Numeric scalar in (0, 1]. Fraction of detected total system RAM considered usable for this run. The remainder is reserved as headroom for the OS and other R objects. Default: 0.75.
leave_free	Integer scalar ≥ 0. Number of physical CPU cores to leave unused when computing the recommended maximum worker count. Default: 2.

Details

The estimate is advisory rather than exact. It is based on utils::object.size() and therefore reflects the size of stored R objects, not all transient allocations made by exactextractr, model refits, or BLAS/LAPACK.

The recommendation is intentionally conservative. Even on Unix-like systems where forking can share memory copy-on-write, worker counts are budgeted as though each worker may require its own copy of the optimization payload.

If fitted_mod is omitted, the RAM recommendation is based on kernel_inputs alone and may understate the true optimization payload.

multiScale_optim() stops before starting parallel workers when n_cores > 1 exceeds the RAM-based limit under the default settings of this helper. Use n_cores = 1 for serial execution, or rebuild kernel_prep() inputs with bin = TRUE and store_cell_data = FALSE when all covariates are kernel-weighted. Landscape and surface metrics require cell-level data and cannot use this lean mode. Recheck the estimate after rebuilding inputs.

Worker recommendations have a minimum of one; this does not guarantee that serial execution fits in memory. The budget check cannot enforce a limit when RAM detection or estimation fails, and a run below the estimated limit can still exhaust memory during a model refit.

Value

A list of class "multiScaleR_ram" with:

requested_n_cores Worker count used for the peak_parallel_estimate.

recommended_n_cores Conservative recommended maximum worker count after applying both CPU and RAM limits.

max_cores_by_cpu Maximum worker count allowed by detected physical CPU cores after leaving leave_free cores unused.

max_cores_by_ram Maximum worker count allowed by the RAM budget, or NA if total system RAM could not be detected.

backend Character string identifying the assumed parallel backend budget: "PSOCK" or "fork".

system Named list describing detected physical cores, logical cores, total RAM, usable RAM, and the reservation settings.

point_summary One-row data frame summarizing the number of points, extracted cells per point, source raster layer count, whether cell-level data were retained (cell_data_stored), and the number of distance bins (nbins).

component_bytes Data frame reporting estimated byte sizes for the major stored objects and worker bundles.

notes Character vector with interpretation notes and any detection warnings.

Examples

```
library(terra)

pts <- vect(cbind(c(3, 5, 7),
                  c(7, 5, 3)))
r <- rast(matrix(rnorm(100), nrow = 10))
names(r) <- "hab"

kernel_inputs <- kernel_prep(
  pts = pts,
  raster_stack = r,
  max_D = 2,
  kernel = "gaussian",
  verbose = FALSE
)

estimate_multiscale_ram(kernel_inputs)
```

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

Converts a kernel scale parameter (`sigma`) into a distance in map units: the distance within which a specified proportion of the kernel weight falls. Distances can be measured along a line through the point (one-dimensional) or as the radius of a circle around it (two-dimensional). Can be used with a fitted `multiScaleR` object to report optimized scale distances, or with manually supplied kernel parameters to explore kernel behavior.

Usage

```
kernel_dist(model, prob = 0.9, dimension = c("2d", "1d"), ...)
```

Arguments

`model`	A fitted `multiScaleR` object from `multiScale_optim`. When provided, scale distances and 95% confidence intervals are returned for all optimized covariates. When omitted, `sigma`, `kernel`, and (for `expow`) `beta` must be supplied via `...`.
`prob`	Numeric between 0 and 1 (exclusive). Cumulative proportion used to define the distance. Default: `0.9`. The proportion refers to weight along a line for `dimension = "1d"` and weight on a plane for `dimension = "2d"`.

`dimension` Character. How the distance is measured. `"2d"` (default) returns the radius of the circle around the point that holds `prob` of the total weight the ideal kernel gives to locations on an unbounded plane. Because raster cells lie across a plane, the `"2d"` radius answers "within what distance does the ideal kernel assign 90% of its total weight?". For distance-weighted kernels it is larger than the `"1d"` distance; hard-radius covariates report the same radius for both. `"1d"` restores the original line-based distance, enclosing `prob` of the area under the kernel curve along a transect through the point. See Details.

`...` Additional parameters used when `model` is not supplied:

`sigma` Numeric (positive). The kernel scale parameter in the same units as the projection of `pts` and `raster_stack` passed to `kernel_prep`. For Gaussian kernels this is the standard deviation; for negative exponential kernels this is the decay length.

`kernel` Character. The kernel function to use. One of `"gaussian"`, `"exp"` (negative exponential), `"fixed"` (fixed-radius buffer), or `"expow"` (exponential power). Required when `model` is not provided.

`beta` Numeric (positive). Shape parameter for the exponential power kernel. Required when `kernel = "expow"` and `model` is not provided. Ignored for all other kernels.

Details

Default change. Version 0.7.4 and later use `dimension = "2d"` for distances, summaries, and kernel plots. Earlier versions used `"1d"`. Set `dimension = "1d"` explicitly to reproduce earlier reported distances. This changes reporting, not fitted sigma values, kernel weights, or likelihoods. Existing simulation extents, projection cutoffs, and optimization diagnostics retain their explicit 1D definitions.

One or two dimensions. A kernel assigns each raster cell a weight that declines with the cell's distance from the point. Two distances can summarize how far that weight reaches:

- `dimension = "1d"` treats the kernel as a curve along a single line (a transect) through the point and returns the distance on either side that encloses `prob` of the area under that curve. For a Gaussian kernel the 90% distance is about 1.65 sigma.
- `dimension = "2d"` accounts for locations lying on a plane, so the number of locations at distance d grows with the circumference 2 pi d. It returns the radius of the circle that holds `prob` of the ideal kernel weight. For a Gaussian kernel the 90% radius is about 2.15 sigma.

With sigma = 300 m, a Gaussian kernel gives a 90% distance of about 493 m in one dimension and 644 m in two. On an unbounded plane, locations within 493 m carry only about 74% of the total weight. When you describe the area associated with kernel weighting, or compare with values defined on the plane (for example, a radius holding 90% of the kernel weight), use `"2d"`. Always state which definition you report.

Kernel formulas. In one dimension:

- **Gaussian**: the inverse normal CDF, `qnorm((1 + prob) / 2) * sigma`.
- **Negative exponential**: `-sigma * log(1 - prob)`.
- **Fixed buffer**: `sigma * prob` (the fraction of the buffer radius).

- **Exponential power**: `sigma * qgamma(prob, shape = 1 / beta)^(1 / beta)`; both `sigma` and `beta` (shape) must be specified.

In two dimensions:

- **Gaussian**: `sigma * sqrt(-2 * log(1 - prob))`.
- **Negative exponential**: `sigma * qgamma(prob, shape = 2)`, about 3.89 sigma at 90%.
- **Fixed buffer**: `sigma * sqrt(prob)`, the radius of the circle holding `prob` of the buffer's area.
- **Exponential power**: `sigma * qgamma(prob, shape = 2 / beta)^(1 / beta)`.

Both definitions describe the ideal continuous kernel. They do not account for truncation at `max_D`, raster boundaries, missing cells, or cell geometry. The realized share of weight among extracted raster cells can therefore differ. If the two-dimensional radius approaches or exceeds `max_D`, consider increasing `max_D` in `kernel_prep` and refitting the model.

Confidence intervals for the fitted `model` case are derived from the Hessian-based standard errors (or profile-likelihood intervals when `profile_sigma` has been run and stored on the object). Each limit of sigma is converted to a distance with the same formula. For exponential-power kernels, beta is held at its fitted estimate during this conversion; the limits do not include uncertainty in beta or in site-specific retained weight.

`summary()` of a fitted model reports both distances; see `summary.multiScaleR`.

Value

When `model` is provided: a data frame with one row per optimized covariate and three columns, all in map units: `Mean` (distance at the estimated sigma), `2.5%` (distance at the lower 95% limit of sigma), and `97.5%` (distance at the upper 95% limit of sigma). Values are rounded to two decimal places. For hard-radius covariates (landscape metrics and unweighted surface metrics) the row reports the radius itself, whichever `dimension` is requested.

When kernel parameters are supplied directly via `...`: a single numeric value giving the distance, in map units, at which the cumulative kernel weight under the requested dimensional definition reaches `prob`.

See Also

`summary.multiScaleR`, `plot.multiScaleR`, `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)

## Uses of `kernel_dist`
kernel_dist(model = opt)                     # radius of a circle (2D)
kernel_dist(model = opt, dimension = "1d")   # original distance along a line
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")


## The two definitions for a Gaussian kernel with sigma = 300
kernel_dist(sigma = 300, kernel = "gaussian")                    # about 644
kernel_dist(sigma = 300, kernel = "gaussian", dimension = "1d")   # about 493
```

`kernel_prep` *Kernel Scale Preparation*

Description

Prepares the data inputs required for multiscale kernel optimization. Extracts raster values within a buffer around each point, computes pairwise distances, and calculates initial kernel-weighted covariate values. The result is passed directly to `multiScale_optim`.

Usage

```
kernel_prep(
  pts,
  raster_stack,
  max_D,
  kernel = c("gaussian", "exp", "expow", "fixed"),
  scale_vars = NULL,
  sigma = NULL,
  shape = NULL,
  projected = TRUE,
  progress = FALSE,
  verbose = TRUE,
  bin = TRUE,
  nbins = 256L,
  store_cell_data = TRUE
)
```

Arguments

pts	Spatial point locations as a SpatVector (terra) or sf object. Points must be in a projected coordinate system that shares units with max_D.
raster_stack	One or more raster layers of class SpatRaster (terra). Layer names must match the variable names in the model formula used with multiScale_optim, unless scale_vars is provided to define derived covariates. Layer names cannot be "x", which is reserved for internal coordinate data.
max_D	Positive numeric. The maximum radius (in the same units as the projection of pts and raster_stack) around each point to sample from the raster. Should be at least 2–3 times the largest expected sigma value. Raster layers must extend far enough beyond the points to fully cover buffers of this radius.
kernel	Kernel function to be used for weighting raster values by distance. One of: "gaussian" (Default) Gaussian (normal) decay: weights fall off as a normal distribution with standard deviation sigma. "exp" Negative exponential decay: weights fall off as exp(-d / sigma). "fixed" Fixed-radius (step) buffer: all cells within sigma receive equal weight; cells beyond receive zero. "expow" Exponential power kernel parameterized by both sigma (scale) and shape (shape). Requires shape to be specified.
scale_vars	Optional variable specifications created with msr_vars. When omitted, each raster layer becomes one kernel-weighted covariate with the same name, preserving historical behavior. Provide scale_vars to define derived model covariates, for example, combining a kernel-weighted mean with landscape composition or edge metrics from the same source layer. See kernel_var and landscape_var.
sigma	Optional numeric vector of initial sigma values for optimization. Length must equal the number of covariates with optimize = TRUE in scale_vars (or the number of raster layers when scale_vars is not provided). Values must be in the same units as the projection. Default: NULL; initial values are generated automatically as max_D / 2, which is recommended.
shape	Optional numeric vector of initial shape values when kernel = "expow". Length requirements same as sigma. Default: NULL; starting values of 2 are generated automatically.
projected	Logical. Whether pts and raster_stack are in a projected (planar) coordinate system. Currently only projected coordinates are supported. Default: TRUE.
progress	Logical. Print progress bars to the console during extraction and distance calculations. Default: FALSE.
verbose	Logical. Print status messages during preparation. Default: TRUE.
bin	Logical. If TRUE (default), precompute distance-binned summaries of the kernel-type covariates so that multiScale_optim can evaluate kernel-weighted means in O(nbins) per point instead of iterating every buffer cell on each optimizer evaluation. This makes optimization substantially faster (and independent of max_D) at the cost of a negligible binning approximation. Landscape-metric covariates are never binned and always use the exact per-cell path.

nbins Positive integer. Number of equal-width distance bins used when bin = TRUE. More bins increase accuracy and memory; the default (256) reproduces exact optimization results to several significant figures. Ignored when bin = FALSE.

store_cell_data
Logical. If TRUE (default), the full per-cell d_list/raw_cov data are retained. If FALSE and every covariate is kernel-type (no landscape metrics) and bin = TRUE, the cell-level data are dropped after binning, dramatically reducing the object's memory footprint (often by 1–2 orders of magnitude). The binned summaries are sufficient to drive optimization, profile_sigma, and the final refit. Ignored (with a message) when landscape metrics are present or bin = FALSE.

Details

Point locations and raster layers must share a defined CRS and both must be projected. All units (including max_D and any user-supplied sigma) must be in the same linear unit as the projection (typically meters or feet).

The max_D buffer should be large enough to encompass the plausible range of the true scale of effect. A common rule of thumb is to set max_D to at least 2–3 times the largest expected sigma. After running multiScale_optim, the max_distance diagnostic will warn if the estimated scale approaches max_D.

Initial sigma values do not need to be precise. The optimizer will refine them. Provide explicit starting values only if the default (max_D / 2) leads to convergence problems.

Row names from pts are preserved throughout the returned object so that downstream model data frames can be joined back to the original point order. Raster extraction is processed in batches of 25 points to limit the peak memory used by temporary extraction tables. Point order, returned values, and distance calculations are unchanged. To assign each observation to a different annual raster while estimating one pooled landscape relationship, use kernel_prep_by_group.

Value

A list of class "multiScaleR_data" containing:

kernel_dat Data frame of scaled (mean-centered, unit-variance) initial kernel-weighted covariate values, one row per point and one column per covariate. Row names match pts row names or sequential integers. Use this directly to fit the initial model passed to multiScale_optim.

d_list Named list (one element per point) of numeric distance vectors from the point to every raster cell within the buffer.

raw_cov Named list (one element per point) of matrices containing raw raster cell values within the buffer, aligned with d_list. Each point is stored as a dense or sparse matrix, whichever is smaller. NULL in lean mode.

kernel Character string identifying the kernel used.

sigma Numeric vector of initial sigma values on the internal (scaled) parameter space.

shape Numeric vector of initial shape values, or NULL.

min_D Numeric. Approximate minimum inter-cell distance, used as the lower bound for sigma during optimization.

max_D Numeric. The max_D value supplied by the user.

`n_covs` Integer. Number of covariates with scale being optimized.

`unit_conv` Numeric. Internal distance scaling factor (equals `max_D`).

`scale_vars` Data frame of class `"multiScaleR_vars"` describing all covariate specifications.

`resolution` Numeric. Raster cell resolution.

`n_cols` Integer. Number of raster columns (used for adjacency landscape metrics).

`scl_params` Named list with elements `mean` and `sd` the centering and scaling parameters applied to `kernel_dat`. Stored for use by `kernel_scale.raster` when `scale_center = TRUE`.

`binned` When `bin = TRUE`, a list of precomputed distance-binned summaries (representative bin distances and per-covariate value/count matrices) used to accelerate optimization. `NULL` when `bin = FALSE`.

`cell_data_stored` Logical flag indicating whether the per-cell `d_list`/`raw_cov` data were retained. `FALSE` only in lean mode (`store_cell_data = FALSE`).

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_prep_by_group` *Prepare Kernel Inputs from Group-Specific Rasters*

Description

Assign each observation to one raster map, then prepare one pooled data object for `multiScale_optim`. Use this when land cover changes across years but the model estimates shared landscape effects and scales.

Usage

```
kernel_prep_by_group(
  pts,
  raster_stacks,
  group,
  max_D,
```

```
  kernel = c("gaussian", "exp", "expow", "fixed"),
  scale_vars = NULL,
  sigma = NULL,
  shape = NULL,
  bin = TRUE,
  nbins = 256L,
  store_cell_data = TRUE,
  progress = FALSE,
  verbose = TRUE
)
```

Arguments

pts	Projected point locations as an sf or SpatVector object. Point order matches group and the returned covariate rows.
raster_stacks	Named list of SpatRaster maps. Each map has the same extent, grid, CRS, layer names, and class coding. Its name identifies a year or other observation group.
group	Character or factor vector, one value per point, identifying the corresponding name in raster_stacks. Every point uses only its assigned map, including cells affected by changes outside the point's immediate location.
max_D	Maximum extraction radius in the projected map units.
kernel	Kernel weighting function; see kernel_prep.
scale_vars	Optional msr_vars specification. By default, each raster layer becomes a kernel-weighted mean covariate.
sigma	Optional starting scales in map units. Defaults to max_D / 2 for each optimized covariate.
shape	Optional starting shapes for the exponential power kernel. Defaults to 2 for each optimized covariate.
bin	Logical. Build pooled distance bins for kernel covariates. Default: TRUE. All observations use one set of bin definitions.
nbins	Number of distance bins when bin = TRUE; default 256.
store_cell_data	Logical. Retain per-cell values and distances. Default: TRUE. Set FALSE for a lean kernel-only binned object; metrics requiring cell detail retain it regardless.
progress	Logical. Display extraction progress. Default FALSE.
verbose	Logical. Print preparation messages. Default TRUE.

Details

Raster maps must be aligned because landscape configuration metrics use cell adjacency and the resulting model assumes common map units. Group-specific maps may contain different missing cells. For kernel covariates, pooled bins track both finite-cell value sums and counts. The fitted model has common slopes and scales unless its formula explicitly includes group interactions. Choose the response likelihood and any group effect according to the sampling design. A pooled model does not establish that effects are stable across years.

Value

A `multiScaleR_data` list containing:

`kernel_dat` Initial covariate data frame, with one row per point. Columns are centered and divided by their pooled standard deviations. Use this in the starting model.

`d_list`, `raw_cov` Named per-point distance vectors (divided by `unit_conv`) and raster-value matrices. Both are `NULL` in lean mode.

`binned` Pooled distance-bin summaries for kernel covariates, or `NULL` when `bin = FALSE`. Each covariate has finite-value sums and counts by point and distance bin.

`raster_group` Named character vector recording the map assigned to each point, in input order.

`scl_params` Pooled means and standard deviations used to create `kernel_dat`; the fitted model replaces these with scaling parameters at the optimized scales.

`kernel`, `sigma`, `shape` Kernel name, initial scales divided by `unit_conv`, and optional initial shape values.

`min_D`, `max_D`, `unit_conv` Lower scale bound, maximum extraction radius, and internal distance scaling factor. These are in map units except the internally scaled `sigma`.

`n_covs`, `scale_vars` Number of optimized covariates and the validated covariate definitions.

`resolution`, `n_cols` Raster cell size and column count, used to calculate landscape configuration metrics.

`cell_data_stored` Whether per-cell values and distances are retained.

Pass the full object to `multiScale_optim`.

See Also

`kernel_prep`, `multiScale_optim`

Examples

```
library(terra)
r1 <- rast(nrows = 20, ncols = 20, xmin = 0, xmax = 200,
           ymin = 0, ymax = 200, crs = "EPSG:26915")
values(r1) <- seq_len(ncell(r1)) / ncell(r1)
names(r1) <- "habitat"
r2 <- r1
r2[seq_len(30)] <- 0
pts <- vect(cbind(c(50, 70, 90, 110, 130, 150),
                  c(60, 90, 120, 140, 100, 70)),
            type = "points", crs = crs(r1))
prep <- kernel_prep_by_group(pts, list(y1 = r1, y2 = r2),
                             group = rep(c("y1", "y2"), 3),
                             max_D = 25, verbose = FALSE)
head(prep$kernel_dat)
```

kernel_scale.raster *Apply kernel smoothing to raster layers*

Description

Applies a kernel smoothing function to one or more raster layers, producing a spatially weighted mean (or landscape metric) at the scale identified by multiScale_optim. Primarily used to generate prediction rasters for spatial model projection.

Usage

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

Arguments

raster_stack	A SpatRaster object containing the source raster layer(s) to be smoothed. Layer names must match the covariate names used during model fitting (or the source names in scale_vars).
sigma	Numeric vector of kernel scale parameter(s) in the same units as the raster projection (e.g., meters). One value per raster layer. Ignored when multiScaleR is provided; sigma values are extracted from the fitted object automatically.
multiScaleR	A fitted object of class "multiScaleR" (from multiScale_optim) or class "multiScaleR_data" (from kernel_prep). When provided, sigma, shape, kernel, and scale_vars are all extracted automatically. Default: NULL.
shape	Numeric vector of shape parameters for the exponential power kernel (kernel = "expow"), one per raster layer. Ignored when multiScaleR is provided. Default: NULL.
kernel	Character. Kernel function used for smoothing. One of "gaussian" (default), "exp", "fixed", or "expow". Ignored when multiScaleR is provided.

scale_vars	Optional variable specifications created with msr_vars. Use when projecting landscape metrics or explicitly defined covariates without passing a fitted multiScaleR object. When multiScaleR is provided, scale_vars is extracted automatically.
pct_wt	Numeric between 0 and 1 (exclusive). Cumulative kernel weight along a line through the focal cell used to determine the smoothing window radius. A larger value (e.g., 0.99) includes more of the kernel tail but increases computation time. This is the historical one-dimensional cutoff, not the proportion of total weight within the planar window. Default: 0.975.
fft	Logical. If TRUE (default), smoothing is performed via Fast Fourier Transform (FFT) convolution, which is substantially faster for large rasters and wide kernels. Some edge effects may occur at raster boundaries. Set to FALSE to use terra::focal, which avoids edge effects but is slower.
scale_center	Logical. If TRUE, the smoothed raster values are centered and scaled using the mean and standard deviation from the model fitting data (extracted from multiScaleR$scl_params). Required when using the output with terra::predict and a fitted model that was trained on scaled covariates. Requires multiScaleR to be provided. Default: FALSE.
clamp	Logical. If TRUE, scaled raster values are clamped to the observed covariate range from the model fitting data, preventing extrapolation beyond the training range. Only active when scale_center = TRUE. Default: FALSE.
pct_mx	Numeric between -0.99 and 0.99. When clamp = TRUE, expands (> 0) or contracts (< 0) the clamping range relative to the observed min/max by this proportion. Default: 0 (exact training range).
na.rm	Logical. If TRUE (default), NA cells are excluded from the kernel-weighted mean calculation. If FALSE, any window containing a NA cell will produce a NA output.
verbose	Logical. Print layer-level progress messages. Default: TRUE.
offset_values	Named numeric vector of values for variables inside model offsets, such as c(n_weeks = 4) for offset(log(n_weeks)). These are raw input values, not transformed offsets. When omitted, a log offset uses 1 (zero on the log scale) and a direct offset uses 0. Set this argument for a meaningful projection scenario.
...	Reserved for deprecated arguments. Currently only scale_opt (deprecated alias for multiScaleR) is handled.

Details

Typical usage after running multiScale_optim:

```
opt_hab <- kernel_scale.raster(raster_stack = hab, multiScaleR = opt)
plot(c(hab, opt_hab))

## Scale and center for prediction
opt_hab_sc <- kernel_scale.raster(raster_stack = hab,
                                  multiScaleR = opt,
                                  scale_center = TRUE)
preds <- terra::predict(opt_hab_sc, opt$opt_mod, type = "response")
```

FFT vs. focal smoothing

The FFT convolution (`fft = TRUE`) is substantially faster for large rasters or wide kernels. It produces minor edge artifacts near raster boundaries, typically within one kernel-width of the edge. For analyses focused on interior areas this is usually negligible. Use `fft = FALSE` for exact focal smoothing when edge accuracy matters.

Dummy rasters for site-level covariates

When a fitted `multiScaleR` object is supplied, the function inspects the model frame to find any predictors that are not represented by raster layers (e.g., survey effort, habitat type measured in the field). These cannot be projected spatially and are therefore assigned constant zero rasters, which correspond to the reference value for centered and scaled covariates. These dummy layers are required for `terra::predict` to work but do not represent real spatial variation. Replace them manually for non-zero projection scenarios. Categorical predictors are skipped with a warning. Offset terms use their underlying input variables as layer names. For `offset(log(n_weeks))`, the default layer is `n_weeks = 1`, representing one week of effort. Set `offset_values = c(n_weeks = 4)` to predict for four weeks. Direct offsets default to zero.

Value

A `SpatRaster` with one layer per covariate defined in `scale_vars` (or per raster layer when `scale_vars` is not used). Layer names match the covariate names from the model. When a fitted `multiScaleR` object is provided and the model contains site-level predictors (i.e., predictors not derived from raster layers), constant ("dummy") raster layers filled with zeros are appended to make the output compatible with `terra::predict`. Offset variables are appended under their input names so transformed offset expressions can be evaluated by `terra::predict`.

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

msr_vars *Define multiScaleR covariate transformations*

Description

These helpers define named model covariates as transformations of one or more source raster layers. They are optional; if omitted, kernel_prep preserves the default behavior where each raster layer becomes one optimized kernel-weighted covariate with the same name.

Use msr_vars() to collect one or more kernel_var(), landscape_var(), or surface_var() specifications into a single object that is passed to the scale_vars argument of kernel_prep and kernel_scale.raster.

Usage

```
msr_vars(...)

kernel_var(source)

landscape_var(
  source,
  metric,
  radius = NULL,
  base = exp(1),
  classes_max = NULL,
  class = NULL
)

surface_var(source, metric, radius = NULL, weighted = FALSE)
```

Arguments

...	Named kernel_var(), landscape_var(), or surface_var() specifications. Each argument must be named; the name becomes the covariate column name in the model data frame. All names must be unique and cannot be "x", which is reserved for internal coordinate data. At least one specification is required.
source	Character scalar. Name of the source raster layer in raster_stack from which the covariate is derived. Must exactly match a layer name in the raster stack provided to kernel_prep.
metric	Character scalar. The metric to compute within the circular buffer. For landscape_var() this is a landscape ecology metric computed on a categorical raster; for surface_var() this is a continuous surface texture metric ("sa" or "sq") computed on a continuous raster. Must be one of the supported metrics (see Details). Matched case-insensitively.
radius	Optional positive numeric. Fixed buffer radius in the same units as the projection. When NULL (default), the radius is treated as a free parameter and optimized

alongside the model. When supplied, the metric is computed at this fixed radius and no scale optimization is performed for this covariate.

`base` Positive numeric (not equal to 1). Logarithm base used by every entropy-based metric: the diversity metrics (`"shdi"`, `"shei"`, `"msidi"`, `"msiei"`) and the information-theory metrics (`"ent"`, `"condent"`, `"joinent"`, `"mutinf"`). Default: `exp(1)` (natural log, so entropies are in nats). Use `2` for bits or `10` for Hartley units. To reproduce the information-theory values reported by FRAGSTATS and the `landscapemetrics` package, set `base = 2`. The base cancels in the ratio metrics (`"relmutinf"`, `"iji"`), so they are unaffected by it.

`classes_max` Optional positive numeric. Maximum number of possible patch types in the landscape, used only for relative patch richness (`"rpr"`). If `NULL` (default), the observed number of classes within each buffer is used as the denominator.

`class` Optional integer-like scalar. The focal class for the class-level metrics (`"clumpy"`, `"pland"`, `"ca"`), which describe one category rather than the whole landscape. It must match one of the integer class codes in the source raster. It is required for those three metrics and is not accepted for any other metric (which describe the whole buffer). Where the focal class is absent from a buffer, `"pland"` and `"ca"` return 0 (zero cover) and `"clumpy"` returns `NA`.

`weighted` Logical, for `surface_var()` only. When `FALSE` (default), the surface metric is computed over a hard-edged circular neighborhood and the optimized scale parameter is that neighborhood radius. When `TRUE`, each cell is weighted by the model's distance kernel (the `kernel` passed to `kernel_prep`) and the optimized parameter is the kernel scale (sigma), exactly as for a kernel-weighted mean. The weighted form estimates the scale of effect of heterogeneity with a likelihood that is smooth in sigma. It is available only for the distribution metrics (`"sa"`, `"sq"`, `"ssk"`, `"sku"`), not the slope metric (`"sdq"`), and the scale is always optimized, so `radius` must be `NULL`.

Details

Kernel vs. landscape covariates

`kernel_var(source)` defines a kernel-weighted mean of the continuous raster values within a circular neighborhood. The neighborhood radius (sigma) is optimized by `multiScale_optim`.

`landscape_var(source, metric)` derives a landscape ecology metric from a categorical (or thresholded) raster layer within a circular buffer. Raster classes must be encoded as finite integer-like values. The buffer radius can be fixed or optimized.

`surface_var(source, metric)` derives a continuous surface texture metric from a continuous raster layer (elevation, NDVI, canopy height, temperature, moisture, and similar surfaces) within a circular buffer. These metrics summarize within-neighborhood heterogeneity rather than an average value, so they complement `kernel_var()` (which summarizes the mean). The buffer radius can be fixed or optimized. Note that these covariates measure variability, not central tendency: changing the radius changes both the ecological scale and the statistical quantity being summarized.

By default a surface metric uses a hard-edged neighborhood. Setting `weighted = TRUE` instead weights each cell by the model's distance kernel and optimizes the kernel scale (sigma), which estimates the scale of effect of heterogeneity with a smooth likelihood (see the `weighted` argument). This applies to the distribution metrics (`"sa"`, `"sq"`, `"ssk"`, `"sku"`).

Supported landscape metrics

Composition metrics (require a categorical raster):

`"shdi"` Shannon diversity index.

`"shei"` Shannon evenness index.

`"sidi"` Simpson diversity index.

`"siei"` Simpson evenness index. Can be uninformative for optimized scales when local evenness is uniform across sample points.

`"msidi"` Modified Simpson diversity index.

`"msiei"` Modified Simpson evenness index. Can be uninformative for optimized scales when local evenness is uniform across sample points.

`"pr"` Patch richness (number of distinct classes).

`"prd"` Patch richness density (pr per 100 ha).

`"rpr"` Relative patch richness (pr / classes_max * 100).

`"ta"` Total area of the buffer (ha).

Edge metrics (require adjacency information; cell IDs are cached internally):

`"ed"` Edge density (m/ha).

`"te"` Total edge length (m).

`"lsi"` Landscape shape index.

Adjacency metrics (configuration; built from cell-pair counts):

`"ai"` Aggregation index (%). Area-weighted percentage of like adjacencies relative to the maximum possible for each class. Higher values mean classes are more spatially aggregated.

`"pladj"` Proportion of like adjacencies (%). Percentage of all cell-pair boundaries that fall between cells of the same class.

`"contag"` Contagion index (%). Ranges from near 0 (maximum interspersion of classes) to 100 (a single class). Requires at least 2 classes within the buffer.

`"iji"` Interspersion and juxtaposition index (%). The evenness of the adjacencies among the different classes, ignoring like adjacencies. Ranges from near 0 (one class pair dominates the between-class boundaries) to 100 (every class is equally adjacent to every other). Undefined, and returned as `NA`, for fewer than 3 classes. For optimized scales, prescreen for usable variation because local windows with too few classes or between-class adjacencies can produce all `NA` values.

Information theory metrics (configuration and composition; built from the class-pair co-occurrence matrix following Nowosad and Stepinski, 2019). The `base` argument sets the logarithm base; set `base = 2` for bits, as in `landscapemetrics`:

`"ent"` Marginal entropy, H(x). The diversity of the cell values entering adjacencies; it increases with the number and evenness of classes.

`"joinent"` Joint entropy, H(x, y). The diversity of the class-pair adjacencies themselves; the total complexity of the pattern.

"condent" Conditional entropy, H(y | x). The configurational complexity that remains once composition is accounted for: the uncertainty about a neighbor's class given the focal cell's class.

"mutinf" Mutual information, I(y, x). How much knowing a cell's class tells you about its neighbor; it rises as like classes cluster together. This is the configuration signal separated from composition.

"relmutinf" Relative mutual information, I(y, x) / H(x). Mutual information rescaled by marginal entropy to the range 0 to 1, so that configuration can be compared across landscapes of differing composition. It is 1 when mutual information is 0 (the landscapemetrics convention) and is unaffected by base.

Class-level metrics (describe a single focal class; require the class argument):

"clumpy" Clumpiness index for the focal class. Ranges from -1 (maximally disaggregated) through 0 (randomly distributed) to 1 (maximally clumped), correcting the proportion of like adjacencies for the class's own abundance. Returned as NA where the class fills or is absent from the buffer.

"pland" Percentage of landscape (%). The focal class's share of the valid buffer cells.

"ca" Class area (hectares) of the focal class within the buffer.

Adjacency and information-theory projections are class-count limited to avoid accidental use of continuous rasters and to keep FFT projection times bounded. The current ceilings are 200 classes for the like-adjacency metrics ("ai", "pladj", "clumpy") and 50 classes for the metrics built from the full class-pair matrix ("contag", "iji", and the information-theory family), whose cost scales with the square of the class count.

Before optimizing the scale of "iji", "siei", or "msiei", inspect a pilot kernel_prep() result or a few fixed radii. These metrics can be undefined or nearly constant when buffers contain too few classes, too few between-class adjacencies, or nearly identical evenness at all sample points. In that case, use a larger radius, increase max_D, or choose a more stable diversity, entropy, or edge metric.

Supported surface texture metrics

Surface metrics (require a continuous raster):

"sa" Average roughness: the mean absolute deviation of the neighborhood values from their mean.

"sq" Root mean square (RMS) roughness: the sample standard deviation of the neighborhood values. More sensitive to large deviations than "sa".

"ssk" Skewness: the asymmetry of the neighborhood value distribution. Positive when occasional high values dominate, negative when occasional low values do.

"sku" Kurtosis (excess): the peakedness and tail weight of the neighborhood value distribution relative to a normal distribution (which has excess kurtosis 0).

"sdq" Root mean square slope: the RMS of the local gradient magnitude, a measure of how steeply the surface changes (terrain or structural ruggedness).

"sdr" Surface area ratio: the percentage by which the true 3D surface area exceeds the flat projected area. Behaves as a continuous edge-density analog (more relief gives a larger ratio).

The dispersion and shape metrics ("sa", "sq", "ssk", "sku") match the geodiv package (sq uses an N - 1 denominator as stats::sd() does; ssk uses the bias-adjusted skewness and sku the excess kurtosis, matching geodiv's defaults). "sdq" is reported as a true slope (the gradient is

divided by the cell resolution); `geodiv::sdq()` instead uses a cell spacing of one, so the two agree after multiplying by the resolution. `"sdr"` is computed in physical units (real elevation values and the real cell size), so a tilted plane of slope `s` returns `(sqrt(1 + s^2) - 1) * 100`; `geodiv::sdr()` instead rescales the surface to a unit range and so is dimensionless. The physical definition is the interpretable one for terrain and, unlike the rescaled version, can be projected by FFT.

Raster projection of `"sq"`, `"ssk"`, `"sku"`, `"sdq"`, and `"sdr"` uses fast Fourier transform (FFT) convolution; projection of `"sa"` uses an exact compiled masked focal pass, because average roughness has no closed-form FFT decomposition. `"sdq"` and `"sdr"` use each cell's neighbors, so (like the landscape edge metrics) they cache raster cell IDs internally, and their projection agrees with point extraction to within a small boundary tolerance rather than bit-for-bit.

Mixing covariate types

Multiple covariate types can be combined in one `msr_vars()` call. For example, a kernel-weighted mean of forest cover and a fixed-radius edge density metric can both be defined and passed together to `kernel_prep`:

```
vars <- msr_vars(
  forest_mean  = kernel_var("forest"),
  forest_ed500 = landscape_var("forest", metric = "ed", radius = 500)
)
kernel_inputs <- kernel_prep(pts, rasters, max_D = 1000,
                             scale_vars = vars)
```

Covariates with a fixed `radius` are computed once and not re-evaluated during optimization, which can meaningfully reduce computation time.

Value

`msr_vars()` A data frame of class `"multiScaleR_vars"` with one row per covariate. Columns are `covariate` (the name assigned in `...`), `source` (source raster layer name), `type` (`"kernel"`, `"landscape"`, or `"surface"`), `metric` (landscape or surface metric, or `NA`), `radius` (fixed radius or `NA` when optimized), `optimize` (logical indicating whether the scale is estimated), `weighted` (logical, the kernel-weighted form of a surface metric), `base` (logarithm base for entropy metrics), `classes_max` (richness denominator for `"rpr"`, or `NA`), and `class` (focal class for the class-level metrics, or `NA`).

`kernel_var()` An internal list of class `"multiScaleR_var"` representing a kernel-weighted mean covariate. Pass one or more of these inside `msr_vars()`.

`landscape_var()` An internal list of class `"multiScaleR_var"` representing a landscape metric covariate. Pass one or more of these inside `msr_vars()`.

`surface_var()` An internal list of class `"multiScaleR_var"` representing a continuous surface texture metric covariate. Pass one or more of these inside `msr_vars()`.

References

Nowosad, J., & Stepinski, T. F. (2019). Information theory as a consistent framework for quantification and classification of landscape patterns. *Landscape Ecology*, 34(9), 2091-2101. doi:10.1007/s1098001900830x

See Also

kernel_prep, multiScale_optim, kernel_scale.raster. For a worked walkthrough of the landscape metrics see vignette("landscape_metric_covariates", package = "multiScaleR"), and for the continuous-surface metrics see vignette("surface_metric_covariates", package = "multiScaleR").

Examples

```
## Kernel-weighted mean only (equivalent to default behavior)
vars <- msr_vars(
  forest_prop = kernel_var("forest")
)

## Combining kernel, landscape, and surface covariates
vars <- msr_vars(
  forest_prop = kernel_var("forest"),
  forest_ed   = landscape_var("forest", metric = "ed"),
  cover_shdi_500 = landscape_var("landcover", metric = "shdi", radius = 500),
  elev_sd     = surface_var("elevation", metric = "sq")
)

## landscape_var with natural-log Shannon diversity (the default)
landscape_var("landcover", metric = "shdi")

## landscape_var with log2 Shannon diversity and a fixed 250 m radius
landscape_var("landcover", metric = "shdi", radius = 250, base = 2)

## Configuration: interspersion (landscape-level), no class argument
landscape_var("landcover", metric = "iji", radius = 500)

## Information theory in bits (set base = 2 to match landscapemetrics)
landscape_var("landcover", metric = "mutinf", radius = 500, base = 2)

## Class-level metrics require `class`: clumpiness and cover of class 3
landscape_var("landcover", metric = "clumpy", radius = 500, class = 3)
landscape_var("landcover", metric = "pland", radius = 500, class = 3)

## surface_var: optimized RMS roughness, and fixed-radius average roughness
surface_var("elevation", metric = "sq")
surface_var("ndvi", metric = "sa", radius = 300)

## Kernel-weighted roughness: optimize the scale of effect of heterogeneity
surface_var("elevation", metric = "sq", weighted = TRUE)

## Define a single optimized kernel-weighted mean covariate
kv <- kernel_var("forest")
```

---

multiScale_optim *Multiscale optimization*

Description

Identifies the kernel scale of effect (sigma) for each raster covariate by maximizing the log-likelihood of a fitted statistical model. Repeatedly replaces kernel-weighted covariate values at different scales and refits the model, using `optim` (or `optimParallel` for parallel execution) with the L-BFGS-B algorithm.

Usage

```
multiScale_optim(
  fitted_mod,
  kernel_inputs,
  join_by = NULL,
  par = NULL,
  start_strategy = c("single", "screen"),
  screen_n_sigma = 5,
  screen_n_jitter = 6,
  screen_maxit = 8,
  screen_jitter_sd = 0.5,
  n_cores = NULL,
  PSOCK = FALSE,
  verbose = TRUE,
  refit_fn = NULL
)
```

Arguments

`fitted_mod`	A fitted model object whose covariates include the kernel-weighted variables defined in `kernel_inputs`. Supported classes include `lm`, `glm`, `gls` (nlme), and `unmarkedFit` (unmarked). Many other model classes are also supported via `stats::update()` or a custom `refit_fn`. A predictor cannot be named `"x"`, which is reserved for internal coordinate data.
`kernel_inputs`	A list of class `"multiScaleR_data"` created by `kernel_prep`. Must contain elements `raw_cov`, `d_list`, `min_D`, `max_D`, `unit_conv`, and `kernel`.
`join_by`	Default: `NULL`. For `unmarked` models, a data frame with one unique key row per spatial site in `kernel_inputs`, in the same order as those sites. Its named key columns must also occur in the fitted model's `siteCovs`. Use it when model rows repeat sites, such as one row per site and year. See Details.
`par`	Optional numeric vector of starting values for the optimizer. Values must be divided by `max_D` to match the internal scaled parameter space. Length must equal the number of optimized covariates (or twice that for `kernel = "expow"`, where shape parameters follow sigma parameters). Default: `NULL`; starting values are chosen automatically.
`start_strategy`	Character. How to choose starting values when `par = NULL`: `"single"` (default) uses one shared default start, while `"screen"` performs a low-cost prescreen of sigma values and short screening optimizations before launching one full optimization.

`screen_n_sigma` Positive integer. Number of log-spaced sigma values evaluated per covariate during the prescreen when `start_strategy = "screen"`. Default: `5`.

`screen_n_jitter` Non-negative integer. Number of jittered candidate starts evaluated with short screening optimizations after the marginal prescreen. Default: `6`. Set to `0` to skip the short jittered runs.

`screen_maxit` Positive integer. Maximum iterations used for each short screening optimization. Default: `8`.

`screen_jitter_sd` Non-negative numeric scalar. Standard deviation of multiplicative sigma jitter on the log scale during screening. Default: `0.5`.

`n_cores` Positive integer. Number of cores for parallel optimization via `optimParallel`. Default: `NULL` (single-threaded). When `start_strategy = "screen"`, the same `n_cores` setting is also used for short screening attempts before the full optimization. Parallel optimization is beneficial for models with many covariates or slow log-likelihood evaluations. When `n_cores > 1`, `multiScale_optim()` checks `estimate_multiscale_ram` and stops before screened starts or worker creation if the requested workers exceed the conservative RAM-based limit for the current inputs. Use `n_cores = 1` for serial execution, or rebuild inputs with `kernel_prep(..., bin = TRUE, store_cell_data = FALSE)` for lean mode when every covariate is kernel-weighted. Landscape and surface metrics cannot use lean mode. An unavailable RAM estimate does not block the run; a passing estimate does not guarantee sufficient memory. No worker count is reduced automatically.

`PSOCK` Logical. On Windows, a PSOCK cluster is always used. On Unix, a FORK cluster is used by default (faster). Set `TRUE` to force a PSOCK cluster on Unix. Default: `FALSE`.

`verbose` Logical. Print optimization status and warnings to the console. Default: `TRUE`.

`refit_fn` Optional function for refitting `fitted_mod` during optimization. When `NULL` (default), `stats::update()` (or the unmarked equivalent) is used. Provide this only when the default refit path is insufficient. See Details.

Details

Optimization approach

The optimizer uses the L-BFGS-B algorithm (bounded quasi-Newton) to maximize the log-likelihood over sigma (and shape for `"expow"`) while holding all regression coefficients at their fitted values for each candidate scale. Standard errors are Hessian-based approximations from the outer optimization. Summary methods report profile-likelihood confidence intervals for sigma when `profile_sigma` has been run on the object; otherwise they fall back to Hessian-based intervals. Candidate scales that make a covariate undefined at a modeled observation are rejected, so the optimizer compares likelihoods for the same sample. The final `sample_size` diagnostic reports prepared and fitted row counts; for repeated-site `unmarked` data it also reports the number of distinct prepared spatial sites. For hard-radius landscape and unweighted surface metrics, Hessian uncertainty is not meaningful; use `summary(x, profile = TRUE)` for profile-likelihood intervals.

Binned acceleration

When kernel_inputs was created by kernel_prep with bin = TRUE (the default), kernel-weighted covariates are evaluated from precomputed distance-binned summaries rather than by iterating every buffer cell on each optimizer evaluation. This makes the per-evaluation cost independent of max_D and typically speeds up optimization by an order of magnitude or more for large buffers, with a negligible binning approximation. Inputs created with store_cell_data = FALSE ("lean" mode) carry only the binned summaries; these still optimize, profile, and refit normally. Landscape-metric covariates always use the exact per-cell path and therefore require cell-level data.

Parallel optimization

To ensure that fitted model function calls are properly serialized for parallel workers, fit models using fully namespace-qualified functions. For example, fit a negative binomial model as fitted_mod <- MASS::glm.nb(y ~ x, data = df) rather than loading the namespace implicitly.

Joining unmarked multi-year data

When unmarked model rows repeat spatial sites, provide a join_by data frame with one row per prepared site. For example, if the spatial sites have IDs c("A", "B") and model rows have site IDs c("A", "B", "A", "B") for two years, use join_by = data.frame(site = c("A", "B")) and include the four-row site column in the model's siteCovs. Multiple key columns are allowed. Prepared keys must be unique and every model row must match. The model's response and site-covariate row order is preserved. Without join_by, the number and order of model rows must equal the prepared spatial sites. When both objects retain the same row IDs, their order is checked. Otherwise, the order must be supplied correctly by the caller. Optimized covariates must be static siteCovs; obsCovs and yearlySiteCovs cannot be optimized through this site-level workflow.

Custom refit functions

By default, multiScale_optim() refits the model using stats::update() (standard models) or the unmarked equivalent. When the default path is insufficient, supply refit_fn. The function must accept arguments model, data, and context, and return a fitted model whose log-likelihood can be extracted by stats::logLik() or insight::get_loglikelihood().

A minimal custom refit function:

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

For models requiring explicit call reconstruction:

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

With PSOCK clusters, refit_fn must serialize cleanly. Prefer namespace-qualified calls (e.g., stats::update()) and avoid closures over large local objects.

Wrapper model objects

Some packages return wrapper objects around another fitted model (e.g., amt::fit_clogit() wraps a survival::clogit() fit in its model component). When possible, multiScale_optim()

unwraps these automatically. For `amt::fit_clogit()`, pass `model = TRUE` when fitting so the nested `clogit` model retains its model frame.

When ‘start_strategy = "screen"‘ and ‘par‘ is left as ‘NULL‘, ‘multiScale_optim()‘ first scouts the sigma space with one-dimensional log-spaced scans, then optionally tests a few jittered starts using short, Hessian-free optimizations. The marginal sigma scans are serial, but the short screening attempts use ‘n_cores‘ when parallel optimization is requested. These screening steps are used only to choose one starting vector for the single full optimization. On Windows, those parallel screening attempts use PSOCK workers, so the same PSOCK serialization guidance applies as in the full optimization. For reproducible screened starts, call ‘set.seed()‘ before ‘multiScale_optim()‘.

Value

A list of class `"multiScaleR"` containing:

`scale_est` Data frame with one row per optimized covariate and two columns: `Mean` (optimized sigma on the original projection scale) and `SE` (Hessian-based standard error). `SE` is `NA` for hard-radius landscape and unweighted surface metrics, whose likelihood changes in steps. Row names are covariate names.

`shape_est` Data frame of the same structure as `scale_est` for the shape parameter, or `NULL` when `kernel != "expow"`.

`optim_results` The raw list returned by `stats::optim` or `optimParallel::optimParallel`, including `par`, `value` (negative log-likelihood), `hessian`, and convergence codes. Also contains `par_unscale` (sigma values back on the projection scale) and `hessian_unscale`.

`opt_mod` The refitted model at the optimized sigma values. This is the model object to use for inference, prediction, and `plot_marginal_effects`.

`fitted_mod_original` The original `fitted_mod` passed by the user, stored for reference.

`min_D` Numeric. Lower bound for sigma during optimization.

`max_D` Numeric. Upper bound for sigma during optimization.

`kernel_inputs` The `kernel_inputs` list (minus `min_D` and `max_D`, which are stored separately).

`scl_params` Named list with `mean` and `sd` vectors for each covariate: the centering and scaling parameters from the optimized kernel data. Used by `kernel_scale.raster` when `scale_center = TRUE`.

`join_by` The `join_by` data frame, or `NULL`.

`opt_context` Internal optimization context object storing the model-class-specific refit logic. Retained for use by `profile_sigma`.

`profile_scale_est` Profile-likelihood CI data frame, or `NULL` until `profile_sigma` has been run.

`diagnostics` List of diagnostic objects; see `diagnostics`.

`next_run` List of recommended values for a follow-up fit. Includes `max_D` for the next `kernel_prep` call, optimized `start_sigma` values in map units, optional `start_shape` values for `kernel = "expow"`, and `start_par` on the internal scaled parameter space expected by `multiScale_optim`. `n_cores` is `NULL`; a conservative parallel worker-count suggestion is available on demand from `estimate_multiscale_ram` (it is no longer computed automatically, as that query can be slow and shells out to an external process).

`warn_message` Integer vector of triggered warning codes: 1 = max-distance, 2 = sigma precision, 3 = shape precision.

`call` The matched call.

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

Plots the optimized kernel weight distribution for each spatial covariate in a fitted `multiScaleR` object, with optional annotations showing the effective scale of effect and its 95% confidence interval.

Usage

```
## S3 method for class 'multiScaleR'
plot(x, dimension = c("2d", "1d"), ...)
```

Arguments

x	A fitted multiScaleR object returned by multiScale_optim.
dimension	Character. "2d" (default) marks the radius containing the requested proportion of ideal planar kernel weight. "1d" restores the original distance along a line through the point.
...	Optional named arguments to customize the plot:
	prob Numeric between 0 and 1 (exclusive). Cumulative kernel weight enclosed by the marked distance under the selected dimension. Default: 0.9.
	scale_dist Logical. If TRUE (default), a vertical dashed line is drawn at the effective distance, and a shaded rectangle spans the 95% confidence interval.
	add_label Logical. If TRUE (default), a caption below the plot reports sigma, the dimensional definition, the effective distance, and 95% CI bounds (in map units).

Details

Development versions after 0.7.3 default to 2D reporting, as do kernel_dist and plot_kernel. Set dimension = "1d" to reproduce the original line-based distances. This switch changes distance markers and converted confidence limits, not the fitted sigma, regression coefficients, or likelihood.

The curve shows cell weight declining with distance, normalized over the plotted sample distances. It is not a radial weight distribution. A planar marker accounts for the increasing area of concentric rings, so the area under the displayed curve to its left is not the stated proportion. Distances refer to an ideal untruncated kernel. Actual retained weight can differ with max_D, raster edges, missing cells, and cell geometry.

Hard-radius covariates show a uniform buffer and mark its full radius, regardless of prob or dimension. A shaded interval is drawn only when finite limits are available.

Value

A list of ggplot2 objects (one per covariate with a finite scale and effective distance), returned invisibly. Plots are printed as a side effect.

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, dimension = "1d")  # Original line-based distances

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

Visualizes the weight (density) of a kernel function as a function of distance, without requiring a fitted multiScaleR object. Useful for exploring how different kernel types and sigma values translate into spatial influence zones before running multiScale_optim. To plot the kernel from a fitted model, use plot(opt) instead.

Usage

```
plot_kernel(
  prob = 0.9,
  sigma,
  beta = NULL,
  kernel,
  scale_dist = TRUE,
  add_label = TRUE,
  dimension = c("2d", "1d"),
  ...
)
```

Arguments

prob	Numeric between 0 and 1 (exclusive). Proportion of ideal kernel weight enclosed by the marked distance under the selected dimension. Default: 0.9.
sigma	Positive numeric. The scale parameter of the kernel, in the same units as the projection used with kernel_prep (e.g., meters when using a metric CRS). Controls the width of the kernel: larger values spread influence over greater distances.
beta	Positive numeric. Shape parameter for the exponential power kernel (kernel = "expow"). Controls kernel shape: values near 1 approximate a Laplace (double-exponential) shape; values near 2 approximate Gaussian; larger values approach a flat-top (platykurtic) shape. Typical range is 1–50. Ignored for all other kernel types. Default: NULL.
kernel	Character. The kernel function to visualize. One of: "gaussian" Gaussian (normal) decay; weights follow a half-normal density with standard deviation sigma. "exp" Negative exponential decay; weights follow exp(-d / sigma). "fixed" Fixed-radius step function; all distances up to sigma receive equal weight. "expow" Exponential power kernel parameterized by sigma and beta.
scale_dist	Logical. If TRUE (default), a vertical dashed line marks the distance enclosing prob of kernel weight under the selected dimensional definition.

add_label	Logical. If TRUE (default), a caption reports the cumulative percentage, dimension, sigma, and effective distance below the plot. Requires scale_dist = TRUE.
dimension	Character. "2d" (default) marks the radius enclosing prob of ideal kernel weight on a plane. "1d" restores the original distance enclosing that proportion along a line through the point.
...	Not used.

Details

The curve shows relative weight assigned to a cell at each distance, normalized over the plotted sample distances. It is not the distribution of total weight in concentric rings. Under the default "2d" definition, the marker accounts for ring area (proportional to distance), so the area under the displayed curve to its left is not prob.

The x-axis covers at least 99.9% of weight under the selected definition, or prob when larger. Marker positions use the unrounded distance; labels round to two decimal places. Sigma remains a kernel parameter, not the marked radius. For a Gaussian kernel with sigma = 300 m, the 90% marker is about 644 m in 2D and 493 m in 1D.

Development versions after 0.7.3 default to 2D; use dimension = "1d" to reproduce the original distance definition. These are ideal continuous-kernel distances, not site-specific retained weight after max_D, raster boundaries, missing cells, or cell geometry limit the extraction. See kernel_dist for both definitions.

For fitted-model kernel plots (with confidence intervals), use plot(multiScaleR_object) instead.

Value

A ggplot2 object showing kernel weight (y-axis) as a function of distance in map units (x-axis), returned for display or customization.

See Also

kernel_dist, plot.multiScaleR

Examples

```
## Default: radius containing 95% of ideal planar weight
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')
```

```
## Original line-based definition
plot_kernel(sigma = 100, kernel = 'gaussian', dimension = '1d')
```

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

Description

Generates marginal effect plots with 95% confidence intervals for each covariate in the optimized model stored within a `multiScaleR` object. Each panel sweeps one covariate across its observed range while holding all other numeric predictors at their sample mean. For `unmarked` models, other categorical predictors are held at their first observed level.

Usage

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

Arguments

`x`	A fitted `multiScaleR` object returned by `multiScale_optim`. Must contain `opt_mod` (the optimized fitted model) and `scl_params` (a list with `mean` and `sd` elements for each covariate, used to back-transform scaled covariates to their original units for the x-axis).
`ylab`	Character scalar. Y-axis label for all marginal effect panels. Default: `"Estimated response"`.
`length.out`	Positive integer. Number of equally spaced points at which to evaluate the marginal effect curve for each covariate. Default: `100`.
`type`	Character. Prediction type for `unmarked` models. One of `"state"` (default) or `"lambda"`. Ignored for non-unmarked models.
`link`	Logical. If `TRUE`, predictions are passed through the model's inverse link function before plotting (i.e., plotted on the response scale). For `glm` objects this is applied automatically. Set to `TRUE` manually if predicted values appear to be on the link scale. Default: `FALSE`.

Details

Marginal effects are computed by constructing a `newdata` data frame where the focal covariate sweeps from its minimum to maximum observed value and all other predictors are held at their sample mean. For kernel-scaled covariates, whose scaled mean is 0, this is equivalent to holding

them at their average landscape value. Variables used only in model offsets are held at their sample mean for prediction but do not receive marginal effect panels.

For `glm` and most GLM-family models, predictions are transformed via the inverse link function automatically (e.g., `exp()` for Poisson log-link models). Confidence intervals are constructed from `predict(..., se.fit = TRUE)` using $\pm 1.96 \times SE$.

For `unmarked` models, `predict(mod, newdata, type = type)` is used, and lower/upper bounds come from the `lower` and `upper` columns of the returned data frame. Predictors are recovered from the selected component's formula, so factor contrasts such as `Year2` are not mistaken for data columns. Factors receive one point and confidence interval per observed level. Site keys not used as fixed predictors and detection-only predictors do not receive state-effect panels. Numeric predictors not optimized by `multiScale_optim()` retain their original units and are held at their sample mean in other panels.

For `unmarked` random-effects models, prediction uses `re.form = NA`, which sets random effects to zero. These curves are conditional predictions at the stated covariate values, not averages over years, sites, or the random-effect distribution. Confidence intervals are conditional on the fitted spatial scale. Explicit nondefault factor contrasts in ordinary `pcount` (Poisson or negative-binomial) and `occu` fits use `unmarked::linearComb()` and `unmarked::backTransform()` to preserve the fitted design and link. Other unmarked classes with such contrasts stop if their prediction method cannot safely retain that design.

For models with interaction terms, a message is printed explaining that each panel represents a conditional slice (at the mean of the interacting variable) rather than the full interaction surface.

`HLfit` (spaMM) and `zeroinfl` (pscl) models are handled with class-specific prediction calls. When `x$opt_mod` is a wrapper object around another fitted model, `plot_marginal_effects()` uses the nested analysis model for predictor discovery, data recovery, and prediction when possible.

Value

A named list of `ggplot2` objects, one per covariate. Plots are printed as a side effect and the list is returned invisibly. Each plot shows:

- The predicted response as a solid line for numeric predictors or points for categorical predictors, with x-axis values back-transformed to the original (unscaled) units when applicable.
- A shaded ribbon or categorical error bars for the 95% confidence interval (when available from the model's `predict` method).
- A subtitle noting any polynomial (`I(x^2)`) or interaction terms that involve the covariate.

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)

## Default marginal effect plot on the response scale
plot_marginal_effects(opt)

## Custom y-axis label
plot_marginal_effects(opt, ylab = "Predicted abundance")

## Finer evaluation grid
plot_marginal_effects(opt, length.out = 200)
```

`print.multiScaleR` *Print method for multiScaleR*

Description

Print method for objects of class `multiScaleR`.

Usage

```
## S3 method for class 'multiScaleR'
print(x, dimension = c("2d", "1d"), prob = 0.9, ...)
```

Arguments

`x`	A `multiScaleR` object
`dimension`	Character. `"2d"` (default) prints the ideal planar radius first; `"1d"` prints the original line-based distance first.
`prob`	Numeric between 0 and 1 (exclusive). Cumulative kernel-weight proportion used for distances. Default: `0.9`.
`...`	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, dimension = NULL, ...)
```

Arguments

`x`	A `summary_multiScaleR` object
`dimension`	Character. `"2d"` or `"1d"`. Selects which distance table is printed first. By default, use the definition stored in `x`; summaries saved before this switch existed retain 1D reporting.
`...`	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,
  n_cores = NULL,
  spacing = c("log", "linear"),
  sigma_values = NULL,
  sigma_range = NULL
)
```

Arguments

x	A fitted multiScaleR object returned by multiScale_optim.
n_pts	Positive integer (>= 3). Number of sigma values to evaluate for each covariate along the profile grid. More points give a smoother profile at the cost of additional model refits. Default: 10. Ignored when sigma_values is supplied.
metric	Character. Metric to display on the y-axis of the profile. One of "AICc" (default) or "LL" (log-likelihood). The optimized sigma minimizes negative log-likelihood, so the "LL" profile should peak at the optimized value and "AICc" should show a minimum.
verbose	Logical. Print per-covariate progress messages. Default: TRUE.
n_cores	Optional positive integer. Number of CPU cores to use when profiling covariates in parallel. Parallel profiling is applied across covariates with a PSOCK cluster. Default: NULL (serial profiling).
spacing	Character. Spacing of the automatically generated sigma grid. "log" (default) concentrates evaluation points at small sigma values where the likelihood surface typically changes more rapidly. "linear" provides equal spacing across the range.
sigma_values	Optional positive numeric vector of sigma values (in the original projection units) to evaluate directly. When supplied, spacing, sigma_range, and n_pts are ignored. Duplicate values are removed and remaining values are sorted before profiling. Must contain at least 3 unique values.

`sigma_range` Optional positive numeric vector of length 2 giving the minimum and maximum sigma values (in projection units) for the generated profile grid. Defaults to the optimization range stored in `x` (`min_D` to `max_D`). Ignored when `sigma_values` is supplied.

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 uses all free fitted likelihood parameters (including ancillary and unmarked detection or variance parameters), plus estimated sigma and shape parameters, and the existing model observation count. Parameter counting follows `aic_tab`. AICc is `NA` when `n <= K + 1`. If all refits return the optimizer penalty, profiling stops with an error instead of returning an artificial flat profile. Keep the model formula and data available to the stored model call, or provide a working `refit_fn` to `multiScale_optim`.

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

Generates simulated response data with known kernel-weighted landscape covariates at a controlled scale of effect. Useful for testing and demonstrating `multiScale_optim` with data where the true parameters are known.

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`	Numeric scalar. Intercept term for the linear predictor. Default: `1`.
`beta`	Numeric vector of slope coefficients, one per raster layer in `raster_stack`. Length must equal `nlyr(raster_stack)`.
`kernel`	Character. Kernel function used to weight raster values by distance when generating the true covariate values. One of `"gaussian"` (default), `"exp"` (negative exponential), `"expow"` (exponential power), or `"fixed"` (fixed-radius buffer).
`type`	Character. Distribution of the simulated response variable. One of: `"count"` (Default) Poisson-distributed counts. `"count_nb"` Negative binomial counts with dispersion `StDev`. `"occ"` Bernoulli (0/1) occupancy data via a logistic link. `"gaussian"` Normally distributed continuous response with standard deviation `StDev`.
`StDev`	Positive numeric. Dispersion parameter for `"count_nb"` (the `size` argument of `rnbinom`) or the standard deviation for `"gaussian"` responses. Default: `0.5`.

n_points	Positive integer or a SpatVector/sf point object. When an integer, that many points are placed on a hexagonal grid covering the raster extent (trimmed to 85% of the x/y range) and then randomly subsampled. When a spatial object is supplied, those points are used directly. Default: 50.
min_D	Positive numeric. Minimum inter-point spacing on the hexagonal grid. If NULL (default), set automatically to 1.55 * max(sigma).
raster_stack	A SpatRaster object. Layer names become covariate names in the returned data frame.
sigma	Positive numeric vector. True kernel scale parameters, one per raster layer. These are the values that multiScale_optim should recover. Must be in the same units as the raster projection.
shape	Positive numeric vector. Shape parameters for the exponential power kernel (kernel = "expow"), one per raster layer. Values between 1 and 50 are typical. Required when kernel = "expow".
max_D	Positive numeric. Maximum buffer radius for kernel_prep during simulation. If NULL (default), set automatically to 110% of the one-dimensional distance enclosing 99% of kernel weight along a line at max(sigma). This historical default is smaller than the radius enclosing 99% of weight on a plane; supply a larger max_D if that planar cutoff is needed.
user_seed	Optional integer seed for reproducibility. Default: NULL.
...	Additional arguments. Not currently used.

Details

Points are distributed on a hexagonal grid across the interior of the raster extent (85% of the range in each direction to avoid edge effects), then randomly subsampled to n_points. The min_D spacing controls the grid resolution; if the grid produces fewer points than n_points, min_D is reduced iteratively by 3% until enough points are generated.

Kernel-weighted covariate values are computed using kernel_prep at the specified sigma (and shape) values. These represent the "true" covariate values that the optimization should recover.

Value

A named list with three elements:

obs Numeric vector of length n_points containing the simulated response values.

df Data frame with n_points rows and one column per raster layer plus a y column. The raster columns contain the true kernel-weighted mean values, scaled to zero mean and unit variance. This data frame can be used to fit a model for multiScale_optim.

pts An sf POINT object with n_points rows. An obs column is appended containing the simulated response values. Pass this to kernel_prep as the pts argument.

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

Generates simulated replicated detection/non-detection or count data formatted for use with the `unmarked` package, with known kernel-weighted landscape covariates at a controlled scale of effect.

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`	Numeric scalar. Intercept for the abundance/occupancy linear predictor. Default: `1`.
`beta`	Numeric vector of slope coefficients, one per raster layer in `raster_stack`. Length must equal `nlyr(raster_stack)`.
`kernel`	Character. Kernel function used to weight raster values by distance. One of `"gaussian"` (default), `"exp"` (negative exponential), `"expow"` (exponential power), or `"fixed"` (fixed-radius buffer).

type	Character. Response type to simulate. One of: "count" (Default) Poisson-distributed abundance with binomial thinning to simulate detection probability det. "count_nb" Negative binomial abundance with dispersion StDev, then binomial thinning. "occ" Bernoulli occupancy data; occupancy is generated from a logistic model and detections follow Bernoulli(occ * det).
StDev	Positive numeric. Dispersion (size) parameter for "count_nb". Default: 0.5.
n_points	Positive integer. Number of spatial sample sites. Points are placed on a hexagonal grid and randomly subsampled. Default: 50.
n_surv	Positive integer. Number of repeated surveys per site, forming the columns of the returned observation matrix. Default: 3.
det	Numeric between 0 and 1. Per-survey detection probability applied via binomial thinning of the true abundance/occupancy. Default: 0.5.
min_D	Positive numeric. Minimum inter-point spacing on the hexagonal grid. If NULL (default), set automatically to 1.55 * max(sigma).
raster_stack	A SpatRaster object. Layer names become covariate names in the returned data frame.
sigma	Positive numeric vector. True kernel scale parameters, one per raster layer. Must be in the same units as the raster projection.
shape	Positive numeric vector. Shape parameters for kernel = "expow", one per raster layer. Required when using the exponential power kernel.
max_D	Positive numeric. Maximum buffer radius for kernel_prep during simulation. If NULL (default), set automatically to 110% of the one-dimensional distance enclosing 99% of kernel weight along a line at max(sigma). Supply a larger max_D if the radius enclosing 99% of weight on a plane is needed.
user_seed	Optional integer seed for reproducibility. Default: NULL.
...	Additional arguments. Not currently used.

Details

Sites are distributed on a hexagonal grid across the interior of the raster extent (85% of the x/y range) and then randomly subsampled. The true abundance or occupancy at each site is a function of the kernel-weighted landscape covariates at the specified scale. Repeated surveys introduce imperfect detection controlled by det.

Use the returned y matrix and df (as siteCovs) to construct an unmarkedFrame, fit a model, and then pass both the fitted model and fresh kernel_prep output to multiScale_optim.

Value

A named list with three elements:

y Integer matrix of dimensions n_points x n_surv containing the simulated replicated observations. Pass as the y argument when constructing an unmarkedFrame.

`df` Data frame with `n_points` rows. The `y` column contains the simulated true abundance/occupancy (before detection thinning). Remaining columns are the true kernel-weighted covariate values, scaled to zero mean and unit variance.

`pts` An `sf` POINT object with `n_points` rows. An `obs` column is appended containing the true abundance/occupancy values. Pass to `kernel_prep` as `pts`.

Examples

```
rs <- sim_rast(dim = 50, 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(40, 80),
                          n_points = 20,
                          n_surv = 3,
                          det = 0.5,
                          type = 'count',
                          raster_stack = rs,
                          max_D = 220,
                          user_seed = 123)
plot(s_dat$df$y ~ s_dat$df$bin1)
plot(s_dat$df$y ~ s_dat$df$cont1)
## unmarked analysis
kernel_inputs <- kernel_prep(pts = s_dat$pts,
                             raster_stack = rs,
                             max_D = 220,
                             kernel = 'gaus',
                             verbose = FALSE)

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

## Base unmarked model
mod0 <- unmarked::pcount(~1 ~ bin1 + cont1,
                         data = umf,
                         K = 30)

## The unmarked optimization is slower than the simulation and base model
## fit, so it is skipped in automated example checks.
if (interactive()) {
  opt1 <- multiScale_optim(fitted_mod = mod0,
                           kernel_inputs = kernel_inputs)

  summary(opt1)
}
```

`sim_rast` *Simulate spatially autocorrelated raster surfaces*

Description

Creates a SpatRaster stack of four simulated landscape surfaces for use with sim_dat and sim_dat_unmarked. The stack contains two binary (0/1) and two continuous (0–1) surfaces with differing spatial autocorrelation ranges, allowing simulation of multiscale ecological processes.

Usage

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

Arguments

dim	Positive integer. Number of cells on each side of the square raster. The output will be a dim x dim grid. Default: 100.
resolution	Positive numeric. Cell size (edge length) in map units. The raster extent will be dim * resolution in each direction. Default: 10.
autocorr_range1	Optional positive numeric. Spatial autocorrelation range (in map cells) for surfaces 1 and 3 (bin1, cont1). Controls the decay rate of the exponential covariance: larger values produce smoother, more broadly correlated patterns. If NULL (default), set to 5% of dim (fine-scale autocorrelation).
autocorr_range2	Optional positive numeric. Autocorrelation range for surfaces 2 and 4 (bin2, cont2). If NULL (default), set to 25% of dim (broad-scale autocorrelation).
sill	Positive numeric. Partial sill (variance) of the Gaussian random field used for all four surfaces. Default: 10.
plot	Logical. If TRUE, the raster stack is plotted using terra::plot after simulation. Default: FALSE.
user_seed	Optional integer seed for reproducibility. Different transformations of user_seed are applied to each of the four surfaces so they are independent but fully reproducible. Default: NULL.
...	Additional arguments. Not currently used.

Details

Each surface is generated as a Gaussian random field using a fast Fourier transform (FFT) circulant embedding approach with an exponential covariance function. The underlying continuous fields are normalized to [0, 1] before thresholding (binary surfaces) or returning directly (continuous surfaces).

The two autocorrelation ranges allow simulation of covariates that operate at different spatial scales, a common scenario in landscape ecology where some resources are patchily distributed at fine scales and others vary broadly across the study area.

When `user_seed` is provided, independent but reproducible seeds are derived for each surface as multiples of `user_seed`.

Value

A `SpatRaster` with four named layers:

`bin1` Binary (0/1) surface with fine-scale autocorrelation (`autocorr_range1`). Values of 1 where the underlying continuous field exceeds its 55th percentile.

`bin2` Binary (0/1) surface with broad-scale autocorrelation (`autocorr_range2`). Values of 1 where the underlying continuous field falls below its 40th percentile.

`cont1` Continuous surface rescaled to [0, 1] with fine-scale autocorrelation (75% of `autocorr_range1`).

`cont2` Continuous surface rescaled to [0, 1] with broad-scale autocorrelation (125% of `autocorr_range2`).

All layers span the same spatial extent: `[0, dim * resolution]` x `[0, dim * resolution]`.

---

summary.multiScaleR *Summarize multiScaleR objects*

Description

Summarizes output from `multiScale_optim`.

Usage

```
## S3 method for class 'multiScaleR'
summary(object, profile = FALSE, dimension = c("2d", "1d"), ...)
```

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.
`dimension`	Character. `"2d"` (default) reports the radius containing `prob` of ideal planar kernel weight as the selected distance. `"1d"` restores the original distance along a line. Both definitions remain available in the returned object and printed output.
`...`	Optional arguments passed to the method (e.g., `prob` for cumulative kernel weight threshold).

Details

Set `dimension = "1d"` to preserve the original `opt_dist` and `opt_distance` values in existing scripts. For a dimension-independent extraction, use `opt_dist_1d` or `opt_dist_2d`. Neither selection changes the fitted sigma, coefficients, or likelihood. The 2D radius is for an ideal continuous kernel; `max_D`, raster edges, missing cells, and cell geometry can change the realized weight in the prepared data. See `kernel_dist` for formulas and examples.

Value

An object of class `summary_multiScaleR`. Kernel-scale limits default to Wald intervals. Scale distances, in map units, come in two forms: `opt_dist` (also `opt_distance`) holds the selected definition, which defaults to 2D rather than the 1D definition used through version 0.7.3. `dimension` records that selection. `opt_dist_1d` always holds the one-dimensional distance, the distance at which the kernel's cumulative density along a line through the point reaches `prob` (about 1.65 sigma for a Gaussian kernel at 90%). `opt_dist_2d` holds the two-dimensional distance, the radius of the circle around the point that holds `prob` of the weight of an ideal, untruncated kernel on a plane (about 2.15 sigma for a Gaussian kernel at 90%). Both tables have columns `Mean`, `2.5%`, and `97.5%`, and hard-radius covariates report their radius in both. See `kernel_dist` for when to use each. For hard-radius landscape and unweighted surface metrics, Hessian standard errors and Wald limits are shown as `NA`; use `profile = TRUE` for profile-likelihood limits. Profiling errors are reported instead of silently returning invalid 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

aic_tab, 3, *6*, *46*
aictabCustom, *4*

bic_tab, 5
bictabCustom, *6*

count_data, 7

diagnostics, 8, *34*

estimate_multiscale_ram, 9, *32*, *34*

hab, 12

kernel_dist, *8*, 12, *35*, *37*, *40*, *55*
kernel_prep, *8–10*, *13, 14*, 15, *19–21*, *25, 26*, *29–31*, *33, 34*, *39*, *49*, *51, 52*
kernel_prep_by_group, *17*, 18
kernel_scale.raster, *18*, 21, *25*, *30*, *34*
kernel_var, *16*
kernel_var *(*msr_vars*)*, 25

landscape, 23
landscape_counts, 24
landscape_var, *16*
landscape_var *(*msr_vars*)*, 25

msr_vars, *16*, *19*, *22*, 25
multiScale_optim, *4, 5*, *8–10*, *12*, *15–18*, *20–22*, *26*, 30, *30*, *37*, *39*, *41*, *45, 46*, *48, 49*, *51*
multiScaleR *(*multiScaleR-package*)*, 3
multiScaleR-package, 3

plot.multiScaleR, *14*, 36, *40*
plot.sigma_profile, 38, *46*
plot_kernel, *14*, *37*, 39
plot_marginal_effects, *34*, 41
print.multiScaleR, 43
print.multiScaleR_data, 44
print.summary_multiScaleR, 44
profile_sigma, *14*, *17*, *32*, *34*, *38*, 45
pts, 47

sim_dat, 48, *53*
sim_dat_unmarked, 50, *53*
sim_rast, 52
summary.multiScaleR, *14*, 54
surface_var *(*msr_vars*)*, 25
surv_pts, 55

utils::object.size(), *10*