Package: terradish (via r-universe)

November 12, 2026

Type Package

Title Fast Gradient-Based Optimization of Resistance Surfaces

Version 1.1.0

Date 2026-10-28

Language en-US

Description Maximum likelihood estimation of landscape conductance surfaces from genetic data on raster graphs, using Newton and quasi-Newton optimization with analytic derivatives. Fits log-linear conductance models with optional joint estimation of Gaussian smoothing scales and spline response shapes, linked to pairwise genetic distances through maximum likelihood population effects models or to allele-frequency covariance through a Wishart likelihood on site contrasts, with the same pairwise distance and environment covariates available in both. Provides exact sparse and iterative solvers for large rasters, fixed-domain spatial cross-validation for comparing conductance formulas, and simulation tools for study design. Works with 'terra' objects. Fitting one parameterized conductance surface to genetic data, rather than regressing on separate resistance distances, follows Peterman and Pope (2021) <doi:10.1111/mec.15716>. Background on population-effects models and circuit-based covariance models is provided by Clarke, Rothery and Raybould (2002) <doi:10.1198/108571102320> and Hanks and Hooten (2013) <doi:10.1080/01621459.2012.724647>, respectively.

URL https://github.com/wpeterman/terradish, https://doi.org/10.5281/zenodo.21225711

BugReports https://github.com/wpeterman/terradish/issues

License BSD_3_clause + file LICENSE

Depends R (>= 4.3.0)

Imports ggplot2, grDevices, landgraph (>= 0.0.3), MASS, Matrix (>= 1.6-2), methods, multiScaleR, nlme, parallel, Rcpp (>= 1.0.4), splines, stats, terra, tools, utils

LinkingTo Matrix (>= 1.6-2), Rcpp, RcppArmadillo, RcppEigen

Suggests adegenet, knitr, numDeriv, patchwork, pkgload, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Encoding UTF-8

Config/testthat/edition 3

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 2026-10-29 13:53:10 UTC

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

RemoteRef HEAD

RemoteSha 960db5a5e42c5e5245d667e40cc82cb44fd8dc20

Contents

aic_table . . . 3
ArmijoControl . . . 5
check_distance_response . . . 6
conductance . . . 8
conductance_surface . . . 10
covariance_response_power . . . 13
crop_to_focal_buffer . . . 16
gaussian_scale_profile . . . 18
gaussian_scale_summary . . . 19
gaussian_smoothed_loglinear_conductance . . . 21
generalized_wishart . . . 24
HagerZhangControl . . . 27
individual_salamanders . . . 29
leastsquares . . . 31
loglinear_conductance . . . 33
lower . . . 35
melip . . . 36
mlpe . . . 38
mlpe_covariates . . . 41
mlpe_response_change . . . 43
NewtonRaphsonControl . . . 45
ohio_ponds . . . 47
pairwise_covariates . . . 49
pairwise_endpoint_covariates . . . 50
pca_dist . . . 52
plot.terradish . . . 53
print.terradish_covariance_power . . . 57

print.terradish_setting_assessment . . . 58
scale_covariates . . . 60
scale_to_0_1 . . . 61
simulate_covariance_response . . . 62
slim_terradish . . . 64
smooth_loglinear_conductance . . . 65
terradish . . . 67
terradish_algorithm . . . 74
terradish_assess_settings . . . 78
terradish_conductance_model_factory . . . 81
terradish_cv_folds . . . 82
terradish_distance . . . 84
terradish_folds . . . 86
terradish_grid . . . 87
terradish_ibe_ratio . . . 89
terradish_inference . . . 90
terradish_measurement_model . . . 92
terradish_methods . . . 92
terradish_rescale_nu . . . 95
terradish_scale_optim . . . 96
terradish_solver_benchmark . . . 99
wishart_covariance . . . 101
wishart_covariates . . . 104

Index **108**

`aic_table` *Rank fitted terradish models by information criterion*

Description

Creates a model-selection table from fitted `terradish` models using AIC, AICc, or BIC.

Usage

```
aic_table(
  mod_list,
  AICc = FALSE,
  BIC = FALSE,
  mod_names = NULL,
  verbose = FALSE
)
```

Arguments

mod_list	List of fitted terradish models.
AICc	Should second-order AIC (Akaike's information criterion with small-sample correction) be used instead of AIC? When TRUE, the correction uses n = number of focal sampling sites (not the number of pairwise observations). A common guideline is to prefer AICc over AIC when $n/K < 40$, where K is the number of estimated parameters.
BIC	Should BIC be used instead of AIC? When TRUE, the penalty uses n = number of focal sampling sites (individuals or populations), the same sample size as AICc, not the number of pairwise observations. Pairs that share a site are not independent, so the number of pairs overstates the information in the data.
mod_names	Optional model names. By default the right-hand side of each fitted formula is used. For MLPE measurement models with additional pairwise covariates, the default appends [mlpe:n], where n is the number of added pairwise covariate columns.
verbose	Should the table be printed to the console?

Details

At nominal Wishart information, information criteria can favor extensions that are unsupported across replicate histories. Use terradish_cv_folds to select conductance terms and terradish_rescale_nu to assess inferential sensitivity.

Information-criterion comparison requires the same observed response, focal sites, graph domain, and likelihood family. Wishart fits must also use the same effective degrees of freedom, nu. leastsquares and mlpe are Gaussian likelihoods for the same pairwise-distance response and can be ranked when those conditions hold. generalized_wishart and wishart_covariance use different response representations and must not be ranked together. Cross-family rankings, such as mlpe versus generalized_wishart, are invalid.

The function checks response values, model family, fitted dimensions, recorded nu, and retained graph signatures. Older fits without signatures require the user to verify the graph domain.

Value

A data frame containing model ranks, parameter counts, information criterion values, delta values, weights, cumulative weights, and log-likelihoods. The converged column is true only when both the outer optimizer and final nuisance optimization record code 0. Missing convergence diagnostics in older objects are treated as unverified (false); boundary marks active parameter bounds or no resistance structure. A warning identifies tables containing nonconverged or boundary fits.

Worked example

See vignette("model-comparison", package = "terradish") for compatible model rankings, coefficient plots, and predictive comparisons.

Examples

```
library(terra)

data(melip)
melip.altitude <- terra::unwrap(melip.altitude)
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords <- terra::unwrap(melip.coords)

keep <- 1:10
melip.Fst_small <- melip.Fst[keep, keep]
# Coarsen the rasters so this example remains quick on CRAN.
covariates <- terra::aggregate(c(melip.altitude, melip.forestcover),
                               fact = 3, na.rm = TRUE)
covariates <- scale_covariates(covariates)
names(covariates) <- c("altitude", "forestcover")
surface_small <- conductance_surface(covariates, melip.coords[keep], directions = 8)

fit1 <- terradish(melip.Fst_small ~ altitude, surface_small,
                  loglinear_conductance, leastsquares,
                  control = NewtonRaphsonControl(maxit = 2, verbose = FALSE))
fit2 <- terradish(melip.Fst_small ~ altitude + forestcover, surface_small,
                  loglinear_conductance, leastsquares,
                  control = NewtonRaphsonControl(maxit = 2, verbose = FALSE))

aic_table(list(fit1, fit2))
aic_table(list(fit1, fit2), AICc = TRUE)
```

`ArmijoControl` *Control settings for Armijo backtracking line search*

Description

Tuning parameters for an objective-only Armijo line search. This line search is currently intended for `optimizer = "bfgs"` fits where trial objective evaluations are cheaper than repeated full objective-plus-gradient evaluations during Hager-Zhang line search.

Usage

```
ArmijoControl(
  initial = 1,
  contraction = 0.5,
  sufficient_decrease = 1e-04,
  maxit = 25L,
  min_alpha = sqrt(.Machine$double.eps),
  verbose = FALSE
)
```

Arguments

initial	Initial trial step length.
contraction	Multiplicative shrinkage applied after rejected trial steps. Must be strictly between 0 and 1.
sufficient_decrease	Armijo sufficient-decrease constant.
maxit	Maximum number of backtracking steps.
min_alpha	Smallest step length to try before stopping.
verbose	Should line-search progress be printed?

Details

`ArmijoControl()` is opt-in. The default optimizer control continues to use `HagerZhangControl`. Armijo trial steps only evaluate the objective, so rejected steps avoid the gradient/backpropagation work required by Hager-Zhang. After a step is accepted, BFGS still evaluates the full gradient once at the accepted point so the quasi-Newton update remains well-defined.

Value

A list of class `"terradish_armijo_control"` holding the validated Armijo line-search settings (`type`, `initial`, `contraction`, `sufficient_decrease`, `maxit`, `min_alpha`, and `verbose`), suitable for the `ls.control` element of a `NewtonRaphsonControl` object.

Examples

```
ctrl <- NewtonRaphsonControl(
  maxit = 25,
  verbose = FALSE,
  ls.control = ArmijoControl()
)
str(ctrl$ls.control)
```

---

check_distance_response

Check a generalized-Wishart distance response

Description

Tests whether a matrix supplied as the response to `generalized_wishart` has squared-Euclidean geometry. The check evaluates the positive semidefiniteness of the centered Gram matrix implied by the observed response. It does not assess whether the chosen genetic summary is biologically appropriate for a particular study.

Usage

```
check_distance_response(S, tol = 1e-07)
```

Arguments

`S`	Numeric square matrix. Entries are interpreted as the **squared-distance response** D that would be supplied to `generalized_wishart`, not as ordinary distances d. If you have ordinary distances, check `S^2` and supply that squared matrix to the model.
`tol`	Positive numeric scalar. Relative tolerance used for structural checks and for classifying negative eigenvalues. An eigenvalue is treated as numerical zero when it is no smaller than $-\mathrm{tol}\,\lambda_{\max}$, where $\lambda_{\max}$ is the largest eigenvalue of the centered Gram matrix.

Details

For n sites, the function forms

$$B = -\frac{1}{2}HDH, \qquad H = I - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathsf{T}}.$$

The response is admissible when B is positive semidefinite within the requested tolerance. Symmetry, nonnegative entries, and a zero diagonal are necessary but are not sufficient. Passing this geometry check does not create a Wishart sampling model. Use covariance-derived squared distances with a defensible effective information parameter. F_{ST} ratio estimators have no Wishart degrees of freedom merely because their distance matrix passes the check; consider `mlpe` for those responses.

This function checks the matrix that you actually intend to analyze because scaling, transformations, locus aggregation, and missing-data handling can change its geometry. It never applies Cailliez, Lingoes, nearest-Euclidean, or other corrections. Those procedures alter the observed response and may alter its biological interpretation.

For an independent check with **ade4**, note that `ade4::is.euclid()` expects ordinary distances and squares them internally. For an already squared response `S`, the corresponding call is `ade4::is.euclid(as.dist(sqrt(S)))` not `ade4::is.euclid(as.dist(S))`.

Value

A list with the following elements:

admissible Logical. `TRUE` when the centered Gram matrix is positive semidefinite within `tol`.

min_eigenvalue Smallest eigenvalue of the centered Gram matrix, on the squared-distance scale of `S`.

relative_min_eigenvalue The smallest eigenvalue divided by the largest eigenvalue. Values close to zero from below usually reflect numerical rounding; materially negative values indicate incompatibility.

n_negative Number of eigenvalues below the scaled negative tolerance.

tolerance The relative tolerance supplied in `tol`.

eigenvalue_threshold The absolute negative-eigenvalue magnitude used for the decision, equal to `tol * max_eigenvalue`.

max_eigenvalue Largest eigenvalue of the centered Gram matrix.

n_sites Number of rows and columns in `S`.

See Also

generalized_wishart, wishart_covariance, mlpe

Examples

```
# Squared Euclidean distances from known coordinates are admissible.
xy <- matrix(c(0, 0,
               1, 0,
               0, 1,
               1, 1), ncol = 2, byrow = TRUE)
D <- as.matrix(dist(xy))^2
check_distance_response(D)

# Symmetry and a zero diagonal alone are not sufficient.
bad <- matrix(c(0, 1, 1,
                1, 0, 9,
                1, 9, 0), nrow = 3, byrow = TRUE)
check_distance_response(bad)$admissible
```

---

conductance *Estimate conductance from a fitted model*

Description

Extracts per-cell conductance values and pointwise confidence intervals from a fitted terradish model, returning either a raster (when the graph stores its stack) or a matrix.

Usage

```
conductance(x, ...)

## S3 method for class 'terradish_graph'
conductance(
  x,
  fit,
  quantile = 0.95,
  support = c("none", "focal"),
  support_probs = c(0, 1),
  clamp_covariates = NULL,
  ...
)
```

Arguments

x	A terradish_graph object returned by conductance_surface.
...	Additional arguments passed to methods.

fit	A fitted model returned by terradish.
quantile	Confidence level for the pointwise conductance interval. The default 0.95 gives a 95 applied to the MLE of theta and its inverse Hessian.
support	Optional support constraint for conductance evaluation. Use "none" (default) to evaluate the fitted conductance model over the full graph covariate domain, or "focal" to clamp selected covariates to focal-site support.
support_probs	Optional length-2 probability vector used with support = "focal" to compute clamping bounds from focal-cell quantiles. The default c(0, 1) uses the focal min/max.
clamp_covariates	Optional character vector of covariate names to clamp when support = "focal". Defaults to all numeric covariates in x$x.

Details

Confidence intervals are computed on the log-conductance scale and then exponentiated, so the point estimate always lies within the interval. The intervals are asymptotic (delta method) and reflect only uncertainty in the conductance parameters theta; they do not account for uncertainty in the nuisance parameters phi.

With support = "focal", selected covariates are clamped to focal-site support before conductance is evaluated. This constrains prediction to empirical covariate support among x$demes and can stabilize extreme tails when focal sampling is sparse.

A no-structure fit leaves conductance coefficients unidentified and cannot produce a fitted conductance surface. Other nuisance-parameter boundaries do not by themselves prevent prediction.

On a new graph, prediction reuses the fitted log-linear terms, spline knots and centers, or Gaussian smoothing parameters and post-smoothing scaling. It does not estimate new centers or scales on the prediction landscape. If the original inputs were scaled before fitting, use scale_covariates(new_rasters, reference = original_scaled_rasters) before constructing the new graph. Other conductance factories are not supported for prediction to a new graph.

Value

If x$stack is non-NULL (the default when saveStack = TRUE in conductance_surface), a three-layer terra::SpatRaster with layer names:

est Point-estimate conductance at each active cell.

lower<Q> Lower bound of the quantile*100% interval (e.g. lower95 for the default quantile = 0.95).

upper<Q> Upper bound of the interval.

Otherwise, a numeric matrix with one row per active cell and columns est, lower<Q>, and upper<Q>.

See Also

terradish, conductance_surface, plot.terradish

Examples

```
library(terra)

data(melip)
melip.altitude <- terra::unwrap(melip.altitude)
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords <- terra::unwrap(melip.coords)

covariates <- c(terra::scale(melip.altitude),
                terra::scale(melip.forestcover))
names(covariates) <- c("altitude", "forestcover")
surface <- conductance_surface(covariates, melip.coords, directions = 8)

fit <- terradish(melip.Fst ~ forestcover + altitude, surface,
             loglinear_conductance, leastsquares,
             control = NewtonRaphsonControl(maxit = 5))

cond <- conductance(surface, fit)
names(cond)   # "est", "lower95", "upper95"
plot(cond)
```

`conductance_surface` *Create a parameterized conductance surface*

Description

Reads spatial covariates and focal-point coordinates and builds a weighted graph suitable for computing conductance-weighted resistance distances. This is the first step of every `terradish` workflow.

Usage

```
conductance_surface(
  covariates,
  coords,
  directions = 8,
  saveStack = TRUE,
  crop_buffer = NULL
)
```

Arguments

`covariates`	A `SpatRaster` (or `PackedSpatRaster`) whose layers contain the spatial covariates to be used in the conductance model. Layer names must match the right-hand side of the formula passed to `terradish`. Scale continuous layers with `scale_covariates` before calling this function.

`coords` Locations of the focal sampling sites, in the same coordinate reference system as `covariates`. Supported inputs: `terra::SpatVector` of points, a two-column numeric matrix (x in column 1, y in column 2), or a data frame with columns named `x` and `y` (or at least two columns, using the first two).

`directions` Adjacency rule for the graph. `4` connects each cell to its four horizontal/vertical neighbors (rook adjacency); `8` additionally includes the four diagonal neighbors (queen adjacency). `8` **is recommended** for landscape genetics: it allows diagonal movement, produces smoother resistance surfaces, and is less sensitive to grid orientation artifacts.

`saveStack` Logical. If `TRUE` (default), the masked `SpatRaster` is stored inside the returned `terradish_graph` object. Required for raster-based conductance visualization (`plot(fit, type = "surface")`) and marginal-association plots. Set to `FALSE` to reduce memory use when the raster is large and visualization is not needed.

`crop_buffer` Optional nonnegative map-unit buffer to crop `covariates` around the focal coordinates before graph construction. When supplied, `crop_to_focal_buffer` is called internally. A scalar applies the same buffer in both x and y directions; a length-two vector gives `c(x_buffer, y_buffer)`. Units are the raster's map units (degrees for lon/lat, meters for projected rasters). See Details.

Details

Missing values: `NA` is propagated uniformly across all layers so every cell is either complete or entirely missing. A warning is issued if some cells are `NA` in only a subset of layers, because this indicates that different covariates cover different areas; model comparison across raster stacks with different `NA` patterns is problematic since the graph vertices differ. Always compare models using the same `terradish_graph` object.

Disconnected components: after masking, the graph may contain cells that are isolated from the main component. The function retains only the largest connected component and warns about the number of pruned cells. The function aborts if any focal site belongs to a disconnected component.

Cropping: when `crop_buffer` is supplied, cropping changes the graph domain. Always use the same `terradish_graph` for all models you intend to compare. A buffer that is too tight may cut off landscape corridors that affect resistance distances; check stability by refitting with several buffer values.

Categorical covariates: raster layers representing categorical variables must be stored as factor-valued `SpatRaster` layers (see `as.factor` and the example below). Level names are taken from the `VALUE` column of the levels table when present; otherwise the first non-`ID` column is used. Factor levels are dummy-coded by `model.matrix` during model fitting, with one level as the reference.

Directions: `directions = 4` gives a more conservative graph that can underestimate resistance in diagonal corridors. `directions = 8` is preferred unless there is a specific reason to restrict movement to the cardinal directions. Eight-neighbor adjacency is the default. Each edge has conductance $c_i + c_j$; diagonal edges receive no additional distance weighting. Duplicate focal cells trigger a warning. Focal sites on pruned disconnected components cannot be fitted on the retained graph.

Cropping removes possible movement paths and can inflate resistance. The supplied release audit measured increases of 15–23 percent with a two-cell buffer and 1–2 percent with ten cells. These are case-specific results; compare a larger-buffer fit before interpreting a cropped landscape.

Value

An object of class "terradish_graph" (also inheriting "radish_graph") containing:

demes Integer indices of cells corresponding to focal sites.

x Data frame of covariate values at non-missing cells.

adj Edge list (upper-triangular, 0-based).

edge_pairs Integer matrix of adjacent cell pairs.

vertex_coordinates xy-coordinate matrix of active cells.

directions The adjacency rule (4 or 8).

covariates Names of the covariate columns.

stack The masked SpatRaster (or NULL if saveStack = FALSE).

Examples

```
library(terra)

data(melip)
melip.altitude <- terra::unwrap(melip.altitude)
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords <- terra::unwrap(melip.coords)

covariates <- c(terra::scale(melip.altitude),
                terra::scale(melip.forestcover))
names(covariates) <- c("altitude", "forestcover")

surface <- conductance_surface(covariates, melip.coords, directions = 8)

# For large rasters where sites occupy only part of the map, crop first.
surface_cropped <- conductance_surface(
  covariates,
  melip.coords,
  directions = 8,
  crop_buffer = 0.05
)

# categorical covariates:
# raster layers of categorical covariates should be factor-valued, see 'details'
forestcover_class <- cut(terra::values(melip.forestcover)[,1], breaks = c(0, 1/6, 1/3, 1))
melip.forestcover_cat <- terra::setValues(melip.forestcover, as.numeric(forestcover_class))
melip.forestcover_cat <- terra::as.factor(melip.forestcover_cat)

RAT <- levels(melip.forestcover_cat)[[1]]
RAT$VALUE <- levels(forestcover_class) # explicitly define level names
levels(melip.forestcover_cat) <- RAT

covariates_cat <- c(melip.forestcover_cat, melip.altitude)
names(covariates_cat) <- c("forestcover", "altitude")
```

covariance_response_power *Assess covariance-response parameter recovery by simulation*

Description

Runs a fast simulation study for covariance-response `terradish` fits. Candidate sampling designs are created by subsetting the focal sites in an existing `conductance_surface` object, responses are generated with `simulate_covariance_response`, and one or more candidate conductance models are refit to each simulated covariance matrix. Gaussian smoothing widths are simulated and reported in map units.

Usage

```
covariance_response_power(
  theta,
  formula,
  data,
  sample_sizes,
  strategies = c("spacefill", "random"),
  conductance_model = loglinear_conductance,
  fit_models = NULL,
  tau = 1,
  sigma = 0,
  nu,
  nsim = 20L,
  n_designs = 1L,
  seed = NULL,
  theta_scale = c("external", "internal"),
  effect_parameters = NULL,
  alpha = 0.05,
  conductance_cor_threshold = 0.9,
  optimizer = c("auto", "newton", "bfgs"),
  control = NewtonRaphsonControl(maxit = 8, verbose = FALSE),
  solver = c("auto", "direct", "amg"),
  solver_control = NULL,
  approximation = c("none", "landmark", "coarse_raster"),
  approximation_control = NULL,
  nonnegative = TRUE,
  leverage = FALSE,
  cores = 1L,
  verbose = FALSE,
  nu_fit = nu
)
```

Arguments

theta	True conductance parameters. By default these are interpreted on the same external scale returned by coef for fitted terradish models. For most conductance models the internal and external scales are identical. For gaussian_smoothed_loglinear_conductance the external scale reports sigma.* parameters in map units.
formula	Formula for the true conductance model. The left hand side, if supplied, is ignored.
data	A conductance_surface object containing all candidate focal sites.
sample_sizes	Integer vector of focal sample sizes to assess.
strategies	Sampling strategies. Supported values are "spacefill", "random", and "sequential".
conductance_model	Conductance-model factory used to simulate the true response.
fit_models	Optional named list of candidate models to refit. Each element may contain formula, conductance_model, theta, optimizer, control, solver, solver_control, approximation, approximation_control, nonnegative, and leverage. When omitted, the true model is refit.
tau	Nonnegative scaling applied to the conductance-implied covariance matrix during simulation.
sigma	Nonnegative nugget variance added to the covariance diagonal during simulation.
nu	Effective Wishart information used to simulate the response. Larger values reduce simulated sampling noise. The fitting value defaults to it through nu_fit = nu; neither is an estimate of information from real data. Report a sensitivity analysis rather than substituting the marker count. Simulation requires nu to be at least the largest assessed covariance-matrix dimension.
nsim	Number of covariance-response simulations per sampling design.
n_designs	Number of independent site designs per sample size for strategy = "random". Deterministic strategies are evaluated once.
seed	Optional random seed for reproducible site and response simulation.
theta_scale	Is theta supplied on the "external" or "internal" conductance-model scale?
effect_parameters	Optional character vector of parameters used in the all-effects recovery summary. Defaults to the nonzero true parameters.
alpha	Significance level for Wald-style parameter recovery summaries.
conductance_cor_threshold	Threshold used for the conductance-surface recovery summary.
optimizer, control, solver, solver_control, approximation, approximation_control, nonnegative, leverage, cores	Defaults passed to terradish for candidate models unless a model-specific value is supplied in fit_models.
verbose	Logical; print progress messages?
nu_fit	Effective Wishart degrees of freedom used to fit each simulation. Defaults to the generating nu. Increasing this value narrows fitted intervals without adding independent information. Compare coverage as well as power when exploring this sensitivity.

Details

This helper avoids forward-time genetic simulation. It assumes a proposed conductance surface and draws covariance responses from the same Wishart likelihood fitted by `wishart_covariance`. That makes it useful for approximate power and sample-size screening before investing in slower individual-based simulations.

Sampling designs are efficient because the raster graph is not rebuilt for each sample size. The helper subsets `data$demes` and the corresponding right-hand-side columns, so all candidate designs share the same graph, covariate matrix, and conductance-model basis expansion.

Fits that land on the no-signal boundary are retained as non-detections in parameter-level power summaries. Their coefficient estimates and standard errors are reported as `NA`, matching explicit fitting failures.

For spline conductance models, individual basis coefficients are not usually the most interpretable recovery target. Inspect `parameter_summary` for basis-level behavior, but prefer `summary$conductance_power`, `summary$mean_conductance_cor`, and model-selection rates when judging whether the non-linear conductance surface was recovered.

Value

An object of class `"terradish_covariance_power"` containing:

`results` One row per fitted model and simulated response.

`summary` Scenario-level fit, model-selection, and conductance-recovery summaries.

`parameter_results` One row per assessed parameter and simulation replicate.

`parameter_summary` Parameter-level power, bias, RMSE, and coverage summaries.

`settings` Simulation settings and true parameters, including `nu` for simulation and `nu_fit` for inference.

Power and coverage are internal to the assumed response model. Matching simulation and fitting values cannot diagnose incorrect information in a real data set. Nonconverged fits count as failed fits and non-detections; both outer and final nuisance optimization must record code 0. Missing convergence diagnostics also count as failed fits. Inspect `fit_rate` before interpreting power or coverage.

Examples

```
data(melip)
melip.altitude <- terra::unwrap(melip.altitude)
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords <- terra::unwrap(melip.coords)

covariates <- c(melip.altitude, melip.forestcover)
names(covariates) <- c("altitude", "forestcover")
covariates <- scale_covariates(covariates)
# Coarsen the raster: a power sweep refits many times, so cell count is the
# dominant cost. Use the full resolution for a real design study.
covariates <- terra::aggregate(covariates, fact = 3, na.rm = TRUE)
surface <- conductance_surface(covariates, melip.coords,
                               directions = 8, saveStack = TRUE)
```

```
spline_model <- smooth_loglinear_conductance(
  ~ forestcover + s(altitude, df = 4), surface$x)
theta <- attr(spline_model, "default")
theta["forestcover"] <- -0.15
theta[grepl("^s\\(altitude\\)", names(theta))] <-
  c(-0.65, 1.1, -0.9, 0.45)

gaussian_model <- gaussian_smoothed_loglinear_conductance(
  surface, scale_vars = "altitude")

power <- covariance_response_power(
  theta = theta,
  formula = ~ forestcover + s(altitude, df = 4),
  data = surface,
  sample_sizes = c(8, 12),
  strategies = "spacefill",
  conductance_model = smooth_loglinear_conductance,
  fit_models = list(
    spline = list(formula = ~ forestcover + s(altitude, df = 4),
                  conductance_model = smooth_loglinear_conductance),
    gaussian = list(formula = ~ altitude + forestcover,
                    conductance_model = gaussian_model)
  ),
  tau = 0.8,
  sigma = 0.12,
  nu = 150,
  nsim = 3,          # raise to 200+ for a real power estimate
  n_designs = 1,
  seed = 1,
  control = NewtonRaphsonControl(maxit = 12, verbose = FALSE)
)
power$summary
power$parameter_summary
```

`crop_to_focal_buffer` *Crop rasters to a focal-site buffer*

Description

Crops a raster stack to the bounding box of a set of focal coordinates plus a user-supplied map-unit buffer. This is a simple pre-processing helper for large landscapes where the sampled sites occupy only a small part of the raster extent.

Usage

```
crop_to_focal_buffer(covariates, coords, buffer)
```

Arguments

covariates	A terra::SpatRaster.
coords	Focal coordinates accepted by conductance_surface: a terra::SpatVector, matrix, or data frame with coordinate columns.
buffer	Nonnegative buffer distance in the raster's map units. A scalar applies the same buffer in both x and y directions; a length-two vector is interpreted as c(x_buffer, y_buffer).

Details

Cropping reduces the number of graph vertices before any Laplacian factorization occurs. It is most useful when sampling locations are spatially clustered inside a much larger raster. It will not help much when focal sites span the whole raster, because the cropped extent will be close to the original extent.

The requested buffer is expanded internally by half a cell width in each direction so the cropped raster safely contains the cells holding the focal sites even when buffer = 0. If the raster is in longitude/latitude, the buffer is in degrees; use a projected raster for direct distance interpretation.

conductance_surface() also accepts crop_buffer = and applies this helper before graph construction. Cropping can increase resistance by removing paths outside the retained extent. The supplied release audit found increases of 15–23 percent at a two-cell buffer and 1–2 percent at ten cells. These results are specific to that landscape. Compare fits over larger buffers as a sensitivity check. Keep the response, focal sites, covariate scaling, and formula fixed, then refit with successively larger buffers. Compare coefficients, confidence intervals, and fitted resistance distances. Report the buffer and whether increasing it changes the substantive conclusion. A message flags buffers smaller than half the maximum intersite distance.

Value

A cropped terra::SpatRaster. Any terradish_scale metadata attached by scale_covariates is preserved.

Examples

```
library(terra)
r <- terra::rast(nrows = 20, ncols = 20, vals = 1:400,
                 xmin = 0, xmax = 20, ymin = 0, ymax = 20)
pts <- matrix(c(4.5, 4.5,
                6.5, 6.5,
                7.5, 5.5),
              ncol = 2, byrow = TRUE)
cropped <- crop_to_focal_buffer(r, pts, buffer = 2)
cropped

# The same crop can be applied automatically while building the graph.
surface <- conductance_surface(r, pts, directions = 4, crop_buffer = 2)
nrow(surface$x)  # number of retained graph cells
```

`gaussian_scale_profile` *Profile a Gaussian smoothing scale*

Description

Fixes one scale at each grid value and reoptimizes all remaining conductance and measurement parameters on the fitted graph.

Usage

```
gaussian_scale_profile(fit, layer, n = 25, level = 0.95)
```

Arguments

`fit`	An unslimmed Gaussian scale fit with its original model closures.
`layer`	Name of the raster layer whose scale is profiled.
`n`	Number of grid points across the fitted scale bounds, at least five. The fitted estimate is also included if it is not already on the grid.
`level`	Confidence level between zero and one.

Details

The interval is the connected likelihood-ratio support region containing the fitted estimate, using the one-degree-of-freedom chi-square cutoff. Grid crossings are refined by root finding. Increase `n` to check for additional support regions. Limits stop at the original bounds. At a boundary the usual chi-square calibration is an approximation. Failed optimizations stop the calculation instead of silently dropping profile points. Both conductance and final measurement-parameter convergence are required. This can be substantially slower than Wald intervals.

Value

A list with layer, estimate, level, interval, grid, and fit. The grid contains scale values and profile log likelihoods. The returned fit stores the interval, so `confint(result$fit)` uses it at the matching level.

Worked example

See `vignette("gaussian-scale-optimization", package = "terradish")` for scale profiles, forest-effect plots, and interpretation in map units.

See Also

`gaussian_scale_summary`, `gaussian_smoothed_loglinear_conductance`, `terradish_inference`

Examples

```
set.seed(24)
r <- terra::rast(nrows = 10, ncols = 10, xmin = 0, xmax = 10,
                 ymin = 0, ymax = 10, crs = "EPSG:3857")
terra::values(r) <- rnorm(100)
names(r) <- "x"
graph <- conductance_surface(r, terra::xyFromCell(r, seq(3, 98, 5)),
                             directions = 4)
factory <- gaussian_smoothed_loglinear_conductance(graph,
  sigma_lower = 0.4, sigma_upper = 1.5)
response <- simulate_covariance_response(c(x = 0.7, sigma.x = 0.8),
  ~x, graph, conductance_model = factory, tau = 1, sigma = 0.1,
  nu = 10000, seed = 14)$covariance
fit <- terradish(response ~ x, graph, conductance_model = factory,
  measurement_model = wishart_covariance, nu = 10000, optimizer = "newton")
profile <- gaussian_scale_profile(fit, "x", n = 5)
profile$interval
confint(profile$fit)
# The stored profile interval replaces the matching scale Wald interval.
```

---

gaussian_scale_summary

Summarize fitted Gaussian smoothing scales

Description

Converts fitted Gaussian `sigma` parameters into map-unit, cell-based, and Gaussian-kernel interpretation summaries for a fitted model produced with `gaussian_smoothed_loglinear_conductance`.

Usage

```
gaussian_scale_summary(
  object,
  probabilities = c(0.5, 0.95),
  distance_per_map_unit = NULL,
  distance_unit = NULL
)
```

Arguments

`object`	A fitted `terradish` object using the Gaussian scale-aware conductance model.
`probabilities`	Probabilities used to summarize the one-dimensional half-width and two-dimensional radial extent of the Gaussian kernel.
`distance_per_map_unit`	Optional scalar conversion factor used to express `sigma` and the derived radii in user-supplied distance units. For example, if the raster CRS is in meters, use `distance_per_map_unit = 0.001` and `distance_unit = "km"` to report kilometers.

`distance_unit` Optional label used when `distance_per_map_unit` is supplied.

Details

The returned table always reports `sigma` on the raster's native map scale and in raster cell widths. When sigma conversion is enabled during fitting, the table also reports the internal optimization scale and its conversion factor back to map units. For each probability `p`, `axis_*` gives the one-dimensional half-width $\sigma\Phi^{-1}((1+p)/2)$, while `radial_*` gives the isotropic two-dimensional radius $\sigma\sqrt{-2\log(1-p)}$ containing proportion `p` of the Gaussian kernel mass. Here Φ^{-1} is the standard normal quantile function and σ is the smoothing scale. These quantities summarize the fitted raster-smoothing kernel. They are not estimates of dispersal distance, movement distance, or home-range size. Use `summary(object)$sigma_table` to inspect standard errors and bound flags, and `confint(object)` for bound-truncated intervals; smoothing scales have no reported z or p values. Use `gaussian_scale_profile` when a profile interval is needed. A positive scale can reflect dispersal averaging, and even the half-cell lower bound applies some smoothing.

If the retained raster is in longitude/latitude, the native-unit results are in degrees. In that case, use a projected raster for direct distance interpretation, or provide an approximate `distance_per_map_unit` conversion if you need a quick descriptive summary.

Value

A data frame with one row per fitted `sigma` parameter. `covariate` identifies the layer. `sigma` is its fitted map-unit scale; `sigma_lower` and `sigma_upper` are fitting bounds. `sigma_internal`, `sigma_conversion`, and `sigma_conversion_mode` describe the optimization units. `native_unit` labels map units or degrees, and `sigma_cells_x`, `sigma_cells_y` express the scale in cell widths. Probability-specific `axis_*` and `radial_*` columns give the theoretical Gaussian distances described above, in native units and, when requested, converted units. They describe the untruncated Gaussian; retained raster windows can omit some of that mass.

Examples

```
data(melip)
forestcover <- terra::unwrap(melip.forestcover)
names(forestcover) <- "forestcover"

# Coarsen the raster so the example runs quickly; the Gaussian model
# re-smooths the layer at every optimizer step, so cost scales with cell
# count. Use the full resolution in a real analysis.
forestcover <- terra::aggregate(forestcover, fact = 3, na.rm = TRUE)

# Feed the model the raw (unsmoothed, unstandardized) raster: it smooths and
# standardizes internally at each candidate sigma.
surface <- conductance_surface(forestcover, terra::unwrap(melip.coords),
                               directions = 8, saveStack = TRUE)
fit <- terradish(melip.Fst ~ forestcover,
                 data = surface,
                 conductance_model =
                   gaussian_smoothed_loglinear_conductance(surface),
                 measurement_model = leastsquares,
                 optimizer = "bfgs")
```

```
# sigma is in map units. The melip rasters are unprojected, so those units
# are degrees; distance_per_map_unit rescales the reported distances.
gaussian_scale_summary(fit)
gaussian_scale_summary(fit, distance_per_map_unit = 111,
                       distance_unit = "km")
```

`gaussian_smoothed_loglinear_conductance` *Gaussian scale-aware log-linear conductance model*

Description

Creates a conductance-model factory that smooths selected raster layers with a native Gaussian FFT operator and treats the corresponding Gaussian scale parameters as part of the optimized conductance parameter vector.

Usage

```
gaussian_smoothed_loglinear_conductance(
  surface,
  scale_vars = NULL,
  standardize = TRUE,
  sigma_lower = NULL,
  sigma_upper = NULL,
  sigma_conversion = c("cell", "map"),
  sigma_conversion_factor = NULL
)
```

Arguments

`surface`	A `conductance_surface` object created with `saveStack = TRUE`. The retained raster stack is used as the native smoothing domain during optimization.
`scale_vars`	Optional character vector naming the raster layers whose Gaussian smoothing scales should be estimated. Defaults to all numeric raster variables present in the model formula.
`standardize`	Should each smoothed layer be centered and scaled across the active graph cells at every parameter evaluation? Default `TRUE`. This is separate from `scale_covariates`: the raw raster is smoothed first, then the smoothed active-cell values are standardized internally.
`sigma_lower, sigma_upper`	Optional lower and upper bounds for the Gaussian scale parameters on their natural distance scale. Scalar values are recycled across all scaled rasters; named vectors may be used to set different bounds by raster name. When omitted, `sigma_lower` defaults to half of the mean raster cell width and `sigma_upper`

defaults to one sixth of the smaller raster dimension in cells times the mean cell width. The Gaussian kernel uses a raster-sized window; larger explicit upper bounds issue a warning because more of the kernel is truncated.

`sigma_conversion`
: Internal scaling applied during optimization. `"cell"` (default) rescales each sigma parameter by the mean cell width of its raster layer so optimization happens in approximate cell units, while reported coefficients are converted back to map units. `"map"` leaves the optimization on the native map-unit scale.

`sigma_conversion_factor`
: Optional positive scalar or named vector used to override the default sigma conversion factors. This is most useful when you want optimization to happen in user-defined distance units while still reporting fitted sigma values in map units.

Details

This is a scale-aware extension of `loglinear_conductance`. Factory settings are evaluated when the factory is created, so factories constructed in a loop retain their own raster choices and scale bounds. Given a model formula such as `~ altitude * forestcover + I(altitude^2)`, the fitted parameter vector contains both conductance coefficients and one natural-scale `sigma` parameter for each scaled raster layer. The spatial kernel is Gaussian and is evaluated with an FFT-based normalized convolution over the retained raster stack from `surface`. The smoothing domain is fixed within each graph evaluation, which keeps the sigma derivatives smooth during optimization. When `terradish` is run with `approximation = "coarse_raster"`, the smoother is rebuilt on each coarse graph while preserving the full-resolution sigma bounds and conversion so coarse-stage starts remain interpretable in map units. Internally, the sigma parameters may be optimized on a converted scale (for example, approximate cell widths) and are then converted back to map units before they are returned to users. These parameters describe Gaussian smoothing of the supplied raster in its map units. They are not direct estimates of dispersal distance, movement distance, home-range size, or the scale of a causal ecological process.

Unlike the standard fixed-raster workflow, pre-scaling with `scale_covariates` is unnecessary when `standardize = TRUE`: the supplied raster is smoothed at each proposed `sigma`, then its active-cell values are centered and scaled. Because `scale_covariates` applies a positive affine transformation to each layer, pre-scaling gives the same standardized smoothed values up to numerical tolerance. Supplying original values keeps the preprocessing and coefficient interpretation clear. With `standardize = FALSE`, pre-scaling changes the covariate and coefficient units. With the default standardization, a coefficient is relative log conductance per standard deviation of the smoothed layer at the current sigma. The layer is re-standardized at every candidate sigma during fitting. The half-cell lower bound still smooths; near the upper bound, the smoothed layer acts as a broad spatial trend. Dispersal averaging can favor a positive sigma even when no separate ecological scale of effect exists.

The Gaussian kernel is truncated to a raster-sized window. The default upper bound keeps approximately three sigma inside half the smaller raster dimension; increasing it cannot recover kernel mass outside that window. Inspect `summary(fit)$sigma_table` for bound flags and `gaussian_scale_profile` for profile intervals. Scale estimates do not receive z or p values because zero is outside the fitting interval. Wald intervals are truncated at the bounds; a bound-limited interval does not demonstrate a precisely identified scale.

The current implementation supports numeric raster formulas built from raw raster names, interactions, and polynomial/arithmetic terms that can be differentiated analytically by `D` after smoothing

the base raster layers. Factor-valued raster layers are rejected because Gaussian smoothing is only defined here for numeric rasters; use an unscaled fixed conductance model if you need categorical raster predictors.

For this conductance model, terradish prefers `optimizer = "bfgs"` when `optimizer = "auto"`, and leverage / `partial_X` calculations are currently disabled because the current leverage machinery assumes a one-to-one mapping between fitted conductance parameters and design-matrix covariates, which does not hold once separate Gaussian `sigma` parameters are introduced.

Value

A function of class `terradish_conductance_model_factory` that can be supplied as the `conductance_model=` argument to terradish.

Worked example

See `vignette("gaussian-scale-optimization", package = "terradish")` for fitting, plotting, and interpreting smoothing scales on the pond landscape.

Examples

```
data(melip)
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords <- terra::unwrap(melip.coords)

# Use the raw raster here. The Gaussian model smooths first and then
# standardizes internally.
names(melip.forestcover) <- "forestcover"

# Coarsened so the example runs quickly: the raster is re-smoothed at every
# candidate sigma. Use the full resolution in a real analysis.
melip.forestcover <- terra::aggregate(melip.forestcover, fact = 3,
                                      na.rm = TRUE)
surface <- conductance_surface(
  melip.forestcover,
  melip.coords,
  directions = 8,
  saveStack = TRUE
)

gaussian_model <- gaussian_smoothed_loglinear_conductance(surface)
fit <- terradish(
  melip.Fst ~ forestcover,
  data = surface,
  conductance_model = gaussian_model,
  measurement_model = mlpe,
  optimizer = "auto",
  leverage = FALSE
)

gaussian_scale_summary(fit)
plot(fit, type = "sigma")
```

```
# On a full-resolution raster, a coarse-raster warm start cuts the cost of
# the scale search. With exact_refine = TRUE the reported fit is refined
# back on the original graph, so the estimates stay exact.
fit_coarse <- terradish(
  melip.Fst ~ forestcover,
  data = surface,
  conductance_model = gaussian_model,
  measurement_model = mlpe,
  approximation = "coarse_raster",
  approximation_control = list(factor = 2, exact_refine = TRUE),
  optimizer = "auto",
  leverage = FALSE
)
```

`generalized_wishart` *Generalized Wishart distance regression*

Description

A function of class `"terradish_measurement_model"` that evaluates the generalized Wishart likelihood for an admissible pairwise genetic squared-distance matrix. This model requires a distance representation that is coherently related to a centered positive semidefinite covariance matrix; it is not valid for an arbitrary dissimilarity matrix. The effective marker degrees of freedom must also be supplied. For covariance-matrix data, use `wishart_covariance`.

Usage

```
generalized_wishart(
  E,
  S,
  phi,
  nu,
  gradient = TRUE,
  hessian = TRUE,
  partial = TRUE,
  nonnegative = TRUE,
  validate = FALSE
)
```

Arguments

`E`	Conductance-implied covariance matrix: the generalized inverse of the graph Laplacian at the current conductance parameters. Passed automatically by the optimizer.

S	Square, symmetric admissible genetic squared-distance matrix with zero diagonal. It must have the same dimensions as E and must correspond to a centered positive semidefinite covariance representation under the scaling used to construct it.
phi	Named numeric vector of nuisance parameters (tau, sigma). Omit to obtain default starting values c(1, 0).
nu	One finite positive number describing effective Wishart information. You must supply it; the fit does not estimate it. It is not the marker count. See the effective-information section and report sensitivity across plausible values with terradish_rescale_nu.
gradient	Logical. Compute gradient of the negative log-likelihood with respect to phi?
hessian	Logical. Compute Hessian with respect to phi?
partial	Logical. Compute second partial derivatives with respect to phi, E, and S?
nonnegative	Unused; present for interface consistency.
validate	Logical. Numerically validate gradients and Hessians via **numDeriv**? Very slow; for debugging small examples only.

Details

The nuisance parameters are:

tau Nonnegative scale applied to the conductance-implied covariance E. A tau near zero signals no detectable contribution from conductance-implied covariance under this model.

sigma Log-scale identity component: the nugget variance added to the diagonal is $\exp(\sigma)$. It represents diagonal variance not captured by the graph covariance, including sampling variation and model mismatch. It is not a smoothing scale or a direct estimate of genetic drift or population size.

The fitted covariance is $\Sigma = \tau E + \exp(\sigma) I$. The model evaluates the generalized Wishart log-likelihood for the observed admissible squared-distance matrix S after projecting out the grand mean, as described in McCullagh (2009).

generalized_wishart and wishart_covariance share the same Σ parameterization but differ in what S represents: this function takes a **squared-distance** matrix; wishart_covariance takes a **covariance** matrix. When the covariance and its implied distance representation carry the same information under the same centering, scaling, sites, and nu, the two functions return the same objective (up to numerical tolerance) because both use the same site-contrast likelihood and omit response-only normalizing terms. This equality does not make AIC values from arbitrary distance and covariance fits interchangeable.

F_{ST} ratio estimators do not have Wishart degrees of freedom. A passed geometry check is necessary but does not establish this sampling likelihood. Prefer covariance-derived squared distances and document the construction; use mlpe for ratio-estimator responses.

The function checks S with check_distance_response before evaluating the likelihood. Substantive violations stop the fit. Tiny structural deviations within the numerical tolerance are symmetrized and set to zero where appropriate. No Euclidean correction is applied.

Value

When phi is missing, a list with elements phi (starting values c(tau = 1, sigma = 0)), lower (c(0, -Inf)), and upper. Otherwise a list containing:

objective	Negative log-likelihood.
fitted	Matrix of expected squared genetic distances.
boundary	Logical; TRUE if tau = 0, indicating no detectable IBR signal.
gradient	Gradient with respect to phi (if gradient = TRUE).
hessian	Hessian matrix with respect to phi (if hessian = TRUE).
gradient_E	Gradient with respect to E (if partial = TRUE).
partial_E	Jacobian of gradient_E with respect to phi (if partial = TRUE).
partial_S	Jacobian of gradient with respect to the lower triangle of S (if partial = TRUE).
jacobian_E	Function for reverse-mode AD through E (if partial = TRUE).
jacobian_S	Function for reverse-mode AD through S (if partial = TRUE).

Numerical nugget limits

Fitting uses the same small-nugget safeguards as wishart_covariance, retaining supplied bounds and fixed parameters. A verified numerical zero residual-variance limit is recorded separately from a zero graph contribution; summary() omits Wald uncertainty for its log-scale nugget coordinate.

Effective Wishart information

For Wishart likelihoods, nu scales the log likelihood. Point estimates do not depend on it, but standard errors, likelihood-ratio tests, and information criteria do. It is a dispersion parameter that you set, rather than the number of markers. Linkage and the population history shared by loci can make effective information much smaller than the SNP count. In forward-time simulations used to validate terradish, standard errors at the SNP count were three to five times smaller than the spread among replicate histories. Those model-based errors understated uncertainty.

Use terradish_cv_folds to compare conductance formulas, and terradish_rescale_nu to report inference across plausible values. For the same successful folds and nuisance settings, changing nu rescales Wishart CV scores without changing their ranking. Neither procedure identifies the true effective information or repairs an unsuitable response matrix.

References

McCullagh P. 2009. Marginal likelihood for distance matrices. Statistica Sinica 19:631-649.

See Also

check_distance_response, wishart_covariance, wishart_covariates, mlpe, terradish

Examples

```
library(terra)

data(melip)
melip.altitude <- terra::unwrap(melip.altitude)
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords <- terra::unwrap(melip.coords)

covariates <- c(melip.altitude, melip.forestcover)
names(covariates) <- c("altitude", "forestcover")
surface <- conductance_surface(covariates, melip.coords, directions = 8)

# Inverse of the graph Laplacian at the null conductance model.
laplacian_inv <- terradish_distance(theta = matrix(0, 1, 2),
                                    formula = ~forestcover + altitude,
                                    data = surface,
                                    terradish::loglinear_conductance,
                                    covariance = TRUE)$covariance[,,1]

# Generate an admissible squared-distance response from a PSD covariance.
Sigma <- 0.8 * laplacian_inv + 0.2 * diag(nrow(laplacian_inv))
observed_covariance <- rWishart(1, df = 40, Sigma = Sigma)[, , 1] / 40
observed_squared_distance <- dist_from_cov(observed_covariance)
generalized_wishart(laplacian_inv, observed_squared_distance, nu = 40,
                    phi = c(tau = 0.8, sigma = log(0.2)))
```

`HagerZhangControl` *Control settings for Hager-Zhang line search*

Description

Tuning parameters for the Hager-Zhang line-search implementation.

Usage

```
HagerZhangControl(
  delta = 0.1,
  sigma = 0.9,
  alphamax = Inf,
  rho = 5,
  epsilon = 1e-06,
  gamma = 0.66,
  linesearchmax = 50,
  psi3 = 0.1,
  c = 1,
  verbose = FALSE
)
```

Arguments

`delta`	Sufficient decrease constant (Wolfe condition 1). Must satisfy `0 < delta < sigma < 1`. Smaller values accept shorter steps more readily.
`sigma`	Curvature condition constant (Wolfe condition 2). Must satisfy `delta < sigma < 1`. Values close to 1 relax the curvature requirement and allow fewer function evaluations per step.
`alphamax`	Upper bound on the trial step length. The default `Inf` lets the bracketing phase grow the step without limit.
`rho`	Multiplicative expansion factor applied during the initial bracketing phase. Must be `> 1`.
`epsilon`	Non-negative tolerance defining an "approximate Wolfe" regime near a local minimum: a small positive value (default `1e-6`) allows the algorithm to accept a slightly increased objective near convergence.
`gamma`	Minimum fractional reduction of the bracket required after each secant step. Steps that shrink by less than this factor trigger bisection instead. Must be in `(0, 1)`.
`linesearchmax`	Maximum number of line-search function evaluations. Raise this if the optimizer reports repeated line-search failures on difficult problems.
`psi3`	Backtracking multiplier applied when the initial trial step produces a non-finite objective. Must be in `(0, 1)`.
`c`	Initial trial step length passed to the line search. The default `1.0` is appropriate for Newton and quasi-Newton methods that produce unit-scale search directions.
`verbose`	Should line-search progress (trial step lengths and objective values) be printed to the console?

Details

`HagerZhangControl()` is the default line search used by `NewtonRaphsonControl`. It implements the approximate Wolfe conditions from Hager and Zhang (2006), which combine sufficient-decrease and curvature checks with a bracketing-plus-secant strategy.

The default parameters work well for most resistance-surface optimization problems. Common reasons to change them:

- **Line-search failures**: increase `linesearchmax` or loosen `sigma` toward 1.
- **Slow convergence with many small steps**: tighten `sigma` (e.g. 0.1) to enforce a stronger curvature condition.
- **Initial non-finite objectives**: increase `psi3` to take larger backtracking steps when `c` is too large.

Value

A named list of class `"HagerZhangControl"` holding the line search settings. The elements are the arguments above, stored unchanged: `delta`, `sigma`, `alphamax`, `rho`, `epsilon`, `gamma`, `linesearchmax`, `psi3`, `c`, and `verbose`. Pass the object to the `ls.control` argument of `NewtonRaphsonControl` rather than to `terradish` directly; inspect it with `str()`. The list carries no fitted quantities, so nothing in it needs back-transforming.

References

Hager WW, Zhang H. 2006. Algorithm 851: CG_DESCENT, a conjugate gradient method with guaranteed descent. ACM Transactions on Mathematical Software 32(1):113-137.

See Also

NewtonRaphsonControl, ArmijoControl

Examples

```
ctrl <- HagerZhangControl(verbose = FALSE, linesearchmax = 10)
str(ctrl)

# Use inside NewtonRaphsonControl:
opt_ctrl <- NewtonRaphsonControl(
  maxit = 50,
  ls.control = HagerZhangControl(sigma = 0.5, linesearchmax = 30)
)
```

---

individual_salamanders

Simulated Individual Salamander Genotypes and Resistance Analyses

---

Description

Matched SNP and microsatellite genotypes for 160 adult terrestrial salamanders in each of two simulated movement scenarios on a real southeastern Ohio landscape. The primary example demonstrates nonlinear isolation by resistance with individual sampling. The animals and drying index are hypothetical; neither species occurrence nor field-calibrated movement is claimed.

Usage

```
data(individual_salamanders)
```

Format

A list with three elements:

`data` Two named scenarios, `compensatory_500` and `control_500`. Each contains `samples`, `snp_dosage`, `microsat_alleles`, and `covariance`, described below.

`models` Twenty checked model records per scenario: uniform and quadratic drying models, two marker types, and partitions 0 (all adults) through 4 (one spatial quadrant withheld). Names combine marker, model, and partition, for example `SNP2500__IBR__0`. Records contain log-conductance coefficients `theta`, graph covariance multiplier `phi$tau` on its natural nonnegative scale, log nugget variance `phi$sigma`, starting-point diagnostics, convergence, curvature, covariance score, RMSE, and predicted-mean range. These are plain lists, not fitted `terradish` objects.

`metadata` Fixed drying center `center`, linear divisor `sd_linear`, and quadratic divisor `sd_quadratic`; the latter is SD of squared deviations, not squared SD. `start_halfwidth` defines additional starting points, not coefficient bounds. `epsg`, `native_resolution_m`, `graph_resolution_m`, `directions`, `extent`, `core`, and `graph_nodes` describe the spatial grid (EPSG:26917, meters). `sample_size`, `main_SNPs`, and `matched_microsatellites` give panel sizes. `objective`, `biological_scope`, and `raw_marker_selection` state interpretation and selection rules.

Details

Within each scenario, `samples` has 160 rows in genotype order. `sample_id` and `sample_key` identify the animal; `id`, `mom`, and `dad` are recorded ecological pedigree identifiers. `stratum` gives one of 16 sampling strata; `quadrant` and `fold` identify the four spatial validation blocks. `age` is in years, `sex` is 0 for female and 1 for male, `x_local` and `y_local` are meters from the domain origin, and `easting` and `northing` are projected UTM coordinates. `drying` is the interpolated hypothetical index from 0 to 1.

`snp_dosage` contains nested integer matrices named `500`, `1000`, and `2500`. Entries 0, 1, and 2 count one allele. Locus names identify synthetic genomic blocks and within-block positions. `microsat_alleles` is a 160 by 48 integer matrix; adjacent columns hold the two repeat-count alleles at each of 24 loci. All marker row names match `samples$sample_id`. Microsatellite allele labels are categorical, not continuous response values.

`covariance` contains centered matrices named `SNP2500` and `microsat24`, averaging unstandardized allele-frequency inner products across loci. Their implied squared distances range from 0 to 2. This is different from the locus-frequency weighting in `cov_from_biallelic` and is not a Dps response.

Each scenario follows 500 years, with maximum age 20 for both sexes. The compensatory scenario allows faster, straighter movement at intermediate drying but greater danger in very dry terrain. The control holds behavioral drying at 0.5 while retaining forest settlement and demography. It is a control for drying-dependent movement, not every possible resistance effect. Samples occupy distinct 10 m cells, are at least 10 m apart, and exclude direct or shared recorded parents and grandparents. Distant kin can remain.

These analyses use `wishart_covariance` with `nu = 1` as a covariance quasi-likelihood, not a literal independent-Gaussian-locus sampling model. Nominal Hessian uncertainty, AIC, and significance tests are not used as genetic-sampling evidence. Two hundred genomic-block resamples and a second genomic assay check conditional marker variation; neither supplies independent ecological replication. Spatial folds were reused during workflow development. Marker discovery conditions on the full sampled panel. The matched microsatellites retain mixed predictive evidence. No separate causal isolation-by-environment effect was simulated.

Installed files in `extdata/individual-salamanders` provide the 10 m drying terms and environment, native 30 m terrain, all 40 original prediction records, comparative scores, the independent assay, and provenance. The 10 m grid interpolates native 30 m information; it adds no new measurements. The vignette explains these files and the longer opt-in reproduction.

Worked examples

```
vignette("data-generation", package = "terradish"), vignette("individual-salamanders",
package = "terradish").
```

Source

Simulated genotypes from recorded individual pedigrees on terrain derived from USGS 3DEP elevation and 2023 Annual NLCD Collection 1.0. See `system.file("extdata", "individual-salamanders", "README.md", package = "terradish")` for data sources, processing, and limitations.

See Also

`cov_from_genetic_data`, `cov_from_biallelic`, `dist_from_cov`, `vignette("individual-salamanders", package = "terradish")`.

Examples

```
data(individual_salamanders)
animals <- individual_salamanders$data$compensatory_500
G <- animals$snp_dosage[["2500"]]
S <- cov_from_genetic_data(G, scale = FALSE, normalize = "none") /
  (2 * ncol(G))
dim(S)                              # 160 individual sampling locations
range(as.matrix(dist_from_cov(S)))   # squared allele-frequency distance
table(animals$samples$fold)           # 40 withheld adults per spatial fold
```

---

leastsquares *Least-squares measurement model*

Description

A function of class `"terradish_measurement_model"` that evaluates a Gaussian likelihood with independent errors and a linear mean structure relating observed genetic distances to resistance distances. This is the fastest measurement model and a useful exploratory or diagnostic choice. For distance-response inference, `mlpe` represents shared-site dependence that this model omits.

Usage

```
leastsquares(
  E,
  S,
  phi,
  nu = NULL,
  gradient = TRUE,
  hessian = TRUE,
  partial = TRUE,
  nonnegative = TRUE,
  validate = FALSE
)
```

Arguments

E	Conductance-implied covariance matrix: the generalized inverse of the graph Laplacian, evaluated at the current conductance parameters. Passed automatically by the optimizer; users normally do not call this function directly.
S	Square, symmetric matrix of observed pairwise genetic distances (e.g. F_{ST}). Must have the same dimensions as E.
phi	Named numeric vector of nuisance parameters (alpha, beta,tau). Omit to obtain maximum-likelihood starting values from an nlme::gls fit.
nu	Unused; present for a common interface with Wishart measurement models.
gradient	Logical. Compute gradient of the negative log-likelihood with respect to phi?
hessian	Logical. Compute Hessian of the negative log-likelihood with respect to phi?
partial	Logical. Compute second partial derivatives with respect to phi, E, and S? Required by the optimizer; set FALSE only for standalone likelihood evaluations.
nonnegative	Logical. Constrain the IBR slope beta to be nonnegative? Default TRUE prevents a nonsensical negative resistance-distance slope.
validate	Logical. Numerically validate gradients and Hessians via **numDeriv**? Very slow; intended for debugging small examples only.

Details

The nuisance parameters are:

alpha Intercept of the regression of genetic distance on resistance distance.

beta IBR slope; constrained ≥ 0 when nonnegative = TRUE.

tau Log-precision of the Gaussian errors: residual variance is $\exp(-\tau)$.

The mean structure is $S_{ij} = \alpha + \beta R_{ij} + e_{ij}$, where R_{ij} is the resistance distance between sites i and j (derived from E) and e_{ij} are independent Gaussian errors with common precision $\exp(\tau)$.

Pairwise genetic distances are not independent: any two pairs sharing a sampling site can be correlated. leastsquares ignores this dependence, so its standard errors can be too small. Use it for fast exploration, diagnostics, or initial parameter searches rather than as the default basis for distance-response inference.

Value

When phi is missing, a list with elements phi (starting values), lower, and upper (parameter bounds). Otherwise a list containing:

objective	Negative log-likelihood.
fitted	Matrix of expected genetic distances (same dimensions as S).
boundary	Logical; TRUE if the MLE is on the boundary (beta = 0), indicating no detectable IBR signal.
gradient	Gradient of the negative log-likelihood with respect to phi (if gradient = TRUE).
hessian	Hessian matrix with respect to phi (if hessian = TRUE).

gradient_E	Gradient with respect to E (if partial = TRUE).
partial_E	Jacobian of gradient_E with respect to phi (if partial = TRUE).
partial_S	Jacobian of gradient with respect to the lower triangle of S (if partial = TRUE).
jacobian_E	Function for reverse-mode algorithmic differentiation through E (if partial = TRUE).
jacobian_S	Function for reverse-mode algorithmic differentiation through S (if partial = TRUE).

See Also

mlpe, generalized_wishart, wishart_covariance, terradish

Examples

```
library(terra)

data(melip)
melip.altitude <- terra::unwrap(melip.altitude)
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords <- terra::unwrap(melip.coords)

covariates <- c(melip.altitude, melip.forestcover)
names(covariates) <- c("altitude", "forestcover")
surface <- conductance_surface(covariates, melip.coords, directions = 8)

# inverse of graph Laplacian at null model (IBD)
laplacian_inv <- terradish_distance(theta = matrix(0, 1, 2),
                                    formula = ~forestcover + altitude,
                                    data = surface,
                                    terradish::loglinear_conductance,
                                    covariance = TRUE)$covariance[,,1]

leastsquares(laplacian_inv, melip.Fst) #without 'phi': return MLE of phi
leastsquares(laplacian_inv, melip.Fst, phi = c(0., 0.5, -0.1))
```

---

loglinear_conductance *Log-link conductance model*

Description

Returns a function of class "terradish_conductance_model" that represents a log-linear mapping from spatial covariates to conductance. This is the recommended conductance model for most applications.

Usage

```
loglinear_conductance(formula, x)
```

Arguments

`formula`	Model formula describing which spatial covariates drive conductance. The left-hand side is ignored; only the right-hand side terms are used (e.g. `~ altitude + forestcover`).
`x`	Data frame of spatial covariates extracted from a `conductance_surface` object (typically `surface$x`). Every variable named on the right-hand side of `formula` must be a column of `x`.

Details

The conductance at grid cell `i` is:

$$C_i = \exp(\theta_1 x_{i1} + \theta_2 x_{i2} + \ldots)$$

where x_{ij} is the value of covariate j at cell i and θ_j is the corresponding conductance parameter.

The intercept is intentionally omitted: multiplying all conductances by a common factor rescales resistance inversely, and the measurement model's resistance coefficient absorbs that factor. The absolute conductance level is therefore not identifiable.

Interpreting θ:

- $\theta_j > 0$: higher values of covariate j increase fitted relative conductance and tend to lower graph resistance distance, conditional on the other terms and graph domain.
- $\theta_j < 0$: higher values lower fitted relative conductance.
- $\theta_j = 0$: the fitted conductance is locally unchanged by that term, conditional on the model.
- A one-standard-deviation increase in covariate j multiplies fitted conductance by $\exp(\theta_j)$ when the covariate was standardized and the term is linear.

These are model-based associations, not direct estimates of movement rate, habitat suitability, or a causal landscape effect.

The exponential link guarantees strictly positive conductances for any real θ.

Categorical covariates must be stored as `factor` columns in `x` (see `conductance_surface` for how to encode them). They are dummy-coded using the default R contrasts via `model.matrix`, with one level dropped as a reference. In-formula transformations such as `I(x^2)` and interaction terms `x * z` are supported.

Value

A function of class `"terradish_conductance_model"` that accepts a numeric vector of conductance parameters `theta` and returns a list with elements `conductance` (a vector of per-cell conductance values), `confint` (a function for confidence intervals), and derivative functions used internally by the optimizer.

See Also

`gaussian_smoothed_loglinear_conductance`, `conductance_surface`, `terradish`

Examples

```
x <- data.frame(altitude = c(-1, 0, 1), forestcover = c(0.2, 0.6, 0.5))
model <- loglinear_conductance(~ altitude + forestcover, x)

# Evaluate at specific parameter values
fit <- model(c(altitude = 0.3, forestcover = -0.2))
fit$conductance  # per-cell conductance values

# Interaction and polynomial terms work too
x2 <- data.frame(altitude = c(-1, -0.25, 0.5, 1),
                 fc = c(0.2, 0.6, 0.5, 0.8))
model2 <- loglinear_conductance(~ altitude + I(altitude^2) + fc, x2)
```

---

lower *Extract the strict lower triangle of a matrix*

Description

Returns the entries strictly below the main diagonal of a square matrix, scanned column-by-column (standard R column-major order).

Usage

```
lower(x)
```

Arguments

x	A square numeric matrix.

Details

This is a thin wrapper around `x[lower.tri(x)]`. It is provided as a convenience for extracting vectorized pairwise measurements from symmetric matrices such as distance or covariance matrices.

Value

A numeric vector of length $n(n-1)/2$ containing the strict lower-triangular entries of `x`, where n is the dimension of `x`.

See Also

`lower.tri`

Examples

```
m <- matrix(1:9, nrow = 3)
lower(m)  # entries (2,1), (3,1), (3,2)

# Useful for extracting pairwise distances:
D <- matrix(c(0, 0.1, 0.3,
              0.1, 0, 0.2,
              0.3, 0.2, 0), nrow = 3)
lower(D)
```

`melip` *Melipona Example Data*

Description

Landscape and genetic data for 37 sampled populations of the stingless bee genus *Melipona* in Brazil, bundled with **terradish** so every help page and vignette can run a complete worked example. The four objects are stored separately in the package data, so `data(melip)` loads all of them into the calling environment at once.

Usage

```
data(melip)
```

Format

Four objects, loaded together by `data(melip)`:

`melip.altitude` A wrapped `terra::SpatRaster`, 156 rows by 180 columns at 0.045 degree resolution, covering roughly 42.9 to 34.8 degrees west and 9.5 to 2.5 degrees south in unprojected longitude/latitude. Values are altitude already rescaled to roughly the unit interval (range 0 to 1.06), not meters. Of the 28,080 cells, 22,443 are land and 5,637 are `NA` (ocean), and only the non-`NA` cells enter the conductance graph.

`melip.forestcover` A wrapped `terra::SpatRaster` on the same grid, giving forest cover as a proportion (range 0.004 to 0.948).

`melip.coords` A wrapped `terra::SpatVector` of 37 point locations, one per sampled population, in the same longitude/latitude coordinate system as the rasters.

`melip.Fst` A symmetric 37 by 37 numeric matrix of pairwise F_{ST} among the same 37 populations, with a zero diagonal and no dimnames. Values range from -0.062 to 0.280. Four off-diagonal entries are slightly negative, which is a normal outcome of unbiased F_{ST} estimators when true differentiation is near zero. Gaussian distance models such as `mlpe` can retain these estimates. Their presence is one reason not to treat this matrix as automatically admissible for `generalized_wishart`.

Details

The two rasters are stored as `terra::wrap()`ped objects because a live `SpatRaster` holds an external pointer that cannot be serialized into a package. Call `terra::unwrap()` on each one before use; the same applies to the `SpatVector` of coordinates.

The rasters are unprojected, so distances are in degrees rather than meters. That matters for any function whose output carries a distance unit: the `sigma` reported by `gaussian_scale_summary`, for instance, is in degrees for this data set. Reproject before interpreting the fitted Gaussian raster-smoothing scale as a distance.

The altitude layer arrives pre-rescaled, but the two layers are still on different scales, so run `scale_covariates` before fitting, as every example here does.

Source

Redistributed from the `radish` package, where the data were assembled for the original resistance-surface optimizer. A primary citation for the underlying *Melipona* sampling and genotyping is not recorded in the package sources.

See Also

`conductance_surface` for turning these layers into a graph, and `terradish` for fitting a model to them. See `vignette("mlpe-pairwise-distance", package = "terradish")` for the full worked example with these data.

Examples

```
data(melip)

# rasters and points are wrapped for storage; unwrap before use
altitude    <- terra::unwrap(melip.altitude)
forestcover <- terra::unwrap(melip.forestcover)
coords      <- terra::unwrap(melip.coords)

altitude                    # 156 x 180 grid, values on ~[0, 1]
nrow(coords)                # 37 sampled populations
dim(melip.Fst)              # 37 x 37 pairwise Fst
range(melip.Fst)            # note the small negative values

# MLPE can retain the negative estimates in this pairwise-distance response
Fst <- melip.Fst
round(Fst[1:4, 1:4], 3)

terra::plot(altitude, main = "Altitude (rescaled)")
terra::points(coords, pch = 19, cex = 0.7, col = "red")
```

`mlpe` *Maximum likelihood population effects (MLPE) measurement model*

Description

A function of class `"terradish_measurement_model"` that evaluates the maximum likelihood population effects (MLPE) likelihood. MLPE accounts for the non-independence of pairwise genetic distances that share a sampling site. It is the preferred distance-response likelihood when a defensible effective Wishart degrees-of-freedom value is unavailable or the response is not admissible for `generalized_wishart`.

Usage

```
mlpe(
  E,
  S,
  phi,
  nu = NULL,
  gradient = TRUE,
  hessian = TRUE,
  partial = TRUE,
  nonnegative = TRUE,
  validate = FALSE
)
```

Arguments

`E`	Conductance-implied covariance matrix: the generalized inverse of the graph Laplacian, evaluated at the current conductance parameters. Passed automatically by the optimizer; users normally do not call this function directly.
`S`	Square, symmetric matrix of observed pairwise genetic distances (e.g. F_{ST}). Must have the same dimensions as `E`.
`phi`	Named numeric vector of nuisance parameters (`alpha`, `beta`,`tau`, `rho`). The stored `tau` is log precision and the stored `rho` is an unconstrained, logit-scale parameter. Omit to obtain starting values from a least-squares fit and a weak positive MLPE correlation.
`nu`	Unused; present for a common interface with Wishart measurement models.
`gradient`	Logical. Compute gradient of the negative log-likelihood with respect to `phi`?
`hessian`	Logical. Compute Hessian of the negative log-likelihood with respect to `phi`?
`partial`	Logical. Compute second partial derivatives with respect to `phi`, `E`, and `S`? Required by the optimizer.
`nonnegative`	Logical. Constrain the IBR slope `beta` to be nonnegative? Default `TRUE`.
`validate`	Logical. Numerically validate gradients and Hessians via **numDeriv**? Very slow; for debugging small examples only.

Details

The likelihood uses maximum likelihood (ML), not restricted maximum likelihood (REML). AIC can therefore compare different conductance formulas fitted to the same response, sites, graph, and likelihood family. That comparison does not validate the correlation model: in package validation, model-based standard errors understated replicate variation about 1.7-fold. Use spatial CV to assess out-of-sample support for conductance terms.

With the standard nuisance bounds, fitting profiles the regression coefficients and residual precision analytically at each shared-site correlation, then searches the correlation before Newton refinement. The resistance predictor is centered and scaled internally so changing its units does not change the nuisance optimum or convergence criterion. Stored coefficients remain in the original units. Fixed predictive parameters and custom nuisance bounds use their supplied constraints. A profile reaching the upper numerical correlation endpoint is rejected because a regular finite nuisance optimum has not been established.

`summary()` reports `rho` on its internal logit scale, with a Wald interval on that scale. Convert an estimate or interval endpoint with `plogis(rho) / 2` to obtain the shared-site correlation, which lies between 0 and 0.5. A value near 0.5 means site identity explains most of the residual variation. When the correlation approaches zero, the logit parameter is poorly identified, so `summary()` reports `rho` as 0 with no standard error and prints a boundary note. This boundary differs from a zero resistance coefficient, which leaves conductance unidentified.

To read the other nuisance parameters, `alpha` is the expected genetic distance at zero resistance distance, `beta` is the increase in genetic distance per unit of resistance distance, and the residual standard deviation is `exp(-tau / 2)`.

The nuisance parameters are:

`alpha` Intercept of the mean structure.

`beta` Resistance-distance (IBR) slope; constrained ≥ 0 when `nonnegative = TRUE`.

`tau` Log-precision parameter: residual variance is $\exp(-\tau)$.

`rho` Unconstrained optimization-scale parameter. The actual shared-site correlation is $\rho_{MLPE} = \text{plogis}(\rho)/2 \in (0, 0.5)$.

The mean structure is $S_{ij} = \alpha + \beta R_{ij} + e_{ij}$, where R_{ij} is the resistance distance derived from `E`. The residual vector e follows the MLPE correlation structure of Clarke et al. (2002): two pairs (i, j) and (i, k) sharing site i have correlation ρ_{MLPE}, while pairs sharing no site are uncorrelated.

Use `mlpe_covariates` to extend the mean structure with additional fixed pairwise covariates (isolation by environment). Coefficients describe conditional associations under the specified mean and correlation structure; they do not isolate causal environmental effects.

When the fitted internal correlation logit is below -8, `summary()` reports the shared-site correlation at zero and omits its Wald uncertainty. The parameter remains in the degrees of freedom because it was estimated; fixing it in advance would define a different model.

Value

When `phi` is missing, a list with elements `phi` (starting values), `lower`, and `upper` (parameter bounds). Otherwise a list containing:

`objective` Negative log-likelihood.

fitted	Matrix of expected genetic distances (same dimensions as S).
boundary	Logical; TRUE if beta = 0, indicating no detectable IBR signal.
gradient	Gradient with respect to phi (if gradient = TRUE).
hessian	Hessian matrix with respect to phi (if hessian = TRUE).
gradient_E	Gradient with respect to E (if partial = TRUE).
partial_E	Jacobian of gradient_E with respect to phi (if partial = TRUE).
partial_S	Jacobian of gradient with respect to the lower triangle of S (if partial = TRUE).
jacobian_E	Function for reverse-mode AD through E (if partial = TRUE).
jacobian_S	Function for reverse-mode AD through S (if partial = TRUE).

Worked example

vignette("mlpe-pairwise-distance", package = "terradish") fits and interprets MLPE models for a pairwise F_{ST} matrix.

References

Clarke RT, Rothery P, Raybould AF. 2002. Confidence limits for regression relationships between distance matrices: estimating gene flow with distance. Journal of Agricultural, Biological, and Environmental Statistics 7(3):361-372.

See Also

leastsquares, mlpe_covariates, generalized_wishart, terradish

Examples

```
library(terra)

data(melip)
melip.altitude <- terra::unwrap(melip.altitude)
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords <- terra::unwrap(melip.coords)

covariates <- c(melip.altitude, melip.forestcover)
names(covariates) <- c("altitude", "forestcover")
surface <- conductance_surface(covariates, melip.coords, directions = 8)

# inverse of graph Laplacian at null model (IBD)
laplacian_inv <- terradish_distance(theta = matrix(0, 1, 2),
                                    formula = ~forestcover + altitude,
                                    data = surface,
                                    terradish::loglinear_conductance,
                                    covariance = TRUE)$covariance[,,1]

mlpe(laplacian_inv, melip.Fst) #without 'phi': return MLE of phi
# Set the actual shared-site correlation to 0.2 with qlogis(2 * 0.2)
mlpe(laplacian_inv, melip.Fst,
```

```
phi = c(0, 0.5, -0.1, qlogis(2 * 0.2)))
```

`mlpe_covariates` *MLPE measurement model with pairwise endpoint-difference covariates*

Description

Creates an MLPE measurement model whose mean structure includes both an IBR term (resistance distance) and one or more fixed pairwise environmental covariates (IBE terms).

Usage

```
mlpe_covariates(
  x,
  coords = NULL,
  transform = c("absdiff", "sqdiff", "euclidean", "manhattan"),
  scale = FALSE,
  rho_start = 0.2
)
```

Arguments

`x`	Pairwise endpoint-difference covariates (the output of `pairwise_endpoint_covariates`), *or* site-level environmental data in any format accepted by that helper (numeric vector, matrix, data frame, or `terra::SpatRaster`).
`coords`	Optional focal-point coordinates, required when `x` is a raster. Accepts the same formats as `conductance_surface`.
`transform`	Pairwise transform applied when `x` is not already a `"terradish_pairwise_covariates"` object. Passed through to `pairwise_endpoint_covariates`.
`scale`	Logical. Standardize site-level covariates before the pairwise transform? Passed to `pairwise_endpoint_covariates`.
`rho_start`	Starting value for the MLPE correlation parameter ρ on its natural scale, which must lie in `(0, 0.5)`. The default `0.2` corresponds to a moderate shared-site correlation. Adjust if the optimizer fails to improve from the starting value.

Details

The fitted mean structure is:

$$S_{ij} = \alpha + \beta R_{ij} + Z_{ij}\gamma + e_{ij}$$

where R_{ij} is the resistance distance (IBR) and Z_{ij} is the row of pairwise endpoint-difference covariates for the pair (i, j) (IBE). The residuals follow the MLPE correlation structure of Clarke et al. (2002).

The γ coefficients are conditional on resistance distance, the other pairwise covariates, and the MLPE correlation model. They describe associations with the genetic-distance response. They do not cleanly separate IBE from IBR, establish causality, or identify a unique ecological mechanism when spatial predictors are correlated.

The nuisance parameters (intercept α, IBR slope β, IBE coefficients γ, log-precision `tau`, and the unconstrained correlation parameter `rho`) are all profiled out during optimization; only the conductance parameters in the formula are optimized by the outer loop. The actual MLPE shared-site correlation is $\text{plogis}(\rho)/2$.

When `x` is site-level data or the output of `pairwise_endpoint_covariates()`, the pairwise covariate matrix is stored as an attribute of the returned function. This allows `terradish_cv_folds` to rebuild the correct pairwise covariates for each train/test split automatically.

Unclassed matrices are interpreted as pairwise columns and must have $n(n-1)/2$ rows. Use a data frame for site-level columns, or `pairwise_endpoint_covariates` to construct their differences. `pairwise_covariates` combines these with other pairwise inputs.

Value

A function of class `"terradish_measurement_model"` suitable for use as the `measurement_model` argument of `terradish`, `terradish_grid`, and `terradish_cv_folds`.

Worked example

See `vignette("ibe-ibr-workflow", package = "terradish")` for paired environmental differences, genetic-response contrasts, and their distinction from covariance kernels.

Interpreting pairwise effects

Construct one `pairwise_covariates` object to use the same environmental transforms and geographic distances in both likelihood families. Their fitted conductance surfaces are conditional on these terms, and conductance signs can change when the mean structure changes. Report fits with and without pairwise effects. A uniform `~ 1` conductance model with pairwise terms represents IBD plus IBE, not IBE alone.

A positive environmental coefficient means differentiation increases with environmental difference conditional on resistance. It does not distinguish assortative mating, dispersal filtering, local adaptation, or site variance that rises at environmental extremes. MLPE coefficients are unconstrained; Wishart kernel weights are nonnegative. Compare their effects in resistance-distance units with `terradish_ibe_ratio`, which divides by the resistance coefficient and propagates joint uncertainty. Ratios are undefined when that denominator is zero.

References

Clarke RT, Rothery P, Raybould AF. 2002. Confidence limits for regression relationships between distance matrices: estimating gene flow with distance. Journal of Agricultural, Biological, and Environmental Statistics 7(3):361-372.

See Also

`mlpe`, `pairwise_endpoint_covariates`, `terradish_cv_folds`

Examples

```
library(terra)

data(melip)
melip.altitude <- terra::unwrap(melip.altitude)
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords <- terra::unwrap(melip.coords)

covariates <- c(terra::scale(melip.altitude), terra::scale(melip.forestcover))
names(covariates) <- c("altitude", "forestcover")
surface <- conductance_surface(covariates, melip.coords, directions = 8)

g_joint <- mlpe_covariates(melip.altitude, melip.coords,
                           transform = "absdiff", scale = TRUE)
fit_joint <- terradish(melip.Fst ~ altitude + forestcover, surface,
                   terradish::loglinear_conductance, g_joint)
summary(fit_joint)
```

---

mlpe_response_change *Response-scale change for MLPE pairwise covariates*

Description

Computes the expected change in the genetic-distance response associated with changing one or more pairwise covariates in an `mlpe_covariates` measurement model. This is useful for interpreting IBE coefficients, which are profiled nuisance parameters rather than conductance-surface coefficients.

Usage

```
mlpe_response_change(
  object,
  covariate = NULL,
  probs = c(0.1, 0.9),
  values = NULL,
  conf.level = 0.95
)
```

Arguments

`object`	A fitted `terradish` model whose measurement model was created with `mlpe_covariates`.
`covariate`	Optional character vector naming the pairwise covariates to summarize. Defaults to all pairwise covariates stored in the measurement model.
`probs`	Numeric vector of length two giving the lower and upper quantiles used to define the contrast when `values` is `NULL`. Defaults to the 10th and 90th percentiles.

values	Optional numeric vector of length two giving explicit lower and upper covariate values for a single covariate. Alternatively, provide a named list with one length-two numeric vector per covariate.
conf.level	Confidence level for the Wald confidence interval.

Details

For a model fitted with mlpe_covariates(), the pairwise mean structure includes terms of the form γZ_{ij}. This function reports $\gamma(z_{\text{high}} - z_{\text{low}})$, the expected response-scale change in genetic distance across the requested covariate contrast, conditional on the fitted conductance surface.

The interval uses the conditional standard error for the MLPE covariate coefficient from summary(object) and treats the selected covariate contrast as fixed. The reported contrast is a conditional association under the fitted MLPE model. It is not a causal effect or proof that IBE has been separated from IBR or other spatially correlated predictors.

Value

A data frame with one row per covariate and columns describing the low and high covariate values, the contrast, estimated response-scale change, confidence limits, and confidence level.

Worked examples

vignette("ibe-ibr-workflow", package = "terradish").

See Also

mlpe_covariates, pairwise_endpoint_covariates, terradish

Examples

```
data(melip)
altitude    <- terra::unwrap(melip.altitude)
forestcover <- terra::unwrap(melip.forestcover)
coords      <- terra::unwrap(melip.coords)

covariates <- c(altitude, forestcover)
names(covariates) <- c("altitude", "forestcover")
covariates <- scale_covariates(terra::aggregate(covariates, fact = 3,
                                                na.rm = TRUE))
surface <- conductance_surface(covariates, coords, directions = 8)

# site-level altitude, turned into a pairwise |difference| IBE predictor
site_altitude <- data.frame(altitude = terra::extract(altitude, coords)[, 2])
g_joint <- mlpe_covariates(site_altitude)

# joint isolation by environment and isolation by resistance
fit_joint <- terradish(melip.Fst ~ altitude + forestcover, data = surface,
                       conductance_model = loglinear_conductance,
                       measurement_model = g_joint)

# Expected change in genetic distance across the covariate's 10th-to-90th
```

```
# percentile spread, on the response scale rather than the log scale.
change <- mlpe_response_change(fit_joint)
change

# A narrower, interquartile contrast
mlpe_response_change(fit_joint, probs = c(0.25, 0.75))
```

`NewtonRaphsonControl` *Control settings for Newton-like optimizers*

Description

Tuning parameters shared by Newton-Raphson and BFGS quasi-Newton optimizers inside `terradish`.

Usage

```
NewtonRaphsonControl(
  maxit = 100,
  ctol = sqrt(.Machine$double.eps),
  ftol = sqrt(.Machine$double.eps),
  etol = 10 * .Machine$double.eps,
  verbose = FALSE,
  eps = 1e-08,
  del = 1,
  ls.control = HagerZhangControl()
)
```

Arguments

`maxit`	Maximum number of Newton or quasi-Newton steps. Increase if the optimizer reports a convergence warning on large or difficult problems.
`ctol`	Gradient convergence tolerance. Optimization stops when the largest absolute projected gradient is below `ctol`.
`ftol`	Objective convergence tolerance. Optimization also stops when the absolute change in the objective is below `ftol` and the largest absolute projected gradient is below `sqrt(ctol)`.
`etol`	Eigenvalue threshold used to detect near-singular Hessians. Absolute eigenvalues smaller than `etol` times the largest absolute Hessian entry are replaced by one when forming the Newton step. Increasing this value regularizes the step at the cost of slower convergence near flat regions.
`verbose`	Logical. If `TRUE`, print the iteration count, objective value, and gradient norm at each step.
`eps`	Box-constraint tolerance. A parameter is considered to be "on the boundary" when it is within `eps * abs(par)` of the bound. Active-constraint gradient components are zeroed before the step is taken.

`del`	Typical initial step length for quasi-Newton (BFGS) methods. Used to scale the initial approximate Hessian before any updates are available.
`ls.control`	Control object for the line search, created by `HagerZhangControl` (default) or `ArmijoControl`. `ArmijoControl` is only valid when `optimizer = "bfgs"` is passed to `terradish`.

Details

The projected gradient sets outward-pointing components at active bounds to zero. A sufficiently small projected gradient gives convergence code 0. A small objective change also gives code 0 if the projected gradient is below `sqrt(ctol)`. Otherwise the optimizer warns that it has stalled and returns code 2. Reaching the iteration limit gives code 1.

For most resistance-surface models the defaults converge reliably. Consider changing them when:

- The optimizer hits `maxit` before converging: increase `maxit`.
- You want a quick exploratory fit: reduce `maxit` (for example 5 to 10).
- Fitting is slow because of oscillation: tighten `ctol` or switch to a more conservative line search via `ls.control`.

Value

A named list of class `"NewtonRaphsonControl"` holding the settings passed to the optimizer. The elements are the arguments above, validated and stored unchanged: `maxit`, `ctol`, `ftol`, `etol`, `verbose`, `eps`, `del`, and `ls.control` (itself a `HagerZhangControl` or `ArmijoControl` list). Pass the object to the `control` argument of `terradish`; inspect it with `str()`. The list carries no fitted quantities, so nothing in it needs back-transforming.

See Also

`HagerZhangControl`, `ArmijoControl`, `terradish`

Examples

```
ctrl <- NewtonRaphsonControl(maxit = 25, verbose = FALSE)
str(ctrl)

# Relaxed tolerances for a quick exploratory run:
ctrl_quick <- NewtonRaphsonControl(maxit = 10, ctol = 1e-3, ftol = 1e-3)


data(melip)
covariates <- c(terra::unwrap(melip.altitude), terra::unwrap(melip.forestcover))
names(covariates) <- c("altitude", "forestcover")
covariates <- scale_covariates(terra::aggregate(covariates, fact = 3,
                                                na.rm = TRUE))
surface <- conductance_surface(covariates, terra::unwrap(melip.coords),
                               directions = 8)

# A tighter, longer-running control than the default
fit <- terradish(melip.Fst ~ forestcover + altitude, data = surface,
```

```
                conductance_model = loglinear_conductance,
                measurement_model = mlpe,
                control = NewtonRaphsonControl(maxit = 200, ctol = 1e-8))
coef(fit)

# The same fit, reporting each optimizer step. `verbose` is the simple
# switch; it overrides whatever `control` says.
fit_loud <- terradish(melip.Fst ~ forestcover + altitude, data = surface,
                      conductance_model = loglinear_conductance,
                      measurement_model = mlpe,
                      control = ctrl_quick, verbose = TRUE)
```

`ohio_ponds` *Simulated Ohio Forest-Pond Genetic Data*

Description

Matched SNP and microsatellite genotypes for 20 retained diploid animals in each of 24 hypothetical forest ponds on a real southeastern Ohio landscape. The neutral SLiM simulation was motivated by a forest pond-breeding salamander. Pond locations, hydroperiods, demographic parameters, and genetic data are simulated. Elevation and land cover are derived from public USGS products.

Usage

```
data(ohio_ponds)
```

Format

A list with 12 elements:

`covariates` A packed `terra::SpatRaster` with 102 rows and 102 columns at 100 m resolution in NAD83 / UTM zone 17N (EPSG:26917). Layers are `forest` (fraction from 0 to 1), `slope` (degrees), and `elevation` (meters).

`forest500` A packed raster on the same grid containing forest fraction smoothed with a Gaussian kernel of standard deviation 500 m. This is the context layer used to generate pond exchange.

`ponds` A 24-row data frame in population order. `pond` is the identifier; `x` and `y` are UTM coordinates in meters; `cell` is the 100 m raster cell; `slope` is in degrees; `edge_m` is distance to the domain edge in meters; `forest_buffer_100m`, `forest_buffer_300m`, and `forest_buffer_500m` are cell-center forest fractions in circular buffers of the indicated radius. `hydroperiod` is a hypothetical pond covariate in days with no simulated genetic effect; `nlcd_30m` is the native land-cover code at the pond; `displacement_from_trace25_m` and `candidate_count_at_selection` record the location-screening procedure; `focal` flags a sampled pond; `fold` gives three spatial validation blocks.

`samples` A 480-row data frame in genotype row order. `individual` and `pond` are identifiers; `population` is the one-based pond index; `pedigree_id`, `parent_1`, and `parent_2` are SLiM pedigree identifiers (`-1` denotes an unavailable parent); `x` and `y` are the pond coordinates.

`raw_samples` The same columns for 780 initial and additional captures before pedigree screening. Sampling began with 30 per pond, with additional captures where necessary to retain 20.

`snp` A 480 by 2500 integer matrix of alternate-allele dosage (0, 1, or 2), with individual row names and locus column names.

`raw_snp` A 780 by 2500 dosage matrix for the raw captures, using exactly the same loci as `snp`.

`snp_positions` The 2500 SNP positions in base pairs along a 50 Mb simulated chromosome, in column order.

`microsat` A 480 by 48 integer matrix of repeat counts at 24 stepwise-mutation loci. Adjacent columns hold the two alleles at one locus. Row names match `snp`; values are repeat counts, not dosage.

`raw_microsat` A 780 by 48 matrix for the raw captures at the same microsatellite loci.

`microsat_loci` A length-48 character vector assigning each microsatellite allele column to its locus.

`simulation` A named list recording simulation and mutation seeds, the 1200 compressed generations, demographic and marker parameters, the known 500 m forest context, the raw log-conductance coefficient of 6, the maximum pond-exchange probability of 0.025, the null hydroperiod effect, and the assumptions used in the teaching analyses.

Details

Call `terra::unwrap()` on the two packed raster elements before use. No downloads or simulation software are needed to analyze the bundled data. All retained individuals have both marker panels from the same ancestry. Screening removes known immediate pedigree relationships within ponds; this is not a claim that the retained animals are unrelated at deeper generations or that such relationships would be known in field samples.

Ponds were screened for native forest, at least 90 percent forest within 100 m, and at least 60 percent within 300 m. These are hypothetical design rules, not estimated habitat thresholds. Exchange has strong reproductive philopatry with a long tail among ponds. The exchange model uses endpoints on a resistance graph; it does not simulate observed animal travel paths.

The SNP panel was filtered for pooled retained-sample minor allele frequency of at least 0.05. Locus spacing does not establish independent loci. The effective degrees of freedom used in a Wishart example is an assumption, distinct from the SNP count. Spatial validation conditions on this globally ascertained panel, rather than validating marker discovery itself.

The source history was specified before fitting. Other histories and SNP panel sizes were checked during development; the bundled history was not selected for the strongest fit. The simulation illustrates an estimable mechanism, not calibration or validation of a species-specific demographic model. See the installed provenance file for sources and processing details.

Worked examples

`vignette("data-generation", package = "terradish")`, `vignette("getting-started", package = "terradish")`.

Source

USGS 3D Elevation Program and USGS Annual National Land Cover Database 2023 (Collection 1.0), processed to a common projected grid. Source URLs, hashes, processing, and simulation provenance are recorded in `system.file("extdata", "ohio-ponds", "PROVENANCE.md", package = "terradish")`.

See Also

cov_from_biallelic, cov_from_genetic_data, conductance_surface, `vignette("getting-started", package = "terradish")`, `vignette("gaussian-scale-optimization", package = "terradish")`.

Examples

```
data(ohio_ponds)
covariates <- terra::unwrap(ohio_ponds$covariates)
dim(ohio_ponds$snp)       # individuals by SNPs
dim(ohio_ponds$microsat)  # individuals by two allele columns per locus
table(ohio_ponds$samples$pond)
terra::res(covariates)   # map units are meters

# Aggregate diploid dosage to counts of alternate alleles in each pond.
groups <- factor(ohio_ponds$samples$pond, levels = ohio_ponds$ponds$pond)
Y <- rowsum(ohio_ponds$snp, groups, reorder = FALSE)
Y <- Y[ohio_ponds$ponds$pond, , drop = FALSE]
N <- 2 * as.numeric(table(groups))
S <- cov_from_biallelic(Y, N) / N[1]
round(S[1:4, 1:4], 3)
```

---

pairwise_covariates *Combine environmental and other pairwise covariates*

Description

Combines differences calculated from site attributes with user-supplied pairwise matrices, such as geographic distance, in one measurement-model input. Values retain their supplied units and scaling.

Usage

```
pairwise_covariates(...)
```

Arguments

`...`	Objects from `pairwise_endpoint_covariates()`, or named symmetric numeric matrices or `dist` objects. Matrices must have a zero diagonal. Every input must describe the same sites in the same order.

Details

No centering, scaling, or geometric correction is applied. For a Wishart model, each supplied distance must additionally produce a positive semidefinite centered kernel; the measurement-model constructor checks it. Site labels, when supplied on square matrices, must agree across inputs.

Value

A classed numeric matrix with one row per unordered site pair and one column per covariate. Rows follow the lower-triangle order used by R's `dist()`. Both `mlpe_covariates()` and `wishart_covariates()` accept this object.

Interpreting pairwise effects

Construct one `pairwise_covariates` object to use the same environmental transforms and geographic distances in both likelihood families. Their fitted conductance surfaces are conditional on these terms, and conductance signs can change when the mean structure changes. Report fits with and without pairwise effects. A uniform `~ 1` conductance model with pairwise terms represents IBD plus IBE, not IBE alone.

A positive environmental coefficient means differentiation increases with environmental difference conditional on resistance. It does not distinguish assortative mating, dispersal filtering, local adaptation, or site variance that rises at environmental extremes. MLPE coefficients are unconstrained; Wishart kernel weights are nonnegative. Compare their effects in resistance-distance units with `terradish_ibe_ratio`, which divides by the resistance coefficient and propagates joint uncertainty. Ratios are undefined when that denominator is zero.

See Also

`pairwise_endpoint_covariates`, `mlpe_covariates`, `wishart_covariates`

Examples

```
climate <- pairwise_endpoint_covariates(data.frame(temp = c(4, 7, 9)))
geography <- dist(cbind(c(0, 1, 3), c(0, 2, 1)))
z <- pairwise_covariates(climate, geographic = geography)
colnames(z)
```

`pairwise_endpoint_covariates` *Pairwise endpoint-difference covariates*

Description

Constructs pairwise environmental-mismatch covariates from site-level values. The result can be used alongside isolation-by-resistance (IBR) terms to model isolation by environment (IBE) within `mlpe_covariates`.

Usage

```
pairwise_endpoint_covariates(
  x,
  coords = NULL,
  transform = c("absdiff", "sqdiff", "euclidean", "manhattan"),
  scale = FALSE
)
```

Arguments

x	Site-level covariates. Supported inputs: • A numeric vector (one covariate per site). • A numeric matrix or data frame (sites in rows, covariates in columns). • A terra::SpatRaster: values are extracted at coords.
coords	Required when x is a raster. Focal-point coordinates in the same projection as x; accepts the same inputs as conductance_surface.
transform	How to collapse site-level values into a pairwise environmental-mismatch measure. "absdiff" Absolute difference between endpoints: $\|x_i - x_j\|$. One output column per covariate. Symmetric; suitable when the hypothesis concerns an association between environmental dissimilarity and differentiation, regardless of direction. "sqdiff" Squared difference: $(x_i - x_j)^2$. One output column per covariate. Like "absdiff" but penalizes large differences more heavily. "euclidean" Euclidean distance across all covariates: $\sqrt{\sum_k (x_{ik} - x_{jk})^2}$. Collapses multivariate data to a single dissimilarity measure. "manhattan" Manhattan (city-block) distance: $\sum_k \|x_{ik} - x_{jk}\|$. Collapses to a single measure; less sensitive to outliers than Euclidean.
scale	Logical. If TRUE, standardize (z-score) site-level covariates before computing the pairwise transform. Recommended when covariates are on different scales.

Details

The function produces a $n(n-1)/2 \times p$ matrix of pairwise endpoint-difference covariates, where n is the number of focal sites and p is the number of output columns (one per input covariate for "absdiff" and "sqdiff"; one for "euclidean" and "manhattan"). Rows correspond to the lower triangle of the pairwise distance matrix, scanned column by column.

The returned object has class "terradish_pairwise_covariates". This class carries the original site-level covariate matrix and the chosen transform as attributes, so that terradish_cv_folds can automatically rebuild the correct pairwise covariates for each train/test split.

If x already has class "terradish_pairwise_covariates", it is returned unchanged (pass-through).

Value

A numeric matrix of class "terradish_pairwise_covariates" with one row per unordered site pair and named columns reflecting the chosen transform (e.g. "absdiff_altitude", "euclidean").

Interpreting pairwise effects

Construct one pairwise_covariates object to use the same environmental transforms and geographic distances in both likelihood families. Their fitted conductance surfaces are conditional on these terms, and conductance signs can change when the mean structure changes. Report fits with and without pairwise effects. A uniform ~ 1 conductance model with pairwise terms represents IBD plus IBE, not IBE alone.

A positive environmental coefficient means differentiation increases with environmental difference conditional on resistance. It does not distinguish assortative mating, dispersal filtering, local adaptation, or site variance that rises at environmental extremes. MLPE coefficients are unconstrained; Wishart kernel weights are nonnegative. Compare their effects in resistance-distance units with terradish_ibe_ratio, which divides by the resistance coefficient and propagates joint uncertainty. Ratios are undefined when that denominator is zero.

See Also

mlpe_covariates, terradish_cv_folds, terradish

Examples

```
library(terra)

data(melip)
melip.altitude <- terra::unwrap(melip.altitude)
melip.coords <- terra::unwrap(melip.coords)

z_abs <- pairwise_endpoint_covariates(melip.altitude, melip.coords,
                                      transform = "absdiff", scale = TRUE)
head(z_abs)

site_env <- data.frame(
  altitude = scale(as.numeric(terra::extract(melip.altitude, melip.coords)[, 2])),
  xcoord = scale(terra::geom(melip.coords)[, "x"])
)
z_env <- pairwise_endpoint_covariates(site_env, transform = "euclidean")
head(z_env)
```

---

pca_dist *Pairwise genetic distances from PCA scores*

Description

Runs a PCA on allele-frequency data from an adegenet::genind object and returns Euclidean distances computed on the retained principal component scores.

Usage

```
pca_dist(gi, n_axes = 64, scale = TRUE)
```

Arguments

gi	A genind object from **adegenet**.
n_axes	Number of principal component axes to retain. Capped at the total number of available axes.
scale	Logical. If TRUE (default), the resulting distance matrix is rescaled to [0, 1] via scale_to_0_1.

Details

Missing genotypes in gi are imputed by the allele-mean method (equivalent to adegenet::tab(gi, NA.method = "mean")), then a standard PCA via prcomp is applied to the allele frequency table. Euclidean distances are then computed on the first n_axes scores.

This is a convenience wrapper designed for users who prefer to work from a genind object. For biallelic SNP data the more direct dist_from_biallelic is generally faster and does not require the **adegenet** package.

Value

A symmetric numeric distance matrix with one row/column per individual in gi.

See Also

dist_from_biallelic, cov_from_genetic_data

Examples

```
if (requireNamespace("adegenet", quietly = TRUE)) {
  data(nancycats, package = "adegenet")
  d <- pca_dist(nancycats, n_axes = 4)
  dim(d)
}
```

---

plot.terradish *Plot methods for fitted terradish models*

Description

Five plot types for objects returned by terradish:

"fit" Observed vs. fitted response values. Distance-response models are shown as pairwise genetic distance against fitted resistance distance; covariance-response models such as wishart_covariance are shown as observed genetic covariance against fitted genetic covariance.

"surface" Fitted conductance surface with asymptotic confidence interval bounds, displayed in three side-by-side panels.

`"marginal"` Marginal association of each raster covariate with conductance, varying one covariate at a time while averaging over the observed values of all other model covariates, with a 95% pointwise confidence band via the delta method. For Gaussian scale-aware conductance models, these curves are conditional on the fitted `sigma` values and are shown on the smoothed covariate scale. For spline-expanded conductance models from `smooth_loglinear_conductance`, panels are shown against the original covariate named inside each `s()` term rather than against the individual basis columns.

`"marginal_response"` Approximate response-scale marginal associations obtained by mapping the marginal conductance curve through the fitted measurement-model mean and adding predictive bands that combine `theta` uncertainty, conditional `phi` uncertainty, and the residual variance implied by `tau` for distance-response models. Covariance-response fits from `wishart_covariance` are shown as the average fitted genetic covariance over the modeled lower triangle. For Gaussian scale-aware conductance models, the predictive bands are likewise conditional on the fitted `sigma` values. Each curve sets one covariate to a common value in every cell. In a model with a single covariate this makes conductance uniform, so the curve is the `"marginal"` conductance curve inverted and rescaled; the plot is most informative when other covariates still vary across the landscape.

`"sigma"` For Gaussian scale-aware conductance models, plots the fitted Gaussian kernel against distance for each `sigma` term, with a dashed line at the fitted distance containing 90% of the one-dimensional kernel mass and a shaded interval from the Wald confidence interval of that kernel-mass distance. These distances summarize the fitted smoothing kernel and are not movement or dispersal distances.

Usage

```
## S3 method for class 'terradish'
plot(
  x,
  type = c("marginal_response", "fit", "surface", "marginal", "sigma"),
  data = NULL,
  covariates = NULL,
  marginal_covariates = NULL,
  conductance_model = NULL,
  distance_per_map_unit = NULL,
  distance_unit = NULL,
  quantile = 0.95,
  n = NULL,
  support = c("none", "focal"),
  support_probs = c(0, 1),
  clamp_covariates = NULL,
  ...
)
```

Arguments

`x`	A fitted `terradish` object.
`type`	One of `"marginal_response"` (default), `"fit"`, `"surface"`, `"marginal"`, or `"sigma"`.

`data` The `terradish_graph` used when fitting `x`. Required for `type = "surface"`, `type = "marginal"`, and `type = "marginal_response"`.

`covariates` Optional `SpatRaster` of covariates on their **original (unscaled)** scale. When supplied for `type = "marginal"`, the x-axis of each panel is back-transformed to the original units using the corresponding per-layer offset and divisor. Layer names must match the covariate names used in the model formula.

`marginal_covariates` Optional character vector used with `type = "marginal"` or `type = "marginal_response"` to plot only selected covariate panels. Values should usually be raw formula variable names such as `"altitude"`; displayed panel labels are also accepted.

`conductance_model` Optional conductance model factory used when fitting `x`. When omitted, `plot()` first tries to reuse the fitted conductance model stored in `x`; if that is unavailable, it falls back to `loglinear_conductance`. Gaussian scale-aware fits reuse the fitted conductance model automatically.

`distance_per_map_unit` Optional positive scalar used by `type = "sigma"` to convert map units into a user-facing distance unit such as kilometers.

`distance_unit` Optional label used with `distance_per_map_unit` for `type = "sigma"`.

`quantile` Confidence level for interval bands. Default `0.95`.

`n` Number of evaluation points for each marginal-association curve. Defaults to `100` for `"marginal_response"` (each point requires a full Laplacian solve) and `200` for `"marginal"`. Supply an explicit integer to override.

`support` Optional support constraint used for `type = "surface"`, `"marginal"`, and `"marginal_response"`. Use `"none"` (default) to evaluate over the full graph covariate domain, or `"focal"` to constrain selected covariates to the empirical support at focal sampling cells (`data$demes`).

`support_probs` Optional length-2 probability vector used with `support = "focal"` to define clamping bounds from focal-cell quantiles. The default `c(0, 1)` uses the focal min/max; for robust trimming, values like `c(0.01, 0.99)` are often useful.

`clamp_covariates` Optional character vector of covariate names to clamp when `support = "focal"`. Defaults to all numeric covariates used by each plot type.

`...` Additional graphical parameters forwarded to `plot` (for `"fit"`) or ignored for the other types.

Details

For `type = "surface"`, `"marginal"`, and `"marginal_response"`, optional support-constrained evaluation can be requested with `support = "focal"` to clamp selected covariates to empirical focal-site support before prediction.

Value

A plot object that is returned visibly so base R auto-printing works as expected:

- `"fit"`: a ggplot object with observed-vs-fitted pairwise distances or genetic covariance, according to the response model.
- `"surface"`, `"marginal"`, `"marginal_response"`, and `"sigma"`: either a single ggplot object (one panel) or a named list of ggplot objects (multiple panels), with one plot per panel/covariate.

Worked examples

`vignette("getting-started", package = "terradish")`, `vignette("gaussian-scale-optimization", package = "terradish")`, `vignette("spline-conductance", package = "terradish")`.

See Also

terradish, conductance, fitted.terradish, loglinear_conductance, smooth_loglinear_conductance

Examples

```
data(melip)
melip.altitude    <- terra::unwrap(melip.altitude)
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords      <- terra::unwrap(melip.coords)

# Build surface with scaled covariates. The raster is coarsened so the
# example runs quickly; use the full resolution in a real analysis.
covariates_scaled <- c(melip.altitude, melip.forestcover)
names(covariates_scaled) <- c("altitude", "forestcover")
covariates_scaled <- terra::aggregate(covariates_scaled, fact = 3, na.rm = TRUE)
covariates_scaled <- scale_covariates(covariates_scaled)
surface <- conductance_surface(covariates_scaled, melip.coords,
                               directions = 8, saveStack = TRUE)

fit <- terradish(melip.Fst ~ altitude + forestcover, surface,
                 loglinear_conductance, mlpe)

# Observed vs. fitted pairwise genetic distances
plot(fit, type = "fit")

# Fitted conductance surface with 95% CI (three-panel figure)
plot(fit, type = "surface", data = surface)

# Marginal-association plots on the original covariate scale
plot(fit, type = "marginal", data = surface)

# Approximate response-scale marginal associations with predictive bands
plot(fit, type = "marginal_response", data = surface)

# For Gaussian scale-aware fits, visualize the fitted sigma kernels
# plot(fit_gaussian, type = "sigma")
```

`print.terradish_covariance_power` *Print a covariance-response power assessment*

Description

Compact display for the object returned by `covariance_response_power`: the settings the sweep ran under, followed by the scenario-level summary table.

Usage

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

Arguments

`x`	An object of class `terradish_covariance_power`, from `covariance_response_power`.
`...`	Ignored, present for compatibility with the `print` generic.

Details

How to read the output. The header records what the sweep did: how many simulated data sets per design (`nsim`), which sample sizes, and which site-selection strategies. Judge the numbers below it against that `nsim`: with a small `nsim` the Monte Carlo error on a rate is substantial, so treat a demonstration run as indicative only.

The table beneath is `x$summary`, one row per scenario (strategy by sample size by fitted model). `fit_rate` is the proportion of simulations that produced a usable fit; a value well below 1 can reflect weak information, numerical difficulty, boundary fits, or a candidate model that is poorly matched to the scenario. `conductance_power` is the proportion exceeding the specified conductance-correlation threshold, `mean_conductance_cor` summarizes correlation with the generating conductance surface, and the model-selection columns report how often each candidate had the lowest AIC or AICc among usable fits.

Per-coefficient results (power, bias, RMSE, and interval coverage) are not printed, because there is one row per parameter per scenario. Read them from `x$parameter_summary`. Power there is the proportion of simulations in which the coefficient was both correctly signed and significant, so it is the quantity to compare against a target such as 0.8.

Value

`x`, invisibly. Called for the printed output.

See Also

`covariance_response_power` for the sweep itself and what each returned component contains, `simulate_covariance_response` for a single simulated data set, and `terradish` for fitting one. See `vignette("simulation-design", package = "terradish")` for a worked study-design walk-through.

Examples

```
data(melip)
covariates <- c(terra::unwrap(melip.altitude), terra::unwrap(melip.forestcover))
names(covariates) <- c("altitude", "forestcover")
# coarsened, and only a handful of simulations, so the example is quick
covariates <- scale_covariates(terra::aggregate(covariates, fact = 3,
                                                na.rm = TRUE))
surface <- conductance_surface(covariates, terra::unwrap(melip.coords),
                               directions = 8, saveStack = TRUE)

power <- covariance_response_power(
  theta        = c(forestcover = 0.5, altitude = -0.4),
  formula      = ~ forestcover + altitude,
  data         = surface,
  sample_sizes = c(8, 12),
  strategies   = "spacefill",
  tau = 1, sigma = 0.2, nu = 100,
  nsim = 3,          # raise to 200+ for a real power estimate
  seed = 1,
  control = NewtonRaphsonControl(maxit = 8)
)

power                     # settings header plus the scenario summary
power$parameter_summary   # per-coefficient power, bias, RMSE, coverage
```

---

print.terradish_setting_assessment

Print a terradish settings assessment

Description

Compact display for the object returned by terradish_assess_settings: the graph profile, the recommended computational settings and why each was chosen, the probe timings behind them, and any warnings the assessment raised.

Usage

```
## S3 method for class 'terradish_setting_assessment'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

x	An object of class terradish_setting_assessment, from terradish_assess_settings.
digits	Number of significant digits used when formatting the probe timing tables.
...	Ignored, present for compatibility with the print generic.

Details

How to read the output. It comes in four parts.

Graph reports the size of the problem: vertices (active raster cells), edges, focal sites, and conductance parameters. This is the number that decides whether any of the rest matters. Below roughly 50,000 vertices the defaults are almost always fine; above a few hundred thousand, solver choice dominates the runtime.

Recommended settings lists the optimizer, line search, solver, and approximation the probes favored, each followed by a parenthesized reason. Pass them straight through with `fit <- terradish(..., optimizer = rec$optimizer, control = rec$control, solver = rec$solver)`, where `rec <- assessment$recommende` These choices change only the computational path, never the estimates.

Defaults comparison is the one-line verdict: whether the assessment agreed with the package defaults. If it says it matched, there is nothing to do and you can drop the assessment from your script.

Probes are the measurements behind the recommendation. The optimizer table gives elapsed time and the log-likelihood each candidate reached in the probe budget, so a candidate that is fast but reaches a worse log-likelihood is not actually better. The solver table shows the three fastest direct factorizations split into setup and solve time; setup dominates when a fit re-factorizes often, solve time when the focal set is large. *Notes* collects anything that needs attention, such as a solver that failed to converge during probing.

Value

`x`, invisibly. Called for the printed output. The machine readable pieces are `x$profile`, `x$recommended`, `x$reasons`, `x$benchmarks`, `x$comparison`, and `x$notes`.

See Also

`terradish_assess_settings` for the assessment itself and what each component holds, `terradish_solver_benchmark` for benchmarking solvers alone, and `terradish` for the fit the recommendations feed. See `vignette("large-landscapes", package = "terradish")` for when this is worth running.

Examples

```
data(melip)
covariates <- c(terra::unwrap(melip.altitude), terra::unwrap(melip.forestcover))
names(covariates) <- c("altitude", "forestcover")
# coarsened for the example; assessment earns its keep on large graphs
covariates <- scale_covariates(terra::aggregate(covariates, fact = 3,
                                                na.rm = TRUE))
surface <- conductance_surface(covariates, terra::unwrap(melip.coords),
                               directions = 8, saveStack = TRUE)

assessment <- terradish_assess_settings(
  melip.Fst ~ forestcover + altitude,
  data = surface, measurement_model = mlpe, probe_maxit = 2)

assessment                      # the four-part report described above
assessment$comparison$summary   # one-line verdict against the defaults
assessment$recommended$solver   # the settings, ready to reuse
```

scale_covariates *Scale raster covariates and retain original-scale metadata*

Description

Transforms each layer of a SpatRaster and stores the original-scale offset and divisor as a "terradish_scale" attribute. When the scaled raster is used to build a conductance_surface, marginal-association plots (plot(fit, type = "marginal", data = surface)) can automatically back-transform x-axes to the original covariate units.

Usage

```
scale_covariates(
  covariates,
  method = c("zscore", "minmax"),
  center = TRUE,
  scale = TRUE,
  reference = NULL
)
```

Arguments

covariates	A SpatRaster or PackedSpatRaster whose layers should be scaled.
method	Scaling method applied to each layer independently. "zscore" (default) subtracts the layer mean and divides by the standard deviation, giving zero mean and unit variance. "minmax" maps the minimum to 0 and the maximum to 1.
center	Logical. Only used when method = "zscore". If TRUE (default), subtract the layer mean before dividing by the standard deviation.
scale	Logical. Only used when method = "zscore". If TRUE (default), divide by the layer standard deviation. If FALSE with center = TRUE, layers are centered but not divided.
reference	A previous scaled raster or its terradish_scale attribute. Applies the stored centers and scales by layer name instead of estimating new values. When supplied, method, center, and scale do not change the stored transformation.

Details

Scaling is strongly recommended before fitting conductance models. In the loglinear model, conductance is $\exp(\theta \cdot x)$. When x has a large range (e.g. altitude in meters), a small change in θ produces enormous conductance differences, making the objective surface poorly conditioned and the optimizer slow to converge.

Missing values (NA) are excluded from the computation of the mean, standard deviation, minimum, and range, and are left as NA in the output.

If a layer is constant (all values identical), the scale is set to 1 to avoid division by zero; the layer is then returned unchanged (or centered to 0 if `center = TRUE`).

The `"terradish_scale"` attribute is a named list with one element per layer; each element is a named numeric vector with entries `center` and `scale` (the subtracted offset and the divisor, respectively). `crop_to_focal_buffer` and `conductance_surface` preserve this attribute.

Value

A `SpatRaster` with transformed values and a `"terradish_scale"` attribute.

See Also

`scale_to_0_1`, `conductance_surface`

Examples

```
library(terra)
r <- c(terra::rast(nrows = 2, ncols = 2, vals = c(100, 200, 300, NA)),
       terra::rast(nrows = 2, ncols = 2, vals = c(0.2, 0.5, 0.4, NA)))
names(r) <- c("altitude", "forestcover")

scaled_r <- scale_covariates(r)
attr(scaled_r, "terradish_scale")  # per-layer center and scale

# min-max scaling to [0, 1]
scaled_mm <- scale_covariates(r, method = "minmax")

# Preserve the training transformation on new values, matched by layer name.
new_scaled <- scale_covariates(r + 10, reference = scaled_r)
terra::values(new_scaled)
# Values can fall outside the training range; they are not re-centered.
```

`scale_to_0_1` *Rescale values to the unit interval*

Description

Rescales numeric values so the minimum non-missing value maps to 0 and the maximum maps to 1. `terra::SpatRaster` inputs are rescaled layer-by-layer using only the finite (non-`NA`) values.

Usage

```
scale_to_0_1(x)
```

Arguments

`x`	Numeric vector, matrix, `dist` object, or `terra::SpatRaster`. `PackedSpatRaster` objects (from `terra::wrap()`) are unwrapped automatically.

Details

The transformation applied to each element is $(x - \min)/(\max - \min)$, computed over non-NA values. NA values are propagated unchanged.

If all non-missing values are identical (constant layer or vector), the output is set to 0 for all non-missing entries rather than producing NaN from a zero-range denominator.

For rasters, each layer is rescaled independently using its own minimum and maximum.

Value

An object of the same class as x with values in [0, 1] (non-NA entries only).

See Also

scale_covariates for standardized (z-score or min-max) scaling with metadata retained for back-transformation.

Examples

```
scale_to_0_1(c(2, 4, 6))
scale_to_0_1(c(5, 5, NA))  # constant non-NA values become 0

library(terra)
r <- terra::rast(nrows = 2, ncols = 2, vals = c(1, 3, 5, NA))
scale_to_0_1(r)
```

---

simulate_covariance_response

Simulate covariance responses from a conductance surface

Description

Simulates one or more covariance response matrices from a known conductance-resistance relationship, using the same Wishart covariance model fitted by wishart_covariance.

Usage

```
simulate_covariance_response(
  theta,
  formula,
  data,
  conductance_model = loglinear_conductance,
  tau = 1,
  sigma = 0,
  nu,
  nsim = 1,
  seed = NULL,
  cores = 1L
)
```

Arguments

theta	Conductance parameters. May be supplied as a numeric vector or a one-row matrix. If unnamed, values are matched in the order implied by `formula`. Gaussian `sigma.<layer>` parameters use the map units reported by `coef()`; conversion to optimizer units is internal.
formula	Model formula specifying the conductance covariates. The left hand side, if supplied, is ignored.
data	A `conductance_surface` object.
conductance_model	Conductance-model factory, such as `loglinear_conductance`.
tau	Nonnegative scaling applied to the conductance-implied covariance matrix.
sigma	Nonnegative nugget variance added to the diagonal.
nu	Effective Wishart degrees of freedom: controls the amount of sampling noise in the simulated covariance: larger `nu` gives tighter draws around `Sigma`. This is a model assumption, not a marker count or an estimate of information from biological data. Examine a range of plausible values. The simulation requires `nu` to be at least the covariance-matrix dimension. See `wishart_covariance` for how `nu` propagates into estimation (it scales standard errors and model-selection statistics but not point estimates).
nsim	Number of covariance matrices to simulate.
seed	Optional random seed.
cores	Number of cores passed to `terradish_distance`.

Details

The helper first computes the conductance-implied covariance matrix `E(theta)` from the supplied resistance surface. It then forms `Sigma = tau * E(theta) + sigma * I` and simulates `S ~ Wishart(nu, Sigma) / nu`. This makes the returned covariance matrices directly compatible with `wishart_covariance`. These results are internal to the assumed model. Simulating and fitting at the same incorrect `nu` cannot reveal that error. Use `covariance_response_power` with different `nu` and `nu_fit` to examine its effect on coverage.

Value

A list containing:

covariance	A covariance matrix if `nsim = 1`, otherwise a three-dimensional array of simulated covariance matrices.
Sigma	The covariance matrix used as the Wishart scale matrix after applying `tau` and `sigma`.
E	The conductance-implied covariance matrix returned by `terradish_distance(covariance = TRUE)`.
theta	The validated conductance parameter matrix used for simulation.
tau	The supplied `tau`.
sigma	The supplied `sigma`.
nu	The supplied `nu`.

Examples

```
r1 <- terra::rast(nrows = 3, ncols = 3, vals = 1:9)
r2 <- terra::rast(nrows = 3, ncols = 3, vals = c(9, 1, 4, 3, 8, 2, 5, 7, 6))
covariates <- c(r1, r2)
names(covariates) <- c("x1", "x2")
pts <- terra::vect(matrix(c(0.5, 0.5,
                            1.5, 1.5,
                            2.5, 2.5), ncol = 2, byrow = TRUE),
                   type = "points")
surface <- conductance_surface(covariates, pts, directions = 4)
sim <- simulate_covariance_response(
  theta = c(x1 = 0.3, x2 = -0.2),
  formula = ~ x1 + x2,
  data = surface,
  tau = 0.8,
  sigma = 0.5,
  nu = 20,
  seed = 1
)
sim$covariance
```

`slim_terradish` *Reduce the storage size of a fitted terradish model*

Description

Removes components that are useful for prediction or diagnostics but can make a fitted `terradish` object expensive to serialize. The fitted coefficients, likelihood, Hessian, fitted response, convergence diagnostics, and approximation metadata are retained unless explicitly dropped.

Usage

```
slim_terradish(object, drop = c("submodels", "leverage"))
```

Arguments

`object`	A fitted `terradish` object.
`drop`	Character vector naming optional components to remove. Supported values are `"submodels"`, `"response"`, and `"leverage"`. Dropping `"submodels"` removes the fitted conductance and measurement-model closures, which usually account for most of the serialized object size. Dropping `"response"` also removes the response copy stored in the comparison contract.

Details

A slim object remains suitable for `print()`, `summary()`, `coef()`, `vcov()`, `confint()`, `logLik()`, `AIC()`, and fitted-value extraction and residual-permutation simulation. Operations that must reevaluate the conductance or measurement model, including conductance- surface prediction and most plot types, require the retained model closures and will stop with an informative error if those closures were dropped.

Slimming is irreversible. Retain the original fit, model formula, graph, and response when later prediction, simulation, or refitting may be needed.

Value

A fitted `terradish` object with a `storage` element that records the removed components and the object's size before and after slimming.

Examples

```
data(melip)
altitude <- terra::unwrap(melip.altitude)
forestcover <- terra::unwrap(melip.forestcover)
coords <- terra::unwrap(melip.coords)
covariates <- c(altitude, forestcover)
names(covariates) <- c("altitude", "forestcover")
surface <- conductance_surface(covariates, coords)
fit <- terradish(melip.Fst ~ altitude, surface,
                 measurement_model = leastsquares,
                 control = NewtonRaphsonControl(maxit = 2))
fit_slim <- slim_terradish(fit)
object.size(fit_slim) <= object.size(fit)
coef(fit_slim)
```

`smooth_loglinear_conductance`

Smooth log-link conductance model

Description

Returns a function of class `"terradish_conductance_model"` that represents a log-linear conductance surface after expanding selected numeric covariates into spline basis columns. The result is an unpenalized regression spline with a fixed number of coefficients, fitted through the selected terradish measurement likelihood.

Usage

```
smooth_loglinear_conductance(
  formula,
  x,
```

```
  df = 4L,
  basis = c("ns", "bs"),
  degree = 3L,
  intercept = FALSE
)
```

Arguments

formula	Model formula describing which spatial covariates drive conductance. Smooth terms are written as s(variable, df = 4) or s(variable, k = 4). Ordinary linear terms can be mixed with smooth terms, e.g. ~ forestcover + s(altitude, df = 4).
x	Data frame of spatial covariates extracted from a conductance_surface object (typically surface$x).
df	Default degrees of freedom for smooth terms that do not supply df or k.
basis	Default spline basis. "ns" uses ns natural splines; "bs" uses bs B-splines.
degree	B-spline polynomial degree used when basis = "bs".
intercept	Logical; include intercept columns within each spline basis? The default is FALSE, matching terradish's usual convention of omitting a global conductance intercept.

Details

The model first expands all s() terms into basis columns and then applies the same positive log-link used by loglinear_conductance:

$$C_i = \exp(B_i\theta)$$

Each spline column is centered over the active graph cells. The fitted knots, boundary knots, degree, and column centers are retained and reused for plotting, focal-support prediction, and coarse warm starts. Centering changes the arbitrary conductance reference level but preserves the likelihood and fitted conductance coefficients. Uncertainty bands are relative to the landscape-mean spline contribution to log conductance.

where B_i is the row of the expanded spline/parametric design matrix for grid cell i. The same conductance factory works with MLPE and Wishart measurement models.

Smoothing parameters are not estimated in this first implementation. The effective smoothness is controlled by fixed basis dimension through df or k; use small values for stable first-pass model comparison. AIC and AICc at nominal Wishart information can over-select flexible curves. Prefer spatial CV for selecting conductance terms, and examine sensitivity to nu before interpreting uncertainty. Tails beyond the sampled covariate range are poorly identified. A selected curve can describe how a process maps onto the graph without identifying an ecological response mechanism.

summary(fit)$spline_monotonicity describes each fitted smooth over the range at focal sites. It reports whether the curve is monotone, its direction, and the number of derivative sign changes. This is a shape diagnostic, not an uncertainty test.

Value

A function of class `"terradish_conductance_model"` that accepts a numeric vector of conductance parameters `theta` and returns conductance values plus derivative functions used internally by the optimizer. Derivatives are with respect to the expanded basis columns.

Worked example

See `vignette("spline-conductance", package = "terradish")` for fitted conductance curves, genetic-response plots, and model comparison.

See Also

`loglinear_conductance`, `mlpe`, `conductance_surface`, `terradish`

Examples

```
x <- data.frame(altitude = seq(-1, 1, length.out = 8),
                forestcover = c(0.2, 0.4, 0.6, 0.7, 0.5, 0.3, 0.2, 0.1))
model <- smooth_loglinear_conductance(~ forestcover + s(altitude, df = 3), x)
theta <- attr(model, "default")
fit <- model(theta)
fit$conductance
```

---

terradish *Optimize a parameterized conductance surface*

Description

Uses maximum likelihood to fit a parameterized conductance surface to genetic data, given a function relating spatial data to conductance (a "conductance model") and a function linking the graph kernel to an observed genetic distance or covariance matrix (a "measurement model").

Usage

```
terradish(
  formula,
  data,
  conductance_model = loglinear_conductance,
  measurement_model = mlpe,
  nu = NULL,
  theta = NULL,
  leverage = FALSE,
  nonnegative = TRUE,
  conductance = TRUE,
  optimizer = c("newton", "bfgs", "auto"),
  verbose = FALSE,
  control = NewtonRaphsonControl(ctol = 1e-06, ftol = 1e-06),
```

```
  validate = FALSE,
  cores = 1L,
  curvature = c("exact", "gauss_newton"),
  solver = c("direct", "auto", "amg"),
  solver_control = NULL,
  approximation = c("none", "landmark", "coarse_raster"),
  approximation_control = NULL,
  measurement_control = NULL,
  slim = FALSE
)
```

Arguments

formula	A formula with an observed genetic distance or covariance matrix on the left-hand side and conductance covariates from data on the right-hand side.
data	An object of class terradish_graph (see conductance_surface)
conductance_model	A function of class terradish_conductance_model_factory (see terradish_conductance_model_fa
measurement_model	A function of class terradish_measurement_model (see terradish_measurement_model)
nu	Effective Wishart degrees of freedom passed to measurement models that require it, such as generalized_wishart and wishart_covariance. It must be supplied and is not estimated. It controls effective information, not a marker count. Report sensitivity across plausible values with terradish_rescale_nu. Ignored by non-Wishart measurement models.
theta	Starting values for optimization
leverage	Compute influence measures and leverage?
nonnegative	Force regression-like measurement_model to have nonnegative slope?
conductance	Retained for backward compatibility. Only conductance = TRUE is currently implemented.
optimizer	The optimization algorithm to use: newton uses the exact Hessian during optimization, with computational cost that grows linearly with the number of parameters; bfgs uses an approximate Hessian during optimization with cheaper iterations but often more steps; and auto selects bfgs when there are more than three conductance parameters and newton otherwise. Regardless of the optimizer, terradish() evaluates the fitted model once more at the final parameter values using the requested curvature, so standard errors and summaries use the selected final derivative machinery.
verbose	Logical. Should the optimizer report its progress? The default FALSE fits silently. Set TRUE to see the iteration count, objective value, gradient norm, and Hessian determinant at each step, which is what you want when a fit fails to converge or you are tuning control. The trace is emitted with message(), so you can also capture or silence it with suppressMessages(). Setting verbose explicitly overrides the verbose field of control; leaving it alone lets a hand-built control object speak for itself. The much noisier line-search trace stays separate: turn it on with control = NewtonRaphsonControl(ls.control = HagerZhangControl(verbose = TRUE)).

`control` A list containing options for the optimization routine (see `NewtonRaphsonControl` for list)

`validate` Numerical validation of leverage via package `numDeriv` (very slow, use for debugging small examples)

`cores` Number of worker processes. `1` evaluates serially. Parallel execution is experimental and uses PSOCK workers on every operating system. Only Hessian and partial solves run in parallel; AMG and cached CHOLMOD ignore this setting. Multiple workers can be slower on small graphs. A one-time message explains these limits when applicable. On Windows, one PSOCK cluster is retained for the duration of the fit instead of being rebuilt for every derivative evaluation.

`curvature` Curvature used for optimization steps and for the returned covariance matrix. `"exact"` (default) uses the exact Hessian. `"gauss_newton"` uses an information-based approximation that drops residual-weighted second-derivative terms and requires only first derivatives of the conductance model. It equals the exact Hessian only when those omitted terms are zero or negligible; convergence alone does not guarantee agreement. Standard errors from `summary()` are then the asymptotic information-based errors. With `leverage = TRUE` the leverage diagnostics inherit the same approximation.

`solver` Linear-system solver. `"direct"` uses sparse Cholesky, `"amg"` uses algebraic multigrid, and `"auto"` chooses AMG above the large-graph threshold regardless of the number of right-hand sides. AMG iteration counts can rise with conductance contrast.

`solver_control` Optional named list of solver settings passed to `terradish_algorithm`. For `solver = "direct"`, supported entries include `factorization`, `supernodal_min_vertices`, `supernodal_max_rhs`, `perm`, and `solve_backend`; the latter can be `"matrix"` or the experimental `"cholmod_cpp"` and `"cholmod_cpp_cached"` backends. `reference = "auto"` grounds a maximum-conductance vertex on high-contrast graphs; `"last"` retains the original reference. This changes the numerical solve while preserving the same graph model. Direct solves also reject relative system residuals exceeding `residual_tolerance = 1e-8`; see `terradish_algorithm`. For `solver = "auto"`, supported selection entries include `auto_direct_max_vertices`, `auto_amg_min_vertices`. The legacy `auto_direct_max_rhs` setting is accepted but no longer affects selection. For `solver = "amg"` or `"auto"`, `terradish()` also understands an adaptive schedule with entries such as `adaptive`, `tol_early`, `tol_mid`, `tol_final`, `maxit_early`, `maxit_mid`, `maxit_final`, and `warmup_evals`. AMG controls also support `reuse_preconditioner` to reuse multigrid hierarchy information across nearby optimization steps when possible, and `reuse_preconditioner_max_age` to periodically rebuild a reused hierarchy. Direct supernodal factorizations can benefit from a threaded BLAS, but those thread counts are controlled by the external R/BLAS build rather than by `terradish()` itself. If a fit stops with a CHOLMOD or non-positive-definite reduced Laplacian message, the current conductance values made the graph numerically singular. Common fixes are to scale raster covariates, start conductance parameters closer to zero, check retained raster cells for missing or infinite values, and retry with `solver = "auto"` or `solver = "amg"` while diagnosing the problem.

`approximation` Exploratory approximation used during optimization. `"none"` uses the full focal set throughout. `"landmark"` optimizes first on a space-filling subset of focal

populations and then refines on the full likelihood when supported by the measurement model. `"coarse_raster"` optimizes first on an aggregated raster and then refines on the full-resolution graph. If multiple coarse factors are supplied, they are evaluated from coarsest to finest before the final full-resolution stage. This is an opt-in warm-start strategy, not a replacement for the exact full-resolution likelihood. Exact refinement is required.

`approximation_control`
: Optional named list controlling the landmark or coarse-raster approximation. Landmark entries include `n_landmarks`, `fraction`, `min_landmarks`, `max_landmarks`, `method` (`"spacefill"`, `"random"`, or `"sequential"`), `seed`, and `exact_refine`. For `approximation = "coarse_raster"`, supported entries include `factor`, `aggregate_fun`, `directions`, `exact_refine`, and `refine_control`; this path requires `conductance_surface(..., saveStack = TRUE)`. `factor` may be a single positive integer or a vector such as `c(8, 4, 2)` for a multilevel coarse-to-fine warm start. The final reported fit is still evaluated on the full data. `refine_control` may be a `NewtonRaphsonControl` list used only for that final exact full-resolution refinement; by default the main `control` is reused. For `measurement_model = leastsquares`, landmark exact-refinement is guarded and the approximation stage is skipped.

`measurement_control`
: Optional `NewtonRaphsonControl` object for the inner nuisance-parameter profile. This is separate from `control`, which governs optimization of the conductance parameters. The fitted object records the inner optimizer's convergence code and iteration count for the final evaluation. A failed nuisance warm start is retried from the model's default start, retaining any fixed values and custom bounds. Warnings from the failed attempt remain visible; the retry is accepted only if it converges. Standard MLPE nuisance fits scale the resistance slope internally; their convergence criterion is therefore independent of resistance units.

`slim`
: Logical. If `TRUE`, apply `slim_terradish` to the returned fit, removing model closures and leverage arrays. Use this for durable storage only when later conductance prediction and model reevaluation are unnecessary.

Details



A parameterized conductance surface maps raster covariates to positive vertex conductance. Vertex values define weighted edges and a graph Laplacian $L(\theta)$. Its generalized inverse $E(\theta) = L(\theta)^{+}$ is the shared graph-process kernel. Distance likelihoods use resistance distances derived from $E(\theta)$, whereas covariance likelihoods use the kernel directly. The selected probability model is called the measurement model. These alternatives describe how related genetic summaries are observed; they are not competing biological process theories.

For example, a log-linear choice of conductance model is:

`vertex_conductance[i] = exp(covariates[i,] %*% theta)`

where `theta` are unknown parameters, `covariates` is a design matrix with each row representing a vertex in the graph (cell in the raster). The conductance of a given edge (an off-diagonal entry in the graph Laplacian) is:

`edge_conductance[i,j] = vertex_conductance[i] + vertex_conductance[j]`

if argument `conductance = TRUE`, and

`edge_conductance[i,j] = 1/(1/vertex_conductance[i] + 1/vertex_conductance[j])`

otherwise. This alternative parameterization is not currently implemented, so `conductance` must be `TRUE`. This differs by a factor of two from CIRCUITSCAPE, where the edge conductance/resistance is the average of the vertex conductance/resistance.

`terradish` estimates `theta` by maximizing the selected measurement likelihood for the observed response. The optimization is done via Newton's method (default; requires computation of Hessian during optimization), via the BFGS algorithm (requires gradient only during optimization) if `optimizer = "bfgs"`, or via a simple parameter-count heuristic if `optimizer = "auto"`. In all cases, the final returned fit is evaluated at the optimized parameters using the requested `curvature` for inference.

Conductance coefficients are conditional model associations. Their sign and magnitude depend on covariate coding, other terms, graph construction, response, measurement likelihood, and candidate set. They are not direct estimates of causal effects, habitat suitability, migration, dispersal, or organismal movement rates.

For an explanation of how categorical spatial covariates are handled, see `details` of `conductance_surface` and the examples below. The dummy coding of categorical covariates is done by the function passed to `conductance_model` (e.g. `terradish::loglinear_conductance`).

Large rasters. For large continuous rasters, two opt-in helpers can reduce runtime. First, build the graph with `conductance_surface(..., crop_buffer = )` when focal sites occupy only part of the raster. This removes vertices outside the buffered sampling extent before fitting. Cropping changes resistance: one sensitivity audit found increases of 15–23% with a two-cell buffer and 1–2% with a ten-cell buffer. These values are case-specific. Refit with a larger buffer to assess sensitivity on your landscape. Second, use `approximation = "coarse_raster"` with `approximation_control = list(factor = c(4, 2), exact_refine = TRUE)` to optimize on one or more aggregated rasters before refining on the original graph. With `exact_refine = TRUE`, the returned coefficients and likelihood are from the full-resolution graph; the coarse fits are only starting values. To keep the full-resolution cleanup deliberately short, pass `refine_control = NewtonRaphsonControl(maxit = 2, ...)` inside `approximation_control`. Exact refinement is required.

For spline terms, `summary()` reports monotonicity over the covariate range at the focal sites, including the number of derivative sign changes. A monotone curve increases or decreases throughout that range; a nonmonotone curve changes direction. This describes the fitted curve, not a confidence statement about its shape outside or within that range.

Gaussian scale-aware conductance. `gaussian_smoothed_loglinear_conductance` can also use `approximation = "coarse_raster"`. In that case the Gaussian smoother is rebuilt on each aggregated raster, while the full-resolution sigma bounds and unit conversion are preserved so a warm-started `sigma` continues to mean the same map-unit distance at every stage.

Currently, all of the built-in conductance models in `terradish` use the default contrast coding (e.g. for a categorical covariate with `K` factors, the estimated parameters are the `K-1` mean differences against a reference category). The intercept is excluded if it is not identifiable.

If the resistance contribution is zero, or the formula specifies uniform conductance, conductance coefficients are not identified and their influence, gradient, and Hessian are unavailable. A zero environmental kernel weight does not by itself remove conductance inference.

Convergence and starting values. Inspect `fit$convergence` before interpreting estimates. Code 0 requires a largest absolute projected gradient below `ctol`, or an objective change below `ftol` together with a projected gradient below `sqrt(ctol)`. Code 1 means the iteration limit was reached; code 2 means a stall or failed line search. The projected gradient ignores components pointing outside active parameter bounds. The record includes the criterion, iterations, gradient, boundary status, and whether a restart was attempted. A small objective change alone is not convergence. Also inspect `fit$fit$subproblem$convergence` for nuisance optimization. Numerical convergence does not establish model adequacy.

If a user-supplied starting point reaches a no-structure boundary, fitting retries the default starting point and keeps the better likelihood. Report starting-value sensitivity when distinct solutions remain plausible.

Value

An object of class `c("terradish", "radish")`, a list. Most users need only the methods: `summary()` for estimates and intervals, `coef()` and `confint()` for conductance coefficients, `conductance()` and `plot()` for maps and curves, and `AIC()`, `anova()`, or `aic_table` for comparison (see `terradish_methods`). The list elements are:

`call`, `formula`	The matched call and the model formula.
`dim`	Named vector with the number of graph vertices (raster cells), focal sites, and graph edges.
`convergence`	List describing the conductance optimization: `code` (0 converged, 1 iteration limit reached, 2 stalled or line search failed), `message`, `iterations`, `max_abs_projected_gradient`, `criterion`, `boundary` (`TRUE` when an estimate sits on a bound or no resistance structure was detected), and `restarted`. Check `object$convergence$code == 0` before interpreting estimates.
`fit`	The final graph and likelihood evaluation. `object$fit$phi` holds the nuisance (measurement-model) estimates on the measurement model's internal scale, for example log precision; `summary()` reports them with intervals. `object$fit$fitted` is the expected response matrix among focal sites. `object$fit$subproblem$convergence` is the nuisance optimizer's code (0 means converged), which should also be 0 before estimates are interpreted.
`loglik`, `df`, `aic`	Maximized log-likelihood, number of estimated parameters (conductance coefficients plus nuisance parameters), and $AIC = -2\ell + 2K$.
`mle`	Conductance estimates (`theta`, on the log-conductance scale per unit of each covariate as supplied to the graph), and the gradient and Hessian of the log-likelihood at the estimate. Elements ending in `_internal` are on the optimizer's scale.
`submodels`	The conductance-model and measurement-model functions used for the fit; `conductance()` and `plot()` reuse them.

`comparison` Measurement family, `nu`, graph fingerprint, and response columns that `aic_table` and `anova()` use to check that fits are comparable.

`cost`, `diagnostics`, `approximation`
Newton steps and function calls, solver timing and evaluation diagnostics, and a record of any landmark or coarse-raster warm start.

`gaussian_scale_info`, `spline_monotonicity`
Metadata for Gaussian smoothing and spline conductance terms, or `NULL`.

`leverage` Leverage arrays for the response (`S`) and spatial covariates (`X`) when `leverage = TRUE`, otherwise `NULL` entries.

`storage` Present only when `slim = TRUE`; records the model closures and leverage arrays that were removed.

Effective Wishart information

For Wishart likelihoods, `nu` scales the log likelihood. Point estimates do not depend on it, but standard errors, likelihood-ratio tests, and information criteria do. It is a dispersion parameter that you set, rather than the number of markers. Linkage and the population history shared by loci can make effective information much smaller than the SNP count. In forward-time simulations used to validate terradish, standard errors at the SNP count were three to five times smaller than the spread among replicate histories. Those model-based errors understated uncertainty.

Use `terradish_cv_folds` to compare conductance formulas, and `terradish_rescale_nu` to report inference across plausible values. For the same successful folds and nuisance settings, changing `nu` rescales Wishart CV scores without changing their ranking. Neither procedure identifies the true effective information or repairs an unsuitable response matrix.

Examples

```
library(terra)

data(melip)
melip.altitude <- terra::unwrap(melip.altitude)
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords <- terra::unwrap(melip.coords)

# scaling spatial covariates helps avoid numeric overflow
covariates <- c(terra::scale(melip.altitude),
                terra::scale(melip.forestcover))
names(covariates) <- c("altitude", "forestcover")

# create parameterized conductance surface
surface <- conductance_surface(covariates, melip.coords, directions = 8)

fit_nnls <- terradish(melip.Fst ~ altitude * forestcover, data = surface,
                   terradish::loglinear_conductance, terradish::leastsquares)
summary(fit_nnls)

# a different "measurement_model" that incorporates dependence
# among pairwise measurements
fit_mlpe <- terradish(melip.Fst ~ altitude + forestcover, data = surface,
```

```
                    terradish::loglinear_conductance, terradish::mlpe)
summary(fit_mlpe)

# conductance surface with 95% CI
fitted_conductance <- conductance(surface, fit_mlpe, quantile = 0.95)

# test for an interaction using a likelihood ratio test
fit_mlpe_interaction <- terradish(melip.Fst ~ forestcover * altitude, data = surface,
                                  terradish::loglinear_conductance, terradish::mlpe)
anova(fit_mlpe, fit_mlpe_interaction)

# test against null model of IBD using a LRT
fit_mlpe_ibd <- terradish(melip.Fst ~ 1, data = surface,
                          terradish::loglinear_conductance, terradish::mlpe)
anova(fit_mlpe, fit_mlpe_ibd)

# categorical covariates:
# categorical raster layers should be factor-valued, see ?terra::as.factor
# and 'details' section of ?conductance_surface
forestcover_class <- cut(terra::values(melip.forestcover)[,1], breaks = c(0, 1/6, 1/3, 1))
melip.forestcover_cat <- terra::setValues(melip.forestcover, as.numeric(forestcover_class))
melip.forestcover_cat <- terra::as.factor(melip.forestcover_cat)

RAT <- levels(melip.forestcover_cat)[[1]]
RAT$VALUE <- levels(forestcover_class) #explicitly define level names
levels(melip.forestcover_cat) <- RAT

covariates_cat <- c(melip.forestcover_cat, melip.altitude)
names(covariates_cat) <- c("forestcover", "altitude")

surface_cat <- conductance_surface(covariates_cat, melip.coords, directions = 8)

fit_mlpe_cat <- terradish(melip.Fst ~ forestcover + altitude, surface_cat,
                          terradish::loglinear_conductance, terradish::mlpe)
summary(fit_mlpe_cat)
```

`terradish_algorithm` *Likelihood of parameterized conductance surface*

Description

Calculates likelihood, gradient, hessian, and partial derivatives of a parameterized conductance surface, given a function mapping spatial data to conductance and a function mapping resistance distance (covariance) to genetic distance; using the algorithm in Pope (in prep).

Usage

```
terradish_algorithm(
```

```
  f,
  g,
  s,
  S,
  theta,
  nu = NULL,
  phi = NULL,
  objective = TRUE,
  gradient = TRUE,
  hessian = TRUE,
  partial = TRUE,
  nonnegative = TRUE,
  validate = FALSE,
  cores = 1L,
  curvature = c("exact", "gauss_newton"),
  solver = c("direct", "auto", "amg"),
  solver_control = NULL,
  solver_warm_start = NULL,
  solver_reuse_state = NULL,
  measurement_control = NULL,
  worker_pool = NULL
)
```

Arguments

f	A function of class 'conductance_model'
g	A function of class 'measurement_model'
s	An object of class "terradish_graph"
S	A matrix of observed genetic distances
theta	Parameters for conductance surface (e.g. inputs to 'f')
nu	Effective Wishart degrees of freedom, passed through to measurement models that require it.
phi	Optional warm-start values for the nuisance-parameter subproblem.
objective	Compute negative loglikelihood?
gradient	Compute gradient of negative loglikelihood wrt theta?
hessian	Compute Hessian matrix of negative loglikelihood wrt theta?
partial	Compute partial derivatives of negative loglikelihood wrt theta and spatial covariates/observed genetic distances
nonnegative	Force regression-like 'measurement_model' to have nonnegative slope?
validate	Numerical validation via 'numDeriv' (very slow, use for debugging small examples)
cores	Number of worker processes to use for per-parameter derivative calculations. 1 evaluates serially.

`curvature` Which curvature to return in `hessian`. `"exact"` (default) returns the exact Hessian of the negative log-likelihood. `"gauss_newton"` returns the Gauss-Newton / Fisher-information approximation, obtained by dropping the two residual-weighted second-derivative terms (the second derivative of the resistance covariance E and of conductance with respect to `theta`). It is an information-based approximation and is positive semidefinite when the retained measurement-model curvature is positive semidefinite. It equals the exact Hessian only when the omitted terms are zero or negligible; convergence alone does not guarantee equality. A large gap can reflect residual structure, model mismatch, or numerical instability and should be investigated rather than assigned a unique cause. It also requires only first derivatives of the conductance model, which is useful for Gaussian scale-aware and spline conductance models whose second derivatives are expensive or unstable. The number of linear solves is the same as for the exact Hessian.

`solver` Linear-system solver. `"direct"` uses sparse Cholesky, `"amg"` uses algebraic multigrid, and `"auto"` chooses between them.

`solver_control` Optional named list of solver settings. For `solver = "direct"`, supported entries include `factorization` (`"auto"`, `"simplicial_ldl"`, `"simplicial_ll"`, or `"supernodal_ll"`), `solve_backend` (`"matrix"` or the experimental `"cholmod_cpp"` and `"cholmod_cpp_cached"` backends), `supernodal_min_vertices`, `supernodal_max_rhs`, and `perm`. For `solver = "auto"`, supported selection entries include `auto_direct_max_vertices`, `auto_amg_min_vertices`. The legacy `auto_direct_max_rhs` is ignored. For `solver = "amg"`, supported entries include `tol`, `maxit`, `coarse_enough`, `npre`, `npost`, `sa_relax`, `aggr_eps_strong`, `estimate_spectral_radius`, `power_iters`, and `reuse_preconditioner`. `reuse_preconditioner_max_age` can be set to a finite nonnegative value to periodically rebuild the AMG hierarchy instead of reusing it indefinitely. Direct solvers also accept `reference = "auto"` (default), which grounds a maximum-conductance vertex when the conductance ratio exceeds one million, or `"last"` and `"max_conductance"`. This changes the numerical reference potential while preserving the graph and model. It improves some stiff solves but cannot guarantee accuracy at arbitrary conductance contrasts. Direct solves check the maximum absolute residual in each right-hand side, divided by its maximum absolute input or one, whichever is larger. `residual_tolerance = 1e-8` rejects unresolved systems by default; `Inf` disables this check for diagnosis. A small residual is necessary but does not guarantee small forward error on an ill-conditioned graph. Before rejection, `refinement_steps = 3` permits up to three correction solves using the same factorization and the current residual. Direct supernodal factorizations can benefit from a threaded BLAS, but the relevant thread counts are controlled by the external R/BLAS build rather than by `terradish_algorithm()`. If the direct solver stops with a CHOLMOD or non-positive-definite reduced Laplacian message, the current conductance values made the graph numerically singular. Check for missing or infinite raster covariates, use scaled covariates and moderate starting values near zero, and try `solver = "auto"` or `solver = "amg"` while diagnosing the fit.

`solver_warm_start` Optional initial guess for the reduced-system solve. This is primarily useful for iterative solvers when evaluating nearby parameter values.

`solver_reuse_state`
: Optional reusable solver state returned by a prior `terradish_algorithm()` call. This is used to reuse AMG hierarchy information or compatible direct CHOLMOD factorization state across nearby evaluations.

`measurement_control`
: Optional `NewtonRaphsonControl` object used to profile nuisance parameters. The default retains the package's high-accuracy profiling tolerances.

`worker_pool`
: Optional reusable worker pool created internally by `terradish`. Most users should leave this as `NULL`.

Value

A list containing at a minimum:

`covariance`
: rows/columns of the generalized inverse of the graph Laplacian for a subset of target vertices

Additionally, if 'objective == TRUE':

`objective`
: (if 'objective') the negative loglikelihood

`phi`
: (if 'objective') fitted values of the nuisance parameters of 'g'

`subproblem`
: (if 'objective') convergence code and iteration count for the nuisance-parameter profile

`boundary`
: (if 'objective') is the solution on the boundary (e.g. no genetic structure)?

`fitted`
: (if 'objective') matrix of expected genetic distances among target vertices

`gradient`
: (if 'gradient') gradient of negative loglikelihood with respect to theta

`hessian`
: (if 'hessian') Hessian matrix of the negative loglikelihood with respect to theta

`partial_X`
: (if 'partial') Jacobian of the gradient with respect to the spatial covariates

`partial_S`
: (if 'partial') Jacobian of the gradient with respect to the observed genetic distances

Examples

```
library(terra)

data(melip)
melip.altitude <- terra::unwrap(melip.altitude)
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords <- terra::unwrap(melip.coords)

covariates <- c(melip.altitude, melip.forestcover)
names(covariates) <- c("altitude", "forestcover")
surface <- conductance_surface(covariates, melip.coords, directions = 8)
conductance_model <- loglinear_conductance(~ altitude + forestcover, surface$x)

terradish_algorithm(conductance_model, terradish::leastsquares,
                    surface, melip.Fst, theta = c(-0.3, 0.3))
```

terradish_assess_settings *Assess terradish speed settings for a data set*

Description

Profiles a terradish graph and, optionally, runs short probe benchmarks to recommend optimizer, line-search, solver, and approximation settings for a fitted model. Use this function before a large or slow fit to identify the fastest computational configuration for your data.

Usage

```
terradish_assess_settings(
  formula,
  data,
  conductance_model = loglinear_conductance,
  measurement_model = mlpe,
  nu = NULL,
  theta = NULL,
  nonnegative = TRUE,
  optimizer_probe = TRUE,
  solver_probe = TRUE,
  coarse_probe = FALSE,
 optimizer_candidates = c("newton_hager_zhang", "bfgs_hager_zhang", "bfgs_armijo"),
  probe_maxit = 2L,
  factorization = c("simplicial_ldl", "simplicial_ll", "supernodal_ll"),
  solve_backend = c("matrix", "cholmod_cpp", "cholmod_cpp_cached"),
  coarse_factor = 2L,
  coarse_refine_maxit = 1L,
  loglik_tolerance = 0.001,
  max_seconds = 120,
  cores = 1L,
  verbose = FALSE
)
```

Arguments

formula	Model formula passed to terradish.
data	A terradish_graph object returned by conductance_surface.
conductance_model	Conductance-model factory (e.g. loglinear_conductance).
measurement_model	Measurement model used during optimizer probes. Should match the model you plan to use for the final fit.

nu	Effective Wishart degrees of freedom passed to measurement models that require it (generalized_wishart, wishart_covariance). For biallelic SNPs, use the number of approximately independent retained SNPs. For microsatellites, use the locus count as the conservative primary value and report sensitivity to larger plausible values. Ignored by mlpe and leastsquares.
theta	Optional starting conductance parameter values. If NULL, the default all-zero start is used.
nonnegative	Should regression measurement models constrain the IBR slope to be nonnegative during probes?
optimizer_probe	Logical. Run short optimization probes to compare Newton-Raphson vs. BFGS and line-search algorithms?
solver_probe	Logical. Benchmark direct Cholesky solver settings (factorization style and solve backend) via terradish_solver_benchmark?
coarse_probe	Logical. Include a coarse-raster warm-start probe? Only applicable when data retains its raster stack (saveStack = TRUE).
optimizer_candidates	Character vector of optimizer configurations to probe. Supported entries: "newton_hager_zhang", "bfgs_hager_zhang", "bfgs_armijo".
probe_maxit	Maximum optimizer steps per probe. Keep small (default 2); the assessment compares early iteration cost, not convergence.
factorization	Character vector of Cholesky factorization styles to benchmark when solver_probe = TRUE. Options: "simplicial_ldl", "simplicial_ll", "supernodal_ll".
solve_backend	Character vector of solve backends to benchmark when solver_probe = TRUE. Options: "matrix", "cholmod_cpp", "cholmod_cpp_cached".
coarse_factor	Raster aggregation factor for the coarse-raster probe (passed to the factor entry of approximation_control).
coarse_refine_maxit	Maximum full-resolution refinement steps for the coarse-raster probe.
loglik_tolerance	Maximum log-likelihood difference below the best probe that is still considered acceptable when choosing the fastest option.
max_seconds	Soft wall-time budget in seconds. Each probe that starts before the budget is exceeded is allowed to finish; later probes are skipped.
cores	Number of parallel worker processes passed to probes.
verbose	Logical. Print progress as probes run?

Details

terradish_assess_settings is an advisory tool, not a model-selection procedure. It does not change package defaults and its output does not affect parameter estimation. Use it to choose *computational* settings; then fit and interpret your model as usual. It is a speed probe. A fast candidate is not evidence of optimizer convergence, identification, correct standard errors, or biological validity.

Short probes work well for comparing early-iteration cost of different optimizers. They are most informative for large graphs (> 100,000 vertices) where a single function evaluation is slow. For

small graphs the default Newton-Raphson + Hager-Zhang settings typically converge in seconds, and assessment adds more overhead than it saves.

The returned `$recommended` list is intentionally a plain named list of arguments (`optimizer`, `control`, `solver`, `solver_control`, `approximation`, `approximation_control`). You can inspect, edit, or pass it directly:

```
fit <- terradish(formula, data,
                 optimizer = assessment$recommended$optimizer,
                 control   = assessment$recommended$control,
                 solver    = assessment$recommended$solver)
```

Value

An object of class `"terradish_setting_assessment"` with components:

`profile` A list of graph statistics (number of vertices, edges, focal sites, parameters, and, for `smooth_loglinear_conductance`, spline-expansion counts.

`optimizer_benchmark` A data frame of optimizer probe results (one row per candidate), or `NULL` if `optimizer_probe = FALSE`.

`solver_benchmark` A data frame of solver probe results, or `NULL` if `solver_probe = FALSE`.

`defaults` The terradish defaults for the profiled model, after resolving `optimizer = "auto"` on the expanded conductance parameter count.

`comparison` A compact comparison between the recommended settings and the terradish defaults, including a one-line summary.

`recommended` A named list of recommended settings: `optimizer`, `control`, `solver`, `solver_control`, `approximation`, `approximation_control`.

`notes` Character vector of advisory messages.

A `print` method summarizes the profile and recommendations.

Examples

```
data(melip)
melip.altitude <- terra::unwrap(melip.altitude)
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords <- terra::unwrap(melip.coords)

covariates <- c(melip.altitude, melip.forestcover)
names(covariates) <- c("altitude", "forestcover")
# Coarsened for the example. Assessment is most useful on the large graphs
# where solver choice actually matters, so use the full resolution there.
covariates <- terra::aggregate(covariates, fact = 3, na.rm = TRUE)
covariates <- scale_covariates(covariates)
surface <- conductance_surface(covariates, melip.coords,
                               directions = 8, saveStack = TRUE)

assessment <- terradish_assess_settings(
  melip.Fst ~ forestcover + altitude,
  data = surface,
```

```
  measurement_model = mlpe,
  probe_maxit = 2
)
assessment

fit <- terradish(
  melip.Fst ~ forestcover + altitude,
  data = surface,
  conductance_model = loglinear_conductance,
  measurement_model = mlpe,
  optimizer = assessment$recommended$optimizer,
  control = assessment$recommended$control,
  solver = assessment$recommended$solver,
  solver_control = assessment$recommended$solver_control
)

smooth_assessment <- terradish_assess_settings(
  melip.Fst ~ forestcover + s(altitude, df = 3),
  data = surface,
  conductance_model = smooth_loglinear_conductance,
  measurement_model = mlpe,
  probe_maxit = 1
)
smooth_assessment$comparison$summary
```

`terradish_conductance_model_factory`

Conductance model factories

Description

Functions that generate objects of class `"terradish_conductance_model"` that represent mappings from spatial data (e.g. rasters) to conductance.

Usage

```
terradish_conductance_model_factory
```

Worked examples

See `vignette("getting-started", package = "terradish")` for a log-linear model, `vignette("categorical-conduct` `package = "terradish")` for class contrasts, and `vignette("spline-conductance", package = "terradish")` for nonlinear curves and their plots.

See Also

`loglinear_conductance`, `smooth_loglinear_conductance`

`terradish_cv_folds` *Fixed-domain cross-validation for terradish models*

Description

Fits one or more conductance formulas to each training fold while retaining the complete landscape graph, then evaluates the trained coefficients on the held-out focal sites. Measurement parameters can be reprofiled on each test fold or held at their training estimates for predictive scoring.

Usage

```
terradish_cv_folds(
  data,
  formulas,
  folds,
  model = "mlpe",
  nu = NULL,
  conductance_model = loglinear_conductance,
  baseline = TRUE,
  keep_fits = c("none", "slim", "full"),
  checkpoint = NULL,
  resume = TRUE,
  cores = 1L,
  measurement_control = NULL,
  nuisance = c("reprofile", "fixed"),
  ...
)
```

Arguments

`data`	A prebuilt `terradish_graph`, usually returned by `conductance_surface`.
`formulas`	A formula or named list of formulas. All formulas must use the same response matrix and likelihood family.
`folds`	Fold membership for each focal site. Use `terradish_folds`, or supply study-specific labels. A list of fold vectors requests repeated cross-validation.
`model`	Measurement model function or one of `"mlpe"`, `"wishart"`, or `"ls"`. With fixed nuisance parameters, a named list matching the formula names can compare models within one family.
`nu`	Effective Wishart degrees of freedom when required by the selected measurement model.
`conductance_model`	One conductance-model factory or a named list matching the formula names, allowing log-linear, Gaussian, and spline models in the same comparison.
`baseline`	Should each test fold also be scored at zero conductance coefficients, the uniform-conductance baseline on the same graph?

`keep_fits` Whether to retain no training fits, slim fits, or full fits.

`checkpoint` Optional file path. Completed fold results are written atomically after every successful or failed fit so a long run can be inspected or resumed.

`resume` Logical. If `TRUE` and `checkpoint` exists, continue a compatible run without reevaluating completed formula-fold combinations.

`cores` Number of workers used inside each model fit. Fold evaluation is serial to avoid nested parallel clusters.

`measurement_control` Optional `NewtonRaphsonControl` object for profiling nuisance parameters.

`nuisance` `"reprofile"` (default) estimates measurement parameters separately on each held-out block. `"fixed"` holds every measurement parameter, including environmental coefficients, at its training estimate.

`...` Additional arguments passed to `terradish`. An explicit `optimizer` is used for each training fit. Otherwise the training call tries Newton and retries with BFGS if Newton raises an error.

Details

The graph domain and raster covariates remain fixed. Only the focal-site columns in `demes` and `rhs`, the response matrix, and any measurement-model site attributes are subset. This evaluates transfer to held-out sites within the observed landscape rather than rebuilding a new landscape for every fold.

Reprofiled scores compare conductance formulas under one measurement model; they must not be compared across calls with different measurement models or effective degrees of freedom. Fixed-parameter scores are predictive marginal densities and can compare measurement models within one likelihood family. Neither mode scores pairs spanning training and test sites. Spatial folds score only within-cluster pairs. Wishart score gains scale with `nu`, although ideal fitted rankings are invariant to its scale.

Checkpoints include a signature of the response, graph, focal cells, covariates, model definitions, formulas, `nu`, and fold assignments. For factor covariates the signature retains class placement, levels, and contrasts. Uniform `~1` formulas are scored with unit conductance. A mismatched signature stops resumption. Nonconverged fits and failed scores count as failed folds. Training fits must record code 0 for both outer and final nuisance optimization; missing diagnostics also fail the fold. Checkpoints predating this convergence requirement are incompatible and must be recomputed (use a new checkpoint or `resume = FALSE`). With fixed nuisance parameters, no-structure boundary fits remain scoreable because predictions do not depend on conductance. A failed uniform-baseline score is recorded separately: model scores remain usable, but baseline gains are unavailable for that fold. Models with failed folds receive NA totals and an explicit success count. Rankings use only folds successful for every model. Paired differences and their standard errors are descriptive; repeated folds share observations and are not independent replicates.

`loglik_gain` is the trained model's held-out log-likelihood minus the uniform-conductance baseline log-likelihood. Larger values indicate improved prediction over that baseline. Raw log-likelihood and gain are comparable only among models using the same response representation, measurement likelihood, folds, graph, and effective degrees of freedom. They are not a common scoring rule for comparing MLPE, generalized-Wishart distance, and covariance-Wishart models.

Value

An object of class `"terradish_cv_folds"`. Its `folds` element records the site assignments, `results` contains one row per formula and fold, and `summary` reports summed and mean held-out scores by formula, common-fold totals, paired differences from the best model, the standard error of their mean, and a one-standard-error flag. Repeated runs also report the SD of repeat totals over common folds. Optional retained fits are stored in `fits`.

Worked examples

See `vignette("model-comparison", package = "terradish")` for pond comparisons and `vignette("individual-sala` `package = "terradish")` for individual sampling, matched markers, and spatial prediction plots.

Examples

```
r <- terra::rast(nrows = 8, ncols = 8, xmin = 0, xmax = 8,
                 ymin = 0, ymax = 8, crs = "EPSG:32617")
terra::values(r) <- sin(seq_len(64) / 5) + seq_len(64) / 64
names(r) <- "x"
sites <- terra::xyFromCell(r, c(1, 5, 8, 12, 20, 27, 33, 37, 45, 53, 57, 64))
surface <- conductance_surface(r, sites)
response <- simulate_covariance_response(c(x = 0.8), ~x, surface,
  nu = 1000, seed = 93)$covariance
folds <- terradish_folds(sites, k = 3, method = "random", seed = 1)
cv <- terradish_cv_folds(
  surface,
  list(uniform = response ~ 1, gradient = response ~ x),
  folds = folds,
  model = wishart_covariance, nu = 1000, nuisance = "fixed",
  control = NewtonRaphsonControl(maxit = 100))
cv$summary
# Check fold success counts before comparing common-fold predictive scores.
# Random folds illustrate the call; choose ecological spatial blocks in practice.
```

---

terradish_distance *Resistance distances from a parameterized conductance surface*

Description

Calculates resistance distances associated with a parameterized conductance surface across a grid of parameter values.

Usage

```
terradish_distance(
  theta,
  formula,
```

```
  data,
  conductance_model = loglinear_conductance,
  conductance = TRUE,
  covariance = FALSE,
  cores = 1L
)
```

Arguments

theta	A matrix of dimension (grid size) x (number of parameters)
formula	A formula with the name of a matrix of observed genetic distances on the lhs, and covariates in the creation of data on the rhs
data	An object of class terradish_graph (see conductance_surface)
conductance_model	A function of class terradish_conductance_model_factory (see terradish_conductance_model_fa
conductance	Retained for backward compatibility. Only conductance = TRUE is currently implemented.
covariance	If TRUE, instead of a matrix of resistance distances, return the associated submatrix of the generalized inverse of graph Laplacian
cores	Number of worker processes to use. 1 evaluates the grid serially.

Value

A list of class c("terradish_grid", "radish_grid"):

theta	Data frame of the evaluated conductance coefficients, one row per grid point.
distance	When covariance = FALSE, a focal-site by focal-site by grid-point array of resistance distances. Larger values mean greater graph resistance between two sites. Their units depend on the conductance scale, so compare them within a grid rather than as map distances.
covariance	When covariance = TRUE, the corresponding array of graph covariance (the generalized inverse of the graph Laplacian), returned in place of distance.

Examples

```
library(terra)

data(melip)
melip.altitude <- terra::unwrap(melip.altitude)
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords <- terra::unwrap(melip.coords)

keep <- 1:10
covariates <- terra::aggregate(c(melip.altitude, melip.forestcover),
                               fact = 5, na.rm = TRUE)
covariates <- scale_covariates(covariates)
names(covariates) <- c("altitude", "forestcover")
surface <- conductance_surface(covariates, melip.coords[keep], directions = 8)
```

```
theta <- as.matrix(expand.grid(forestcover = seq(-0.5, 0.5, length.out = 3),
                               altitude = seq(-0.5, 0.5, length.out = 3)))

distances <- terradish_distance(theta, ~forestcover + altitude,
                            surface, terradish::loglinear_conductance, cores = 1)

ibd <- which(theta[,1] == 0 & theta[,2] == 0)
round(distances$distance[,,ibd], 4)
```

`terradish_folds` *Construct reproducible cross-validation folds*

Description

Creates either spatially blocked folds by k-means clustering projected site coordinates or balanced random folds. Fold labels can also be supplied directly to `terradish_cv_folds` when study-specific blocks are more defensible than an automated partition.

Usage

```
terradish_folds(
  pts,
  k = 5L,
  method = c("spatial_kmeans", "random"),
  seed = 1L,
  projected = NULL,
  nstart = 25L
)
```

Arguments

`pts`	Focal-site coordinates as a `terra::SpatVector`, matrix, or data frame.
`k`	Number of folds. Must be at least 2 and smaller than the number of sites.
`method`	Fold construction method. `"spatial_kmeans"` applies k-means to the unscaled projected coordinates. `"random"` assigns sites to balanced random folds.
`seed`	Optional random seed.
`projected`	Logical indicating whether the coordinates are projected. For spatial k-means with a `SpatVector`, `NULL` uses its coordinate reference system. For spatial k-means with a matrix or data frame, this argument must be supplied explicitly. It is ignored for random folds.
`nstart`	Number of random starts used by k-means.

Details

Coordinates are deliberately not standardized before spatial clustering. Consequently, Euclidean separation is measured in the map units of the projected coordinate reference system. Automated k-means blocks are a reproducible diagnostic, not a substitute for ecological blocking based on watersheds, barriers, sampling campaigns, or other design information.

Value

An integer vector of fold labels with attributes recording the construction method, seed, and number of folds.

Examples

```
data(melip)
coords <- terra::unwrap(melip.coords)
coords_projected <- terra::project(coords, "EPSG:5070")
folds <- terradish_folds(coords_projected, k = 3, seed = 1)
table(folds)
```

---

terradish_grid *Evaluate likelihood of a parameterized conductance surface*

Description

Calculates the profile likelihood of a parameterized conductance surface across a grid of parameter values (e.g. the nuisance parameters are optimized at each point on the grid).

Usage

```
terradish_grid(
  theta,
  formula,
  data,
  conductance_model = loglinear_conductance,
  measurement_model = mlpe,
  nu = NULL,
  nonnegative = TRUE,
  conductance = TRUE,
  covariance = FALSE,
  cores = 1L,
  approximation = "none",
  approximation_control = NULL
)
```

Arguments

theta	A matrix of dimension (grid size) x (number of parameters)
formula	A formula with the name of a response matrix compatible with the selected measurement model on the left-hand side, and covariates used to create data on the right-hand side.
data	An object of class terradish_graph (see conductance_surface)
conductance_model	A function of class terradish_conductance_model_factory (see terradish_conductance_model_fa
measurement_model	A function of class terradish_measurement_model (see terradish_measurement_model)
nu	Effective Wishart degrees of freedom passed to measurement models that require it. This is a user-supplied effective-information parameter, not the marker count. Report sensitivity to plausible values; see wishart_covariance.
nonnegative	Force regression-like measurement_model to have nonnegative slope?
conductance	Retained for backward compatibility. Only conductance = TRUE is currently implemented.
covariance	If TRUE, additionally return (a submatrix of) the generalized inverse of graph Laplacian across the grid
cores	Number of worker processes to use. 1 evaluates the grid serially.
approximation	Only "none" is supported; every grid point uses the full graph and focal set.
approximation_control	Reserved for compatibility; unused.

Value

A list of class c("terradish_grid", "radish_grid"), with one entry per row of theta:

theta	Data frame of the evaluated conductance coefficients (log conductance per unit of each covariate).
loglik	Numeric vector of profile log-likelihoods, with nuisance parameters optimized at each grid point. Larger values indicate better fit, and differences between grid points are on the log-likelihood scale.
phi	Matrix of optimized nuisance parameters, one column per grid point, on the measurement model's internal scale.
covariance	When covariance = TRUE, a focal-site by focal-site by grid-point array of graph covariance (the generalized inverse of the graph Laplacian); otherwise NULL.
approximation	Record of the approximation setting (always "none").

Examples

```
library(terra)

data(melip)
melip.altitude <- terra::unwrap(melip.altitude)
```

```
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords <- terra::unwrap(melip.coords)

covariates <- c(terra::scale(melip.altitude),
                terra::scale(melip.forestcover))
names(covariates) <- c("altitude", "forestcover")
surface <- conductance_surface(covariates, melip.coords, directions = 8,
                               saveStack = TRUE)

fit_mlpe <- terradish(melip.Fst ~ altitude + forestcover, data = surface,
                  terradish::loglinear_conductance, terradish::mlpe,
                  control = NewtonRaphsonControl(maxit = 5, verbose = FALSE))

theta <- as.matrix(expand.grid(forestcover = seq(-0.5, 0.5, length.out = 3),
                               altitude = seq(-0.5, 0.5, length.out = 3)))

grid <- terradish_grid(theta, melip.Fst ~ forestcover + altitude, surface,
                   terradish::loglinear_conductance, terradish::mlpe, cores = 1)

cbind(grid$theta, loglik = grid$loglik)
coef(fit_mlpe)
```

`terradish_ibe_ratio` *Environmental effects in resistance-distance units*

Description

Reports the resistance-distance equivalent of one unit of environmental difference, using the environmental coefficient divided by the resistance coefficient in the additive measurement model.

Usage

```
terradish_ibe_ratio(fit)
```

Arguments

`fit`	A fitted model from `mlpe_covariates()` or `wishart_covariates()`, or a fit with no environmental terms.

Details

The ratios are γ_k/β for MLPE and λ_k/τ for Wishart models. They require the same environmental transform and scaling for comparison. Their interpretation agrees exactly for the same covariance-derived distance and approximately for distances that scale linearly with it. Separate fits estimate their own conductance surfaces, so their resistance distances can differ. Absolute measurement coefficients are not comparable across families.

A zero resistance coefficient gives undefined ratios and uncertainty, returned as NA. A zero Wishart kernel coefficient receives a one-sided upper limit and no symmetric lower limit. Boundary inference remains approximate. The distance-equivalent ratio interpretation also appears in BEDASSLE, although that model uses geographic rather than resistance distance.

Value

A data frame with covariate, ratio, SE, lower, upper, and family. Intervals use a 95 percent delta-method approximation based on the joint covariance of both coefficients, including fitted-conductance uncertainty. Fits without environmental terms return a zero-row data frame.

References

Bradburd GS, Ralph PL, Coop GM (2013). Disentangling the effects of geographic and ecological isolation on genetic differentiation. Evolution, 67, 3258–3273. doi:10.1111/evo.12193.

See Also

pairwise_covariates, mlpe_covariates, wishart_covariates, terradish_rescale_nu

Examples

```
# A small model-based example, with fixed information supplied explicitly.
r <- terra::rast(nrows = 8, ncols = 8, xmin = 0, xmax = 8, ymin = 0, ymax = 8)
terra::values(r) <- sin(seq_len(64) / 5) + seq_len(64) / 64
names(r) <- "x"
sites <- terra::xyFromCell(r, c(1, 8, 20, 37, 57, 64))
graph <- conductance_surface(r, sites)
simulated <- simulate_covariance_response(c(x = 0.6), ~x, graph,
                                          nu = 1000, seed = 93)
response <- simulated$covariance
fit <- terradish(response ~ x, graph, measurement_model = wishart_covariance,
                 nu = 1000)
environment <- pairwise_covariates(
  pairwise_endpoint_covariates(data.frame(climate = c(2, 4, 1, 7, 5, 9))))
measurement <- wishart_covariates(environment, model = "wishart_covariance")
kernel <- attr(measurement, "kernel_covariates")[, , 1]
response <- response + 0.1 * kernel
joint <- terradish(response ~ x, graph, measurement_model = measurement,
                   nu = 1000)
terradish_ibe_ratio(joint)
# Each ratio is the resistance-distance equivalent of one unit of difference.
```

---

terradish_inference *Covariance and intervals for fitted conductance parameters*

Description

Extracts observed-curvature uncertainty from a full or slim fit.

Usage

```
## S3 method for class 'terradish'
vcov(object, ...)

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

## S3 method for class 'terradish'
confint(object, parm, level = 0.95, ...)
```

Arguments

object	A fitted terradish object.
...	Unused arguments for generic compatibility.
parm	Names or indices of parameters to include.
level	Confidence level between zero and one.

Details

Gaussian scale intervals are Wald intervals truncated at the fitted bounds. Stored profile intervals take precedence when their confidence level matches. The observation count is the number of focal sites (individuals or populations), the sample size that AICc and BIC use in aic_table. Pairwise observations are not counted, because pairs that share a site are not independent.

Value

vcov() returns a covariance matrix; confint() returns a two-column interval matrix; nobs() returns the number of focal sites.

See Also

terradish, slim_terradish, gaussian_scale_profile, terradish_rescale_nu

Examples

```
# A small model-based example, with fixed information supplied explicitly.
r <- terra::rast(nrows = 8, ncols = 8, xmin = 0, xmax = 8, ymin = 0, ymax = 8)
terra::values(r) <- sin(seq_len(64) / 5) + seq_len(64) / 64
names(r) <- "x"
sites <- terra::xyFromCell(r, c(1, 8, 20, 37, 57, 64))
graph <- conductance_surface(r, sites)
simulated <- simulate_covariance_response(c(x = 0.6), ~x, graph,
                                          nu = 1000, seed = 93)
response <- simulated$covariance
fit <- terradish(response ~ x, graph, measurement_model = wishart_covariance,
                 nu = 1000)
coef(fit)       # Relative log conductance per unit of x.
vcov(fit)       # Joint uncertainty in conductance parameters.
confint(fit)    # A model-based interval conditional on the supplied nu.
nobs(fit)       # Number of focal sites, the sample size for AICc and BIC.
```

terradish_measurement_model *Measurement models*

Description

Functions that fit a matrix observed genetic distances to a submatrix of the inverse graph Laplacian

Usage

```
terradish_measurement_model
```

Details

For the ordinary two-component wishart_covariance and generalized_wishart models with their standard bounds, nuisance fitting first profiles the overall covariance variance analytically and searches the relative graph/nugget contribution on site contrasts. This deterministic initialization avoids false convergence when a warm log nugget has an almost zero derivative. The existing Newton optimizer then refines the result and computes derivatives at the same parameters. Fixed predictive parameters and custom bounds retain their supplied values and constraints. The finite profile search does not guarantee a global optimum for every possible covariance matrix. A nugget approaching zero remains a limiting estimate and can imply weak nuisance identification.

See Also

leastsquares, mlpe, generalized_wishart

terradish_methods *Methods for fitted terradish models*

Description

S3 methods for working with fitted objects returned by terradish and summaries returned by summary().

Usage

```
## S3 method for class 'terradish'
print(x, digits = max(3L, getOption("digits") - 3L), ...)

## S3 method for class 'terradish'
summary(object, conf.level = 0.95, ...)

## S3 method for class 'summary.terradish'
print(
  x,
```

```
  digits = max(3L, getOption("digits") - 3L),
  signif.stars = getOption("show.signif.stars"),
  ...
)

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

## S3 method for class 'terradish'
fitted(object, type = c("response", "distance", "covariance"), ...)

## S3 method for class 'terradish'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  method = c("permutation", "parametric"),
  ...
)

## S3 method for class 'terradish'
anova(object, ..., alternative = NULL)

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

## S3 method for class 'terradish'
AIC(object, ..., k = 2)

## S3 method for class 'terradish'
residuals(object, ...)
```

Arguments

x	A fitted terradish object.
digits	Number of digits to print.
...	Additional arguments passed through to generic methods.
object, alternative	Fitted terradish objects.
conf.level	Confidence level for nuisance-parameter confidence intervals reported by summary(). These intervals are conditional on the fitted conductance surface.
signif.stars	Should significance stars be printed in coefficient tables?
type	Type of fitted values to return.
nsim	Number of simulated response matrices to generate.
seed	Optional random seed used by simulate().
method	Simulation method.
k	Penalty multiplier supplied to AIC().

Details

Nuisance-parameter estimates and intervals from `summary()` remain on the parameterization used by the measurement model. For example, MLPE `tau` is log precision and its stored `rho` is unconstrained; the actual shared-site correlation is plogis$(\rho)/2$. Consult the selected measurement-model help page before interpreting `phi`. For a verified numerical zero residual-variance limit in a Wishart fit, `summary()` retains the finite log-scale representation of `sigma` but suppresses its Wald standard error and interval. A zero residual variance does not imply a zero graph contribution. Covariance-response residuals are centered on both site dimensions: $H(S-\Sigma)H$, where H is the site-centering matrix. Fitted covariance matrices retain their full $n \times n$ form.

`anova()` is a likelihood-ratio test for nested conductance formulas fitted to the same response, sites, graph domain, conductance-model factory, measurement model, and pair subset; Wishart fits must also use the same `nu`. Both fits must record successful outer and final nuisance convergence; missing diagnostics require refitting. The function checks formula-term nesting, but the analyst must still verify that the full measurement and conductance parameterization is substantively nested. Its chi-squared reference distribution assumes the tested parameters are regular interior parameters. It can be unreliable when a null value is on a parameter boundary or other regularity conditions fail; use a justified alternative or simulation calibration in those cases.

Value

- `print()` returns its input invisibly.
- `summary()` returns an object of class `summary.terradish`, with legacy `summary.radish` compatibility retained.
- `coef()` returns the fitted conductance coefficients.
- `fitted()` returns fitted responses, distances, or covariance.
- `simulate()` returns one or more simulated response matrices.
- `anova()` returns a likelihood-ratio comparison table.
- `logLik()` returns a `logLik` object.
- `AIC()` returns $-2\ell + kK$; the default `k = 2` gives Akaike's Information Criterion.
- `residuals()` returns the residual response matrix.

See Also

`terradish` for fitting, `conductance` for the fitted surface as a raster, `plot.terradish` for the five diagnostic plot types, `aic_table` for ranking a model set, and `mlpe_response_change` for putting an MLPE covariate association on the response scale. See `vignette("getting-started", package = "terradish")` for a walkthrough of reading these outputs and `vignette("model-comparison", package = "terradish")` for `anova()` and `AIC()` in context.

Examples

```
data(melip)
covariates <- c(terra::unwrap(melip.altitude), terra::unwrap(melip.forestcover))
names(covariates) <- c("altitude", "forestcover")
covariates <- scale_covariates(terra::aggregate(covariates, fact = 3,
                                                na.rm = TRUE))
```

```
surface <- conductance_surface(covariates, terra::unwrap(melip.coords),
                               directions = 8)

fit <- terradish(melip.Fst ~ altitude + forestcover, data = surface,
                 conductance_model = loglinear_conductance,
                 measurement_model = mlpe)

fit                    # coefficients, log-likelihood, and AIC
summary(fit)           # adds standard errors, Wald tests, and the phi table

# Conductance coefficients are on the LOG scale, so exponentiate for a
# multiplicative reading: exp(theta) is the change in conductance per
# one-unit (one standard deviation, here) increase in the covariate.
coef(fit)
exp(coef(fit))

logLik(fit)            # carries a df attribute, so AIC()/BIC() work
AIC(fit)               # compare only fits using the same response and likelihood family
head(fitted(fit)[, 1:4])     # fitted genetic distances
head(residuals(fit)[, 1:4])  # observed minus fitted

# A nested comparison: is forest cover worth its parameter?
fit_alt <- terradish(melip.Fst ~ altitude, data = surface,
                     conductance_model = loglinear_conductance,
                     measurement_model = mlpe)
anova(fit_alt, fit)
```

`terradish_rescale_nu` *Rescale the effective information in a Wishart fit*

Description

Changes `nu` without refitting the coefficients. Wishart objectives and their derivatives are proportional to this user-supplied quantity.

Usage

```
terradish_rescale_nu(fit, nu)
```

Arguments

`fit`	A full or slim fitted Wishart model.
`nu`	New finite, positive value of `nu`.

Details

This is a sensitivity calculation, not an estimate of effective information. It does not remedy misspecification or dependent histories. If `nu` is divided by 20, standard errors grow by `sqrt(20)`; the likelihood and likelihood-ratio statistics are divided by 20. AIC changes because its likelihood term changes while its parameter penalty stays fixed. Report conclusions over a scientifically plausible range.

Value

A fitted object with rescaled likelihood, curvature, AIC, and comparison metadata. Estimates are unchanged; `nu_original` records the original value across repeated rescalings. Standard errors computed by `summary()` and `vcov()` use the rescaled curvature.

See Also

`terradish_inference`, `aic_table`, `terradish_cv_folds`

Examples

```
# A small model-based example, with fixed information supplied explicitly.
r <- terra::rast(nrows = 8, ncols = 8, xmin = 0, xmax = 8, ymin = 0, ymax = 8)
terra::values(r) <- sin(seq_len(64) / 5) + seq_len(64) / 64
names(r) <- "x"
sites <- terra::xyFromCell(r, c(1, 8, 20, 37, 57, 64))
graph <- conductance_surface(r, sites)
simulated <- simulate_covariance_response(c(x = 0.6), ~x, graph,
                                          nu = 1000, seed = 93)
response <- simulated$covariance
fit <- terradish(response ~ x, graph, measurement_model = wishart_covariance,
                 nu = 1000)
sensitivity <- terradish_rescale_nu(fit, nu = 50)
rbind(original = coef(fit), sensitivity = coef(sensitivity))
rbind(original = sqrt(diag(vcov(fit))),
      sensitivity = sqrt(diag(vcov(sensitivity))))
```

`terradish_scale_optim` *Optimize raster smoothing scales around a terradish fit*

Description

Performs an outer optimization over raster scale parameters by repeatedly rescaling raster covariates, rebuilding or updating the conductance surface, and refitting `terradish`.

Usage

```
terradish_scale_optim(
  formula,
  covariates,
  coords,
  scales = NULL,
  lower = NULL,
  upper = NULL,
  scale_grid = NULL,
  search = c("coordinate", "grid"),
  objective = c("aic", "logLik"),
  grid_points = 5L,
  maxit = 10L,
  tol = 0.1,
  scale_fun = "terradish",
  scale_args = NULL,
  postprocess = scale_covariates,
  directions = 8L,
  theta = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

formula	A terradish model formula.
covariates	A terra::SpatRaster containing the raster layers whose smoothing scales will be optimized.
coords	Focal coordinates passed to conductance_surface.
scales	Optional starting values for the scale parameters. Required for search = "coordinate" unless scale_grid is supplied.
lower, upper	Lower and upper bounds for each scale parameter, in map units for both built-in smoothing methods. The default uses the same normalized, truncated Gaussian as the joint scale model. A useful upper bound is one sixth of the smaller raster dimension times its cell width. Used to construct scale_grid when that is not supplied, and required for search = "coordinate".
scale_grid	Optional named list of candidate scale values for search = "grid".
search	One of "coordinate" or "grid".
objective	Objective used for the outer optimization: "aic" or "logLik".
grid_points	Number of equally spaced candidate scales per raster when search = "grid" and scale_grid is not supplied.
maxit	Maximum number of outer coordinate-search iterations.
tol	Convergence tolerance for coordinate search, measured as the maximum absolute change in the scale vector between iterations.

scale_fun	Smoothing method: "terradish" (default) or "multiScaleR". Both use map units, but their kernel constructions differ, so their fitted scales are not interchangeable. A custom function may instead accept a single-layer raster and a numeric scale= argument. NULL selects the default.
scale_args	Optional named list passed to scale_fun.
postprocess	Optional function applied to the full scaled raster stack before constructing the conductance surface. Defaults to scale_covariates so each scaled layer is re-standardized. Set to NULL to use the scaled rasters as returned by scale_fun.
directions	Neighborhood definition passed to conductance_surface.
theta	Optional starting values passed to terradish at each outer evaluation.
verbose	Print outer-optimization progress?
...	Additional arguments passed to terradish.

Details

This helper treats scale optimization as an outer problem around the usual terradish fit, rather than as an inner Newton/BFGS parameter. That makes it compatible with raster-scaling approaches such as multiScaleR::kernel_scale.raster(), while keeping the terradish core optimizer unchanged. The selected values describe the transformation performed by scale_fun in that function's units. They are tuning parameters for the raster representation, not direct estimates of dispersal distance, home-range size, organismal movement, or a causal ecological-process scale. Report the search range and objective, and account for this outer selection when describing model-selection uncertainty. Each searched scale counts toward the parameter total used for information criteria. Prefer the joint Gaussian model when its kernel matches the scientific question; this outer search is a secondary option for other kernels or an explicit scale grid. The default terradish kernel matches the joint model, whereas multiScaleR uses a different construction even when both scales are expressed in map units.

Coordinate search updates one raster scale at a time using optimize while holding the others fixed. Grid search evaluates every combination in scale_grid. In both modes, failed terradish fits receive an infinite outer objective and are retained in the returned evaluation table.

Value

An object of class "terradish_scale_optim", a list with:

par	The selected scale for each raster layer, in the units scale_fun uses (map units for both built-in methods).
value	The outer objective at par. Lower is better, whichever objective was used.
fit	The fitted terradish model at par. Read its conductance coefficients with coef() the same way as any other fit; they are on the log-conductance scale.
surface	The conductance surface built from the rasters smoothed at par, ready to pass to conductance() or plot().
evaluations	A data frame with one row per candidate scale combination: the scale for each layer, the objective value, aic, loglik, and a status column that records "OK" or the error for candidates that failed to fit. Inspect this to see how sharply the objective distinguishes the scales; a flat column means the objective does not identify a smoothing scale within the searched range.

`search` The search strategy used, `"grid"` or `"coordinate"`.

`objective` The name of the objective minimized, `"aic"` or `"logLik"`.

`iterations, convergence`
Number of outer iterations, and `0` if the coordinate search converged (always `0` for a grid search).

Examples

```
data(melip)
melip.altitude <- terra::unwrap(melip.altitude)
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords <- terra::unwrap(melip.coords)
covariates <- c(melip.altitude, melip.forestcover)
names(covariates) <- c("altitude", "forestcover")

# Every candidate scale re-smooths the rasters and refits, so coarsen the
# grid for the example. Use the full resolution in a real analysis.
covariates <- terra::aggregate(covariates, fact = 3, na.rm = TRUE)

# Express the bounds in map units, using the coarsened cell width.
cell_width <- mean(terra::res(covariates))
# A 2-point grid over two layers is 4
# candidate fits; use a wider grid in practice.
fit_scale <- terradish_scale_optim(
  melip.Fst ~ altitude + forestcover,
  covariates = covariates,
  coords = melip.coords,
  lower = cell_width,
  upper = 6 * cell_width,
  search = "grid",
  grid_points = 2,
  measurement_model = leastsquares,
  optimizer = "bfgs"
)
fit_scale$par          # selected smoothing scale per layer, in map units
fit_scale$value        # objective value there (lower is better)
fit_scale$objective    # which objective was minimized ("aic" by default)
fit_scale$evaluations  # every candidate tried, with its AIC and loglik
coef(fit_scale$fit)    # conductance coefficients at the selected scale
```

---

terradish_solver_benchmark

Benchmark linear solver settings for a terradish graph

Description

Times the linear-system setup and solve used internally by `terradish_algorithm` for a fixed graph and conductance vector. This is intended as a lightweight hardware-specific diagnostic for choosing direct factorization settings on large rasters.

Usage

```
terradish_solver_benchmark(
  data,
  conductance = NULL,
  factorization = c("simplicial_ldl", "simplicial_ll", "supernodal_ll"),
  solve_backend = "matrix",
  n_replicates = 1L,
  perm = TRUE
)
```

Arguments

data	A terradish_graph object.
conductance	Optional numeric conductance vector with one value per graph vertex. Defaults to constant conductance.
factorization	Direct solver factorization modes to compare. Supported values are "simplicial_ldl", "simplicial_ll", and "supernodal_ll".
solve_backend	Direct-solver RHS solve backend to compare. "matrix" uses Matrix's standard solve() method. The experimental "cholmod_cpp" backend calls Matrix's CHOLMOD C API directly from compiled code after the factorization has been built. The experimental "cholmod_cpp_cached" backend also keeps the CHOLMOD factorization/update state inside a compiled external pointer.
n_replicates	Number of times to repeat each setting.
perm	Should CHOLMOD use a fill-reducing permutation?

Details

The benchmark isolates the sparse linear algebra step. It does not run a full likelihood optimization, line search, measurement-model subproblem, or Gaussian smoothing update. Use it to learn whether "simplicial_ldl", "simplicial_ll", or "supernodal_ll" is fastest on a particular machine and graph size, then pass the preferred choice through solver_control = list(factorization = "...") in terradish when using the direct solver. Use solve_backend to evaluate whether either experimental C++ CHOLMOD path is faster for the right-hand-side count used by the graph. The cached backend is most informative when compared inside a full optimization, where nearby evaluations can reuse the same symbolic analysis.

Very small rasters can give noisy timings because setup and R overhead are a large fraction of elapsed time. The comparison is most informative on graphs large enough that Cholesky factorization is a visible part of the fit time.

Value

A data frame with one row per benchmark replicate and columns for graph size, factorization mode, solve backend, setup time, solve time, and total time.

Examples

```
library(terra)
data(melip)
melip.altitude <- terra::unwrap(melip.altitude)
melip.coords <- terra::unwrap(melip.coords)
surface <- conductance_surface(melip.altitude, melip.coords, directions = 8)
terradish_solver_benchmark(
  surface,
  factorization = c("simplicial_ldl", "simplicial_ll"),
  n_replicates = 1
)
```

---

wishart_covariance *Wishart covariance measurement model*

Description

A function of class `"terradish_measurement_model"` that evaluates a Wishart likelihood for an observed genetic **covariance** matrix. This is the natural model when raw genotype data are summarized with `cov_from_genetic_data` or `cov_from_biallelic` rather than as pairwise distances. For an admissible squared-distance representation coherently related to a centered positive semidefinite covariance matrix, use `generalized_wishart`. For other pairwise distances, use `mlpe` or `leastsquares`.

Usage

```
wishart_covariance(
  E,
  S,
  phi,
  nu,
  gradient = TRUE,
  hessian = TRUE,
  partial = TRUE,
  nonnegative = TRUE,
  validate = FALSE
)
```

Arguments

`E`	Conductance-implied covariance matrix: the generalized inverse of the graph Laplacian at the current conductance parameters. Passed automatically by the optimizer; users normally do not call this function directly.
`S`	Square, symmetric matrix of observed population-level genetic covariance. Typically the output of `cov_from_genetic_data` or `cov_from_biallelic`. Must have the same dimensions as `E` and should be positive (semi-)definite.

phi	Named numeric vector of nuisance parameters (tau, sigma). Omit to obtain least-squares starting values.
nu	One finite positive number describing effective Wishart information. You must supply it; the fit does not estimate it. It is not the marker count. See the effective-information section and report sensitivity across plausible values with terradish_rescale_nu.
gradient	Logical. Compute gradient of the negative log-likelihood with respect to phi?
hessian	Logical. Compute Hessian with respect to phi?
partial	Logical. Compute second partial derivatives with respect to phi, E, and S?
nonnegative	Unused; present for interface consistency.
validate	Logical. Numerically validate gradients and Hessians via **numDeriv**? Very slow; for debugging small examples only.

Details

The nuisance parameters are:

tau Nonnegative scale applied to the conductance-implied covariance E. A value near zero signals no detectable graph-kernel contribution under this response and model.

sigma Log-scale nugget: the identity component added to the model covariance is $\exp(\sigma)$. It captures diagonal variation not represented by the graph covariance, including sampling variation and model mismatch. It is not a smoothing scale or a direct estimate of genetic drift or population size.

The fitted covariance is $\Sigma = \tau E + \exp(\sigma) I$. The likelihood is evaluated on orthonormal site contrasts. Let L have $n - 1$ orthonormal columns perpendicular to the constant vector, and let $\tilde{\Sigma} = L^T \Sigma L$ and $\tilde{S} = L^T S L$. The negative log-likelihood, omitting terms constant in the fitted parameters for a fixed response and supplied ν, is:

$$-\ell = \frac{\nu}{2} \left[\log |\tilde{\Sigma}| + \operatorname{tr}(\tilde{\Sigma}^{-1} \tilde{S}) \right]$$

The response is a sample covariance on site contrasts with effective degrees of freedom ν. Adding a constant to every entry of S or centering it across sites does not change this objective. This is the same objective as generalized_wishart(dist_from_cov(S)); the nugget contributes $2\exp(\sigma)$ to the expected squared distances. Working on contrasts discards only the overall site-mean direction, which carries no information in a centered genetic covariance.

Preparing the response. Use a coherent covariance construction, such as diagonal = "gower" for grouped data in landgraph. A "within" diagonal is on a different scale and is unsuitable for Wishart fitting, even when the matrix happens to be positive definite. Per-locus standardization upweights rare variants; filter by minor-allele frequency and assess sensitivity. Unequal group sizes create unequal sampling variance that a common diagonal nugget does not automatically fix.

During fitting, terradish checks very small nugget estimates for false convergence on the log scale. Improving finite-variance starts retain the supplied bounds. An unrestricted nugget can instead be represented at a verified numerical zero limit: its finite log value gives a positive variance below floating-point covariance precision, and the one-sided variance gradient must satisfy the stopping tolerance. This does not change the likelihood or override fixed parameters. The fitted object records fit$residual_variance_zero_limit; summary() omits Wald uncertainty for sigma at that limit.

Value

When `phi` is missing, a list with elements `phi` (least-squares starting values for `(tau, sigma)`), `lower` (`c(0, -Inf)`), and `upper`. Otherwise a list containing:

`objective`	Negative log-likelihood.
`fitted`	Fitted covariance matrix Σ (same dimensions as `S`).
`boundary`	Logical; `TRUE` if `tau = 0`, indicating no detectable IBR signal.
`gradient`	Gradient with respect to `phi` (if `gradient = TRUE`).
`hessian`	Hessian matrix with respect to `phi` (if `hessian = TRUE`).
`gradient_E`	Gradient with respect to `E` (if `partial = TRUE`).
`partial_E`	Jacobian of `gradient_E` with respect to `phi` (if `partial = TRUE`).
`partial_S`	Jacobian of `gradient` with respect to the lower triangle of `S` (if `partial = TRUE`).
`jacobian_E`	Function for reverse-mode AD through `E` (if `partial = TRUE`).
`jacobian_S`	Function for reverse-mode AD through `S` (if `partial = TRUE`).

Worked examples

`vignette("wishart-covariance", package = "terradish")`, `vignette("individual-salamanders", package = "terradish")`.

Effective Wishart information

For Wishart likelihoods, `nu` scales the log likelihood. Point estimates do not depend on it, but standard errors, likelihood-ratio tests, and information criteria do. It is a dispersion parameter that you set, rather than the number of markers. Linkage and the population history shared by loci can make effective information much smaller than the SNP count. In forward-time simulations used to validate terradish, standard errors at the SNP count were three to five times smaller than the spread among replicate histories. Those model-based errors understated uncertainty.

Use `terradish_cv_folds` to compare conductance formulas, and `terradish_rescale_nu` to report inference across plausible values. For the same successful folds and nuisance settings, changing `nu` rescales Wishart CV scores without changing their ranking. Neither procedure identifies the true effective information or repairs an unsuitable response matrix.

See Also

`cov_from_genetic_data`, `cov_from_biallelic`, `generalized_wishart`, `wishart_covariates`, `simulate_covariance_response`, `terradish`

Examples

```
E <- matrix(c(1.2, 0.3, 0.2,
              0.3, 1.5, 0.4,
              0.2, 0.4, 1.1), nrow = 3, byrow = TRUE)
Sigma <- 0.8 * E + 0.2 * diag(3)
S <- rWishart(1, df = 20, Sigma = Sigma)[,,1] / 20
wishart_covariance(E, S, phi = c(0.8, log(0.2)), nu = 20)
```

`wishart_covariates` *Wishart measurement models with site-level environmental covariance kernels*

Description

Creates a Wishart measurement model whose fitted covariance includes the resistance-implied covariance plus one or more positive semidefinite kernels built from site-level environmental covariates. Use this as the Wishart analogue of `mlpe_covariates` when your measurement model is `generalized_wishart` (squared-distance responses) or `wishart_covariance` (covariance-matrix responses).

Usage

```
wishart_covariates(
  x,
  coords = NULL,
  transform = c("absdiff", "sqdiff", "euclidean", "manhattan"),
  scale = FALSE,
  model = c("wishart_covariance", "generalized_wishart")
)
```

Arguments

`x`	Site-level covariates. Supported inputs are the same as `pairwise_endpoint_covariates`: a numeric vector, matrix, data frame, or `terra::SpatRaster`. A previously constructed `terradish_pairwise_covariates` object is accepted unchanged.
`coords`	Required when `x` is a raster. Focal-point coordinates in the same projection as `x`; accepts the same inputs as `conductance_surface`.
`transform`	Pairwise environmental difference: `"absdiff"` (default), `"sqdiff"`, `"euclidean"`, or `"manhattan"`. See `pairwise_endpoint_covariates`. Ignored for an existing pairwise-covariate object.
`scale`	Logical. Standardize site-level covariates to zero mean and unit variance before constructing kernels? Recommended when covariates use different units or scales.
`model`	Which Wishart likelihood to use. `"generalized_wishart"` is appropriate only when `S` is an admissible squared-distance matrix coherently related to a centered positive-semidefinite covariance matrix. An arbitrary distance matrix, including an unchecked F_{ST} matrix, is not sufficient. `"wishart_covariance"` is appropriate when `S` is a **covariance** matrix (e.g. from `cov_from_genetic_data`). In both cases the same Σ parameterization is used; they differ in how Σ is compared to the observed data.

Details

A zero kernel coefficient sets the boundary flag while retaining conductance inference when the resistance coefficient is positive. Summaries mark zero kernel coefficients and give one-sided upper limits instead of symmetric intervals. Tests adding only kernel coefficients use a chi-bar-square reference with binomial weights. This reference is exact for information-orthogonal kernels and approximate otherwise. MLPE environmental coefficients are unconstrained and retain ordinary chi-square tests.

`wishart_covariates()` is the Wishart analogue of `mlpe_covariates`, but environmental covariates enter the model through positive semidefinite kernels. For each pairwise difference matrix D_k, the kernel is $K_k = -\frac{1}{2} H D_k H$, where H centers sites. A kernel with a negative eigenvalue beyond numerical tolerance is rejected. The `"sqdiff"` transform gives the centered outer product of the original covariate. The fitted covariance is:

$$\Sigma = \tau E(\theta) + \sum_k \lambda_k K_k + \exp(\sigma) I$$

where $E(\theta)$ is the resistance-implied covariance (generalized inverse of the graph Laplacian at conductance parameters θ), τ is the weight on conductance-implied covariance, $\lambda_k \geq 0$ are nonnegative kernel weights, and $\exp(\sigma)$ is diagonal residual variance not represented by the graph or environmental kernels.

The nuisance parameters estimated alongside θ are:

`tau` Nonnegative scale on the resistance-implied covariance E. A value near zero indicates no detectable contribution from conductance-implied covariance under this model.

`lambda_<covariate>` Nonnegative weight per unit of the corresponding pairwise environmental difference. A negative association is represented at the zero bound; MLPE environmental slopes are unconstrained. The weight is not a causal IBE effect.

`sigma` Log-scale nugget: identity component added as $\exp(\sigma) I$, representing diagonal variance not captured by the other covariance components. It is not a smoothing scale or direct drift or population-size estimate.

This design naturally preserves positive definiteness of Σ when the nugget is positive and all kernel weights are nonnegative. It is evaluated on site contrasts, as in `wishart_covariance`. The implied squared-distance mean is $\tau R_{ij} + \sum_k \lambda_k D_{k,ij} + 2\exp(\sigma)$, using the same additive environmental terms as MLPE. The likelihoods and coefficient constraints still differ. Use `wishart_covariates()` when the response satisfies the selected Wishart model's requirements and you can justify an effective degrees-of-freedom value `nu`; use `mlpe_covariates` when `nu` is unknown or when the distance-regression framework better matches the response. Neither formulation cleanly separates causal IBE from IBR when predictors and kernels are spatially correlated. Scaling is applied to site values before the pairwise transform; kernels are not normalized afterward. Site subsets retain these original pairwise values and rebuild centered kernels without restandardizing the subset. In the package's null-simulation audit, AIC selected an environmental kernel in 19 of 40 runs at the SNP-count value of `nu`, and in none at a replicate-calibrated value. These are results of that design, not general error-rate guarantees. Use `terradish_rescale_nu` for sensitivity and `terradish_cv_folds(..., nuisance = "fixed")` to assess predictive support for a measurement-model extension.

Value

A function of class "terradish_measurement_model" suitable for the measurement_model argument of terradish and terradish_grid. The nuisance parameter vector ϕ estimated by terradish contains tau, one lambda_<covariate> per environmental kernel, and sigma (in that order). The function stores the kernel array in attribute "kernel_covariates" and supports site-subsetting for model fitting through terradish.

Numerical nugget limits

Fitting uses the same small-nugget safeguards as wishart_covariance, retaining supplied bounds and fixed parameters. A verified numerical zero residual-variance limit does not remove the graph or environmental kernel coefficients. summary() omits Wald uncertainty for the log-scale nugget at that limit.

Worked example

See vignette("ibe-ibr-workflow", package = "terradish") for environmental covariance kernels and their distinction from MLPE slopes.

Interpreting pairwise effects

Construct one pairwise_covariates object to use the same environmental transforms and geographic distances in both likelihood families. Their fitted conductance surfaces are conditional on these terms, and conductance signs can change when the mean structure changes. Report fits with and without pairwise effects. A uniform ~ 1 conductance model with pairwise terms represents IBD plus IBE, not IBE alone.

A positive environmental coefficient means differentiation increases with environmental difference conditional on resistance. It does not distinguish assortative mating, dispersal filtering, local adaptation, or site variance that rises at environmental extremes. MLPE coefficients are unconstrained; Wishart kernel weights are nonnegative. Compare their effects in resistance-distance units with terradish_ibe_ratio, which divides by the resistance coefficient and propagates joint uncertainty. Ratios are undefined when that denominator is zero.

Effective Wishart information

For Wishart likelihoods, nu scales the log likelihood. Point estimates do not depend on it, but standard errors, likelihood-ratio tests, and information criteria do. It is a dispersion parameter that you set, rather than the number of markers. Linkage and the population history shared by loci can make effective information much smaller than the SNP count. In forward-time simulations used to validate terradish, standard errors at the SNP count were three to five times smaller than the spread among replicate histories. Those model-based errors understated uncertainty.

Use terradish_cv_folds to compare conductance formulas, and terradish_rescale_nu to report inference across plausible values. For the same successful folds and nuisance settings, changing nu rescales Wishart CV scores without changing their ranking. Neither procedure identifies the true effective information or repairs an unsuitable response matrix.

References

McCullagh P. 2009. Marginal likelihood for distance matrices. Statistica Sinica 19:631-649.

See Also

check_distance_response, generalized_wishart, wishart_covariance, mlpe_covariates, pairwise_endpoint_covariates, terradish

Examples

```
library(terra)

data(melip)
melip.altitude    <- terra::unwrap(melip.altitude)
melip.forestcover <- terra::unwrap(melip.forestcover)
melip.coords      <- terra::unwrap(melip.coords)

covariates <- c(melip.altitude, melip.forestcover)
names(covariates) <- c("altitude", "forestcover")
covariates <- scale_covariates(terra::aggregate(covariates, fact = 3,
                                                na.rm = TRUE))
surface <- conductance_surface(covariates, melip.coords, directions = 8)

# Build an altitude kernel for a covariance-Wishart model.
g_wc <- wishart_covariates(melip.altitude, coords = melip.coords,
                           model = "wishart_covariance", scale = TRUE)
inherits(g_wc, "terradish_measurement_model")  # TRUE

# Simulate an admissible covariance response, then fit the extended model.

simulated <- simulate_covariance_response(
  theta = c(altitude = 0.2, forestcover = -0.1),
  formula = ~ altitude + forestcover,
  data = surface, tau = 0.8, sigma = 0.2, nu = 40, seed = 1
)
fit_joint_w <- terradish(
  simulated$covariance ~ altitude + forestcover,
  data              = surface,
  conductance_model = loglinear_conductance,
  measurement_model = g_wc,
  nu                = 40
)
# phi table shows tau, lambda_altitude, and sigma.
summary(fit_joint_w)
```

Index

∗ **datasets**
individual_salamanders, 29
melip, 36
ohio_ponds, 47

AIC.terradish *(*terradish_methods*)*, 92
aic_table, 3, *72, 73*, *91*, *94*, *96*
anova.terradish *(*terradish_methods*)*, 92
ArmijoControl, 5, *29*, *46*
as.factor, *11*

bs, *66*

check_distance_response, 6, *25, 26*, *107*
coef, *14*
coef.terradish *(*terradish_methods*)*, 92
conductance, 8, *56*, *94*
conductance_surface, *8, 9*, 10, *13, 14*, *17*, *21*, *34*, *37*, *41*, *49*, *51*, *60, 61*, *63*, *66–68*, *71*, *78*, *82*, *85*, *88*, *97, 98*, *104*
confint.terradish *(*terradish_inference*)*, 90
cov_from_biallelic, *30, 31*, *49*, *101*, *103*
cov_from_genetic_data, *31*, *49*, *53*, *101*, *103, 104*
covariance_response_power, 13, *57*, *63*
crop_to_focal_buffer, *11*, 16

D, *22*
dist_from_biallelic, *53*
dist_from_cov, *31*

fitted.terradish, *56*
fitted.terradish *(*terradish_methods*)*, 92

gaussian_scale_profile, 18, *20*, *22*, *91*
gaussian_scale_summary, *18*, 19, *37*
gaussian_smoothed_loglinear_conductance, *14*, *18, 19*, 21, *34*, *71*
generalized_wishart, *6*, *8*, 24, *33*, *36*, *38*, *40*, *68*, *79*, *92*, *101*, *103, 104*, *107*

HagerZhangControl, *6*, 27, *46*

individual_salamanders, 29

leastsquares, 31, *40*, *79*, *92*, *101*
logLik.terradish *(*terradish_methods*)*, 92
loglinear_conductance, *22*, 33, *55, 56*, *63*, *66, 67*, *78*, *81*
lower, 35
lower.tri, *35*

melip, 36
mlpe, *7, 8*, *25, 26*, *31*, *33*, *36*, 38, *42*, *67*, *79*, *92*, *101*
mlpe_covariates, *39, 40*, 41, *43, 44*, *50*, *52*, *90*, *104, 105*, *107*
mlpe_response_change, 43, *94*
model.matrix, *11*, *34*

NewtonRaphsonControl, *6*, *28, 29*, 45, *69, 70*, *77*, *83*
nobs.terradish *(*terradish_inference*)*, 90
ns, *66*

ohio_ponds, 47
optimize, *98*

pairwise_covariates, *42*, 49, *50*, *52*, *90*, *106*
pairwise_endpoint_covariates, *41, 42*, *44*, 50, *50*, *104*, *107*
pca_dist, 52
plot, *55*
plot.terradish, *9*, 53, *94*
prcomp, *53*
print.summary.terradish *(*terradish_methods*)*, 92
print.terradish *(*terradish_methods*)*, 92
print.terradish_covariance_power, 57
print.terradish_setting_assessment, 58

residuals.terradish *(*terradish_methods*)*, 92

scale_covariates, *10*, *17*, *21, 22*, *37*, 60, *62*, *98*
scale_to_0_1, *53*, 61, *61*
simulate.terradish *(*terradish_methods*)*, 92
simulate_covariance_response, *13*, *57*, 62, *103*
slim_terradish, 64, *70*, *91*
smooth_loglinear_conductance, *54*, *56*, 65, *80, 81*
summary(), *92*
summary.terradish *(*terradish_methods*)*, 92

terradish, *9, 10*, *14*, *19*, *22, 23*, *26*, *28*, *33, 34*, *37*, *40*, *42*, *44–46*, *52, 53*, *56, 57*, *59*, 67, *67*, *77, 78*, *83*, *91, 92*, *94*, *96*, *98*, *100*, *102, 103*, *106, 107*
terradish_algorithm, *69*, 74, *99*
terradish_assess_settings, *58, 59*, 78
terradish_conductance_model_factory, *68*, 81, *85*, *88*
terradish_cv_folds, *4*, *26*, *42*, *51, 52*, *73*, 82, *86*, *96*, *103*, *106*
terradish_distance, *63*, 84
terradish_folds, *82*, 86
terradish_grid, *42*, 87, *106*
terradish_ibe_ratio, *42*, *50*, *52*, 89, *106*
terradish_inference, *18*, 90, *96*
terradish_measurement_model, *68*, *88*, 92
terradish_methods, *72*, 92
terradish_rescale_nu, *4*, *25, 26*, *68*, *73*, *90, 91*, 95, *102, 103*, *105, 106*
terradish_scale_optim, 96
terradish_solver_benchmark, *59*, *79*, 99

vcov.terradish *(*terradish_inference*)*, 90

wishart_covariance, *8*, *15*, *24–26*, *33*, *53, 54*, *62, 63*, *68*, *79*, *88*, *92*, 101, *104–107*
wishart_covariates, *26*, *50*, *90*, *103*, 104