Package 'spatialkit'

August 7, 2026

Title Spatial Tessellation, Modeling, and Cross-Validation Toolkit

Version 1.0.0

Description A modular toolkit for spatial analysis workflows including coordinate reference system management, Voronoi/Delaunay/grid tessellation, feature-to-polygon assignment, geographically weighted regression (GWR, via 'GWmodel'), Bayesian spatial Gaussian process regression (via 'brms'), spatial cross-validation with block and buffered strategies, and model comparison. Provides an S3 class system ('spatial_fit') with consistent predict, fitted, and residuals methods across model backends.

License MIT + file LICENSE

URL https://github.com/elkronos/gis_modeling_toolkit

BugReports https://github.com/elkronos/gis_modeling_toolkit/issues

Encoding UTF-8

Depends R (>= 4.1.0)

Imports sf (>= 1.0), dplyr (>= 1.0), logger, digest, stats, methods, utils, parallel

Suggests sp, GWmodel, brms, cmdstanr, loo, tibble, geometry, gstat, ggplot2, patchwork, FNN, Matrix, testthat (>= 3.0.0), knitr, rmarkdown

Additional_repositories https://stan-dev.r-universe.dev

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Justin Chase [aut, cre, cph]

Maintainer Justin Chase <jchase.msu@gmail.com>

Repository CRAN

Date/Publication 2026-08-07 16:40:02 UTC

Contents

`assign_features_to_polygons` *Assign features to polygons and attach a polygon ID*

Description

Joins an sf layer of input features to a polygon layer via spatial join.

Usage

```
assign_features_to_polygons(
  features_sf,
  polygons_sf,
  polygon_id_col = "poly_id",
  keep_unassigned = FALSE,
  predicate = sf::st_intersects,
  largest = TRUE,
  tie_break = c("smallest_area", "first")
)
```

Arguments

`features_sf`	An sf object containing features to assign.
`polygons_sf`	An sf or sfc polygonal layer.
`polygon_id_col`	Name of the polygon identifier column. Default "poly_id".
`keep_unassigned`	Logical; retain unmatched features. Default FALSE.
`predicate`	Binary spatial predicate function. Default sf::st_intersects.
`largest`	Logical; for polygon-on-polygon joins with overlapping polygons, keep the polygon with the largest overlap. Default TRUE. Only effective with predicates that support it (e.g. st_intersects).
`tie_break`	Strategy for resolving features that match multiple polygons: `"smallest_area"` (default) keeps the polygon with the smallest area, `"first"` keeps the first match (original order-dependent behavior).

Value

An sf object with polygon_id_col attached.

Examples

```
library(sf)
set.seed(1)
pts <- st_as_sf(
  data.frame(x = runif(20, 0, 100), y = runif(20, 0, 100), val = rnorm(20)),
  coords = c("x", "y"), crs = 32632
```

```
)
bnd <- st_sf(geometry = st_sfc(st_polygon(list(rbind(
  c(0, 0), c(100, 0), c(100, 100), c(0, 100), c(0, 0)
))), crs = 32632))
grid <- create_grid_polygons(bnd, target_cells = 9, type = "square")
assigned <- assign_features_to_polygons(pts, grid)
table(assigned$poly_id)
```

`build_tessellation` *Build a tessellation (Voronoi, Delaunay triangles, hex grid, or square grid)*

Description

Build a tessellation (Voronoi, Delaunay triangles, hex grid, or square grid)

Usage

```
build_tessellation(
  points_sf,
  boundary = NULL,
  method = c("voronoi", "triangles", "hex", "square"),
  approx_n_cells = NULL,
  cellsize = NULL,
  expand = 0,
  clip = TRUE,
  keep_duplicates = FALSE,
  crs = NULL,
  quiet = FALSE
)
```

Arguments

`points_sf`	An sf object with POINT/MULTIPOINT geometry.
`boundary`	Optional polygonal sf/sfc.
`method`	One of "voronoi", "triangles", "hex", "square".
`approx_n_cells`	Approximate number of cells (grid methods). For hex grids the target is adjusted for packing density; the actual count after clipping to an irregular boundary may differ noticeably.
`cellsize`	Numeric cell size (grid methods).
`expand`	Buffer distance for Voronoi envelope.
`clip`	Logical; clip to boundary.
`keep_duplicates`	Logical; keep duplicate points.
`crs`	Optional target CRS.
`quiet`	Logical; suppress messages.

Value

A list with cells, index, boundary, method, params.

See Also

Other tessellation: `create_grid_polygons()`, `create_voronoi_polygons()`, `get_voronoi_seeds()`, `plot_tessellation_map()`

Examples

```
library(sf)
set.seed(1)
pts <- st_as_sf(
  data.frame(x = runif(20, 0, 100), y = runif(20, 0, 100)),
  coords = c("x", "y"), crs = 32632
)
tess <- build_tessellation(pts, method = "voronoi", quiet = TRUE)
tess$cells
```

---

clear_fitted_cache *Clear cached fitted values for a Bayesian spatial model*

Description

Removes the lazily-cached `fitted()` result so that the next call recomputes from the posterior. This is only necessary if the underlying `brmsfit` engine or training data has been manually mutated after fitting — normal usage never requires it.

Usage

```
clear_fitted_cache(object)
```

Arguments

`object`	A `bayesian_fit` object.

Value

`object`, invisibly (called for side effect).

clear_grid_cache *Clear the in-session grid cache*

Description

Removes all memoized grid results from the internal cache environment.

Usage

```
clear_grid_cache(cache_env = .gmt_cache)
```

Arguments

cache_env	Environment to clear. Default .gmt_cache.

Value

Invisibly, the number of entries removed.

clip_target_for *Build a polygonal clip target from points and/or a boundary*

Description

Build a polygonal clip target from points and/or a boundary

Usage

```
clip_target_for(points_sf, boundary = NULL, expand = 0, quiet = FALSE)
```

Arguments

points_sf	An sf object with POINT/MULTIPOINT geometry.
boundary	Optional polygonal sf object.
expand	Numeric expansion distance or fraction (0–1 = fraction of extent).
quiet	Logical; suppress messages.

Value

An sf polygon layer representing the clip target.

coef.bayesian_fit *Extract Bayesian model fixed-effect summaries*

Description

Extract Bayesian model fixed-effect summaries

Usage

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

Arguments

object	A bayesian_fit object.
...	Ignored.

Value

A data.frame of fixed-effect posterior summaries.

coef.gwr_fit *Extract GWR local coefficients*

Description

Extract GWR local coefficients

Usage

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

Arguments

object	A gwr_fit object.
...	Ignored.

Value

A data.frame of local coefficient estimates (one row per obs).

coerce_to_points *Coerce arbitrary geometries to representative points*

Description

Converts the geometry column of an sf object to POINTs using one of several strategies.

Usage

```
coerce_to_points(
  x,
  mode = c("auto", "centroid", "point_on_surface", "surface", "line_midpoint",
    "bbox_center"),
  tmp_project = TRUE
)
```

Arguments

x	An sf object.
mode	One of "auto", "centroid", "point_on_surface", "surface", "line_midpoint", "bbox_center".
tmp_project	Logical; temporarily project for line-based midpoints.

Value

An sf object with geometry coerced to POINTs.

Examples

```
library(sf)
poly <- st_sf(
  id = 1,
  geometry = st_sfc(st_polygon(list(rbind(
    c(0, 0), c(2, 0), c(2, 2), c(0, 2), c(0, 0)
  ))), crs = 32632)
)
coerce_to_points(poly, "auto")  # interior representative point
```

compare_models *Side-by-side comparison of fitted spatial models*

Description

Takes a named list of already-fit `spatial_fit` objects and produces a tidy comparison table including in-sample metrics and model-specific information criteria (AICc, LOOIC).

Usage

```
compare_models(fits, newdata = NULL, ...)
```

Arguments

fits	A named list of spatial_fit objects.
newdata	Optional sf for out-of-sample evaluation.
...	Extra arguments passed to predict().

Value

A data.frame comparing all models.

See Also

Other model evaluation: compare_models_cv(), evaluate_insample(), residual_morans_i()

---

compare_models_cv *Cross-validated comparison of spatial models*

Description

Fits and cross-validates one or more model types, returning a unified comparison table. Unlike compare_models(), this function does perform fitting (inside CV folds), because CV inherently requires repeated fitting.

Usage

```
compare_models_cv(
  data_sf,
  response_var,
  predictor_vars,
  models = c("GWR", "Bayesian"),
  k = 5,
  seed = 123,
  folds = NULL,
  boundary = NULL,
  pointize = "auto",
  gwr_args = list(),
  bayes_args = list(),
  summary = c("mean", "median"),
  quiet = FALSE
)
```

Arguments

data_sf	An sf object.
response_var	Response column name.
predictor_vars	Predictor column names.
models	Character vector: subset of c("GWR", "Bayesian").
k	Number of folds. Default 5.
seed	RNG seed. Default 123.
folds	Optional precomputed fold splits.
boundary	Optional polygon sf/sfc.
pointize	Geometry coercion strategy.
gwr_args	Extra arguments for fit_gwr_model() / cv_gwr().
bayes_args	Extra arguments for fit_bayesian_spatial_model().
summary	"mean" or "median" for Bayesian predictions.
quiet	Logical; suppress messages.

Value

A list with overall, by_fold, and per-model cv_results.

See Also

Other model evaluation: compare_models(), evaluate_insample(), residual_morans_i()

---

create_grid_polygons *Create square or hexagonal grid polygons over a boundary*

Description

Create square or hexagonal grid polygons over a boundary

Usage

```
create_grid_polygons(
  boundary,
  target_cells = NULL,
  type = c("square", "hex"),
  cellsize = NULL,
  n = NULL,
  clip = TRUE,
  crs = NULL,
  quiet = FALSE
)
```

Arguments

boundary	Polygonal sf or sfc object.
target_cells	Optional approximate desired number of cells. For hex grids the count is adjusted for hexagonal packing density, but the final cell count after clipping to an irregular boundary may deviate substantially from the requested value.
type	Grid type: "square" or "hex".
cellsize	Optional numeric cell size (length 1 or 2).
n	Optional grid resolution (integer, length 1 or 2).
clip	Logical; clip grid to boundary.
crs	Optional target CRS.
quiet	Logical; suppress messages.

Value

An sf polygon layer with poly_id column.

See Also

Other tessellation: build_tessellation(), create_voronoi_polygons(), get_voronoi_seeds(), plot_tessellation_map()

Examples

```
library(sf)
bnd <- st_sf(geometry = st_sfc(st_polygon(list(rbind(
  c(0, 0), c(100, 0), c(100, 100), c(0, 100), c(0, 0)
))), crs = 32632))
grid_sq  <- create_grid_polygons(bnd, target_cells = 25, type = "square")
grid_hex <- create_grid_polygons(bnd, target_cells = 25, type = "hex")
nrow(grid_sq)
nrow(grid_hex)  # near-target thanks to hex packing correction
```

---

create_grid_polygons_cached

Create and cache grid polygons over a boundary

Description

Builds a grid via create_grid_polygons() and memoizes the result so repeated calls with the same inputs return instantly.

Usage

```
create_grid_polygons_cached(
  boundary,
  target_cells,
  type = c("hex", "square"),
  ...,
  cache_env = .gmt_cache
)
```

Arguments

boundary	An sf or sfc polygonal object.
target_cells	Approximate desired number of cells.
type	Grid type: "hex" or "square".
...	Additional arguments forwarded to create_grid_polygons().
cache_env	Environment for memoized grids. Default .gmt_cache.

Value

An sf data frame with a stable poly_id column.

---

create_voronoi_polygons

Create Voronoi polygons from points with robust CRS and optional clipping

---

Description

Create Voronoi polygons from points with robust CRS and optional clipping

Usage

```
create_voronoi_polygons(
  points_sf,
  boundary = NULL,
  expand = 0,
  clip = TRUE,
  keep_duplicates = FALSE,
  crs = NULL,
  quiet = FALSE
)
```

Arguments

points_sf	An sf object with POINT/MULTIPOINT geometries.
boundary	Optional polygonal sf object.
expand	Numeric; absolute buffer distance for the envelope.
clip	Logical; intersect cells with boundary.
keep_duplicates	Logical; keep coincident points for graph construction.
crs	Optional target CRS.
quiet	Logical; suppress messages.

Value

A list with cells, index, boundary, method, params.

See Also

Other tessellation: build_tessellation(), create_grid_polygons(), get_voronoi_seeds(), plot_tessellation_map()

Examples

```
library(sf)
set.seed(1)
pts <- st_as_sf(
  data.frame(x = runif(15, 0, 100), y = runif(15, 0, 100)),
  coords = c("x", "y"), crs = 32632
)
res <- create_voronoi_polygons(pts, quiet = TRUE)
res$cells   # one polygon per unique point, with stable cell_id
res$index   # cell_id assignment for each input point
```

---

cv_bayes *K-fold cross-validation for the Bayesian spatial model*

Description

K-fold cross-validation for the Bayesian spatial model

Usage

```
cv_bayes(
  data_sf,
  response_var,
  predictor_vars,
  folds = NULL,
  k = 5,
```

```
  seed = 123,
  boundary = NULL,
  pointize = "auto",
  fit_args = list(),
  summary = c("mean", "median"),
  compute_pred_intervals = TRUE,
  coverage_levels = c(0.5, 0.8, 0.95),
  block_size = NULL,
  auto_range = FALSE,
  parallel = FALSE
)
```

Arguments

data_sf	An sf object.
response_var	Response column name.
predictor_vars	Predictor column names.
folds	Optional list of fold definitions.
k	Number of folds. Default 5.
seed	RNG seed. Default 123.
boundary	Optional polygonal sf/sfc for CRS alignment.
pointize	Geometry coercion strategy.
fit_args	Named list of extra arguments for fit_bayesian_spatial_model(). A user-supplied gp_k is respected in every fold; when omitted, the GP rank is auto-selected per training fold. compute_loo, boundary, and pointize are always overridden by the CV internals.
summary	"mean" or "median" for posterior predictions.
compute_pred_intervals	Logical; compute predictive intervals.
coverage_levels	Numeric vector of coverage levels.
block_size	Optional minimum block edge length for spatial CV blocks (projected CRS units).
auto_range	Logical. If TRUE and folds is NULL, estimate the autocorrelation range and use it as the minimum block size. Default FALSE.
parallel	Logical or positive integer. If TRUE, auto-detect the number of cores and fit folds in parallel via parallel::mclapply() (macOS / Linux; falls back to sequential on Windows). If an integer > 1, use that many cores. Default FALSE (sequential). Bayesian folds with full MCMC runs are the primary beneficiary of this option.

Value

A list with overall, fold_metrics, predictions, folds, formula, and predictive_coverage.

See Also

Other cross-validation: cv_gwr(), cv_spatial(), estimate_sac_range(), make_folds()

Examples

```
## Not run:
if (requireNamespace("brms", quietly = TRUE)) {
  library(sf)
  set.seed(1)
  n <- 60
  dat <- st_as_sf(
    data.frame(x = runif(n, 0, 1000), y = runif(n, 0, 1000), elev = rnorm(n)),
    coords = c("x", "y"), crs = 32632
  )
  dat$price <- 10 + 0.01 * st_coordinates(dat)[, 1] + 2 * dat$elev + rnorm(n)
  cv <- cv_bayes(dat, "price", "elev", k = 2,
                 fit_args = list(chains = 2, iter = 500))
  cv$overall
  cv$predictive_coverage  # coverage at 50/80/95% plus mean CRPS
}

## End(Not run)
```

---

cv_gwr *K-fold cross-validation for GWR*

Description

K-fold cross-validation for GWR

Usage

```
cv_gwr(
  data_sf,
  response_var,
  predictor_vars,
  folds = NULL,
  k = 5,
  seed = 123,
  adaptive = TRUE,
  bandwidth = NULL,
  kernel = c("bisquare", "gaussian", "tricube", "boxcar", "exponential"),
  boundary = NULL,
  pointize = "auto",
  block_size = NULL,
  auto_range = FALSE,
  parallel = FALSE
)
```

Arguments

data_sf	An sf object.
response_var	Response column name.
predictor_vars	Predictor column names.
folds	Optional list of fold definitions.
k	Number of folds. Default 5.
seed	RNG seed. Default 123.
adaptive	Logical; use adaptive bandwidth. Default TRUE.
bandwidth	Optional bandwidth value.
kernel	Kernel function type.
boundary	Optional polygonal sf/sfc for CRS alignment.
pointize	Geometry coercion strategy.
block_size	Optional minimum block edge length for spatial CV blocks (projected CRS units). Ensures blocks are at least as large as the spatial autocorrelation range.
auto_range	Logical. If TRUE and folds is NULL, estimate the autocorrelation range and use it as the minimum block size. Default FALSE.
parallel	Logical or positive integer. If TRUE, auto-detect the number of cores and fit folds in parallel via parallel::mclapply() (macOS / Linux; falls back to sequential on Windows). If an integer > 1, use that many cores. Default FALSE (sequential).

Value

A list with overall, fold_metrics, predictions, folds, formula, adaptive.

See Also

Other cross-validation: cv_bayes(), cv_spatial(), estimate_sac_range(), make_folds()

Examples

```
if (requireNamespace("GWmodel", quietly = TRUE) &&
    requireNamespace("sp", quietly = TRUE)) {
  library(sf)
  set.seed(1)
  n <- 60
  dat <- st_as_sf(
    data.frame(x = runif(n, 0, 1000), y = runif(n, 0, 1000), elev = rnorm(n)),
    coords = c("x", "y"), crs = 32632
  )
  dat$price <- 10 + 0.01 * st_coordinates(dat)[, 1] + 2 * dat$elev + rnorm(n)
  cv <- cv_gwr(dat, "price", "elev", k = 3, bandwidth = 30)
  cv$overall
  cv$fold_metrics
}
```

`cv_spatial` *Model-agnostic spatial cross-validation*

Description

Run K-fold CV for any model that returns a `spatial_fit` object. This is the extensibility point: to plug in a new model type, supply a `fit_fn(train_sf)` that returns a `spatial_fit`.

Usage

```
cv_spatial(
  data_sf,
  response_var,
  predictor_vars,
  fit_fn,
  folds = NULL,
  k = 5,
  seed = 123,
  boundary = NULL,
  pointize = "auto",
  predict_args = list(),
  fold_info_fn = NULL,
  p = NULL,
  block_size = NULL,
  auto_range = FALSE,
  parallel = FALSE
)
```

Arguments

`data_sf`	An sf object.
`response_var`	Response column name.
`predictor_vars`	Predictor column names.
`fit_fn`	A function(train_sf) that returns a `spatial_fit`.
`folds`	Optional fold definitions. Built via block_kfold if NULL.
`k`	Number of folds.
`seed`	RNG seed.
`boundary`	Optional boundary for fold construction.
`pointize`	Geometry coercion strategy.
`predict_args`	Extra arguments for predict().
`fold_info_fn`	Optional function for per-fold extras.
`p`	Number of predictors for Adj R^2 (NULL to skip). Only meaningful for models with a fixed global parameter count; pass NULL for models with spatially varying coefficients (e.g. GWR).

block_size	Optional minimum block edge length for spatial CV blocks (projected CRS units).
auto_range	Logical. If TRUE and folds is NULL, estimate the autocorrelation range and use it as the minimum block size. Default FALSE.
parallel	Logical or positive integer. If TRUE, auto-detect the number of cores and fit folds in parallel via parallel::mclapply() (macOS / Linux; falls back to sequential on Windows). If an integer > 1, use that many cores. Default FALSE (sequential).

Value

A list with overall, fold_metrics, predictions, folds.

See Also

Other cross-validation: cv_bayes(), cv_gwr(), estimate_sac_range(), make_folds()

---

determine_optimal_levels

Determine an optimal number of spatial levels via an elbow heuristic

---

Description

Computes a WSS curve over k=1..K_max using k-means on projected feature coordinates and selects candidate k values around the elbow.

Usage

```
determine_optimal_levels(
  data_sf,
  max_levels = 12L,
  top_n = 3L,
  sample_n = 1500L,
  set_seed = 123L,
  response_var = NULL,
  predictor_vars = NULL,
  criterion = c("geometric", "morans_i", "combined")
)
```

Arguments

data_sf	An sf object.
max_levels	Integer upper bound on levels. Default 12.
top_n	Integer; how many candidates to return. Default 3.
sample_n	Integer; subsample size for speed. Default 1500.
set_seed	Integer RNG seed. Default 123.

response_var	Optional response column name. When provided alongside predictor_vars, enables model-aware level selection via Moran's I on OLS residuals.
predictor_vars	Optional predictor column names.
criterion	One of "geometric" (default when no response given), "morans_i" (select k that minimizes |Moran's I|), or "combined" (rank-average of WSS elbow distance and |Moran's I|). Falls back to "geometric" if response/predictors are unavailable.

Details

When response_var and predictor_vars are provided, the geometric WSS elbow is supplemented with Moran's I computed on OLS residuals at each candidate k. The Moran's I profile measures how much spatial autocorrelation in the response remains *unexplained* at a given tessellation resolution — a direct reflection of the spatial process being modeled, rather than mere geometric compactness of coordinates. The combined criterion selects the k that best balances geometric parsimony and residual spatial independence.

To keep memory use and runtime bounded for large max_levels, the initial k-means sweep records only within-cluster sum-of-squares (WSS) without retaining cluster assignments. Moran's I is then evaluated lazily: k-means is re-run only for a focused neighbourhood around the elbow (±4 by default, or ±top_n if larger), so that only the most promising candidate k values incur the cost of the full Moran's I computation.

Value

An integer vector of candidate level counts. When criterion != "geometric", an attribute "diagnostics" is attached with per-k Moran's I values.

Examples

```
library(sf)
set.seed(1)
# Two clearly separated clusters: the elbow should sit near k = 2
pts <- st_as_sf(
  data.frame(x = c(runif(25, 0, 10), runif(25, 90, 100)),
             y = c(runif(25, 0, 10), runif(25, 90, 100))),
  coords = c("x", "y"), crs = 32632
)
determine_optimal_levels(pts, max_levels = 6)
```

---

ensure_projected *Ensure an object has a projected CRS (with sensible defaults)*

Description

Coerces spatial objects to a projected coordinate reference system suitable for distance/area calculations.

Usage

```
ensure_projected(x, target_crs = NULL)
```

Arguments

x	An sf or sfc object (other objects returned unchanged).
target_crs	Optional target CRS (sf object, integer EPSG, or crs).

Value

x, potentially with a new projected CRS.

Examples

```
library(sf)
pts_ll <- st_as_sf(
  data.frame(lon = c(9.1, 9.2), lat = c(48.7, 48.8)),
  coords = c("lon", "lat"), crs = 4326
)
st_crs(ensure_projected(pts_ll))$epsg  # auto-selected UTM zone (32632)
```

---

ensure_stable_poly_id *Create deterministic, stable polygon IDs based on spatial sort keys*

Description

Ensures that a polygon layer has a reproducible, deterministic identifier column by sorting features using representative point coordinates (and secondary tie-breakers) and then assigning sequential IDs.

Usage

```
ensure_stable_poly_id(
  polygons_sf,
  id_col = "poly_id",
  method = c("centroid", "surface_point", "bbox_center"),
  make_valid = TRUE,
  transform_for_sort = 4326
)
```

Arguments

polygons_sf	An sf or sfc object containing polygonal features.
id_col	Character scalar; name of the identifier column.
method	One of "centroid", "surface_point", "bbox_center".
make_valid	Logical; apply st_make_valid() first. Default TRUE.
transform_for_sort	CRS used only for computing sort-key coordinates. Set to NULL to disable.

Value

An sf polygon layer re-ordered with sequential IDs in id_col.

`estimate_sac_range` *Estimate the spatial autocorrelation range from data*

Description

Fits exponential (or spherical) variogram models and returns the *effective range* — the distance at which the semivariance reaches ~95 \

Usage

```
estimate_sac_range(
  points_sf,
  response_var,
  predictor_vars = NULL,
  n_max = 5000L,
  cutoff = 0.5,
  seed = NULL
)
```

Arguments

`points_sf`	An sf object with point geometries (will be projected automatically if in geographic CRS).
`response_var`	Character(1) name of the response column.
`predictor_vars`	Optional character vector. When supplied, an OLS residual variogram is fitted instead of a raw-response variogram, which better reflects the autocorrelation that the spatial model must handle.
`n_max`	Maximum number of points to subsample before fitting. Variogram estimation is O(n²) so this keeps runtime bounded.
`cutoff`	Fraction of the maximum inter-point distance to use as the variogram lag cutoff. Default 0.5.
`seed`	Optional RNG seed for subsampling reproducibility.

Details

To guard against anisotropy, the function first estimates directional variograms at 0° (N–S) and 90° (E–W) azimuths (tolerance 22.5°, which avoids double-counting point pairs near the 45° diagonal but requires denser point clouds for stable estimates). When both fits succeed the **maximum** of the two directional ranges is returned, which is the conservative choice for spatial block CV — blocks must be at least as large as the longest autocorrelation range to avoid information leakage.

If either directional fit fails (e.g., too few point pairs in a direction), the function falls back to an omnidirectional (isotropic) variogram.

A log warning is emitted when notable anisotropy is detected (ratio of directional ranges > 1.5).

The returned range is in the coordinate units of the (projected) data and can be passed directly to `make_folds(block_size = ...)` to ensure that CV blocks are at least as wide as the autocorrelation range.

Value

A single positive numeric value (the effective range in projected coordinate units), or `NA_real_` if estimation fails.

See Also

Other cross-validation: `cv_bayes()`, `cv_gwr()`, `cv_spatial()`, `make_folds()`

Examples

```
if (requireNamespace("gstat", quietly = TRUE)) {
  library(sf)
  set.seed(1)
  n <- 80
  xy <- data.frame(x = runif(n, 0, 1000), y = runif(n, 0, 1000))
  xy$z <- sin(xy$x / 200) + rnorm(n, sd = 0.2)
  pts <- st_as_sf(xy, coords = c("x", "y"), crs = 32632)
  estimate_sac_range(pts, response_var = "z")
}
```

`evaluate_insample` *Compute in-sample (or out-of-sample) metrics for fitted spatial models*

Description

Accepts a single `spatial_fit` object or a named list of them. Does NOT refit — uses `fitted()` for in-sample and `predict()` for new data.

Usage

```
evaluate_insample(fits, newdata = NULL, ...)
```

Arguments

`fits`	A `spatial_fit` object, or a named list of them (e.g. `list(GWR = gwr_obj, Bayesian = bayes_obj)`).
`newdata`	Optional sf object for out-of-sample evaluation. Must contain the response variable and all predictors. If NULL, in-sample metrics are computed.
`...`	Extra arguments passed to predict().

Value

A data.frame with one row per model and columns for model name and all regression metrics.

See Also

Other model evaluation: `compare_models()`, `compare_models_cv()`, `residual_morans_i()`

`evaluate_models` *Evaluate spatial models (legacy interface)*

Description

Thin wrapper that preserves the original `evaluate_models()` call signature. New code should use `compare_models()` (for already-fit objects) or `compare_models_cv()` (for CV) instead.

Usage

```
evaluate_models(
  data_sf,
  response_var,
  predictor_vars,
  do_cv = TRUE,
  folds = NULL,
  k = 5,
  seed = 123,
  boundary = NULL,
  pointize = "auto",
  gwr_args = list(),
  bayes_args = list(),
  summary = c("mean", "median"),
  models = c("GWR", "Bayesian"),
  quiet = FALSE
)
```

Arguments

`data_sf`	An sf object.
`response_var`	Response column name.
`predictor_vars`	Predictor column names.
`do_cv`	Logical; use cross-validation. Default TRUE.
`folds`	Optional precomputed fold splits.
`k`	Number of folds. Default 5.
`seed`	RNG seed. Default 123.
`boundary`	Optional polygon sf/sfc.
`pointize`	Geometry coercion strategy.
`gwr_args, bayes_args`	Extra arguments for model fitting.
`summary`	"mean" or "median" for Bayesian predictions.
`models`	Character vector: subset of c("GWR", "Bayesian").
`quiet`	Logical; suppress messages.

Value

A list with CV or in-sample results.

`evaluate_models_cv` *Cross-validated comparison with optional tessellation (legacy interface)*

Description

Thin wrapper preserving the original `evaluate_models_cv()` call signature. New code should use `compare_models_cv()` directly.

Usage

```
evaluate_models_cv(
  data_sf,
  response_var,
  predictor_vars,
  k = 5,
  seed = 123,
  folds = NULL,
  boundary = NULL,
  pointize = "auto",
  tess_method = c("grid", "hex", "square", "voronoi", "triangles"),
  tess_args = list(),
  summary = c("mean", "median"),
  models = c("GWR", "Bayesian"),
  gwr_args = list(),
  bayes_args = list(),
  quiet = FALSE
)
```

Arguments

`data_sf`	An sf object.
`response_var`	Response column name.
`predictor_vars`	Predictor column names.
`k`	Number of folds. Default 5.
`seed`	RNG seed. Default 123.
`folds`	Optional precomputed fold splits.
`boundary`	Optional polygon sf/sfc.
`pointize`	Geometry coercion strategy.
`tess_method`	Tessellation type for diagnostics.
`tess_args`	Extra arguments for tessellation builders.

summary	"mean" or "median" for Bayesian predictions.
models	Character vector: subset of c("GWR", "Bayesian").
gwr_args	Extra arguments for fit_gwr_model() / cv_gwr().
bayes_args	Extra arguments for fit_bayesian_spatial_model().
quiet	Logical; suppress messages.

Value

A list with overall, by_fold, tessellation.

`fit_bayesian_spatial_model` *Fit a Bayesian spatial regression with a 2D Gaussian Process (via brms)*

Description

Fit a Bayesian spatial regression with a 2D Gaussian Process (via brms)

Usage

```
fit_bayesian_spatial_model(
  data_sf,
  response_var,
  predictor_vars,
  family = NULL,
  gp_k = NULL,
  gp_c = 1.5,
  prior = NULL,
  chains = 4,
  iter = 2000,
  warmup = floor(iter/2),
  cores = max(1L, parallel::detectCores() - 1L),
  seed = 123,
  backend = c("auto", "cmdstanr", "rstan"),
  control = list(adapt_delta = 0.9, max_treedepth = 12),
  compute_loo = TRUE,
  standardize_predictors = FALSE,
  check_convergence = TRUE,
  pointize = "auto",
  boundary = NULL,
  .already_prepped = FALSE
)
```

Arguments

data_sf	An sf object with response, predictors, and geometries.
response_var	Response column name.
predictor_vars	Predictor column names.
family	A model family accepted by brms::brm() (a stats family function or a brms family object). Default NULL (resolved to stats::gaussian()).
gp_k	Positive integer for GP rank, or NULL (default) for automatic selection based on dataset size: min(n/3, max(15, sqrt(n))).
gp_c	Positive numeric for GP scale. Default 1.5.
prior	Optional brms prior specification. When NULL and standardize_predictors = TRUE, weakly informative normal(0, 5) priors are set on regression coefficients. A data-informed GP length-scale prior is always appended automatically unless prior already contains an entry with class = "lscale".
chains	Number of MCMC chains. Default 4.
iter	Total iterations per chain. Default 2000.
warmup	Warmup iterations. Default floor(iter/2).
cores	Number of parallel cores.
seed	Integer seed. Default 123.
backend	"auto", "cmdstanr", or "rstan".
control	Named list of sampler controls.
compute_loo	Logical; compute PSIS-LOO. Default TRUE.
standardize_predictors	Logical; center and scale numeric predictors before fitting. Default FALSE. When TRUE, the scaling parameters are stored in the return value so predictions can be computed correctly.
check_convergence	Logical; after fitting, check for divergences, low ESS, and high R-hat and issue warnings. Default TRUE.
pointize	Strategy for non-point geometry coercion.
boundary	Optional polygonal sf/sfc for CRS harmonization.
.already_prepped	Logical (internal). If TRUE, skip the prep_model_data() call because the caller has already projected, coerced, and filtered the data. The data must then have plain POINT geometry (an error is raised otherwise). Used by the CV internals to avoid a redundant second pass on every fold. End users should leave this at the default FALSE.

Details

Coordinate scaling and anisotropy. Before fitting the GP, X and Y coordinates are each centered and divided by their own standard deviation (lines 110–111). Because the two axes are scaled independently, an isotropic squared-exponential kernel in the *scaled* space corresponds to

an **anisotropic** kernel in the original CRS: the effective length-scale in the X direction (in CRS units) differs from the Y direction whenever `sd(X) != sd(Y)`.

This per-axis standardization is deliberate — it stabilises the GP numerically when the coordinate extents differ dramatically (common in projected CRSs where easting and northing span very different ranges) — but users who expect the GP to be isotropic in geographic distance should be aware of this behaviour.

If true isotropy in the original CRS is desired, one could use a single scaling factor such as `max(sd(X), sd(Y))` for both axes. The stored `$info$coord_scaling` list includes a `scaling_type` element (`"anisotropic"`) so downstream code can detect which strategy was used.

Value

A `bayesian_fit` object (inherits from `spatial_fit`). Supports `predict()`, `fitted()`, `residuals()`, `coef()`, `summary()`, and `model_metrics()`. Model-specific metadata lives in `$info` (coord_scaling, predictor_scaling, gp_k, loo, looic, convergence_ok, convergence_diagnostics). The raw brmsfit is in `$engine`.

See Also

Other model fitting: `fit_gwr_model()`, `prep_model_data()`

Examples

```
## Not run:
if (requireNamespace("brms", quietly = TRUE)) {
  library(sf)
  set.seed(1)
  n <- 60
  dat <- st_as_sf(
    data.frame(x = runif(n, 0, 1000), y = runif(n, 0, 1000), elev = rnorm(n)),
    coords = c("x", "y"), crs = 32632
  )
  dat$price <- 10 + 0.01 * st_coordinates(dat)[, 1] + 2 * dat$elev + rnorm(n)
  fit <- fit_bayesian_spatial_model(dat, "price", "elev",
                                    chains = 2, iter = 500,
                                    compute_loo = FALSE)
  summary(fit)
  head(predict(fit, newdata = dat))
}

## End(Not run)
```

`fit_gwr_model` *Fit a Geographically Weighted Regression (GWR) via GWmodel*

Description

Fits a GWR using GWmodel on an sf dataset with either adaptive or fixed bandwidth.

Usage

```
fit_gwr_model(
  data_sf,
  response_var,
  predictor_vars,
  adaptive = TRUE,
  bandwidth = NULL,
  kernel = c("bisquare", "gaussian", "tricube", "boxcar", "exponential"),
  .already_prepped = FALSE
)
```

Arguments

data_sf	An sf object with response, predictors, and geometries.
response_var	Response column name.
predictor_vars	Predictor column names.
adaptive	Logical; use adaptive bandwidth. Default TRUE. When TRUE, bandwidth is an integer number of nearest neighbours. When FALSE, bandwidth is a fixed distance in CRS units.
bandwidth	Optional numeric bandwidth value. For adaptive mode this is an integer (number of neighbours); for fixed mode a distance in CRS units. If NULL (default), bandwidth is selected automatically via GWmodel::bw.gwr().
kernel	Kernel function type. One of "bisquare" (default), "gaussian", "tricube", "boxcar", "exponential".
.already_prepped	Logical (internal). If TRUE, skip the prep_model_data() call because the caller has already projected, coerced, and filtered the data. Used by the CV internals to avoid a redundant second pass on every fold. End users should leave this at the default FALSE.

Value

A gwr_fit object (inherits from spatial_fit). Supports predict(), fitted(), residuals(), coef(), summary(), and model_metrics(). Model-specific metadata lives in $info (bandwidth, adaptive, kernel, AICc). The raw GWmodel result is in $engine.

Collinearity diagnostics

The function checks the condition number of the predictor matrix and warns when it exceeds a threshold. A **global** condition number is computed on the full predictor matrix. In addition, a **local** spot-check is performed at a small random sample of locations: for each sampled point, the nearest neighbours within the bandwidth window are selected and the condition number of that local (weighted) design sub-matrix is evaluated. If the fraction of sampled locations with an extreme local condition number (> 1e6) exceeds 25\ separate warning is issued.

Because the local spot-check examines only a subset of locations (up to 30 by default), it may not detect every problematic neighbourhood. Users working with highly clustered data or near-collinear predictors should consider a full local-collinearity audit as a post-fit diagnostic.

See Also

Other model fitting: `fit_bayesian_spatial_model()`, `prep_model_data()`

Examples

```
if (requireNamespace("GWmodel", quietly = TRUE) &&
    requireNamespace("sp", quietly = TRUE)) {
  library(sf)
  set.seed(1)
  n <- 60
  dat <- st_as_sf(
    data.frame(x = runif(n, 0, 1000), y = runif(n, 0, 1000), elev = rnorm(n)),
    coords = c("x", "y"), crs = 32632
  )
  dat$price <- 10 + 0.01 * st_coordinates(dat)[, 1] + 2 * dat$elev + rnorm(n)
  fit <- fit_gwr_model(dat, "price", "elev", bandwidth = 30)
  summary(fit)
  head(predict(fit, newdata = dat))   # newdata is re-projected if needed
}
```

---

get_voronoi_seeds *Generate seed points for Voronoi tessellation*

Description

Creates an sf POINT layer of "seed" locations. Multiple strategies are supported: user-provided points, uniform random sampling within a boundary, or k-means clustering of a sampling cloud.

Usage

```
get_voronoi_seeds(
  boundary = NULL,
  method = c("kmeans", "random", "provided"),
  n = NULL,
  seeds = NULL,
  sample_points = NULL,
  kmeans_nstart = 10,
  kmeans_iter = 100,
  set_seed = NULL
)
```

Arguments

boundary	Optional polygonal sf object defining the sampling area.
method	One of "kmeans", "random", "provided".
n	Integer; number of seeds to return.

seeds	sf POINT object of user-provided seeds (method = "provided").
sample_points	Optional sf POINT cloud for k-means clustering.
kmeans_nstart	Integer; nstart for kmeans(). Default 10.
kmeans_iter	Integer; iter.max for kmeans(). Default 100.
set_seed	Optional integer RNG seed.

Value

An sf POINT object with seed_id and method columns.

See Also

Other tessellation: `build_tessellation()`, `create_grid_polygons()`, `create_voronoi_polygons()`, `plot_tessellation_map()`

Examples

```
library(sf)
bnd <- st_sf(geometry = st_sfc(st_polygon(list(rbind(
  c(0, 0), c(100, 0), c(100, 100), c(0, 100), c(0, 0)
))), crs = 32632))
get_voronoi_seeds(bnd, method = "random", n = 5, set_seed = 1)
```

---

gp_lengthscale_bounds *Heuristic length-scale bounds for a squared-exponential GP*

Description

Computes sensible prior bounds for the GP length-scale parameter ℓ of a squared-exponential (exponentiated-quadratic) kernel, $k(h) = \exp(-h^2/(2\ell^2))$. The "effective range" where correlation drops to ~5\ $\ell\sqrt{2\ln 20} \approx 2.45\,\ell$.

Usage

```
gp_lengthscale_bounds(coords_xy, q_small = 0.25, max_n = 1000L)

phi_prior_bounds(coords_xy, q_small = 0.25, max_n = 1000L)
```

Arguments

coords_xy	Numeric matrix of (x, y) coordinates.
q_small	Numeric quantile for the lower bound. Default 0.25. Previous versions used 0.1, but the 10th percentile can be dominated by within-cluster spacing in clustered data, producing a misleadingly small lower bound.
max_n	Maximum number of points to use in distance computation. Default 1000. Set to `Inf` to use all points.

Details

Subsamples large datasets to avoid O(n^2) memory and time cost.

`phi_prior_bounds` is a deprecated alias retained for backward compatibility. Previous versions computed bounds calibrated for an exponential covariance (`3 / d`); the current implementation delegates to `gp_lengthscale_bounds` which is calibrated for the squared-exponential kernel used by `brms::gp()`.

Value

Named numeric vector `c(lower, upper)` on the length-scale.

Examples

```
set.seed(1)
xy <- cbind(runif(50), runif(50))
gp_lengthscale_bounds(xy)
```

---

harmonize_crs *Harmonize CRS between two spatial objects*

Description

Aligns two sf objects to a common CRS.

Usage

```
harmonize_crs(
  a,
  b,
  prefer = c("a", "b"),
  target_crs = NULL,
  on_transform_error = c("stop", "set_crs")
)
```

Arguments

a, b	Objects of class sf or sfc.
prefer	Which object's CRS to keep ("a" or "b").
target_crs	Optional target CRS to apply to both.
on_transform_error	What to do when st_transform() fails: `"stop"` (default) raises an error immediately; `"set_crs"` falls back to st_set_crs() (UNSAFE — coordinates are NOT reprojected, only the CRS label is overwritten). The `"set_crs"` option exists only for rare edge cases where you are certain the coordinates already match the target CRS definition.

Value

A named list with components a and b.

`make_folds` *Create spatial cross-validation folds*

Description

Builds train/test splits using random K-fold, spatial block K-fold, or buffered leave-one-out strategies.

Usage

```
make_folds(
  points_sf,
  k,
  method = c("random_kfold", "block_kfold", "buffered_loo"),
  seed = NULL,
  block_nx = NULL,
  block_ny = NULL,
  block_multiplier = 3,
  block_size = NULL,
  auto_range = FALSE,
  response_var = NULL,
  predictor_vars = NULL,
  boundary = NULL,
  buffer = NULL,
  drop_empty_blocks = TRUE
)
```

Arguments

`points_sf`	An sf object.
`k`	Integer; number of folds.
`method`	One of "random_kfold", "block_kfold", "buffered_loo".
`seed`	Optional integer RNG seed.
`block_nx, block_ny`	Optional grid dimensions for block_kfold. Ignored when `block_size` or `auto_range` override them.
`block_multiplier`	Numeric; target blocks multiplier. Default 3.
`block_size`	Optional positive numeric minimum block edge length (in projected CRS units). When supplied, grid dimensions are clamped so that every block is at least this wide and tall. Takes precedence over `block_nx`/`block_ny` and `block_multiplier`.

auto_range	Logical. If TRUE, the spatial autocorrelation range is estimated via estimate_sac_range() — which fits directional variograms to account for anisotropy — and used as the minimum block_size. Requires response_var. An explicit block_size takes precedence. Default FALSE.
response_var	Character(1) response column name. Required when auto_range = TRUE.
predictor_vars	Optional character vector of predictor column names. Passed to estimate_sac_range() for residual variogram estimation.
boundary	Optional polygonal sf/sfc for block_kfold.
buffer	Positive numeric distance for buffered_loo.
drop_empty_blocks	Logical. Default TRUE.

Details

For block_kfold, the default grid sizing is purely geometric and unrelated to the autocorrelation range of the data. When blocks are smaller than the autocorrelation range, spatially correlated observations leak across folds and CV metrics become optimistic. Use block_size to set a minimum block edge length (in CRS units), or set auto_range = TRUE to estimate the range from an empirical variogram and enforce it automatically.

Value

A list with method, k, folds, assignment, params. The train/test elements of each fold contain ..row_id values (equal to row positions when the input has no pre-existing ..row_id column), consistent with the assignment tibble.

See Also

Other cross-validation: cv_bayes(), cv_gwr(), cv_spatial(), estimate_sac_range()

Examples

```
library(sf)
set.seed(1)
pts <- st_as_sf(
  data.frame(x = runif(30, 0, 1000), y = runif(30, 0, 1000)),
  coords = c("x", "y"), crs = 32632
)
folds <- make_folds(pts, k = 3, method = "block_kfold", seed = 42)
folds$assignment          # fold membership per row
lengths(folds$folds[[1]]) # train/test row-ID splits

# Buffered leave-one-out: neighbours within 100 units excluded from training
loo <- make_folds(pts, k = 1, method = "buffered_loo", buffer = 100)
```

model_metrics *Compute goodness-of-fit metrics for a spatial model*

Description

Compute goodness-of-fit metrics for a spatial model

Usage

```
model_metrics(object, ...)

## S3 method for class 'spatial_fit'
model_metrics(object, newdata = NULL, ...)
```

Arguments

object	A spatial_fit object.
...	Additional arguments passed to predict().
newdata	Optional sf object for out-of-sample evaluation. If NULL, in-sample (fitted) values are used.

Value

A data.frame with n, RMSE, MAE, MAPE, SMAPE, R2, Adj_R2.

new_spatial_fit *Build a spatial_fit S3 object*

Description

Low-level constructor used by fit_gwr_model() and fit_bayesian_spatial_model(). Users should not call this directly.

Usage

```
new_spatial_fit(
  subclass,
  engine,
  formula,
  response_var,
  predictor_vars,
  data_sf,
  info = list()
)
```

Arguments

subclass	Character scalar: "gwr_fit" or "bayesian_fit".
engine	The raw model object.
formula	A formula.
response_var	Character(1).
predictor_vars	Character vector.
data_sf	An sf object used for fitting.
info	Named list of model-specific extras.

Value

An object of class `c(subclass, "spatial_fit")`.

---

plot_tessellation_map *Plot a tessellation map with optional boundary, seeds, and features*

Description

Builds a layered ggplot2 map of polygon tessellations and optional overlays for a study boundary, seed points, and additional features.

Usage

```
plot_tessellation_map(
  tessellation_sf,
  boundary = NULL,
  seeds_sf = NULL,
  features_sf = NULL,
  fill_col = NULL,
  palette = "viridis",
  na_fill = "grey90",
  tile_alpha = 0.9,
  outline_col = "white",
  outline_size = 0.2,
  features_col = "#333333",
  features_size = 0.5,
  seeds_col = "#1f77b4",
  seeds_size = 1.5,
  boundary_col = "#111111",
  boundary_size = 0.6,
  labels = FALSE,
  label_col = "grid_id",
  label_size = 2.7,
  legend = TRUE,
  legend_title = NULL,
```

```
  theme = ggplot2::theme_void(),
  target_crs = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  xlim = NULL,
  ylim = NULL,
  expand = TRUE
)
```

Arguments

tessellation_sf	An sf POLYGON/MULTIPOLYGON layer. Required.
boundary	Optional sf/sfc polygon outline layer.
seeds_sf	Optional sf/sfc point layer of seed locations.
features_sf	Optional sf/sfc layer of additional features.
fill_col	Column name in tessellation_sf to map to fill. NULL = no fill.
palette	Viridis palette name. Default "viridis".
na_fill	Fill for NA values. Default "grey90".
tile_alpha	Alpha for filled polygons. Default 0.9.
outline_col, outline_size	Tessellation outline aesthetics.
features_col, features_size	Feature overlay aesthetics.
seeds_col, seeds_size	Seed point aesthetics.
boundary_col, boundary_size	Boundary outline aesthetics.
labels	Logical; draw per-cell labels. Default FALSE.
label_col	Column for label text. Default "grid_id".
label_size	Label text size. Default 2.7.
legend	Logical; show fill legend. Default TRUE.
legend_title	Optional legend title.
theme	A ggplot2 theme. Default theme_void().
target_crs	Optional CRS for plotting.
title, subtitle, caption	Plot annotations.
xlim, ylim	Optional numeric vectors of length 2 for coordinate limits (in the plot CRS). Default NULL (auto).
expand	Logical; expand plot area slightly beyond data limits. Default TRUE.

Value

A ggplot2 object.

See Also

Other tessellation: `build_tessellation()`, `create_grid_polygons()`, `create_voronoi_polygons()`, `get_voronoi_seeds()`

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(sf)
  set.seed(1)
  pts <- st_as_sf(
    data.frame(x = runif(20, 0, 100), y = runif(20, 0, 100)),
    coords = c("x", "y"), crs = 32632
  )
  tess <- build_tessellation(pts, method = "voronoi", quiet = TRUE)
  p <- plot_tessellation_map(tess$cells, features_sf = pts,
                             fill_col = "cell_id", legend = FALSE)
  p
}
```

`predict.bayesian_fit` *Predict from a Bayesian spatial GP model*

Description

Applies the same newdata preparation pipeline as `predict.gwr_fit()`: non-point geometries are coerced to points, the data is projected to the CRS used during fitting (via `ensure_projected()`), and rows with missing or non-finite values are dropped. Coordinate scaling and predictor standardisation stored at fit time are then applied before delegating to `brms::posterior_epred()` or `brms::posterior_predict()`.

Usage

```
## S3 method for class 'bayesian_fit'
predict(
  object,
  newdata = NULL,
  summary = c("mean", "median"),
  type = c("epred", "predict"),
  draws = FALSE,
  ...
)
```

Arguments

object	A bayesian_fit object.
newdata	An sf object with the same predictors. The response variable need not be present (true out-of-sample prediction is supported). NULL = fitted values.
summary	"mean" (default) or "median" over posterior draws.
type	"epred" (default) for expected predictions (no obs noise), or "predict" for full posterior predictive draws (includes obs noise).
draws	If TRUE, return the full posterior draw matrix instead of a point summary. Default FALSE.
...	Ignored.

Value

Numeric vector (or matrix when draws=TRUE).

---

predict.gwr_fit *Predict from a GWR spatial model*

Description

When newdata is NULL, returns the in-sample fitted values. Otherwise uses GWmodel::gwr.predict() on the new locations. newdata is first transformed to the CRS used during fitting (via ensure_projected()), so predictions are computed in a single coordinate system regardless of the CRS newdata arrives in.

Usage

```
## S3 method for class 'gwr_fit'
predict(object, newdata = NULL, ...)
```

Arguments

object	A gwr_fit object.
newdata	An sf object with the same predictors. The response variable need not be present (true out-of-sample prediction is supported). NULL = fitted values.
...	Ignored.

Value

Numeric vector of predictions.

prep_model_data *Prepare and sanitize an sf dataset for spatial modeling*

Description

Ensures point geometry, projected CRS, and removes rows with missing or non-finite values in modeling columns. All non-POINT geometries (including MULTIPOINT) are coerced to representative points via `coerce_to_points()`, so downstream coordinate extraction always aligns one row per observation.

Usage

```
prep_model_data(
  data_sf,
  response_var,
  predictor_vars,
  boundary = NULL,
  pointize = c("auto", "surface", "centroid", "line_midpoint", "bbox_center"),
  require_response = TRUE
)
```

Arguments

`data_sf`	An sf object.
`response_var`	Response variable column name.
`predictor_vars`	Predictor column names.
`boundary`	Optional sf/sfc for CRS alignment.
`pointize`	Strategy for non-point geometry coercion.
`require_response`	Logical; if FALSE the response column is not required to be present (useful for out-of-sample prediction where the response is unknown). Default TRUE.

Value

An sf object (points) in a projected CRS, cleaned.

See Also

Other model fitting: `fit_bayesian_spatial_model()`, `fit_gwr_model()`

Examples

```
library(sf)
dat <- st_as_sf(
  data.frame(x = 1:5, y = 5:1,
             resp = c(1, 2, NA, 4, 5),
```

```
            pred = c(1, 2, 3, 4, Inf)),
  coords = c("x", "y"), crs = 32632
)
prep_model_data(dat, "resp", "pred")  # drops rows 3 (NA) and 5 (Inf)
```

`residual_morans_i` *Compute Moran's I on the residuals of a fitted spatial model*

Description

Given a `spatial_fit` object (GWR or Bayesian), extracts the residuals and the observation coordinates, builds a spatial weight matrix, and computes the Moran's I statistic together with its analytical expectation and variance under the randomisation assumption (Cliff & Ord). A z-score and two-sided p-value are provided so the caller can assess whether statistically significant spatial autocorrelation remains after fitting.

Usage

```
residual_morans_i(
  fit,
  alternative = c("two.sided", "greater", "less"),
  weights = NULL,
  k = 8L
)
```

Arguments

`fit`	A `spatial_fit` object (from `fit_gwr_model` or `fit_bayesian_spatial_model`).
`alternative`	Character: `"two.sided"` (default), `"greater"` (positive autocorrelation), or `"less"`.
`weights`	Optional user-supplied n x n weight matrix — a base matrix or a **Matrix**-package matrix (e.g. a sparse dgCMatrix). When `NULL` (the default), a k-nearest-neighbour binary weight matrix (k = 8, row-standardised) is built from the observation coordinates. If a non-row-standardised matrix is supplied (i.e. rows do not all sum to 1), the Cliff & Ord variance formula is still valid for general W and the computation proceeds, but a warning is emitted because the magnitude of I is not directly comparable to results obtained with row-standardised weights.
`k`	Integer number of nearest neighbours used when building the default weight matrix (ignored when `weights` is supplied). Default 8.

Details

By default, weights are constructed as a k-nearest-neighbour (k = 8) binary matrix, row-standardised. Users may supply their own weight matrix via the `weights` argument.

Value

A list with components:

observed Numeric scalar, Moran's I statistic.

expected Expected I under the null of no spatial autocorrelation, $-1/(n-1)$.

sd Standard deviation of I under the randomisation assumption.

z Standardised z-score, $(I - E[I])/sd(I)$.

p_value Two-sided (or one-sided) p-value from the normal approximation.

n Number of observations used.

Returns `NULL` with a warning if computation fails (e.g. fewer than 4 valid residuals).

See Also

Other model evaluation: `compare_models()`, `compare_models_cv()`, `evaluate_insample()`

Examples

```
if (requireNamespace("GWmodel", quietly = TRUE) &&
    requireNamespace("sp", quietly = TRUE)) {
  library(sf)
  set.seed(1)
  n <- 60
  dat <- st_as_sf(
    data.frame(x = runif(n, 0, 1000), y = runif(n, 0, 1000), elev = rnorm(n)),
    coords = c("x", "y"), crs = 32632
  )
  dat$price <- 10 + 0.01 * st_coordinates(dat)[, 1] + 2 * dat$elev + rnorm(n)
  fit <- fit_gwr_model(dat, "price", "elev", bandwidth = 30)
  residual_morans_i(fit)  # z near 0 / p large = no residual structure
}
```

`summarize_by_cell` *Summarize features by polygon/cell ID*

Description

Aggregates an sf point dataset into one row per cell. By default computes counts and means, but the aggregation function is configurable.

Usage

```
summarize_by_cell(
  assigned_points_sf,
  response_var = NULL,
  predictor_vars = NULL,
  id_col = "poly_id",
  agg_funs = list(mean = function(x) mean(x, na.rm = TRUE)),
  cells_sf = NULL,
  deff = 1,
  quiet = TRUE
)
```

Arguments

`assigned_points_sf`
: An sf object with a cell identifier column.

`response_var`
: Optional response column name for per-cell aggregation.

`predictor_vars`
: Optional predictor column names for per-cell aggregation.

`id_col`
: Preferred name of the polygon/cell ID column.

`agg_funs`
: Named list of aggregation functions. Default `list(mean = \(x) mean(x, na.rm = TRUE))`. Additional common options: `median`, `sum`, `sd`.

`cells_sf`
: Optional polygon sf layer to join cell geometries onto the output. When supplied, the return value is an sf object with the polygon geometry from cells_sf. When NULL (default), a plain data.frame/tibble is returned (previous behaviour).

`deff`
: Design-effect adjustment for standard errors. One of:

  `1` **(default)** No adjustment; classic IID standard error. Equivalent to previous behaviour but now emits a message (when `quiet = FALSE`) reminding that SEs assume independence.

  `"kish"` Estimate per-variable-type intra-class correlations (ICCs) from the grouped data using a one-way random-effects ANOVA decomposition — one ICC for the response variable and a separate ICC for the predictor variables — then apply Kish's formula per cell: `deff_i = 1 + (n_i - 1) * rho`. When multiple columns are pooled for a single ICC estimate (e.g. several predictor variables), each column is z-scored before pooling so that variables with different scales contribute equally to the variance decomposition. The response-specific ICC is used for response SEs and the predictor-specific ICC for predictor SEs. Requires at least 2 cells with 2+ observations; falls back to `deff = 1` otherwise.

  A positive number Applied as a uniform design effect to every cell. Use when you have an external estimate of the design effect.

`quiet`
: Logical; suppress messages. Default TRUE.

Details

In addition to user-specified aggregation functions, this function always computes within-cell standard deviation (`..sd_<var>`) and standard error (`..se_<var>`) for every numeric response/predictor

column, plus a `cell_weight` column equal to the observation count. These columns let downstream models account for the fact that a cell with 2 observations carries more aggregation uncertainty than one with 200.

Value

A tibble/data.frame (or sf if cells_sf given) with per-cell summaries including `n`, `cell_weight`, and `..sd_*` / `..se_*` columns. When `deff != 1`, an attribute `"deff_applied"` is attached to the result recording the design effect(s) used.

Spatial autocorrelation and standard-error bias

Important: By default (`deff = 1`), the `..se_*` columns are computed as `sd / sqrt(n)`, which assumes observations within each cell are independent. When data are spatially autocorrelated — the common case for the spatial workflows this package supports — within-cell observations are typically positively correlated, so the effective sample size is smaller than `n`. The naive SE is therefore **anticonservative** (too small), and downstream weighted regressions using `cell_weight` or `..se_*` columns will produce overconfident standard errors for cells with strong intra-cell correlation.

Setting `deff = "kish"` applies an approximate correction using Kish's design effect. Separate intra-class correlations (ICCs) are estimated for response and predictor variables via a one-way random-effects decomposition across all cells. Each variable type's ICC is used for its own SE adjustment, and each cell's effective sample size is reduced to `n_i / (1 + (n_i - 1) * rho)`. This is a first-order correction that does not require a full spatial covariance model but does require enough cells and observations for a stable ICC estimate. You may also pass a fixed numeric design effect (e.g. `deff = 2`) to uniformly inflate standard errors.

Even with the Kish correction, the adjusted SE is an approximation. For rigorous inference under spatial dependence, consider fitting an explicit spatial covariance model (e.g. via `fit_bayesian_spatial_model`).

Examples

```
library(sf)
set.seed(1)
pts <- st_as_sf(
  data.frame(x = runif(40, 0, 100), y = runif(40, 0, 100), val = rnorm(40)),
  coords = c("x", "y"), crs = 32632
)
bnd <- st_sf(geometry = st_sfc(st_polygon(list(rbind(
  c(0, 0), c(100, 0), c(100, 100), c(0, 100), c(0, 0)
))), crs = 32632))
grid <- create_grid_polygons(bnd, target_cells = 9, type = "square")
assigned <- assign_features_to_polygons(pts, grid)

# IID standard errors (default) vs Kish design-effect adjustment
cells <- summarize_by_cell(assigned, response_var = "val", deff = "kish")
cells
attr(cells, "deff_applied")
```

voronoi_seeds_kmeans *K-means seed generation from point coordinates*

Description

K-means seed generation from point coordinates

Usage

```
voronoi_seeds_kmeans(points_sf, k, set_seed = 456)
```

Arguments

points_sf	An sf object with POINT geometries.
k	Integer; requested number of clusters.
set_seed	Optional integer RNG seed. Default 456.

Value

An sf object of k cluster center POINTs.

voronoi_seeds_random *Random seed generation within a polygonal boundary*

Description

Random seed generation within a polygonal boundary

Usage

```
voronoi_seeds_random(boundary, k, set_seed = 456)
```

Arguments

boundary	An sf or sfc polygonal object.
k	Integer; number of random seeds.
set_seed	Integer RNG seed. Default 456.

Value

An sf object of k random POINTs.

Index