Package 'geosmooth'

November 12, 2026

Title Geometric Methods for Nonparametric Regression and Density Estimation

Version 0.2.1

Description Provides geometric methods for nonparametric regression and density estimation on data represented as coordinate matrices or weighted graphs.
Methods include local polynomial smoothing, model-averaged local polynomial smoothing, local polynomial lifting trend filtering, synchronized local polynomial lifting trend filtering, order-2 Gaussian and logistic trend filtering, graph low-pass filtering, and
Hessian-energy regression. Methodological references include Gajer and Ravel (2025) ``Adaptive Geometric Regression for High-Dimensional Structured Data" <doi:10.48550/arXiv.2511.03817>, Fan and Gijbels (1996, ISBN:9780412983214), Wang et al. (2016) ``Trend Filtering on Graphs" <https://www.jmlr.org/papers/v17/15-147.html>, and Kim et al. (2009) ``Semi-Supervised Regression Using Hessian Energy" <https://papers.nips.cc/paper/3741-semi-supervised-regression-using-hessian-energy-with-an-application-to-semi-supervised-dimer

Copyright file inst/COPYRIGHTS

License GPL (>= 3)

URL https://pgajer.github.io/geosmooth/, https://github.com/pgajer/geosmooth

BugReports https://github.com/pgajer/geosmooth/issues

Encoding UTF-8

Language en-US

SystemRequirements GNU make

LinkingTo Rcpp, dgraphs (>= 0.4.0)

Depends R (>= 4.1.0)

Imports dgraphs (>= 0.4.0), digest, jsonlite, MASS, Matrix, methods, parallel, Rcpp, stats, utils

Suggests genlasso, grip (>= 0.3.0), knitr, rmarkdown, splines, testthat (>= 3.0.0), waldo, withr

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/start-first synthetic-registry, ps-lps, state-density-odcv3-lps-outer-visit-cv, synthetic-legacy-ssrhe, coupled-kd-selection-csd4, graph-trend-filtering, simplex-charts

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Pawel Gajer [aut, cre], Gael Guennebaud [ctb] (Eigen), Benoit Jacob [ctb] (Eigen), Authors of Eigen [cph] (Authorship and copyright in the included Eigen library), Yixuan Qiu [ctb, cph] (Spectra), Contributors to Spectra [cph] (Copyright in the included Spectra library), Sunil Arya [ctb, cph] (ANN), David M. Mount [ctb, cph] (ANN), University of Maryland [cph] (ANN)

Maintainer Pawel Gajer <pgajer@gmail.com>

Repository CRAN

Date/Publication 2026-11-12 13:20:10 UTC

Contents

geosmooth-package 4
apply.metric.graph.lowpass.path 6
as.data.frame.synthetic_dataset 7
assess.gaussian.trend.filtering 8
assess.logistic.trend.filtering 10
bootstrap.malps 14
compare.masked.graph.fields 15
compare.synthetic.dataset 16
cv.masked.graph.field 17
cv.projection.trend.filtering 18
fit.chart.kernel 20
fit.density 23
fit.gaussian.trend.filtering 24
fit.graph.trend.filtering 26
fit.local.likelihood 29
fit.logistic.trend.filtering 32
fit.lpl.tf 35
fit.lps 37
fit.malps 42
fit.masked.graph.field 47

fit.metric.graph.lowpass . . . 48
fit.neighbor.probability . . . 51
fit.projection.trend.filtering . . . 53
fit.ps.lps . . . 56
fit.pttf.trend.filtering . . . 60
fit.slpl.tf . . . 63
fit.ssrhe.hessian.l1.regression . . . 65
fit.ssrhe.hessian.regression . . . 70
fit.subject.od . . . 75
get.region.boundary . . . 77
graph.trend.filtering.operator . . . 79
harmonic.smoother . . . 80
lpl.tf.operator . . . 83
lps.backend.diagnostics . . . 85
lps.grouped.foldid . . . 86
lps.nested.cv . . . 87
lps.pointwise.band . . . 89
malps.gcv . . . 91
materialize.synthetic . . . 92
materialize.synthetic.instance . . . 93
metric.graph.heat.eta.grid . . . 94
metric.graph.heat.extend.lower . . . 95
metric.graph.lowpass.basis . . . 96
metric.graph.lowpass.operator . . . 98
model-summary . . . 101
model-values . . . 103
normalize.density . . . 106
perform.harmonic.smoothing . . . 108
plot.harmonic_smoother . . . 110
plot.synthetic_dataset . . . 111
predict.gaussian.trend.filtering.fit . . . 112
predict.logistic.trend.filtering.fit . . . 113
predict.lpl_tf . . . 114
predict.lps . . . 115
predict.malps . . . 116
predict.neighbor.probability.fit . . . 117
predict.projection.trend.filtering.fit . . . 118
predict.slpl_tf . . . 119
print.graph.trend.filtering.fit . . . 120
print.harmonic_smoother . . . 122
print.summary.harmonic_smoother . . . 123
print.synthetic_dataset . . . 124
pttf.geometry . . . 124
pttf.operator . . . 126
pttf.operator.filter.rows . . . 128
refit . . . 129
refit.lpl_tf . . . 130
refit.malps . . . 131

refit.metric.graph.lowpass.fit . . . 133
refit.slpl_tf . . . 134
refit.ssrhe.hessian.fit . . . 135
refit.ssrhe.hessian.l1.fit . . . 136
select.masked.graph.field . . . 138
slpl.tf.operator . . . 139
smoother.matrix . . . 142
smoother.matrix.lps . . . 143
smoother.matrix.malps . . . 144
ssrhe.hessian.operator . . . 145
ssrhe.support.grid . . . 149
summary.harmonic_smoother . . . 150
support.masked.graph.field . . . 151
synthetic.dataset.checksum . . . 152
synthetic.registry.ids . . . 152
synthetic.registry.seed . . . 153
synthetic.registry.spec . . . 153
synthetic.response.bernoulli . . . 154
synthetic.response.clustered.gaussian . . . 155
synthetic.response.gaussian . . . 155
synthetic.response.heteroskedastic.gaussian . . . 156
synthetic.response.laplace.outlier . . . 156
synthetic.sampling.stratified . . . 157
synthetic.spec . . . 158
synthetic.truth.gaussian.mixture . . . 159
synthetic.truth.logit . . . 160
synthetic.truth.named . . . 160
synthetic.truth.occupation.mixture . . . 161
synthetic.truth.polynomial . . . 162
transported.graph.hessian.operator . . . 162
unfold.simplex.chart . . . 168
validate.synthetic.dataset . . . 169

Index **171**

`geosmooth-package` *geosmooth: Geometric Methods for Nonparametric Regression and Density Estimation*

Description

`geosmooth` estimates a smooth response, or a probability mass, from data given as a matrix of coordinates (one row per observation) or as a weighted graph. It provides local polynomial smoothing, model-averaged local polynomial smoothing, several forms of trend filtering, graph low-pass filtering, Hessian-energy regression, and estimators of probability mass and occupation density.

Start here

`vignette("function-guide", package = "geosmooth")` is organized by task. It lists every exported function, says which fitted models can predict at new points, and gives examples. `browseVignettes("geosmooth")` lists all installed guides. The online version of the guide links to the full reference.

Finding a function

The main functions, by task:

Local regression `fit.lps` fits a local polynomial smoother and selects its settings by cross-validation. `fit.malps` averages the predictions of overlapping local polynomial fits.

Lifting trend filters `fit.lpl.tf` penalizes the absolute residuals of local polynomial predictions. `fit.slpl.tf` adds a penalty that makes overlapping local predictions agree.

One-dimensional trend filters `fit.gaussian.trend.filtering` fits a continuous response and `fit.logistic.trend.filtering` a binary response. `assess.gaussian.trend.filtering` and `assess.logistic.trend.filtering` test whether the response varies with the predictor and assess held-out prediction.

Graph smoothing `fit.metric.graph.lowpass` smooths a response on the vertices of a graph by filtering its spectrum, and `fit.graph.trend.filtering` penalizes differences along the edges.

Hessian penalties `fit.ssrhe.hessian.regression` penalizes estimated second derivatives with a quadratic penalty, and `fit.ssrhe.hessian.l1.regression` with an absolute-value penalty.

Probability mass `fit.density` estimates probability mass over a fixed set of support points, `fit.subject.od` does so from the repeated visits of one subject, and `normalize.density` converts an existing fitted field to probability mass.

Synthetic examples `synthetic.spec` describes a synthetic data set, and `materialize.synthetic` draws the data.

Author(s)

Maintainer: Pawel Gajer <pgajer@gmail.com>

Authors:

- Pawel Gajer <pgajer@gmail.com>

Other contributors:

- Gael Guennebaud (Eigen) [contributor]
- Benoit Jacob (Eigen) [contributor]
- Authors of Eigen (Authorship and copyright in the included Eigen library) [copyright holder]
- Yixuan Qiu (Spectra) [contributor, copyright holder]
- Contributors to Spectra (Copyright in the included Spectra library) [copyright holder]
- Sunil Arya (ANN) [contributor, copyright holder]
- David M. Mount (ANN) [contributor, copyright holder]
- University of Maryland (ANN) [copyright holder]

See Also

vignette("function-guide", package = "geosmooth"), vignette("synthetic-datasets", package = "geosmooth")

Examples

```
X <- matrix(seq(0, 1, length.out = 30), ncol = 1)
y <- sin(2 * pi * X[, 1])
fit <- fit.lps(X, y, support.grid = 8L, degree.grid = 1L,
               kernel.grid = "tricube", foldid = rep(1:3, 10))
head(predict(fit))
```

---

apply.metric.graph.lowpass.path

Apply a Metric Graph Low-Pass Filter Path

Description

Applies every requested low-pass parameter to one response or a matrix of responses using a reusable spectral basis.

Usage

```
apply.metric.graph.lowpass.path(
  basis,
  y,
  eta.grid,
  filter.type = c("heat_kernel", "tikhonov", "cubic_spline", "gaussian", "exponential",
    "butterworth"),
  block.size = NULL,
  truncation.tol = 1e-04,
  unresolved.action = c("warn", "error", "allow"),
  exact.zero = TRUE,
  numerical.policy = NULL
)
```

Arguments

basis	A "metric.graph.lowpass.basis" object.
y	Numeric response vector or matrix with one row per graph vertex. Missing responses may be NA for complete-basis heat filtering.
eta.grid	Numeric filter-parameter grid.
filter.type	Spectral low-pass filter family.
block.size	Optional number of response columns processed together.

`truncation.tol` Positive tolerance smaller than one used to classify truncated-basis candidates as spectrally resolved.

`unresolved.action`
Action when a truncated-basis candidate exceeds `truncation.tol`: `"warn"`, `"error"`, or `"allow"`.

`exact.zero` Logical. For heat filtering, return the input exactly at `eta = 0`. This avoids treating a truncated spectral projection as the identity.

`numerical.policy`
NULL inherits the basis policy; otherwise `"conservative"` or `"verified.fallback"`. See `metric.graph.lowpass.basis` for the bounded fallback contract.

Value

A list of class `"metric.graph.lowpass.path"`. For one response, `fitted.values` is an N by J matrix. For multiple responses it is an N by J by S array.

Examples

```
adj <- list(2L, c(1L, 3L), c(2L, 4L), 3L)
lengths <- list(1, c(1, 1), c(1, 1), 1)
basis <- metric.graph.lowpass.basis(adj, lengths, n.eigenpairs = 4L,
                                    eigen.solver = "dense")
apply.metric.graph.lowpass.path(basis, 1:4, eta.grid = c(0, 0.1, 1))
```

---

as.data.frame.synthetic_dataset

Summarize a synthetic dataset as one metadata row

Description

Summarize a synthetic dataset as one metadata row

Usage

```
## S3 method for class 'synthetic_dataset'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x` A `synthetic_dataset`.

`row.names` Optional row names.

`optional` Ignored.

`...` Ignored.

Value

A one-row data frame.

Examples

```
x <- materialize.synthetic(synthetic.registry.spec("G1"), n = 20L, seed = 1L)
as.data.frame(x)
```

`assess.gaussian.trend.filtering` *Test Constancy and Assess Gaussian Trend-Filtering Prediction*

Description

Assess a continuous outcome using order-2 trend filtering, retuned outcome permutations and a separate nested-CV comparison with a constant mean.

Usage

```
assess.gaussian.trend.filtering(
  object,
  permutations = 999L,
  inner.folds = 5L,
  outer.folds = 5L,
  seed = 1L,
  predictive = TRUE,
  warn = TRUE,
  additional.tests = character()
)

## S3 method for class 'gaussian.trend.filtering.assessment'
print(x, ...)
```

Arguments

`object`	A `gaussian.trend.filtering.fit` with independent, unit-weight observations, no covariates and no offset. Predictor choice, penalty path, ridge penalties and numerical settings must be specified independently of outcomes. The assessment uses order 2 and reuses these explicit settings.
`permutations`	Number of random outcome permutations, at least one. The default 999 gives a minimum possible p-value of 0.001. Small values in examples illustrate usage only.
`inner.folds`	Number of balanced, outcome-independent tuning folds. Used for the global test and within each outer training set.
`outer.folds`	Number of outer folds for predictive assessment.
`seed`	Nonnegative integer random seed. The caller's RNG state is restored. Realized folds, permutation seeds and RNG kind are returned.
`predictive`	Whether to run nested CV in addition to the global test.
`warn`	Warn if the observed fit, any permutation, or an outer fit fails.

additional.tests
: Unique optional names "absolute.deviation" and/or "total.variation". Default character() tests squared-error improvement only. Both observed curve summaries are always returned.

x
: A gaussian.trend.filtering.assessment to print.

...
: Reserved for generic compatibility.

Details

The randomization null is an outcome distribution that is unchanged across the fixed predictor locations. Independent outcomes must be exchangeable under the null; a constant mean plus identically distributed independent errors is a sufficient model. Normality is unnecessary. This is not a test of constant mean under arbitrary heteroskedasticity. Repeated participants, serial dependence, covariate adjustment and unequal weights require other null procedures and are rejected where detectable. Independence and exchangeability cannot be checked from a fit.

Fresh balanced folds are generated independently of outcomes; stored fit selections and fold labels are ignored. The explicit lambda path is tuned by CV on the full data and again on every shuffled outcome vector, always using the same tuning folds. The primary statistic is `max(0, sum((y-mean(y))^2) - sum((y-f.hat(x))^2))`, the nonnegative reduction in residual sum of squares, in squared response units. It is not a penalized-objective difference or a chi-squared statistic. Constant outcomes have zero statistics and permutation p-values of one.

With B requested permutations the inclusive plus-one p-value is `(1 + sum(permuted.statistic >= observed.statistic))/(B + 1)`, allowing a conservative numerical tie tolerance of `1e-10 * max(1, abs(observed.statistic))`. Retuning is part of each randomization. Failed candidates are ineligible, and a replicate fails if none is eligible. Failed replicates are retained without retries or denominator changes. Any failure suppresses the point p-value; lower and upper bounds count missing statistics as nonexceedances or exceedances. Observed-fit failure leaves both bounds unavailable. These are failure bounds, not confidence intervals.

Absolute deviation is `mean(abs(f.hat(x)-mean(y)))`, averaging over observations with repeated predictors counted at their observed frequency. Total variation sums absolute changes along the continuous fitted curve over `[min(x),max(x)]`, including every knot and interior quadratic turning point. Both are in response units. Total variation is not the third- difference penalty. These describe the newly tuned observed fit; they are not bias-corrected population effects or uncertainty intervals. Optional tests reuse the same fitted permutations, with per-statistic failure bounds and Holm adjustment across squared error and the requested tests. An incomplete family has no Holm p-values. This adjustment does not cover additional predictors or outcomes. Choose tests and penalties in advance.

Nested CV inherits the input fit's `cv.penalty.scale`. Under `"training_mass"`, outer training fits scale the reference lambda grid and ridge penalties by their sample-size fraction; inner folds apply their own training fraction. Full-data observed and permutation refits retain the reference grid. Older saved objects without this field use `"none"` to preserve their earlier convention. Nested CV tunes using only each outer training set. Its constant baseline is that set's sample mean. Pooled mean squared error (`mse`) and mean absolute error (`mae`) use the same held-out rows; improvements are constant minus smoother loss. Positive values favor trend filtering. The baseline remains the training mean for both scores. Boundary predictions use the fitter's constant-boundary policy and are flagged. Incomplete predictions leave overall smoother scores unavailable. No standard errors

are computed from overlapping CV folds. Prediction scores assess training at the outer-training sample size, and need not improve when the global association test rejects.

Runs are sequential, with the same fitting cost and failure policy as `assess.logistic.trend.filtering`. The test does not establish nonlinearity, causality or a useful predictive effect.

Value

A `gaussian.trend.filtering.assessment` with the same evidence structure as `assess.logistic.trend.filtering`, replacing `null.probability` with `null.mean` and the primary `likelihood` test with `squared.error`. Scalar `statistic`, `p.value`, `p.value.bounds`, `status` and the `permutations` table refer to squared-error improvement. `fit` is the newly tuned observed fit. `curve.summaries`, `tests`, and `permutation.statistics` report curve summaries and requested tests. Fold assignments, seeds, candidate diagnostics and failure messages are retained. Optional `prediction` includes row-level `mean` and `constant.mean` predictions, losses, fold diagnostics, model scores and baseline-minus-smoother improvements. No confidence intervals are supplied.

References

Phipson and Smyth (2010). Permutation p-values should never be zero. doi:10.2202/15446115.1585

See Also

`fit.gaussian.trend.filtering`, `assess.logistic.trend.filtering`

Examples

```
x <- seq(0, 1, length.out = 40)
y <- sin(2*pi*x) + 0.2*cos(seq_along(x))
fit <- fit.gaussian.trend.filtering(x, y, c(Inf, 0.01), backend = "banded")
# Nine permutations illustrate the API; use more for inference.
assessment <- assess.gaussian.trend.filtering(fit, permutations = 9,
    inner.folds = 3, outer.folds = 3, seed = 42,
    additional.tests = c("absolute.deviation", "total.variation"))
assessment$tests
assessment$prediction$improvement
```

`assess.logistic.trend.filtering`

Test Constancy and Assess Logistic Trend-Filtering Prediction

---

Description

Assess whether an order-2 logistic trend filter detects a nonconstant binary probability, and separately estimate predictive improvement over a constant. This is an additional assessment, not an option that changes the fitted objective or supplies confidence bands.

Usage

```
assess.logistic.trend.filtering(
  object,
  permutations = 999L,
  inner.folds = 5L,
  outer.folds = 5L,
  seed = 1L,
  predictive = TRUE,
  warn = TRUE,
  additional.tests = character()
)

## S3 method for class 'logistic.trend.filtering.assessment'
print(x, ...)
```

Arguments

`object`	A `logistic.trend.filtering.fit` with independent binary observations, unit weights, no covariates and no offset. Predictor choice, penalty path, ridge penalties and solver settings must have been specified independently of the outcomes. Backend and explicit penalties are reused.
`permutations`	Number of random outcome permutations, at least one. The default 999 gives a minimum possible p-value of 0.001. Small values in examples illustrate usage only.
`inner.folds`	Number of balanced, outcome-independent tuning folds. Used for the global test and within each outer training set.
`outer.folds`	Number of outer folds for predictive assessment.
`seed`	Nonnegative integer random seed. The caller's RNG state is restored. Realized folds, permutation seeds and RNG kind are returned.
`predictive`	Whether to run nested CV in addition to the global test.
`warn`	Warn if the observed fit, any permutation, or an outer fit fails.
`additional.tests`	Character vector selecting optional permutation tests: `"absolute.deviation"`, `"total.variation"`, or both. Default `character()` tests likelihood improvement only. Both curve summaries are always returned for the observed fit. Specify the tests before inspecting outcomes.
`x`	A `logistic.trend.filtering.assessment` to print.
`...`	Reserved for generic compatibility.

Details

The null is a common unknown Bernoulli probability at all observed predictor locations. The constant estimate is the sample event rate, not a uniformly weighted integral over the continuous predictor domain. Independent observations must be exchangeable under this null. Repeated participants, serial dependence, confounding adjustment and grouped binomial counts require other null procedures and are not supported here. Independence cannot be established from a fitted object.

The input object's selected penalty and fold labels are not reused: those labels may depend on outcomes. A new outcome-independent fold assignment selects a penalty from the supplied path, including a full-data refit. Every permutation repeats this complete selection procedure on the same folds with the same solver settings. The statistic is the nonnegative part of twice the ordinary binomial log-likelihood improvement over the constant. It is not a penalized-objective difference and is not compared with a chi-squared distribution. Lambda order determines selection ties. Changing the path after inspecting the assessment invalidates the stated test unless that additional selection is included in the randomization.

Absolute deviation is `mean(abs(p.hat(x) - mean(y)))`, evaluated at all observed predictor locations, including repeats. It averages over sampled observations, not uniformly over the predictor domain. A value of 0.04 means an average absolute departure of four percentage points. Total variation is the sum of absolute probability changes along the fitted curve over `[min(x), max(x)]`. It includes every knot and interior turning point of the piecewise quadratic logit, up to floating-point precision; it is not a coarse-grid approximation or the trend-filtering penalty. For a monotone curve it equals the endpoint probability difference in absolute value; an oscillating curve can have total variation above one. These are descriptive summaries of the newly tuned fit, not uncertainty intervals or bias-corrected population effect estimates.

Optional tests use the same selected fits, folds and permutations as the likelihood test, with large values indicating departure from constancy. No additional model fits or random draws are needed. Each statistic has its own inclusive comparison tolerance and failure bounds. `tests` reports individual p-values and Holm-adjusted p-values across likelihood and the requested additional tests. Holm adjustment remains valid under dependence; it does not adjust across other predictors or outcomes. If any requested test is incomplete, all Holm values are unavailable. Choosing the smallest unadjusted p-value after inspecting several tests does not preserve the stated test level. The legacy scalar fields always describe the unadjusted likelihood test, regardless of `additional.tests`.

Outcomes are shuffled uniformly with the predictor fixed, preserving the total event count. With B successful permutations the Monte Carlo p-value is `(1 + sum(permuted.statistic >= observed.statistic))/(B + 1)`. Under exchangeability this is a valid conditional randomization test, subject to Monte Carlo discreteness. For numerical ties, comparisons allow a conservative tolerance of `1e-10 * max(1, abs(observed.statistic))`. A failed candidate is ineligible under the fitter's usual CV rule. If no candidate is eligible, that replicate fails. Failed replicates are never dropped or retried: the point p-value is NA and `p.value.bounds` gives the smallest/largest possible Monte Carlo p-value counting each missing statistic as a nonexceedance/exceedance. These bounds are not a confidence interval. An observed-fit failure leaves both bounds unavailable.

Predictive assessment uses separate outer folds. Each outer training set inherits the input fit's `cv.penalty.scale`. Under `"training_mass"`, its lambda grid and ridge penalties are multiplied by the outer/full sample-size ratio, and its inner folds apply their own training ratio. Observed and permuted full-data refits use the same unscaled reference grid. Older saved fits without this field retain `"none"` for reproduction. Each outer training set tunes the path using only its inner folds and predicts its outer test set. The constant predictor is that outer training set's event rate. Both use identical test observations. Pooled mean natural-log loss and Brier score are reported; improvements are constant loss minus trend-filtering loss, so positive values favor the smoother. These assess training on the outer training-set size, not the full sample. No standard error is computed from overlapping CV folds. Incomplete smoother predictions leave its overall scores and improvements unavailable rather than scoring a selected subset.

Computation is sequential. A complete run uses approximately `(permutations + 1) * (inner.folds + 1)` native path fits for the test, plus `outer.folds * (inner.folds + 1)` for prediction. Per-

replicate diagnostics are retained, but full permuted fitted objects are not. Rejecting constancy is evidence of association, not necessarily nonlinearity, predictive usefulness or a causal effect.

Value

A `logistic.trend.filtering.assessment` list containing `statistic`, `p.value`, `p.value.bounds`, `status`, `null.probability`, the newly tuned observed `fit`, permutation statistics and diagnostic records, optional nested-CV `prediction`, realized `folds`, seeds and solver settings. `curve.summaries` is a named vector with `absolute.deviation` and `total.variation`; `curve.summary.details` records their conventions and any calculation failure. `tests` has one row per requested test, including likelihood, with statistics, status, completed/failed permutation counts, exceedances, comparison tolerances, raw p-values, lower/upper failure bounds and `p.value.holm`. `permutation.statistics` contains a `replicate` column and one column per tested statistic. The original `permutations` table retains its likelihood-only schema. `prediction$scores` contains model losses; `prediction$improvement` contains baseline-minus-smoother gains; `prediction$predictions` and `prediction$diagnostics` retain row-level and fold-level evidence.

References

Ojala and Garriga (2010). Permutation Tests for Studying Classifier Performance. https://www.jmlr.org/papers/v11/ojala10a.html

Phipson and Smyth (2010). Permutation p-values should never be zero. doi:10.2202/15446115.1585

See Also

`fit.logistic.trend.filtering`

Examples

```
x <- seq(0, 1, length.out = 60)
y <- rep(c(0, 1, 0, 1, 1), 12)
fit <- fit.logistic.trend.filtering(x, y, lambda = c(Inf, 0.01),
    backend = "banded", curve.ridge = 0.1)
# Only nine permutations for a quick illustration; use more for inference.
assessment <- assess.logistic.trend.filtering(fit, permutations = 9,
    inner.folds = 3, outer.folds = 3, seed = 42)
print(assessment)
assessment$prediction$improvement
assessment$curve.summaries
# Optional tests reuse the fits within an assessment run:
shape <- assess.logistic.trend.filtering(fit, permutations = 9,
    inner.folds = 3, seed = 42, predictive = FALSE,
    additional.tests = c("absolute.deviation", "total.variation"))
shape$tests
```

`bootstrap.malps` *Bootstrap Uncertainty For A Fixed-Selection MALPS Fit*

Description

Reuses the supports, local charts, prediction supports, averaging weights, selected degree, and local-solver controls from an existing `fit.malps` object, then repeatedly calls `refit` with bootstrap-style case weights. This is a fixed-selection uncertainty diagnostic: it measures refit variability conditional on the fitted MALPS support profile and does not rerun cross-validation or rebuild supports in each replicate.

Usage

```
bootstrap.malps(
  object,
  B = 200L,
  weight.type = c("bayesian", "multinomial"),
  y = NULL,
  probs = NULL,
  conf.level = 0.95,
  seed = NULL,
  max.failures = max(10L, B),
  keep.weights = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

`object`	A `"malps"` object from `fit.malps` or `refit`.
`B`	Number of successful bootstrap replicates requested.
`weight.type`	Bootstrap weight generator, one of `"bayesian"` or `"multinomial"`.
`y`	Optional response vector. Defaults to `object$y`.
`probs`	Optional nonnegative sampling probabilities for `weight.type = "multinomial"`.
`conf.level`	Pointwise interval level used in the returned summary.
`seed`	Optional integer seed for reproducible bootstrap weights.
`max.failures`	Maximum failed refit attempts allowed before stopping.
`keep.weights`	Logical; store the generated bootstrap weights in the returned object.
`verbose`	Logical; emit progress messages every ten successful replicates.
`...`	Reserved for future extensions.

Details

Two weight generators are available. `"bayesian"` draws positive exponential weights and rescales them to have mean one. This is the default because it preserves positive local support weights and therefore avoids many local-support degeneracies. `"multinomial"` draws ordinary nonparametric bootstrap counts with total count `nrow(object$X)`; this can fail when a replicate assigns zero weight to all observations in a local support, and such failures are recorded.

Value

A list of class `"malps_bootstrap"` containing:

- `replicate.fitted.values`: an $n \times B$ matrix of successful replicate fitted values;
- `replicate.weights`: an optional $n \times B$ matrix of bootstrap weights when `keep.weights = TRUE`;
- `failures`: a data frame with columns `attempt`, `failure`, and `message`;
- `summary`: pointwise fitted values, bootstrap summaries, and empirical lower/upper intervals;
- bootstrap metadata such as `B.completed`, `n.failures`, `conf.level`, and `weight.type`.

Examples

```
X <- matrix(seq(0, 1, length.out = 20), ncol = 1)
fit <- fit.malps(X, X[, 1]^2, degree = 1L,
                 support.type = "knn", support.size = 8L)
bootstrap.malps(fit, B = 5L, seed = 1L)
```

---

compare.masked.graph.fields

Compare Masked Graph Fields with Independent Graph Workers

Description

Runs one candidate graph per process, with dynamically scheduled jobs and no nested fold parallelism. On Unix, forked workers inherit the loaded package sources. Run from Rscript, not a GUI or multithreaded parent process.

Usage

```
compare.masked.graph.fields(operators, studies, workers = 1L, ...)
```

Arguments

`operators`	Named list of metric graph low-pass operators.
`studies`	Named list of argument lists containing vertex, y and group.
`workers`	Number of graph workers; choose using available memory and CPU. Defaults to serial execution. Windows currently requires one worker.
`...`	Additional arguments passed to cv.masked.graph.field.

Value

Named graph results, each containing named study results or an explicit error record. Retains per-observation predictions for common-support scoring.

Examples

```
op <- metric.graph.lowpass.operator(list(2L, c(1L, 3L), 2L),
                                    list(1, c(1, 1), 1))
studies <- list(example = list(vertex = rep(c(1, 3), 6), y = 1:12,
                               group = letters[1:12]))
comparison <- compare.masked.graph.fields(list(path = op), studies,
    workers = 1, lambdas = c(0.01, 1), outer.folds = 3, inner.folds = 2)
comparison$path$example$coverage
```

---

compare.synthetic.dataset

Compare two synthetic datasets scientifically

Description

Compare two synthetic datasets scientifically

Usage

```
compare.synthetic.dataset(x, y, tolerance = c(1e-12, 1e-10))
```

Arguments

x, y	Synthetic datasets.
tolerance	Absolute-plus-relative comparison tolerance.

Value

A list with equality and mismatch details.

Examples

```
spec <- synthetic.registry.spec("G1")
x <- materialize.synthetic(spec, n = 20L, seed = 1L)
y <- materialize.synthetic(spec, n = 20L, seed = 1L)
compare.synthetic.dataset(x, y)
```

`cv.masked.graph.field` *Participant-Grouped Nested Cross-Validation for a Masked Graph Field*

Description

Select smoothing within outer training folds, predict held-out groups, and finally tune and refit on all observations. Graph construction is label-free and fixed (transductive prediction). Unsupported predictions remain NA.

Usage

```
cv.masked.graph.field(
  operator,
  vertex,
  y,
  group,
  lambdas = 10^seq(-6, 2, length.out = 17),
  outer.folds = 5L,
  inner.folds = 4L,
  seed = 1L,
  fold.id = NULL,
  progress = NULL
)
```

Arguments

`operator`	An unnormalized `metric.graph.lowpass.operator`.
`vertex`	Integer vertex index for each observed response.
`y`	Finite numeric observed responses (no missing-value placeholders).
`group`	Participant or leakage-block identifier per observation.
`lambdas`	Positive finite candidate penalties.
`outer.folds`, `inner.folds`	Numbers of grouped folds.
`seed`	Fold seed, independent of worker scheduling.
`fold.id`	Optional preassigned outer fold per observation; a group must occupy exactly one fold.
`progress`	Optional function called after each outer fold with fold and selected penalty; intended for experiment progress records.

Value

Outer predictions, baseline predictions, scores on supported cases, coverage, tuning records, fold assignments and final full-data field. Scores do not account for selecting a graph after comparing graphs.

Examples

```
op <- metric.graph.lowpass.operator(list(2L, c(1L, 3L), 2L),
                                    list(1, c(1, 1), 1))
cv <- cv.masked.graph.field(op, rep(c(1, 3), 6), 1:12, letters[1:12],
    lambdas = c(0.01, 1), outer.folds = 3, inner.folds = 2)
cv$coverage
cv$final$fitted
```

`cv.projection.trend.filtering` *Cross-Validate Projection Trend Filtering*

Description

Select explicit mean-loss penalties and component counts using supplied folds. Every candidate in every training split repeats preprocessing, direction selection, refinement and joint fitting.

Usage

```
cv.projection.trend.filtering(
  X,
  y,
  foldid,
  lambda.grid,
  nterms.grid = c(0L, 1L, 2L),
  curve.ridge.grid = 0.001,
  family = c("gaussian", "binomial"),
  directions = NULL,
  direction.search = c("learned", "fixed", "gn", "pls"),
  weights = NULL,
  offset = NULL,
  groups = NULL,
  preprocessing = c("scale", "isotropic", "none"),
  seed = 1L,
  control = list(),
  backend = c("auto", "dense", "banded"),
  keep.fits = FALSE,
  warn = TRUE
)
```

Arguments

`X`	Finite numeric predictor matrix or data frame.
`y`	Finite numeric response. For `family="binomial"`, values must lie in `[0,1]`; proportions can be accompanied by trial weights.
`foldid`	Integer vector, or matrix with one column per CV repeat. Each column must contain at least two folds; labels need not be consecutive.

`lambda.grid` Nonnegative finite penalties or `Inf`.

`nterms.grid` Nonnegative integer component counts. In fixed mode the nonzero count must match the supplied direction matrix.

`curve.ridge.grid`
Nonnegative finite component ridge coefficients.

`family` `"gaussian"` or `"binomial"`.

`directions` Matrix with one direction per row and one column per original predictor. Directions refer to the preprocessed coordinates. Entries for dropped constant columns must be zero. In learned mode these are additional candidates; in fixed mode these are the fitted directions.

`direction.search`
`"fixed"` fits supplied directions (coordinate directions when omitted); `"learned"` selects and refines directions with pattern search. `"gn"` uses multistart damped Gauss–Newton proposals; `"pls"` uses one covariance start per component with the same Gauss–Newton refinement. The latter two modes construct their own starts and do not accept supplied directions.

`weights` Positive finite observation weights, default one.

`offset` Known additive predictor, default zero.

`groups` Optional participant/group labels. All rows of a group must belong to the same fold in each repeat. Groups validate supplied splits; they do not automatically change observation weights.

`preprocessing` `"scale"` centers and scales each nonconstant column; `"isotropic"` centers and uses a common scale, preserving rotations and distance ratios; `"none"` leaves retained columns unchanged. All transformations use training weights only. The default is `"isotropic"` for `"gn"` and `"pls"`, and `"scale"` otherwise.

`seed` Integer random seed. Local randomness leaves the caller's RNG state unchanged.

`control` Named list. `max.sweeps=200`, `tolerance=1e-7`, `gap.tolerance=1e-8`, and `objective.tolerance=1e-10` govern joint fitting. `inner=list()` passes controls to the univariate solver. `joint.solver="backfit"` retains alternating component updates; `joint.solver="dense"` attempts a simultaneous dense solve when there are exactly two components, with guarded fallback to backfitting. `joint.max.workspace.bytes=1024^3` bounds the estimated dense workspace. The joint certificate and objective safeguard apply to both solvers. Search controls are `starts=2`, `sparse.directions=8`, `dense.directions=8`, `refinement.passes=2`, `refinement.proposals=16`, `steps=c(0.25,0.1,0.04,0.01)`, and `improvement.tolerance=1e-9`. Zero refinement passes select from a fixed candidate bank. The `"gn"` and `"pls"` modes instead use `gn.max.steps=8` accepted proposals and `gn.max.fits=40` native fits per start, with `gn.shared.fits=400` shared fits for previously selected components. `stream.key=character()` can identify a dataset or representation; local start streams also depend on the seed and component number.

`backend` Univariate solver: `"auto"`, `"dense"`, or `"banded"`.

`keep.fits` Retain training-split fits for inspection, default `FALSE`.

`warn` Warn when joint fitting or a terminal search failure makes the returned fit ineligible. Failed candidates are retained in `search.history`.

Details

Scores pool weighted held-out losses within each repeat and then average repeats. Gaussian selection uses MSE; binomial selection uses stable log loss from logits. Every required fold must converge and predict every held-out row. Fixed and pattern-search modes select the first grid row on equal scores. Gauss–Newton modes also require explicit search eligibility and resolve equal CV scores by fewer requested components, stronger penalty, then grid order. Their random streams are keyed by repeat and fold. Penalties already use mean loss, so no additional fold-mass multiplier is applied by this wrapper. A final full-data refit is performed only for the selected candidate. Its failure is reported without substituting another candidate. CV scores used for selection are not an independent assessment of the selected procedure; use outer folds or a test set.

Value

A `projection.trend.filtering.cv` containing `cv.table`, the candidate-by-split `fold.results`, held-out `predictions` (logits for binomial data), `foldid`, optional `fold.fits`, `selected.index`, `fit`, and `status`. `selected.index` is NULL if validation or the full refit failed; `cv.selected.index` retains the validation choice when applicable. For Gauss–Newton modes, `fold.results` separately records `numerical.certified`, `search.eligible`, `cv.eligible`, `search.status`, requested/effective component counts and native-fit counts, including when `keep.fits=FALSE`.

Examples

```
X <- matrix(seq(-1, 1, length.out = 24), ncol = 1)
cv <- cv.projection.trend.filtering(X, sin(2*X[,1]),
    foldid = rep(1:3, 8), nterms.grid = 1,
    lambda.grid = c(Inf, 0.001), direction.search = "fixed")
cv$cv.table
```

`fit.chart.kernel` *Fit a Chart-Kernel Smoother*

Description

Computes a kernel-weighted average of the response at each evaluation point, in the manner of the Nadaraya–Watson estimator, with distances measured in a local coordinate chart. The result is a fitted field of general use. To obtain a density from it, apply `normalize.density` to the returned fit.

Usage

```
fit.chart.kernel(
  X,
  y,
  X.eval = NULL,
  support.size = min(15L, nrow(X)),
  kernel = c("gaussian", "tricube", "epanechnikov", "triangular"),
  bandwidth.multiplier = 1,
```

```
  support.grid = NULL,
  kernel.grid = NULL,
  bandwidth.multiplier.grid = NULL,
  foldid = NULL,
  cv.folds = 5L,
  cv.seed = 1L,
  coordinate.method = c("coordinates", "local.pca"),
  chart.dim = NULL,
  chart.dim.grid = NULL,
  selection.strategy = c("grid", "sparse_kd", "plateau_kd"),
  chart.dim.max = NULL,
  geometry.margin = 0L,
  auto.chart.support.metric = c("coordinates", "operator", "both"),
  auto.chart.selection.metric = c("coordinates", "operator"),
  quadrature.weights = NULL,
  denominator.floor = sqrt(.Machine$double.eps),
  return.details = TRUE
)
```

Arguments

`X`	Numeric matrix with one row per source/support point.
`y`	Numeric response or mass vector of length `nrow(X)`.
`X.eval`	Optional numeric matrix of evaluation points. Defaults to `X`.
`support.size`	Number of source points in each local support.
`kernel`	Kernel name. Supported values are `"gaussian"`, `"tricube"`, `"epanechnikov"`, and `"triangular"`.
`bandwidth.multiplier`	Positive multiplier applied to the local support radius.
`support.grid`	Optional integer candidate neighborhood sizes. If supplied, or if `foldid` is supplied, the function performs row-wise cross-validation and refits the selected candidate on all rows.
`kernel.grid`	Optional kernel candidates for cross-validation.
`bandwidth.multiplier.grid`	Optional bandwidth-multiplier candidates for cross-validation.
`foldid`	Optional positive integer vector assigning source rows to cross-validation folds.
`cv.folds`	Number of folds used when `foldid` is not supplied.
`cv.seed`	Random seed used to generate folds when `foldid` is not supplied.
`coordinate.method`	Local coordinate method. `"coordinates"` uses centered ambient coordinates. `"local.pca"` projects centered support points onto a local PCA basis.
`chart.dim`	Local PCA dimension when `coordinate.method = "local.pca"`. If `NULL`, the dimension is `min(ncol(X), support.size - 1)`. The deployable input-only policies `"auto"` and `"local.auto"` use the same local-PCA dimension diagnostics as `fit.lps`; `"auto"` resolves one global chart dimension, while `"local.auto"` resolves one dimension per evaluation anchor.

`chart.dim.grid` Optional candidate chart dimensions for experimental row-wise cross-validation. Numeric grids can be evaluated with `selection.strategy = "sparse_kd"`.

`selection.strategy`
Candidate-selection strategy. `"grid"` evaluates the requested candidate grid. `"sparse_kd"` evaluates a sparse support-size by chart-dimension skeleton when `chart.dim.grid` is supplied. `"plateau_kd"` uses the same geometry-only support-size and chart-dimension plateau rule as `fit.lps`.

`chart.dim.max` Optional explicit maximum chart dimension for the sparse coupled candidate family.

`geometry.margin`
Nonnegative integer subtracted from the local-PCA rank cap `min(ncol(X), support.size - 1)`. The default zero applies only the Nadaraya–Watson chart-geometry requirement; this is not a local polynomial design margin.

`auto.chart.support.metric`
Support system used by `chart.dim = "auto"` or `"local.auto"`. Chart-kernel smoothers currently use coordinate supports for both coordinate and operator diagnostics.

`auto.chart.selection.metric`
Which auto chart-dimension diagnostic to use when both coordinate and operator summaries are requested.

`quadrature.weights`
Optional positive reference-measure weights `q_i`. Defaults to unit weights.

`denominator.floor`
Positive floor used when the local denominator is numerically zero.

`return.details` Logical; if `TRUE`, keep per-evaluation diagnostics.

Details

For an evaluation point x_u, the method chooses a local support U_u, builds either centered ambient coordinates or a local PCA chart, and computes

$$\widehat{f}(x_u) = \frac{\sum_{i \in U_u} y_i K_h(z_{ui})}{\sum_{i \in U_u} q_i K_h(z_{ui})}.$$

Here z_{ui} is the local chart coordinate of $x_i - x_u$, q_i is an optional quadrature weight, and K_h is the selected kernel.

Value

A list with class `"chart_kernel"` containing `fitted.values`, source/evaluation supports, selected controls, and denominator diagnostics.

Examples

```
X <- matrix(seq(0, 1, length.out = 21), ncol = 1)
y <- exp(-20 * (X[, 1] - 0.5)^2)
fit.chart.kernel(X, y, support.size = 7L, kernel = "gaussian")
```

`fit.density` *Fit a Density Estimator*

Description

Estimates probability mass over a fixed set of support points from nonnegative counts or masses given at those points, using the estimator selected by `method`. One application is the occupation density of a subject, for which the input is the number of visits to each support point; see `fit.subject.od`.

Usage

```
fit.density(
  X,
  weights = NULL,
  method = c("empirical", "graph_random_walk"),
  graph = NULL,
  graph.control = list(),
  density.control = list(),
  return.details = TRUE,
  ...
)
```

Arguments

`X`	Numeric matrix with one row per support point.
`weights`	Optional nonnegative mass/count vector of length `nrow(X)`. Required by count-based density methods.
`method`	Density method identifier.
`graph`	Optional precomputed graph object.
`graph.control`	Named list for random walks: `walk.steps` (or `walk.step`), `affinity.method`, `affinity.scale`, `affinity.epsilon`, and logical `normalize`. Underscore aliases are accepted, but conflicting aliases and unknown names are rejected. Must be empty for the empirical estimator.
`density.control`	List controlling clipping, normalization, and accounting checks. Recognized entries are `mass.tol`, `neg.tol`, logical `clip.negative` and `renormalize`, and diagnostic controls `smoothness.adj.list` and logical `smoothness.auto.1d`. Unknown or duplicated names are errors.
`return.details`	Logical; if `TRUE`, keep diagnostic details in the result.
`...`	Reserved for future methods; unused arguments are errors.

Value

A list of class `"density_fit"` with fields `method.id`, `status`, `rho`, `empirical.rho`, `fitted.raw`, `theta`, `accounting`, `smoothness`, `timing`, `diagnostics`, and `warnings`.

Examples

```
X <- matrix(seq(0, 1, length.out = 6), ncol = 1)
fit.density(X, weights = c(1, 0, 2, 0, 0, 1), method = "empirical")
```

`fit.gaussian.trend.filtering` *Fit Order-2 Gaussian Trend Filtering*

Description

Fit a continuous response with a piecewise-quadratic curve and an L1 penalty on third divided differences. Dense and banded native C++ solvers share the order-2 geometry of `fit.logistic.trend.filtering`.

Usage

```
fit.gaussian.trend.filtering(
  x,
  y,
  lambda,
  weights = NULL,
  covariates = NULL,
  offset = NULL,
  nuisance.ridge = 0,
  curve.ridge = 0,
  lambda.selection = c("fixed", "cv"),
  foldid = NULL,
  control = list(),
  warn = TRUE,
  backend = c("auto", "dense", "banded"),
  cv.penalty.scale = c("training_mass", "none")
)
```

Arguments

`x`	Finite numeric predictor, with at least four distinct values.
`y`	Finite numeric continuous response, with one value per observation. Constant responses are supported.
`lambda`	Nonnegative penalties, or `Inf` for the quadratic limit. Required explicitly: no data-dependent default penalty is assumed.
`weights`	Positive observation weights, default one.
`covariates`	Optional finite numeric matrix of additional predictors, without an intercept. Columns are centered and scaled on training data.
`offset`	Optional finite vector added to the linear predictor.
`nuisance.ridge, curve.ridge`	Nonnegative explicit ridge penalties, default zero. See Details for their precise scaling.

`lambda.selection`
: `"fixed"` returns the supplied penalty path; `"cv"` selects minimum held-out weighted mean squared error (MSE).

`foldid`
: Integer fold labels, required for `lambda.selection="cv"`. Supply the same label for observations that must be held out together.

`control`
: Named list overriding `tolerance=1e-9`, `gap.tolerance=1e-10`, `max.iterations=1000L`, or `max.workspace.bytes=1024^3`. Tolerances apply to mean weighted loss.

`warn`
: Logical; warn when a path fit fails convergence checks.

`backend`
: Native solver: `"auto"` uses `"banded"` above 256 distinct locations or when the dense workspace estimate exceeds its limit, and `"dense"` otherwise. The banded backend uses compact working sets and checks every knot; it does not thin the predictor locations.

`cv.penalty.scale`
: `"training_mass"` (default) multiplies each fold's lambda and ridge penalties by its training/full weight-sum ratio. This preserves their coefficients relative to mean loss. `"none"` retains the earlier same-numerical-penalty convention. Fixed-path fits are unchanged.

Details

With training predictor locations normalized to `[0,1]`, minimize

$$\frac{1}{2}\sum_i w_i(y_i-\eta_i)^2+\lambda\|D^{(3)}f\|_1+\frac{r_z}{2}\|\gamma\|^2+\frac{r_f}{2\sum_i w_i}\sum_i w_i(f_i-\bar{f}_w)^2.$$

Here eta is the curve plus standardized covariates and offset. The intercept is unpenalized. Lambda is on the summed-loss scale; the default ridge penalties are zero. `lambda=Inf` restricts the curve to a global quadratic, not a constant. At `lambda=0` with no covariates, offsets or ridge, predictions at distinct training locations are their weighted response means. Exact ties retain their individual loss contributions. Order is fixed at 2; no order-selection step is performed.

Gaussian refers to squared-error fitting. Fitting and CV do not estimate a noise variance, supply confidence intervals, or assume normality for the permutation assessment. Additional covariates and offsets are supported for fitting; `assess.gaussian.trend.filtering` supports only independent, unit-weight observations without adjustment.

Every CV split recomputes its training-only predictor range, knots and transformations. The default `cv.penalty.scale="training_mass"` scales lambda and ridge penalties by the training/full weight-sum ratio; `"none"` retains the earlier raw-penalty convention. This holds the mean-loss coefficients constant, not a common raw-coordinate operator across splits. Selection requires convergence on the full data and all folds. Equal CV scores select the first supplied lambda. Predictions outside the training range hold the curve at the nearest boundary; covariates and offsets continue to contribute. The banded backend scans all knots without thinning locations. Both solvers require finite arithmetic, stationarity, dual feasibility and L1 complementarity checks; failed fits remain explicit. Tolerances apply to mean weighted loss.

Value

A `gaussian.trend.filtering.fit` containing `order=2L`, the original data, penalty path, `fits`, convergence `diagnostics`, training `design`, and, when requested, `cv.table` with MSE and eligibility, fold-level loss sums and status, and `selected.index`. A fixed path has no selected index.

`cv.penalty.scale`, `cv.fold.lambda`, `cv.fold.penalty.factors`, and `cv.fold.ridge` record the actual convention and training penalties. Standard prediction returns NA for unconverged candidates.

References

Wang, Smola and Tibshirani (2014). The Falling Factorial Basis and Its Statistical Applications. https://arxiv.org/abs/1405.0558

See Also

`assess.gaussian.trend.filtering`, `fit.logistic.trend.filtering`

Examples

```
x <- seq(0, 1, length.out = 40)
y <- sin(2*pi*x) + 0.1*cos(seq_along(x))
fit <- fit.gaussian.trend.filtering(x, y, lambda = c(Inf, 0.01),
    backend = "banded")
predict(fit, c(0.25, 0.75), lambda.index = 2)
```

`fit.graph.trend.filtering` *Fit Graph Trend Filtering*

Description

Fits graph trend filtering on a supplied undirected graph. Supported orders are `0L`, `1L`, and `2L`:

$$\widehat{\beta}_\lambda = \arg\min_{\beta \in \mathbb{R}^n} \left\{ \frac{1}{2}\|y - \beta\|_2^2 + \lambda \|\Delta_w^{(k+1)} \beta\|_1 \right\}.$$

Usage

```
fit.graph.trend.filtering(
  adj.list,
  weight.list = NULL,
  y,
  order = 0L,
  lambda.grid = NULL,
  lambda.selection = c("cv", "fixed"),
  weight.rule = c("conductance", "sqrt.conductance", "unit"),
  operator.family = c("graph.laplacian.recursive", "path.divided.difference"),
  path.family = c("branch.continuation", "all.simple"),
  path.weighting = c("unit", "anchor.mean"),
  n.lambda = 40L,
  nfolds = 5L,
  foldid = NULL,
```

```
  maxsteps = 2000L,
  minlam = 0,
  approx = FALSE,
  rtol = 1e-07,
  btol = 1e-07,
  eps = 1e-04,
  verbose = FALSE,
  cv.penalty.scale = c("training_mass", "none")
)
```

Arguments

adj.list	List of integer neighbor vectors using 1-based vertex indices. The graph must be undirected: every edge must appear in both endpoint adjacency lists.
weight.list	Optional list of positive edge weights parallel to adj.list. For operator.family = "graph.laplacian.recursive", weights are graph trend-filtering weights or conductances. For operator.family = "path.divided.difference", weights are metric edge lengths used to define path coordinates. If NULL, all values are one.
y	Numeric response vector of length length(adj.list).
order	Integer trend-filtering order. Supported values are 0L, 1L, and 2L.
lambda.grid	Optional non-negative lambda grid. For lambda.selection = "fixed", this must contain exactly one value. For lambda.selection = "cv", NULL builds a default grid from the fitted full-data solution path.
lambda.selection	"cv" for vertex K-fold cross-validation or "fixed" for a supplied fixed lambda.
weight.rule	Character scalar. "conductance" uses weight.list directly as ω_e; "sqrt.conductance" uses $\sqrt{w_e}$; "unit" ignores weight.list. This applies to operator.family = "graph.laplacian.recursive".
operator.family	Character scalar selecting the penalty operator family. The default "graph.laplacian.recursive" preserves the existing graph trend-filtering semantics. "path.divided.difference" uses local pathwise finite differences and is designed to reproduce classical one-dimensional trend filtering on path graphs.
path.family	Character scalar used for operator.family = "path.divided.difference". "branch.continuation" keeps paths whose internal vertices do not pass through branch points. "all.simple" keeps all simple paths of the required length.
path.weighting	Character scalar used for operator.family = "path.divided.difference". "unit" assigns every canonical path row weight one. "anchor.mean" divides rows by the number of retained canonical paths sharing the same canonical start vertex.
n.lambda	Number of default lambda candidates when lambda.grid = NULL and lambda.selection = "cv".
nfolds	Number of CV folds.
foldid	Optional integer fold assignments of length length(y).

`maxsteps` Maximum number of path steps passed to **genlasso**.

`minlam` Minimum lambda passed to **genlasso**.

`approx` Logical; passed to **genlasso**.

`rtol`, `btol`, `eps` Numerical controls passed to **genlasso**.

`verbose` Logical. If `TRUE`, pass verbose output to **genlasso**.

`cv.penalty.scale`
`"training_mass"` (default) multiplies fold lambda by the training/full observation-count ratio, keeping the penalty coefficient on the mean-loss scale. `"none"` retains the earlier same-numerical-lambda convention. Fixed-lambda fits are unchanged.

Details

Graph trend filtering is intended as an ℓ_1-adaptive comparator to the ℓ_2 graph smoothers now housed in **geosmooth** and the legacy rdgraph comparator. The low-pass smoother penalizes broad quadratic graph roughness through a term like

$$\eta f^\top L f,$$

which shrinks high-frequency variation everywhere. Graph trend filtering instead penalizes absolute graph differences. For `order = 0L`, this is

$$\lambda \sum_e \omega_e |\beta_j - \beta_i|,$$

so many edge differences can become exactly zero while selected edges carry jumps. In this sense `order = 0L` is locally adaptive: fitted values can be piecewise constant over graph regions. Higher orders use

$$\Delta_w^{(1)} = D_w, \qquad \Delta_w^{(2)} = L_w, \qquad \Delta_w^{(3)} = D_w L_w,$$

with $L_w = D_w^\top D_w$.

The supplied `weight.list`, when used, is interpreted as a graph trend-filtering edge weight or conductance for `operator.family = "graph.laplacian.recursive"`. For `operator.family = "path.divided.difference"`, `weight.list` is interpreted as metric edge length and is used to form local path coordinates. Use `weight.rule = "sqrt.conductance"` when the recursive graph square-root convention is desired, and `weight.rule = "unit"` for an unweighted recursive graph fused-lasso penalty.

The solver backend is currently **genlasso**. Cross-validation refits the generalized-lasso problem on vertex-held-out training sets using a selection matrix X; this is useful for small and moderate diagnostic problems but is not yet a scalable production graph-trend-filtering solver.

Value

A list of class `"graph.trend.filtering.fit"` containing fitted values, residuals, selected lambda, the operator, path metadata, and CV diagnostics when requested.

References

Wang, Y.-X., Sharpnack, J., Smola, A., and Tibshirani, R. J. (2016). Trend filtering on graphs. *Journal of Machine Learning Research*, 17(105), 1–41. https://www.jmlr.org/papers/v17/15-147.html

Examples

```
if (requireNamespace("genlasso", quietly = TRUE)) {
  adj <- list(2L, c(1L, 3L), c(2L, 4L), 3L)
  fit.graph.trend.filtering(
    adj, y = c(0, 0.2, 1.8, 2), lambda.grid = 0.3,
    lambda.selection = "fixed", weight.rule = "unit"
  )
}
```

`fit.local.likelihood` *Fit a Local-Likelihood Density or Bernoulli Smoother*

Description

Fits a local likelihood model around each evaluation point and returns the fitted value at that point. When the aim is the occupation density of a subject, convert the returned values with `normalize.density`.

Usage

```
fit.local.likelihood(
  X,
  y,
  X.eval = NULL,
  likelihood.family = c("density", "bernoulli"),
  support.size = min(15L, nrow(X)),
  degree = 1L,
  kernel = c("gaussian", "tricube", "epanechnikov", "triangular"),
  bandwidth.multiplier = 1,
  support.grid = NULL,
  degree.grid = NULL,
  kernel.grid = NULL,
  bandwidth.multiplier.grid = NULL,
  lambda.ridge.grid = NULL,
  foldid = NULL,
  cv.folds = 5L,
  cv.seed = 1L,
  coordinate.method = c("coordinates", "local.pca", "local.pls"),
  chart.dim = NULL,
  chart.dim.grid = NULL,
  selection.strategy = c("grid", "sparse_kd", "plateau_kd"),
  chart.dim.max = NULL,
  design.margin = 2L,
  auto.chart.support.metric = c("coordinates", "operator", "both"),
  auto.chart.selection.metric = c("coordinates", "operator"),
  quadrature.weights = NULL,
  lambda.ridge = 1e-08,
```

```
  min.local.mass = sqrt(.Machine$double.eps),
  min.nonzero.mass = 1L,
  fallback = c("degree0", "zero", "chart_kernel", "na"),
  optimizer = c("newton", "optim"),
  max.iter = 50L,
  tol = 1e-08,
  return.details = TRUE,
  chart.geometry = c("legacy", "euclidean", "linf.simplex"),
  unfold.faces = FALSE,
  intercept.mass = 0.5
)
```

Arguments

`X`	Numeric matrix with one row per source/support point.
`y`	Numeric response vector. For `likelihood.family = "density"`, this must be a nonnegative mass/intensity vector with positive total mass. For `"bernoulli"`, values must lie in `[0, 1]`.
`X.eval`	Optional numeric matrix of evaluation points. Defaults to `X`.
`likelihood.family`	Local likelihood family.
`support.size`	Number of source points in each local support.
`degree`	Local chart feature degree. Supported values are 0, 1, and 2. The density branch omits an intercept because the intercept is not identifiable in the normalized local density likelihood. The Bernoulli branch includes an intercept in its internal logistic feature map.
`kernel`	Kernel name. Supported values are `"gaussian"`, `"tricube"`, `"epanechnikov"`, and `"triangular"`.
`bandwidth.multiplier`	Positive multiplier applied to the local support radius.
`support.grid`	Optional integer candidate neighborhood sizes for cross-validation. CV selection is currently implemented for `likelihood.family = "bernoulli"`.
`degree.grid`	Optional local polynomial degree candidates for Bernoulli cross-validation.
`kernel.grid`	Optional kernel candidates for Bernoulli cross-validation.
`bandwidth.multiplier.grid`	Optional bandwidth-multiplier candidates for Bernoulli cross-validation.
`lambda.ridge.grid`	Optional ridge-penalty candidates for Bernoulli cross-validation.
`foldid`	Optional positive integer vector assigning source rows to cross-validation folds.
`cv.folds`	Number of folds used when `foldid` is not supplied.
`cv.seed`	Random seed used to generate folds when `foldid` is not supplied.
`coordinate.method`	Local coordinate method. `"coordinates"` uses centered ambient coordinates. `"local.pca"` projects centered support points onto a local PCA basis.

`chart.dim` Local PCA dimension when `coordinate.method = "local.pca"`. If `NULL`, the dimension is `min(ncol(X), support.size - 1)`. The deployable input-only policies `"auto"` and `"local.auto"` use the same local-PCA dimension diagnostics as `fit.lps`; `"auto"` resolves one global chart dimension, while `"local.auto"` resolves one dimension per evaluation anchor.

`chart.dim.grid` Optional candidate chart dimensions for experimental cross-validation. Numeric grids can be evaluated with `selection.strategy = "sparse_kd"`.

`selection.strategy`
Candidate-selection strategy. `"grid"` evaluates the requested candidate grid. `"sparse_kd"` evaluates a sparse support-size by chart-dimension skeleton when `chart.dim.grid` is supplied. `"plateau_kd"` uses the same geometry-only support-size and chart-dimension plateau rule as `fit.lps`.

`chart.dim.max` Optional explicit maximum chart dimension for the sparse coupled candidate family.

`design.margin` Nonnegative integer feasibility margin used to screen local polynomial design size before candidate evaluation.

`auto.chart.support.metric`
Support system used by `chart.dim = "auto"` or `"local.auto"`. Local-likelihood smoothers currently use coordinate supports for both coordinate and operator diagnostics.

`auto.chart.selection.metric`
Which auto chart-dimension diagnostic to use when both coordinate and operator summaries are requested.

`quadrature.weights`
Optional positive reference-measure weights. Defaults to unit weights.

`lambda.ridge` Nonnegative ridge penalty on identifiable coefficients.

`min.local.mass` Minimum local kernel-weighted mass needed before attempting a higher-degree density fit. For Bernoulli fits this is used only as diagnostic telemetry.

`min.nonzero.mass`
Minimum number of locally weighted positive-mass source points needed before attempting a higher-degree local fit.

`fallback` Fallback policy for underidentified or failed higher-degree local fits. Zero local mass always uses `"zero"`.

`optimizer` Optimizer for nonzero-degree density fits. `"newton"` is implemented; `"optim"` is accepted and delegates to BFGS.

`max.iter` Maximum optimizer iterations.

`tol` Convergence tolerance for gradient norm and step norm.

`return.details` Logical; if `TRUE`, keep per-evaluation diagnostics.

`chart.geometry` Opt-in chart implementation. Default `"legacy"` preserves existing behavior. `"euclidean"` uses ambient distances; `"linf.simplex"` max-normalizes rows and uses intrinsic Euclidean distances, including intermediate faces. Both new modes support Bernoulli degree 0/1 fits, weighted `"local.pca"` or `"local.pls"`, numeric dimensions and grid CV only. Supports and kernels use the original metric, independent of projection. Directions, weighted centering and score scaling are learned from training rows only, including within every CV fold.

`unfold.faces` Certify and unfold local simplex neighborhoods using `unfold.simplex.chart`? Requires simplex geometry. Unsupported charts use the declared `fallback`: `"degree0"` or `"na"`.

`intercept.mass` In nonlegacy modes, positive symmetric pseudo-count for successes and failures at the weighted chart center. Default 0.5 keeps constant-response supports finite. Kernel times quadrature weights are normalized to support.size before fitting; slopes have the ridge penalty. Convergence requires the maximum absolute gradient below tol, not just a small step. Failed fits retain diagnostic status and use the fallback.

Details

The `likelihood.family = "density"` branch uses a local exponential tilt of the chart reference measure. The `"bernoulli"` branch uses a weighted local logistic likelihood and returns fitted probabilities at the evaluation anchors.

Value

A list with class `"local_likelihood"` containing `fitted.values`, selected controls, and local solver diagnostics.

Examples

```
X <- matrix(seq(0, 1, length.out = 21), ncol = 1)
y <- exp(-20 * (X[, 1] - 0.5)^2)
y <- y / sum(y)
fit.local.likelihood(X, y, likelihood.family = "density",
                     support.size = 7L, degree = 0L)
```

`fit.logistic.trend.filtering`

Fit Order-2 Logistic Trend Filtering

Description

Fits the probability of a binary outcome as a function of one continuous predictor. The logit of the probability is a piecewise-quadratic curve, estimated by maximum likelihood with an L1 penalty on its third divided differences. The computation is done by a dense or a banded solver written in C++; no external optimizer is needed.

Usage

```
fit.logistic.trend.filtering(
  x,
  y,
  lambda,
  weights = NULL,
  covariates = NULL,
```

```
  offset = NULL,
  nuisance.ridge = 0,
  curve.ridge = 0,
  lambda.selection = c("fixed", "cv"),
  foldid = NULL,
  control = list(),
  warn = TRUE,
  backend = c("auto", "dense", "banded"),
  cv.penalty.scale = c("training_mass", "none")
)
```

Arguments

`x`	Finite numeric predictor, with at least four distinct values.
`y`	Numeric response in `[0, 1]`. Binary observations use unit weights; binomial proportions use trial counts as weights.
`lambda`	Nonnegative penalties, or `Inf` for the quadratic limit. Required explicitly: no data-dependent default penalty is assumed.
`weights`	Positive observation weights, default one.
`covariates`	Optional finite numeric matrix of additional predictors, without an intercept. Columns are centered and scaled on training data.
`offset`	Optional finite vector added to the linear predictor.
`nuisance.ridge, curve.ridge`	Nonnegative explicit ridge penalties, default zero. See Details for their precise scaling.
`lambda.selection`	`"fixed"` returns the supplied path without selecting a penalty; `"cv"` selects minimum held-out weighted binomial log loss.
`foldid`	Integer fold labels, required for `lambda.selection="cv"`. Supply the same label for observations that must be held out together.
`control`	Named list overriding `tolerance=1e-9`, `gap.tolerance=1e-10`, `max.iterations=1000L`, or `max.workspace.bytes=1024^3`. Tolerances apply to mean weighted loss.
`warn`	Logical; warn when a path fit fails convergence checks.
`backend`	Native solver: `"auto"` uses `"banded"` above 256 distinct locations or when the dense workspace estimate exceeds its limit, and `"dense"` otherwise. The banded backend uses compact working sets and checks every knot; it does not thin the predictor locations.
`cv.penalty.scale`	`"training_mass"` (default) multiplies each fold's lambda and ridge penalties by its training/full weight-sum ratio. This preserves their coefficients relative to mean loss. `"none"` retains the earlier same-numerical-penalty convention. Fixed-path fits are unchanged.

Details

Predictor locations are normalized to `[0,1]` using the training range. Lambda refers to third divided differences on that normalized scale, not on the original units of x. The minimized objective is

$$\sum_i w_i[\log(1+\exp(\eta_i)) - y_i\eta_i] + \lambda\|D^{(3)}f\|_1 + \frac{r_z}{2}\|\gamma\|^2 + \frac{r_f}{2\sum_i w_i}\sum_i w_i(f_i - \bar{f}_w)^2.$$

Here eta is the curve plus standardized covariates and offset, gamma contains standardized covariate coefficients, and the curve includes an unpenalized intercept. Third differences annihilate quadratic polynomials. Both backends preserve the falling-factorial penalty exactly. The dense solver uses a scaled global basis and a logarithmic barrier. The banded solver uses locally supported linear combinations of the same functions, a fully corrective working set, banded curve Hessians, and a small Schur complement for covariates/centering. Every stopping check scans all knots; omitted working-set coefficients remain exactly zero and violating knots are admitted until the global KKT checks pass. It does not approximate the penalty or restrict the permitted knot locations. Exact ties share a curve value but retain their own likelihood contributions, covariates, and offsets. No jittering or approximate thinning is performed. Extra covariates can be confounded with the curve; without identifying constraints, their individual coefficients are not effects identifiable separately from the curve.

Convergence requires finite arithmetic, small stationarity residual, dual feasibility and small L1 complementarity. The reported dual vector lies in `[-1,1]`; complementarity checks whether it is also an approximate subgradient at the returned coefficients. This is an approximate KKT check, not a separately computed primal-dual objective gap. Iteration limits and numerical failures remain explicit. Separation can prevent a finite minimizer, including in the unpenalized quadratic component. The classical Gaussian trend-filtering minimax result is not a guarantee for this model.

The dense backend is intended for moderate knot counts. The banded backend avoids observation-by-full-knot and full-knot-square arrays during fitting and prediction. Its storage is linear in observations and knots for a fixed number of covariates and penalties; computation also depends on working-set changes and Newton convergence. The conservative workspace estimate is checked before allocation. It is a solver workspace guard, not a process- wide memory limit. The quadratic initialization uses the dense solver only on the three polynomial columns and optional covariates.

CV recomputes training normalization, centering, and scaling in each fold. With the default `cv.penalty.scale="training_` the penalty coefficients relative to weighted mean loss are held constant. The predictor range and standardized covariates still belong to each training split; scaling does not hold a common raw-coordinate operator fixed. A candidate with any failed training fit is ineligible for selection. All fits are refitted on the complete training data. Prediction outside its range uses the nearest boundary curve value; covariates and offsets continue to contribute normally. This boundary policy is not extrapolation of a quadratic polynomial.

Value

An object of class `logistic.trend.filtering.fit`, with `fits`, `lambda`, `diagnostics`, training transformations, optional `cv.table` and `cv.fold.lambda`, `cv.fold.penalty.factors`, `cv.fold.ridge`, the selected `cv.penalty.scale`, and `selected.index`. Failed fits are preserved for diagnosis but standard fitted/prediction methods return NA for them unless explicitly permitted.

See Also

assess.logistic.trend.filtering for a global constancy test and nested-CV predictive improvement for independent binary observations.

Examples

```
x <- seq(0, 1, length.out = 40)
y <- rep(c(0, 1, 0, 1, 1), 8)
model <- fit.logistic.trend.filtering(x, y, lambda = c(1, 0.1))
predict(model, seq(0, 1, length.out = 10), lambda.index = 2)
```

`fit.lpl.tf` *Fit A Local Polynomial Lifting Trend Filter*

Description

Fits an LPL-TF estimator with a fixed operator:

$$\hat{f} = \arg\min_f \frac{1}{2}\|y - f\|_2^2 + \lambda \|A_{\mathrm{LPL}} f\|_1.$$

Usage

```
fit.lpl.tf(
  X = NULL,
  y,
  operator = NULL,
  adj.list = NULL,
  weight.list = NULL,
  graph = NULL,
  degree = 2L,
  lambda.grid = NULL,
  lambda = NULL,
  lambda.selection = c("cv", "fixed"),
  operator.grid = NULL,
  foldid = NULL,
  cv.folds = 5L,
  cv.loss = c("mse", "rmse", "mae"),
  cv.seed = NULL,
  cv.repeats = 1L,
  solver = c("genlasso"),
  selection = c("min", "one.se"),
  n.lambda = 80L,
  maxsteps = 2000L,
  minlam = 0,
  approx = FALSE,
  rtol = 1e-07,
```

```
  btol = 1e-07,
  eps = 1e-04,
  verbose = FALSE,
  cv.penalty.scale = c("training_mass", "none"),
  ...
)
```

Arguments

X	Optional coordinate matrix used to build lpl.tf.operator when operator is NULL.
y	Numeric response vector.
operator	Optional "lpl_tf_operator" object.
adj.list, weight.list, graph	Optional graph inputs passed to lpl.tf.operator when operator is NULL.
degree	Polynomial degree passed to lpl.tf.operator when operator is NULL.
lambda.grid	Optional nonnegative lambda grid.
lambda	Optional fixed lambda shortcut. If supplied, it is used as the single fixed lambda and lambda.selection must be "fixed".
lambda.selection	"cv" or "fixed". CV requires an explicit lambda.grid so the candidate grid is not generated from the full response vector before fold scoring.
operator.grid	Optional operator candidate grid. Supply a data frame with one row per candidate or a list of named lists. Candidate fields may include operator-construction arguments such as degree, support.type, support.size, min.support, support.buffer, kernel, row.normalize, and local.solver.
foldid	Optional deterministic fold assignments for CV.
cv.folds	Number of generated folds when foldid = NULL.
cv.loss	Cross-validation loss, "mse", "rmse", or "mae". "rmse" uses the same lambda ordering as MSE and reports square-rooted fold errors.
cv.seed	Reserved for reproducible generated folds. Generated folds are deterministic even when this is NULL.
cv.repeats	Number of CV repeats. The current implementation supports only one repeat.
solver	Solver backend. The current implementation supports only "genlasso".
selection	Lambda selection rule, "min" or "one.se".
n.lambda	Number of generated lambdas when a genlasso path is used and lambda.grid = NULL.
maxsteps, minlam, approx, rtol, btol, eps	Genlasso controls.
verbose	Logical.
cv.penalty.scale	"training_mass" (default) scales fold lambda by the training/full weight-sum ratio, preserving its coefficient relative to mean loss. "none" uses the earlier unchanged numerical lambda. All folds must have finite predictions for a candidate to be eligible.

... Additional arguments passed to lpl.tf.operator when the operator is built internally.

Value

A list of class "lpl_tf".

Examples

```
if (requireNamespace("genlasso", quietly = TRUE)) {
  X <- matrix(seq(0, 1, length.out = 18), ncol = 1)
  fit.lpl.tf(X, sin(2 * pi * X[, 1]), degree = 1L,
             support.type = "knn", support.size = 7L,
             lambda = 0.1, lambda.selection = "fixed")
}
```

---

fit.lps *Fit a Local Polynomial Smoother*

Description

Fits a local polynomial smoother (LPS) and selects the size of its neighborhoods (the support size), the degree of the polynomial, and the kernel by cross-validation. By default the smoother works in the supplied coordinates. To predict at a point, it takes the nearest training points in Euclidean distance, centers their coordinates at the point, fits a kernel-weighted polynomial to their responses, and uses the intercept of that polynomial as the prediction.

Usage

```
fit.lps(
  X,
  y,
  foldid = NULL,
  support.grid = c(10L, 15L, 20L),
  degree.grid = 0:2,
  kernel.grid = c("gaussian", "tricube"),
  cv.folds = 5L,
  cv.seed = 1L,
  X.eval = NULL,
  coordinate.method = c("coordinates", "local.pca"),
  chart.dim = NULL,
  chart.dim.grid = NULL,
  local.chart.method = c("pca", "second.order.svd"),
  auto.chart.support.metric = c("coordinates", "operator", "both"),
  auto.chart.selection.metric = c("coordinates", "operator"),
  backend = c("auto", "R", "cpp", "cpp.local.pca"),
  design.basis = c("orthogonal.polynomial.drop", "monomial", "weighted.qr",
    "weighted.qr.drop"),
```

```
  design.drop.tol = 1e-08,
  ridge.multiplier.grid = c(0, 1e-10, 1e-08),
  ridge.condition.max = 1e+12,
  unstable.action = c("na", "mean"),
  outcome.family = c("gaussian", "bernoulli", "binomial"),
  bandwidth.multiplier.grid = 1,
  keep.cv.predictions = FALSE,
  ridge.shrinkage.target = c("zero", "local.mean"),
  selection.strategy = c("grid", "sparse_kd", "plateau_kd"),
  chart.activation = c("none", "subject.od"),
  chart.activation.response = NULL,
  chart.activation.control = list(),
  chart.dim.max = NULL,
  design.margin = 2L
)
```

Arguments

X	Numeric design/coordinate matrix with one observation per row.
y	Numeric response vector with length nrow(X).
foldid	Optional positive integer vector assigning rows to CV folds.
support.grid	Integer candidate neighborhood sizes.
degree.grid	Integer polynomial degrees. Currently degrees 0, 1, and 2 are supported.
kernel.grid	Candidate kernels. Supported kernels are "gaussian", "tricube", "epanechnikov", and "triangular".
cv.folds	Number of folds used when foldid is not supplied.
cv.seed	Random seed used to generate folds when foldid is not supplied.
X.eval	Optional matrix of prediction locations. Defaults to X.
coordinate.method	Local coordinate system. "coordinates" uses centered ambient coordinates. "local.pca" uses a local PCA chart.
chart.dim	Chart dimension for coordinate.method = "local.pca". If NULL, defaults to ncol(X). The special value "auto" estimates one global chart dimension from observed X only. The experimental special value "local.auto" estimates a local chart dimension separately for each prediction anchor in the ordinary local-PCA R backend.
chart.dim.grid	Optional candidate chart dimensions for experimental coupled support-size by chart-dimension selection. When supplied, coordinate.method must be "local.pca" and the evaluated candidates use scalar numeric chart dimensions after feasibility filtering. The default NULL preserves the historical single-chart.dim behavior.
local.chart.method	Local chart constructor used when coordinate.method = "local.pca". "pca" preserves the ordinary local-PCA chart path. "second.order.svd" uses an experimental curvature-corrected second-order local SVD chart and records compact chart fallback diagnostics. This option is opt-in and does not affect ambient coordinate fits.

auto.chart.support.metric
: Support system used when chart.dim = "auto" or "local.auto". Included for consistency with LPL-TF and S-LPL-TF; because this smoother uses coordinate supports, "operator" is equivalent to "coordinates".

auto.chart.selection.metric
: Which auto chart-dimension diagnostic to select when both diagnostics are requested.

backend
: Computation backend. "auto" uses the C++ backend for coordinate.method = "coordinates" and the R reference backend for coordinate.method = "local.pca". "R" always uses the reference implementation. "cpp" requires ambient coordinates. "cpp.local.pca" is an opt-in prototype backend for coordinate.method = "local.pca" with local.chart.method = "pca".

design.basis
: Local polynomial design backend. "monomial" uses the ordinary raw monomial design. "weighted.qr" solves the same design using an explicit weighted-QR numerical path. "weighted.qr.drop" first drops numerically dependent local design columns by weighted QR before solving. "orthogonal.polynomial.drop" replaces the local monomial design by a weighted-orthogonal basis spanning its estimable polynomial directions, dropping numerically rank-deficient directions first.

design.drop.tol
: Relative QR tolerance used by design.basis = "weighted.qr.drop" or design.basis = "orthogonal.polynomial.drop".

ridge.multiplier.grid
: Nonnegative scale-relative ridge multipliers tried by the R local-solve backend. The smallest multiplier whose penalized local normal equations pass ridge.condition.max is used.

ridge.condition.max
: Maximum allowed condition number for the penalized local normal equations. Use Inf to disable this guard.

unstable.action
: Action when no local solve passes the rank and condition guards. "mean" preserves the historical weighted-mean fallback. "na" returns NA, causing CV candidates with unstable predictions to be avoided.

outcome.family
: Response family. "gaussian" preserves the ordinary numeric-response LPS behavior. "bernoulli" treats 0/1 responses as numeric conditional-expectation targets for $\Pr(Y = 1 \mid X)$, keeps the same local least-squares fitting core, clips reported response-scale probabilities to [0,1], selects candidates by the observed CV Brier score of the clipped predictions (E2.12: the selection score is the deployed metric, which requires per-point CV predictions, so "bernoulli" – like "binomial" – always uses the R backend: backend = "auto" resolves to "R" and an explicit C++ backend is an error), and records Brier/log-loss probability diagnostics. "binomial" uses local weighted logistic polynomial fits, selects candidates by observed CV log loss with the log-loss probability truncation pinned at 1e-6 (E2.12) and with any candidate having a non-finite CV prediction scored Inf – unselectable, the same rule as the gaussian/bernoulli selection scores (E2.15) – and reports probability diagnostics on the fitted probabilities. The local logistic IRLS uses deviance step-halving (E2.14): a Newton

update is accepted only if the weighted binomial deviance does not increase by more than 1e-8; otherwise the step is halved toward the current iterate (at most 30 times, after which the solve is declared non-convergent). The deviance is evaluated on the same [-35, 35] clamped linear predictor the IRLS update uses, so it is finite for every finite iterate. A solve that does not converge within the iteration cap (including under exact separation, where the unpenalized logistic MLE does not exist) falls back deterministically to the local weighted event rate under unstable.action = "mean" or to NA under unstable.action = "na"; every fallback is counted in logistic.diagnostics (fallback.path, event.rate.fallback, na.failure).

bandwidth.multiplier.grid

Nonnegative bandwidth multipliers added to the CV candidate grid. For a candidate with multiplier b, the local kernel bandwidth becomes $h = b \cdot d_{(K)}$ where $d_{(K)}$ is the distance to the K-th nearest support neighbor, decoupling the kernel scale from the support size. The default 1 reproduces the historical behavior exactly (the bandwidth equals the support radius). Any grid other than exactly 1 requires the R reference backend: backend = "auto" then resolves to "R", and explicit backend = "cpp" or "cpp.local.pca" is an error. The selected multiplier is returned in $selected$bandwidth.multiplier and as a bandwidth.multiplier column of cv.table.

keep.cv.predictions

Logical; if TRUE, the returned object additionally carries cv.predictions, the matrix of out-of-fold CV predictions with one column per cv.table row, so selection scores can be recomputed from the actual CV predictions (E2.12). NULL on the C++ CV paths (reachable only for outcome.family = "gaussian"), which do not materialize per-point predictions. Default FALSE leaves the returned object exactly as before.

ridge.shrinkage.target

Shrinkage target of the local ridge penalty in the least-squares solve (E2.13). The default "zero" preserves the historical behavior bit-for-bit: in the orthogonal design bases the penalty acts on every transformed direction, including the constant, so a large ridge multiplier shrinks the local prediction toward 0. "local.mean" – the statistically recommended setting – leaves the constant direction unpenalized via a weighted-centering reparametrization, so a large ridge shrinks the prediction toward the local weighted mean instead, and rho = 0 remains the unpenalized weighted least-squares solve. The two settings coincide for the non-orthogonal design bases ("monomial", "weighted.qr", "weighted.qr.drop"), whose constant column is already unpenalized, and at rho = 0. The setting applies to the least-squares solve (outcome.family "gaussian" / "bernoulli"); it has *no effect* on the "binomial" local logistic solver, whose ridge keeps the historical structure (a warning is issued if combined).

selection.strategy

Candidate-selection strategy. "grid" preserves the full Cartesian candidate grid. The experimental "sparse_kd" strategy evaluates a sparse coupled support-size by chart-dimension skeleton when chart.dim.grid is supplied. "plateau_kd" is a geometry-only selector: from observed X only, it estimates the local PCA variance dimension over the supplied support grid, finds the initial support-size plateau where that dimension is stable from the smallest support size, aggregates

plateau endpoints across representative anchors, and evaluates the resulting single (support.size, chart.dim) candidate.

chart.activation
: Optional sparse-response chart activation rule. "none" preserves the ordinary LPS behavior. "subject.od" is intended for subject-occupation density workflows: a local chart whose support contains too little subject occupation mass, too few positive subject-visited points, or only fringe occupation receives fitted value zero without constructing the local polynomial fit.

chart.activation.response
: Optional nonnegative response used only by chart.activation = "subject.od" to decide whether a chart is active. When omitted, y is used.

chart.activation.control
: List controlling sparse chart activation. Supported fields are mass.min, n.positive.min, positive.tol, core.weight.rule, core.weight.quantile, and core.weight.min. The OD default is two positive support points and a chart-specific 0.25 weight quantile.

chart.dim.max
: Optional explicit maximum chart dimension for the experimental coupled selector.

design.margin
: Nonnegative prefit margin used to mark coupled (support.size, chart.dim) candidates infeasible when the full local polynomial design would be underdetermined.

Details

With coordinate.method = "local.pca", the neighborhoods and the kernel weights are the same, but the polynomial is fitted in a local coordinate chart computed by PCA and centered at the prediction point. With chart.dim = "auto", a single chart dimension for all points is estimated from X alone, by the same procedure that LPL-TF and S-LPL-TF use. The experimental chart.dim = "local.auto" estimates a chart dimension separately for each prediction point, again from X alone.

Value

A list of class "lps" with response-scale fitted.values, unmodified local least-squares fitted.values.raw, selected parameters, a candidate CV table, the requested local.chart.method, and the effective chart method used for reporting. In outcome.family = "bernoulli" or "binomial" mode, fitted.values are response-scale probabilities in [0,1], fitted.values.raw are the un-clipped conditional-expectation estimates for "bernoulli" and the fitted probabilities for "binomial", cv.table$cv.brier.observed is the observed CV Brier score of the response-scale (clipped) predictions with Inf for candidates having any non-finite prediction, and probability.diagnostics records raw/clipped probability ranges, out-of-range fractions, and Brier/log-loss diagnostics. The Brier and log-loss diagnostics are defined only when the fitted predictions have the same length as the training response, which is the default X.eval = X path. In "binomial" mode, logistic.diagnostics records local logistic solve attempts, convergence statuses, fallback-path counts, event-rate fallback counts, and NA failure counts separately for CV and final fitting.

Choosing controls

First fit: supply X, y, foldid, and small support.grid, degree.grid, and kernel.grid candidates. See the installed function guide.

Selection: `cv.folds`, `cv.seed`, and `selection.strategy` control the search. Candidate combinations and folds multiply the fitting work; use `lps.nested.cv()` for an outer assessment of the tuning procedure.

Geometry: `coordinate.method`, `chart.dim`, and local chart controls describe neighborhoods and coordinate systems, not a different response model.

Numerics: `backend`, `design.basis`, rank thresholds and ridge controls govern stable local solves. Inspect `lps.backend.diagnostics()` before changing them. A numerical ridge is distinct from neighborhood selection.

Diagnostics: `selected`, `cv.table`, and `diagnostics` retain selection and local-fit information. A dense `smoother.matrix` needs one entry per evaluation/training pair; it is restricted to supported configurations.

References

Fan, J. and Gijbels, I. (1996). *Local Polynomial Modelling and Its Applications*. Chapman and Hall/CRC. ISBN 9780412983214.

Examples

```
X <- matrix(seq(0, 1, length.out = 20), ncol = 1)
y <- sin(2 * pi * X[, 1])
fit <- fit.lps(
  X, y, support.grid = 8L, degree.grid = 1L,
  kernel.grid = "tricube", cv.folds = 2L, backend = "R"
)
head(fit$fitted.values)
```

`fit.malps` *Fit Model-Averaged Local Polynomial Smoothing*

Description

Fits a model-averaged local polynomial smoother (MALPS) to a response observed at the rows of a coordinate matrix. MALPS fits a local polynomial regression around each of a set of anchor points, which by default are the observations themselves. The fitted value at a training point is the weighted average of the predictions of all local fits that give that point positive weight.

Usage

```
fit.malps(
  X,
  y,
  graph = NULL,
  adj.list = NULL,
  weight.list = NULL,
  graph.stage = "final",
  anchor.index = NULL,
```

```
  anchor.coordinates = NULL,
  degree = 2L,
  degree.grid = NULL,
  support.type = c("adaptive.radius", "knn", "fixed.radius"),
  support.size = NULL,
  support.grid = NULL,
  radius = NULL,
  radius.grid = NULL,
  min.support = NULL,
  min.support.grid = NULL,
  support.buffer = 3L,
  kernel = c("epanechnikov", "triangular", "gaussian", "tricube"),
  kernel.grid = NULL,
  model.weight.rule = c("none", "condition", "support", "boundary", "quality"),
  duplicate.action = c("keep", "error"),
  coordinate.method = c("coordinates", "local.pca"),
  chart.dim = NULL,
  support.metric = c("auto", "coordinates", "graph.geodesic"),
  auto.chart.support.metric = c("coordinates", "operator", "both"),
  auto.chart.selection.metric = c("coordinates", "operator"),
  support.selection = c("fixed", "cv", "gcv"),
  foldid = NULL,
  cv.folds = 5L,
  cv.loss = c("rmse", "mae", "mse"),
  cv.repeats = 1L,
  cv.seed = NULL,
  cv.one.se = FALSE,
  gcv.exact.max.n = 1000L,
  local.solver = c("auto", "normal.equations", "qr", "svd"),
  normal.equations.max.condition = 1e+08,
  robust.iterations = 0L,
  robust.tuning.constant = 6,
  verbose = FALSE,
  ...
)
```

Arguments

X	Numeric coordinate matrix with one row per observation.
y	Numeric response vector with length nrow(X).
graph	Optional supported **dgraphs** graph object, for example from dgraphs::create.rknn.graph(). When support.metric = "graph.geodesic" or "auto" with a graph supplied, weighted shortest-path distances from this graph are used for support construction.
adj.list	Optional 1-based undirected adjacency list. Must be supplied together with weight.list when using a supplied graph payload.
weight.list	Optional positive edge-length list parallel to adj.list. Edge weights are interpreted as graph lengths.

graph.stage Graph lifecycle stage used when graph is a **dgraphs** graph object. Supported stages include "final", "raw", and "pruned" when those payloads are available on the supplied graph.

anchor.index Optional integer vector of observed rows used as local model anchors. Defaults to all rows when anchor.coordinates is NULL.

anchor.coordinates
Optional numeric coordinate matrix of off-sample local model anchors with the same number of columns as X. This option is currently supported only with coordinate support distances, not with support.metric = "graph.geodesic".

degree Local polynomial degree, one of 0L, 1L, or 2L.

degree.grid Optional degree grid used when support.selection is "cv" or "gcv".

support.type Support construction rule. "knn" uses exactly support.size nearest observations including the anchor itself for full-data local fitting supports. Prediction supports are then rebuilt from the induced local radius, so tied targets at the kNN boundary can be covered even when they were not part of the exact-size fitting support. "adaptive.radius" uses a radius large enough to include at least min.support observations. "fixed.radius" uses the supplied radius.

support.size Number of nearest observations for support.type = "knn". If NULL, defaults to min.support.

support.grid Optional kNN support-size grid used when support.selection is "cv" or "gcv".

radius Radius for support.type = "fixed.radius" and optional base radius for support.type = "adaptive.radius".

radius.grid Optional radius grid used when support.selection is "cv" or "gcv".

min.support Minimum positive-weight support size. If NULL, defaults to the local polynomial design size plus support.buffer.

min.support.grid
Optional minimum-support grid used when support.selection is "cv" or "gcv".

support.buffer Nonnegative integer added to the local design size when deriving the default min.support.

kernel Kernel used for local fitting and model averaging.

kernel.grid Optional kernel grid used when support.selection is "cv" or "gcv".

model.weight.rule
Optional per-anchor model-quality multiplier applied to the ordinary kernel averaging weights. "none" preserves the kernel-only averaging from earlier phases. "condition" downweights poorly conditioned local designs with $q_u^{\mathrm{cond}} = 1/\max(\kappa_u, 1)$. "support" upweights anchors with larger positive fitting supports using support size divided by the maximum support size. "boundary" downweights asymmetric boundary-like supports with $q_u^{\mathrm{bdry}} = 1/(1+b_u)$. "quality" multiplies these three component weights. For every non-"none" rule, diagnostics$model.weight.raw stores the selected unnormalized rule multiplier, while model.weights and diagnostics$model.weight store the final multipliers after median-one normalization over positive raw values. Raw zero or non-finite multipliers are retained diagnostically but are replaced by a tiny positive final floor before prediction averaging, so model-quality

rules reweight existing prediction support rather than removing anchors from support membership. These weights affect prediction averaging only; they do not change local supports or local coefficient estimation.

`duplicate.action`
: How duplicate coordinate rows should be handled. `"keep"` allows duplicates, records duplicate diagnostics, and preserves observed-anchor self-inclusion. `"error"` rejects duplicate rows. Jittering duplicates is deliberately not implemented.

`coordinate.method`
: Coordinate system used for each local polynomial design. `"coordinates"` uses supplied coordinates centered at the anchor. `"local.pca"` uses a deterministic local PCA chart estimated from the anchor support.

`chart.dim`
: Local chart dimension for `coordinate.method = "local.pca"`. If `NULL`, defaults to `ncol(X)`. The special value `"auto"` estimates a single observed-data local PCA dimension without using responses, truth values, latent coordinates, or labels. For `coordinate.method = "coordinates"`, `chart.dim` must be `NULL`.

`support.metric`
: Distance system used for support construction. `"coordinates"` uses Euclidean distances in `X`. `"graph.geodesic"` uses weighted shortest-path distances from `graph` or `adj.list`/`weight.list`. `"auto"` uses graph geodesic distances when graph input is supplied and coordinate distances otherwise.

`auto.chart.support.metric`
: Support system used when `chart.dim = "auto"`. `"coordinates"` uses Euclidean coordinate neighborhoods, `"operator"` uses the resolved MALPS support metric, and `"both"` computes both diagnostics side by side.

`auto.chart.selection.metric`
: Which auto chart-dimension diagnostic to use for the fitted MALPS model when both diagnostics are available. The default `"coordinates"` preserves historical behavior.

`support.selection`
: `"fixed"` for a single support profile, `"cv"` to tune support parameters by cross-validation, or `"gcv"` to tune support parameters by exact dense generalized cross-validation. The GCV path is cached across candidates and is exact for fixed-weight linear MALPS fits; it rejects robust residual reweighting.

`foldid`
: Optional integer fold assignments. If supplied, `cv.repeats` is forced to one.

`cv.folds`
: Number of folds generated when `foldid = NULL`.

`cv.loss`
: Cross-validation loss.

`cv.repeats`
: Number of independent fold assignments generated when `foldid = NULL`.

`cv.seed`
: Optional seed for generated folds.

`cv.one.se`
: Logical; use a one-standard-error rule for support selection.

`gcv.exact.max.n`
: Maximum number of observations allowed for exact dense GCV support selection. Larger datasets should use `"cv"` until a sparse or trace-estimated GCV path is implemented.

`local.solver`
: Local weighted least-squares solver. `"auto"` uses normal equations only for full-rank, well-conditioned local designs and otherwise falls back to SVD.

`normal.equations.max.condition`
: Maximum local design condition number for normal equations under `local.solver = "auto"`.

`robust.iterations`
: Number of Cleveland-style robust residual reweighting iterations applied inside each local polynomial fit. The default `0L` preserves ordinary MALPS. Positive values multiply the geometric fitting weights by Tukey bisquare residual weights recomputed from the local fit.

`robust.tuning.constant`
: Positive bisquare tuning constant. The default `6` matches the usual LOWESS convention in which residuals are scaled by `6 * median(abs(residuals))`.

`verbose`
: Logical; reserved for future progress messages.

`...`
: Reserved for future extensions. Supplying unused arguments is an error.

Details

The neighborhoods (supports) of the local fits can be given fixed parameters, or their parameters can be selected by cross-validation or by exact GCV. The local polynomials can be fitted in the supplied coordinates or in local PCA charts. A fitted model can be refitted to a new response, with or without observation weights (`refit`), can predict at new points when its supports are based on coordinate distances, and supports linear-smoother diagnostics (`smoother.matrix`, `malps.gcv`) and bootstrap summaries (`bootstrap.malps`). Supports based on graph-geodesic distances can be used for fitting and refitting, but prediction at new points is not yet available for them.

The defaults are recommended for most uses: `support.selection = "cv"`, `robust.iterations = 0L`, `model.weight.rule = "none"`, and the observations as anchors (`anchor.coordinates = NULL`). `support.selection = "gcv"` is a faster alternative for fits without robust reweighting, particularly for exploratory smoothing of a continuous response. A binary response is treated as numeric, so the fit estimates a conditional expectation, and the fitted values are not restricted to $[0, 1]$. Grid anchors, empirical-Bayes shrinkage, robust reweighting of local residuals, and the conservative and smoothed GCV rules are experimental.

For graph-geodesic supports, supply a graph constructed with **dgraphs**, or an adjacency list and a weight list (`adj.list`, `weight.list`). Shortest-path distances are computed by **dgraphs**.

Value

A list of class `"malps"` with fitted values, local model coefficients, supports, averaging weights, diagnostics, and selection metadata.

Choosing controls

First fit: supply `X`, `y`, `degree`, and `support.type` with a suitable `support.size` or `radius`. Coordinate-distance fits support new predictions.

Selection: support and degree grids, `foldid`, and `cv.folds` govern held-out selection. Robust refits recompute residual weights; selection is not automatically repeated by `refit`.

Geometry: `support.metric`, graph inputs, anchors, and chart controls determine the local models. Graph-geodesic fits require aligned vertices and do not currently predict at new targets.

Numerics: `local.solver`, conditioning controls, and robust weighting have distinct meanings. Keep their defaults until diagnostics identify a need.

Diagnostics: inspect `selected`, model weights, and stored local fits. `smoother.matrix` allocates a dense matrix and applies only to fixed weighting; its `max.n` guard makes that cost explicit.

Examples

```
X <- matrix(seq(0, 1, length.out = 20), ncol = 1)
fit <- fit.malps(X, sin(2 * pi * X[, 1]), degree = 1L,
                 support.type = "knn", support.size = 8L)
head(fit$fitted.values)
```

---

fit.masked.graph.field

Fit a Graph Field from Partially Observed Responses

Description

Estimate a conditional mean by weighted least squares with an unnormalized graph Laplacian penalty. Repeated observations at a vertex remain separate. Unobserved components return NA rather than an arbitrary constant.

Usage

```
fit.masked.graph.field(operator, vertex, y, lambda, weights = NULL)
```

Arguments

`operator`	An unnormalized `metric.graph.lowpass.operator`.
`vertex`	Integer vertex index for each observed response.
`y`	Finite numeric observed responses (no missing-value placeholders).
`lambda`	Positive smoothing parameter.
`weights`	Positive observation weights; normalized to sum to one.

Value

A list with vertex fitted values, support flags, observation weights, objective, and relative linear-system residual.

Examples

```
op <- metric.graph.lowpass.operator(list(2L, c(1L, 3L), 2L),
                                    list(1, c(1, 1), 1))
fit <- fit.masked.graph.field(op, vertex = c(1, 3), y = c(2, 8), lambda = 0.5)
fit$fitted
```

`fit.metric.graph.lowpass` *Fit Metric-Conductance Graph Low-Pass Regression*

Description

Smooths a response defined on the vertices of a graph by keeping the low-frequency part of its expansion in the eigenvectors of the graph Laplacian. The amount of smoothing is selected by generalized cross-validation. Edge values can be conductances, or lengths that are converted to conductances. With the heat filter and a complete set of eigenvectors, responses with missing values can also be fitted, by penalized least squares on the observed vertices.

Usage

```
fit.metric.graph.lowpass(
  adj.list,
  weight.list,
  y,
  conductance.rule = c("inverse.length.power", "exp.length", "exp.length.squared",
    "self.tuned.gaussian"),
  conductance.epsilon = 1e-08,
  conductance.alpha = 1,
  conductance.sigma = NULL,
  conductance.sigma.rule = c("edge.quantile", "median", "local.k"),
  conductance.sigma.quantile = 0.75,
  conductance.local.k = 5L,
  laplacian.type = c("unnormalized", "symmetric.normalized"),
  n.eigenpairs = 50L,
  filter.type = c("heat_kernel", "tikhonov", "cubic_spline", "gaussian", "exponential",
    "butterworth"),
  eta.grid = NULL,
  n.candidates = 40L,
  eigen.solver = c("auto", "sparse", "dense"),
  dense.eigen.threshold = 200L,
  dense.fallback.threshold = 5000L,
  dense.fallback = c("auto", "never", "always"),
  verbose = FALSE,
  eta.search = c("fixed", "guarded.gcv"),
  eta.expansion.factor = 3,
  eta.max.expansions = 3L,
  eta.identity.departure = 0.01,
  eta.truncation.tol = 1e-04,
  weight.type = c("length", "conductance"),
  spectrum = c("partial", "full"),
  numerical.policy = c("conservative", "verified.fallback")
)
```

Arguments

adj.list	List of integer neighbor vectors using 1-based vertex indices.
weight.list	List of weights parallel to adj.list: positive metric lengths or nonnegative conductances, according to weight.type.
y	Numeric response vector of length length(adj.list). NA marks unobserved vertices and requires a complete basis and heat filtering. Every component must have an observed response.
conductance.rule	Character scalar. One of "inverse.length.power", "exp.length", "exp.length.squared", or "self.tuned.gaussian".
conductance.epsilon	Positive numeric regularizer used by inverse-power conductances and as a local-scale floor.
conductance.alpha	Positive numeric exponent for "inverse.length.power".
conductance.sigma	Optional positive global scale for exponential rules. If NULL, it is selected by conductance.sigma.rule.
conductance.sigma.rule	Rule for selecting a global scale when needed.
conductance.sigma.quantile	Quantile used when conductance.sigma.rule = "edge.quantile".
conductance.local.k	Positive integer local incident-edge order statistic for "self.tuned.gaussian".
laplacian.type	Laplacian operator. "unnormalized" uses the weighted graph Laplacian $L = D - C$. "symmetric.normalized" uses $L_{\mathrm{sym}} = I - D^{-1/2} C D^{-1/2}$.
n.eigenpairs	Positive integer number of eigenpairs to compute.
filter.type	Spectral low-pass filter family.
eta.grid	Optional numeric eta grid. Values must be positive except for filter.type = "heat_kernel", where eta = 0 is allowed as an identity/no-smoothing diagnostic with undefined GCV, excluded from selection. A grid containing only zero therefore cannot produce a fit; use apply.metric.graph.lowpass.path to evaluate that endpoint. If NULL, the existing package helper generate.eta.grid() is used.
n.candidates	Number of eta candidates when eta.grid = NULL.
eigen.solver	"auto", "sparse", or "dense". "auto" uses dense decomposition for n <= dense.eigen.threshold or a request exceeding the sparse backend's n - 2 capacity; otherwise it tries sparse first.
dense.eigen.threshold	Exact dense threshold for auto mode. Default 200L, intended for small reference/testing problems.
dense.fallback.threshold	Maximum graph size for emergency dense fallback when sparse decomposition fails and fallback is allowed.

`dense.fallback` `"auto"`, `"never"`, or `"always"`.

`verbose` Logical. Reserved for future diagnostic messages.

`eta.search` Heat-time search controller. `"fixed"` evaluates the supplied or generated grid once. `"guarded.gcv"` is available only for `filter.type = "heat_kernel"` and proposes one lower positive time whenever GCV selects the current lower endpoint.

`eta.expansion.factor`
Geometric divisor used by guarded GCV lower-time expansion.

`eta.max.expansions`
Maximum number of guarded GCV lower-time extensions.

`eta.identity.departure`
Smallest allowed departure from the identity at the largest retained graph-Laplacian eigenvalue.

`eta.truncation.tol`
Omitted-mode attenuation tolerance used to certify guarded proposals for a truncated basis.

`weight.type` Weight interpretation. `"length"` transforms lengths; `"conductance"` uses positive weights unchanged and removes zero edges. Reciprocal conductances must agree; self-loops must be removed by the caller. Conductance-transform settings are unused for direct conductances.

`spectrum` `"partial"` retains the requested number of modes. `"full"` requests and verifies all vertex eigenpairs with a dense solver, overriding `n.eigenpairs`; sparse-only requests are rejected.

`numerical.policy`
See `metric.graph.lowpass.basis`. The default is conservative; the bounded fallback must be requested explicitly.

Details

Missing-response heat fits minimize observed squared error plus $f^T(\exp(tL) - I)f$. This reduces to ordinary heat filtering when all vertices are observed. GCV is $q\,RSS/(q - \mathrm{tr}(S_O))^2$, with q observed responses and observed-data operator S_O. Zero heat time is an identity diagnostic with undefined GCV and cannot be selected. A grid with no valid GCV candidate fails explicitly.

Value

A list of class `"metric.graph.lowpass.fit"`. The GCV record includes per-candidate status, observed count and degrees of freedom. Missing-response residuals remain `NA`. The `fitting.mode` field distinguishes the observed heat-penalty fit from a fully observed spectral filter. For missing responses, the stored spectral filter weights describe the base heat kernel, not eigenvalues of the observed fitting matrix.

References

Gajer, P. and Ravel, J. (2025). Adaptive geometric regression for high-dimensional structured data. arXiv:2511.03817. doi:10.48550/arXiv.2511.03817

Examples

```
adj <- list(2L, c(1L, 3L), c(2L, 4L), 3L)
lengths <- list(1, c(1, 1), c(1, 1), 1)
fit.metric.graph.lowpass(
  adj, lengths, y = c(0, 0.2, 1.8, 2), n.eigenpairs = 4L,
  eta.grid = c(0.1, 1), eigen.solver = "dense"
)
```

---

fit.neighbor.probability

Nearest-Person and Nearest-Case Binary Probability Estimators

---

Description

Estimates the probability of a binary outcome at a point from the outcomes of its nearest neighbors, with one of three estimators. The ordinary nearest-neighbor estimator uses the `k` nearest people. The two nearest-case estimators instead extend the neighborhood until it contains `k` cases. All three are computed by the same single-threaded C++ code, which handles one query point at a time and never forms the matrix of all pairwise distances.

Usage

```
fit.neighbor.probability(
  X = NULL,
  y,
  method = c("knn", "case_raw", "case_corrected"),
  k = NULL,
  k.grid = NULL,
  distances = NULL,
  metric = c("euclidean", "chebyshev", "linf.simplex"),
  normalization = c("none", "sum", "max"),
  id = NULL,
  group = NULL,
  foldid = NULL,
  selection = c("min", "one.se", "locality.one.se"),
  search = c("auto", "ann", "brute"),
  eps = 0
)
```

Arguments

`X`	Numeric matrix with one training record per row. Supply either `X` or `distances`.
`y`	Binary responses, with `NA` for unanswered outcomes.
`method`	`"knn"`, `"case_raw"`, or `"case_corrected"`.
`k`	Fixed positive integer neighborhood size. Counts people for `"knn"` and cases for the other methods. Corrected sizes must be at least two. Omit for cross-validation using `foldid`.

k.grid	Optional candidate sizes for cross-validation. If omitted, use every size up to the minimum feasible training count across folds.
distances	Optional square, nonnegative training dissimilarity matrix in record order. Positive infinity denotes an unreachable record. No graph or powered path distances are constructed internally.
metric	Coordinate distance: "euclidean", "chebyshev", or "linf.simplex". The last computes exact intrinsic Euclidean-length paths on the nonnegative max-normalized surface, using dgraphs' C++ solver. Paths can cross intermediate faces; this is not Chebyshev distance.
normalization	Row normalization: "none", "sum", or "max". The latter two require nonnegative rows with positive mass. Supply already-computed distances with normalization = "none". Normalization changes coordinates, not the distance definition: Euclidean distances on max-normalized profiles are ambient chords, not intrinsic paths on the cube surface. Use metric = "linf.simplex" for those paths, with normalization = "max". Alternatively, "none" requires nonnegative rows whose maxima equal one within sqrt(.Machine$double.eps); admitted rows are divided by their maxima. Sum normalization is rejected for intrinsic coordinate input.
id	Unique nonmissing record IDs. Defaults to row numbers as strings. Supply persistent IDs to obtain ordering invariant to row permutations.
group	Nonmissing participant IDs, defaulting to id.
foldid	Optional fold vector or matrix (one repeat per column). Every participant must stay in one fold per repeat. Missing outcomes are excluded from scoring, but their groups still obey the split.
selection	Minimum Brier error, largest size within one standard error ("one.se"), or smallest such size ("locality.one.se").
search	Neighbor search: "auto" uses ANN for Euclidean coordinates and brute force otherwise; "ann" explicitly requests the ANN kd-tree; "brute" is the exhaustive reference backend. ANN requires ambient Euclidean coordinates, including normalized coordinates. Intrinsic surface search always exhausts eligible candidates using one query's workspace, with no all-pairs distance matrix.
eps	Nonnegative ANN approximation tolerance, default zero (exact). Positive values require ANN. The distance approximation guarantee does not bound probability error or case stopping ranks after exclusions.

Details

Rank answering training records by distance, then by the SHA-256 hash of paste0("nearest-cases-v1|", id). Hash collisions use ID order. Keep only the closest answering record per participant, before inspecting its outcome. Exclude the query's entire participant when its group is supplied.

If T_k is the rank including the kth case, the estimates are $K^{-1}\sum_{j=1}^{K} y_{(j)}$, k/T_k, and $(k-1)/(T_k-1)$. No clipping, pseudocount or automatic baseline fallback is applied. Insufficient reachable people or cases return NA. The number of cases reachable can depend on the query when a participant's visits have discordant outcomes.

ANN expands candidate neighborhoods until the requested people or cases are available. Exact search completes the boundary before applying the ID tie rule. Approximate search sorts retrieved

candidates by exact distance; for each retrieved participant it also checks all answering visits, keeping the closest. It can miss other participants and tied records. Approximate boundary diagnostics describe retrieved participants, not a complete shell. Available people/cases diagnostics still describe the full training set. Intrinsic surface search prepares sparse training profiles once per prediction call or CV split and streams a complete eligible distance row for each query. It uses dgraphs' shared intrinsic solver directly, without constructing a sample graph, limiting paths to endpoint faces, or truncating candidate neighbors. All three estimators share the same participant and tie rules. Per-pair worst-case work is quadratic in the number of features present in either profile. Distance workspace is linear in training records.

Trees are reused across queries and sizes, and across participants within each CV split. They are rebuilt for each prediction call and are not stored in the serializable fit. CV retains one tree or sparse intrinsic profile cache per split and repeat; a new prediction call rebuilds its cache. Native caches are released on return or error from prediction or tuning.

Tuning averages losses within participant, then across repeats, then across participants. Its standard error is the standard deviation of those participant losses divided by the square root of the number of people. Candidates with any unsupported validation prediction are unselectable. Numerical ties within `1e-14` favor larger sizes for minimum-error selection. The tuning score is not an independent performance estimate; use external participant-held outer folds for assessment.

This implementation matches the sPTB study's prediction arithmetic and minimum-error tuning when there is one record per participant, the same distances, IDs and folds. The repeated-record policy is an explicit extension. At a tied distance the prefix can split a tied shell, as in that study.

Value

A `neighbor.probability.fit` with selected `k`, `method`, `backend = "C++"`, candidate scores, folds, and the `metric`, `distance.backend`, and training data needed for prediction. No training predictions are computed during fitting; `predict(fit)` computes leave-participant-out values.

Examples

```
X <- matrix(0:11, ncol = 1)
y <- rep(c(0, 1, 1), 4)
fit <- fit.neighbor.probability(X, y, method = "case_corrected", k = 2)
predict(fit, newdata = matrix(c(1.5, 5.5), ncol = 1))
tuned <- fit.neighbor.probability(X, y, method = "knn",
                                  foldid = rep(1:3, 4), k.grid = 1:4)
tuned$cv.table
# Intrinsic surface neighbors can differ from ambient Euclidean neighbors.
profiles <- rbind(c(0, 1, 0, 0), c(1, 1, 1, 1))
surface <- fit.neighbor.probability(profiles, c(1, 0), k = 1,
    metric = "linf.simplex", normalization = "max")
predict(surface, rbind(c(1, 0, 0, 0)))
```

`fit.projection.trend.filtering`

Projection Pursuit with Order-2 Trend Filtering

Description

Fit a sum of piecewise-quadratic curves on fixed or learned linear projections. Gaussian models use squared loss; binomial models add the curves on the logit scale.

Usage

```
fit.projection.trend.filtering(
  X,
  y,
  lambda,
  family = c("gaussian", "binomial"),
  directions = NULL,
  direction.search = c("learned", "fixed", "gn", "pls"),
  nterms = NULL,
  weights = NULL,
  offset = NULL,
  curve.ridge = 0.001,
  preprocessing = c("scale", "isotropic", "none"),
  seed = 1L,
  control = list(),
  backend = c("auto", "dense", "banded"),
  warn = TRUE
)
```

Arguments

`X`	Finite numeric predictor matrix or data frame.
`y`	Finite numeric response. For `family="binomial"`, values must lie in `[0,1]`; proportions can be accompanied by trial weights.
`lambda`	Nonnegative scalar or `Inf`, on the mean-loss scale.
`family`	`"gaussian"` or `"binomial"`.
`directions`	Matrix with one direction per row and one column per original predictor. Directions refer to the preprocessed coordinates. Entries for dropped constant columns must be zero. In learned mode these are additional candidates; in fixed mode these are the fitted directions.
`direction.search`	`"fixed"` fits supplied directions (coordinate directions when omitted); `"learned"` selects and refines directions with pattern search. `"gn"` uses multistart damped Gauss–Newton proposals; `"pls"` uses one covariance start per component with the same Gauss–Newton refinement. The latter two modes construct their own starts and do not accept supplied directions.
`nterms`	Number of supplied directions in fixed mode; maximum number of components in learned mode. Defaults to the supplied row count, all nonconstant coordinates in fixed mode, or two in learned mode. Zero fits an intercept only.
`weights`	Positive finite observation weights, default one.
`offset`	Known additive predictor, default zero.

curve.ridge	Nonnegative component ridge coefficient on the mean-loss scale; default 0.001. Positive ridge stabilizes component decomposition.
preprocessing	"scale" centers and scales each nonconstant column; "isotropic" centers and uses a common scale, preserving rotations and distance ratios; "none" leaves retained columns unchanged. All transformations use training weights only. The default is "isotropic" for "gn" and "pls", and "scale" otherwise.
seed	Integer random seed. Local randomness leaves the caller's RNG state unchanged.
control	Named list. max.sweeps=200, tolerance=1e-7, gap.tolerance=1e-8, and objective.tolerance=1e-10 govern joint fitting. inner=list() passes controls to the univariate solver. joint.solver="backfit" retains alternating component updates; joint.solver="dense" attempts a simultaneous dense solve when there are exactly two components, with guarded fallback to backfitting. joint.max.workspace.bytes=1024^3 bounds the estimated dense workspace. The joint certificate and objective safeguard apply to both solvers. Search controls are starts=2, sparse.directions=8, dense.directions=8, refinement.passes=2, refinement.proposals=16, steps=c(0.25,0.1,0.04,0.01), and improvement.tolerance=1e-9. Zero refinement passes select from a fixed candidate bank. The "gn" and "pls" modes instead use gn.max.steps=8 accepted proposals and gn.max.fits=40 native fits per start, with gn.shared.fits=400 shared fits for previously selected components. stream.key=character() can identify a dataset or representation; local start streams also depend on the seed and component number.
backend	Univariate solver: "auto", "dense", or "banded".
warn	Warn when joint fitting or a terminal search failure makes the returned fit ineligible. Failed candidates are retained in search.history.

Details

For training weight mass W, the objective is

$$W^{-1}\sum_i w_i\ell(y_i,\eta_i)+\lambda\sum_j\|D_j^{(3)}h_j\|_1+\frac{\rho}{2W}\sum_{ij}w_ih_j(u_j^Tz_i)^2.$$

Each component has training weighted mean zero; directions have unit norm and a canonical sign. Projection ranges and the falling-factorial basis follow fit.gaussian.trend.filtering and fit.logistic.trend.filtering. Native lambda and curve ridge are each multiplied by W exactly once. Gaussian loss includes a factor of one half. lambda=Inf restricts components to quadratics; it does not select a constant model.

Fixed-direction fitting is convex. Backfitting updates a curve with the other components supplied as an offset and checks joint stationarity, dual feasibility, complementarity and the original objective after each sweep. The "learned" mode uses greedy additions and derivative-free tangent proposals, followed by joint refitting. The search is nonconvex; its recorded stopping status is not a global optimality certificate. Gauss–Newton modes use one-sided derivatives of the falling-factorial curve, solve in the tangent space of the unit direction, and accept only certified joint-objective improvements. Binomial curvature is floored at 1e-6 without changing the likelihood score. All curve fits use the original responses and offsets, including for binomial outcomes. Use preprocessing="isotropic" to preserve the geometry of supplied score coordinates. PCA and variance-threshold tuning are not performed automatically by this function. In "gn" mode the ordered starts are coordinate axes, weighted ridge, weighted covariance, sliced inverse regression, and

twice as many random directions as retained coordinates. Sign-equivalent duplicates retain their first origin. SIR uses two outcome classes for the first Bernoulli component with unit trial weights; otherwise it slices the centered working response by weight mass, keeping exact response ties together. Both supervised seeds and fitted curves eliminate a weighted intercept. A start must obtain a completed joint certificate before it can be selected. A per-start limit can retain that certificate; exhaustion of the candidate or shared allowance makes the candidate ineligible. With q retained coordinates and requested K components, the default native-fit caps are `40*K*(3*q+3)+400` for `"gn"` and `40*K+400` for `"pls"`. An intercept-only fit makes no native curve-fitting calls. Directions with fewer than four distinct projections are ineligible. Exact ties are retained. Distinct projected locations that collapse during floating-point normalization produce an explicit failed fit; locations are never rounded, merged or jittered to repair a candidate. One-class binomial responses have no finite unpenalized intercept and produce an error.

Value

A `projection.trend.filtering.fit` containing centered `terms`, `intercept`, canonical `directions` in retained predictor coordinates, component fits, saved `preprocessing`, `checks`, `converged`, `status`, `objective.history`, and a candidate-level `search.history`. `checks$converged` reports joint numerical optimality; `converged` also excludes terminal search failures. Search termination is recorded separately in `search.status`. Ordinary predictions are NA for failed fits. Gauss–Newton modes additionally return `numerical.certified`, `cv.eligible`, `counters`, `curve.diagnostics`, and `search.banks`. An interrupted search may preserve a numerical certificate while being ineligible for CV. Requested and effective component counts can differ. In `counters`, `candidate.fits` counts returned attempts, while `completed.candidate.fits` counts eligible, certified results.

Examples

```
x <- seq(-1, 1, length.out = 24)
X <- cbind(x, cos(3*x))
fit <- fit.projection.trend.filtering(X, sin(2*x), lambda = 0.002,
    directions = matrix(c(1, 0), 1), direction.search = "fixed")
predict(fit, X[1:3, , drop = FALSE])
```

---

fit.ps.lps *Fit Prediction-Synchronized Local Polynomial Smoothing*

Description

Fits prediction-synchronized local polynomial smoothing (PS-LPS). For given local polynomial settings, PS-LPS fits one local polynomial for each anchor point and adds a quadratic penalty that makes the predictions of different local polynomials agree at the points where their neighborhoods overlap. This is an experimental implementation written in R.

Usage

```
fit.ps.lps(
  X,
  y,
```

```
  foldid = NULL,
  support.size = NULL,
  degree = 2L,
  kernel = "gaussian",
  chart.dim = NULL,
  support.grid = NULL,
  degree.grid = NULL,
  kernel.grid = NULL,
  auto.chart.support.metric = c("coordinates", "operator", "both"),
  auto.chart.selection.metric = c("coordinates", "operator"),
  chart.dim.grid = NULL,
  selection.strategy = c("grid", "sparse_kd", "plateau_kd"),
  chart.dim.max = NULL,
  design.margin = 2L,
  lambda.sync.grid = c(0, 0.001, 0.01, 0.1, 1, 10),
  lambda.sync.search = c("grid", "guarded"),
  lambda.sync.selection = c("cv", "fixed"),
  local.candidate.search = c("screened", "full", "subgrid"),
  local.candidate.search.control = list(),
  lambda.sync.search.control = list(),
  lambda.diagnostics = c("all", "selected"),
  lambda.ridge = 1e-08,
  design.basis = c("monomial", "weighted.qr", "weighted.qr.drop",
    "orthogonal.polynomial.drop"),
  design.drop.tol = sqrt(.Machine$double.eps),
  ridge.multiplier.grid = NULL,
  ridge.condition.max = Inf,
  sync.neighbor.size = NULL,
  overlap.weight = c("normalized.product", "product"),
  chart.activation = c("none", "subject.od"),
  chart.activation.response = NULL,
  chart.activation.control = list(),
  ps.lps.geometry.cache = NULL,
  ps.lps.local.pca.supports = NULL,
  cv.folds = 5L,
  cv.seed = 1L
)
```

Arguments

X	Numeric covariate matrix.
y	Numeric response vector.
foldid	Optional integer cross-validation fold assignment.
support.size	Optional single neighborhood size for the fixed local setup path. Use support.grid for CV selection over neighborhood sizes.
degree	Single local polynomial degree for the fixed local setup path. Use degree.grid for CV selection over degrees.

`kernel` Single kernel name for the fixed local setup path. Use `kernel.grid` for CV selection over kernels.

`chart.dim` Chart dimension for the local PCA charts. A scalar fixes one dimension for all anchors; an integer vector supplies one local chart dimension per anchor. The special value `"auto"` estimates one global chart dimension from observed `X` only. The experimental special value `"local.auto"` estimates one local chart dimension per anchor, using the same local-PCA dimension rule as `fit.lps`.

`support.grid` Candidate neighborhood sizes. When supplied, PS-LPS selects over support size, degree, kernel, and synchronization strength by materialized-fold CV. If both `support.size` and `support.grid` are absent, defaults to `c(10L, 15L, 20L)`, matching `fit.lps`.

`degree.grid` Candidate local polynomial degrees for grid selection. Defaults to the scalar `degree`.

`kernel.grid` Candidate kernels for grid selection. Defaults to the scalar `kernel`.

`auto.chart.support.metric`
Support system used when `chart.dim = "auto"` or `"local.auto"`. Because PS-LPS uses coordinate supports, `"operator"` is equivalent to `"coordinates"`.

`auto.chart.selection.metric`
Which auto chart-dimension diagnostic to select when both diagnostics are requested.

`chart.dim.grid` Optional candidate chart dimensions for experimental local-candidate selection. Numeric grids can be evaluated with `selection.strategy = "sparse_kd"`.

`selection.strategy`
Candidate-selection strategy. `"grid"` evaluates the requested candidate grid. `"sparse_kd"` evaluates a sparse support-size by chart-dimension skeleton when `chart.dim.grid` is supplied. `"plateau_kd"` uses the same geometry-only support-size and chart-dimension plateau rule as `fit.lps`.

`chart.dim.max` Optional explicit maximum chart dimension for the sparse coupled candidate family.

`design.margin` Nonnegative integer feasibility margin used to screen local polynomial design size before candidate evaluation.

`lambda.sync.grid`
Candidate synchronization strengths.

`lambda.sync.search`
Lambda-search policy. `"grid"` evaluates the supplied grid exactly. `"guarded"` uses an experimental guarded coarse-to-refine search with boundary expansion.

`lambda.sync.selection`
Lambda-selection mode. `"cv"` performs the usual internal fold-weighted lambda selection. `"fixed"` treats the single value in `lambda.sync.grid` as preselected and skips internal lambda CV; this is intended for outer selection loops that already score scalar candidates.

`local.candidate.search`
Local-candidate search policy when `support.grid`, `degree.grid`, or `kernel.grid` contains more than one candidate. `"screened"` is the routine default: it first ranks candidates by ordinary LPS materialized-fold CV, then runs PS-LPS only

on a screened subset plus guard candidates. "full" evaluates PS-LPS lambda search for every local candidate and is the exact audit/reference path. "subgrid" skips the ordinary-LPS screening pass and evaluates only a deterministic support/kernel guard subgrid; this is intended for high-dimensional preflight runs where the screening pass is itself too expensive.

local.candidate.search.control
: Optional list controlling screened local-candidate search. Supported fields are top.n (default 8), max.candidates (default 12), neighbor.radius (default 1), and guard.support.quantiles (default c(0, 0.5, 1)).

lambda.sync.search.control
: Optional list controlling guarded search. Supported fields are coarse.size (default 5), refine.radius (default 2), rel.tol (default 0.002), boundary.guard.rel.tol (default 0.01), boundary.expand (default TRUE), boundary.factor (default 3), max.boundary.expansions (default 2), and max.candidates (default 25). Boundary expansion may evaluate positive candidates outside the supplied lambda.sync.grid. max.candidates is a global cap on distinct evaluated candidates; a very small cap can prevent local refinement or boundary expansion.

lambda.diagnostics
: Lambda-diagnostic evaluation policy. "all" computes synchronization-energy and local-GCV diagnostics for every evaluated lambda and is the audit/reference mode. "selected" computes those diagnostics only for the selected lambda, which is faster for routine selection.

lambda.ridge
: Nonnegative scale-relative ridge used in the chart coefficient solve. Use 0 for the unregularized least-squares model.

design.basis
: Local polynomial design backend. See fit.lps. In PS-LPS, "weighted.qr.drop" drops numerically dependent columns separately in each anchor chart before the synchronized system is assembled, and "orthogonal.polynomial.drop" builds each synchronized chart in a weighted-orthogonal polynomial basis.

design.drop.tol
: Relative QR tolerance used by design.basis = "weighted.qr.drop" or design.basis = "orthogonal.polynomial.drop".

ridge.multiplier.grid
: Optional nonnegative ridge multipliers for adaptive scale-relative ridge selection. When supplied, the solver uses the smallest multiplier whose penalized normal equations pass ridge.condition.max. If NULL, lambda.ridge is used as the single multiplier for backward compatibility.

ridge.condition.max
: Maximum allowed condition number for adaptive ridge selection. Use Inf to disable the condition-number guard.

sync.neighbor.size
: Number of nearby anchor pairs considered for synchronization from each anchor support.

overlap.weight
: Overlap weighting rule.

chart.activation
: Optional sparse-response chart activation rule. "none" preserves ordinary PS-LPS. "subject.od" is intended for subject-occupation density workflows and

deactivates local charts with no subject mass, too few positive subject-visited support points, or only fringe occupation.

`chart.activation.response`
: Optional nonnegative vector used only for `chart.activation = "subject.od"` to decide whether each chart is active. When omitted, `y` is used.

`chart.activation.control`
: List controlling sparse chart activation; see `fit.lps`.

`ps.lps.geometry.cache`
: Optional precomputed geometry cache used by occupation-density cross-validation. This is an implementation detail; ordinary callers should leave it as `NULL`.

`ps.lps.local.pca.supports`
: Optional precomputed local-PCA supports used when activation-specific frames must be rebuilt. This is an implementation detail; ordinary callers should leave it as `NULL`.

`cv.folds`
: Number of folds when `foldid` is absent.

`cv.seed`
: Fold seed when `foldid` is absent.

Value

A list with fitted values, selected lambda, CV table, diagnostics, and fitted chart coefficients.

Examples

```
X <- matrix(seq(0, 1, length.out = 20), ncol = 1)
fit.ps.lps(
  X, sin(2 * pi * X[, 1]), support.size = 7L, degree = 1L,
  chart.dim = 1L, lambda.sync.grid = 0,
  lambda.sync.selection = "fixed", cv.folds = 2L
)
```

`fit.pttf.trend.filtering` *Fit A Parallel-Transport Trend Filter From A PTTF Operator*

Description

Fits experimental PTTF regression models from a transported difference operator produced by `pttf.operator`. The implementation focuses on one response vector and explicit operator-row policies, especially the 1-D boundary policy used to separate interior transported rows from endpoint rows.

Usage

```
fit.pttf.trend.filtering(
  geometry = NULL,
  operator = NULL,
  X = NULL,
  y,
  derivative.order = 3L,
  penalty = c("l1", "l2"),
  lambda.grid = NULL,
  lambda.selection = c("cv", "fixed"),
  n.lambda = 80L,
  nfolds = 5L,
  foldid = NULL,
  cv.loss = c("mse", "mae"),
  selection = c("min", "one.se"),
  lambda.extension = c("none", "auto"),
  solver = c("genlasso", "admm", "auto"),
  operator.row.policy = c("all", "drop.line.boundary", "diagnostic.only"),
  line.order = NULL,
  boundary.trim = NULL,
  row.mass.rule = c("none", "node.mass"),
  row.normalize = c("none", "l2"),
  weights = NULL,
  maxsteps = 2000L,
  minlam = 0,
  approx = FALSE,
  rtol = 1e-07,
  btol = 1e-07,
  eps = 1e-04,
  admm.rho = NULL,
  admm.maxiter = 2000L,
  admm.abstol = 1e-04,
  admm.reltol = 0.001,
  verbose = FALSE,
  admm.adaptive.rho = TRUE,
  ...
)
```

Arguments

geometry	Optional "pttf_geometry" object. Used only when operator is NULL.
operator	Optional "pttf_operator" object. Supplying an operator is recommended for validation so geometry/operator construction is separated from fitting.
X	Optional data matrix used to build pttf.geometry when both geometry and operator are NULL.
y	Numeric response vector.

`derivative.order`
: Integer derivative order passed to `pttf.operator` when an operator must be built.

`penalty`
: `"l1"` for generalized-lasso style fitting or `"l2"` for the quadratic energy fit.

`lambda.grid`
: Optional lambda grid. Required for `lambda.selection = "fixed"`.

`lambda.selection`
: `"cv"` or `"fixed"`.

`n.lambda`
: Number of lambdas in a generated grid.

`nfolds`
: Number of folds used only when `foldid = NULL`.

`foldid`
: Optional integer fold assignments. Validation lanes should pass a materialized `foldid`; otherwise folds are generated by R's current RNG state.

`cv.loss`
: Cross-validation loss, `"mse"` or `"mae"`.

`selection`
: Lambda selection rule, `"min"` or `"one.se"`.

`lambda.extension`
: Currently recorded as metadata; no extension heuristic is implemented.

`solver`
: L1 solver backend.

`operator.row.policy`
: Which operator rows to use for fitting.

`line.order`
: Vertex order for 1-D boundary-row policies.

`boundary.trim`
: Number of endpoint ranks to drop for `operator.row.policy = "drop.line.boundary"`. Defaults to `derivative.order`.

`row.mass.rule`, `row.normalize`
: Passed to `pttf.operator` when this function builds an operator. `row.normalize` is also used as the L1 solver row scaling for the supplied fit operator.

`weights`
: Optional nonnegative observation weights.

`maxsteps`, `minlam`, `approx`, `rtol`, `btol`, `eps`
: Genlasso controls for L1 fits.

`admm.rho`, `admm.maxiter`, `admm.abstol`, `admm.reltol`
: ADMM controls used when the L1 backend is `"admm"` or falls back to ADMM. A `NULL` initial penalty is chosen from the operator scale. The shared solver's convergence checks, failure behavior, and CV eligibility rules are described in `fit.ssrhe.hessian.l1.regression`.

`verbose`
: Logical.

`admm.adaptive.rho`
: Logical; adapt the ADMM penalty during the initial iterations. Set `FALSE` to keep its initial value fixed.

`...`
: Additional arguments passed to `pttf.geometry` or `pttf.operator` when those objects are built internally.

Value

A list of class `"pttf.trend.filtering.fit"`.

Examples

```
n <- 10L
X <- matrix(seq(0, 1, length.out = n), ncol = 1)
adj <- lapply(seq_len(n), function(i) intersect(c(i - 1L, i + 1L), 1:n))
lengths <- Map(function(i, j) abs(X[j, 1] - X[i, 1]), seq_len(n), adj)
geometry <- pttf.geometry(
  X, adj, lengths, graph = "supplied", tangent.dim = 1L
)
operator <- pttf.operator(geometry, derivative.order = 2L)
fit.pttf.trend.filtering(
  operator = operator, y = sin(2 * pi * X[, 1]), penalty = "l2",
  lambda.grid = 0.1, lambda.selection = "fixed"
)
```

`fit.slpl.tf` *Fit A Fixed-Operator Synchronized LPL-TF Model*

Description

Fits the fixed-operator S-LPL-TF estimator

$$\hat{f} = \arg\min_f \frac{1}{2}\|y - f\|_2^2 + \lambda_1 \|A_{\mathrm{LPL}} f\|_1 + \frac{\lambda_2}{2}\|C_{\mathrm{sync}} f\|_2^2.$$

Usage

```
fit.slpl.tf(
  X = NULL,
  y,
  operator = NULL,
  lambda1 = NULL,
  lambda2 = 0,
  lambda1.grid = NULL,
  lambda2.grid = NULL,
  lambda.selection = c("fixed", "cv"),
  operator.grid = NULL,
  foldid = NULL,
  cv.folds = 5L,
  cv.loss = c("mse", "rmse", "mae"),
  cv.seed = NULL,
  cv.repeats = 1L,
  solver = c("genlasso"),
  selection = c("min", "one.se"),
  maxsteps = 2000L,
  minlam = 0,
  approx = FALSE,
  rtol = 1e-07,
  btol = 1e-07,
```

```
  eps = 1e-04,
  verbose = FALSE,
  ...
)
```

Arguments

`X`	Optional coordinate matrix used to build `slpl.tf.operator` when `operator` is `NULL`.
`y`	Numeric response vector.
`operator`	Optional `"slpl_tf_operator"` object.
`lambda1`	Fixed LPL-TF ℓ_1 penalty, or a shortcut for `lambda1.grid` when `lambda.selection = "cv"`.
`lambda2`	Fixed quadratic synchronization penalty, or a shortcut for `lambda2.grid` when `lambda.selection = "cv"`.
`lambda1.grid`	Optional nonnegative λ_1 grid for CV.
`lambda2.grid`	Optional nonnegative λ_2 grid for CV.
`lambda.selection`	`"fixed"` or `"cv"`. CV uses materialized folds and a deterministic Cartesian grid over `lambda1.grid` and `lambda2.grid`.
`operator.grid`	Optional operator candidate grid. Supply a data frame with one row per candidate or a list of named lists.
`foldid`	Optional deterministic fold assignments for CV.
`cv.folds`	Number of generated folds when `foldid = NULL`.
`cv.loss`	Cross-validation loss, `"mse"`, `"rmse"`, or `"mae"`.
`cv.seed`	Optional seed for reproducible generated folds.
`cv.repeats`	Number of CV repeats. The current implementation supports only one repeat.
`solver`	Current implementation supports only `"genlasso"`.
`selection`	Selection rule, `"min"` or `"one.se"`. For the two-parameter one-standard-error rule, the most regularized eligible pair is chosen by decreasing `lambda1` and then decreasing `lambda2`.
`maxsteps`, `minlam`, `approx`, `rtol`, `btol`, `eps`	Genlasso controls.
`verbose`	Logical.
`...`	Additional arguments passed to `slpl.tf.operator` when the operator is built internally.

Value

A list of class `"slpl_tf"`.

Examples

```
if (requireNamespace("genlasso", quietly = TRUE)) {
  X <- matrix(seq(0, 1, length.out = 18), ncol = 1)
  fit.slpl.tf(X, sin(2 * pi * X[, 1]), degree = 1L,
              support.type = "knn", support.size = 7L,
              lambda1 = 0.1, lambda2 = 0, lambda.selection = "fixed")
}
```

`fit.ssrhe.hessian.l1.regression`

Fit SSRHE-Style Hessian L1 Regression

Description

Fits a regression function on a point cloud by least squares with an ℓ_1 (absolute-value) penalty on the estimated second derivatives Af, where A is the operator returned by `ssrhe.hessian.operator`:

$$\widehat{f}_\lambda = \arg\min_f \left\{ \frac{1}{2} \sum_i w_i (y_i - f_i)^2 + \lambda \|Af\|_1 \right\}.$$

Usage

```
fit.ssrhe.hessian.l1.regression(
  X,
  y,
  k = NULL,
  tangent.dim,
  lambda.grid = NULL,
  lambda.selection = c("cv", "fixed"),
  weights = NULL,
  n.lambda = 40L,
  nfolds = 5L,
  fold.id = NULL,
  loss = c("mse", "mae"),
  selection = c("min", "one.se"),
  nn.index = NULL,
  neighborhood.type = c("knn", "adaptive.radius", "supplied"),
  support.index = NULL,
  adaptive.k.scale = NULL,
  radius.rule = c("geomean", "max", "min"),
  radius.factor = 1.25,
  min.support = NULL,
  max.support = NULL,
  support.buffer = 2L,
  support.topup = c("nearest", "none"),
  tangent.dim.rule = c("fixed", "eigen.cumulative"),
  eigen.tolerance = 0.95,
```

```
  derivative.order = 2L,
  pinv.tol = sqrt(.Machine$double.eps),
  local.solver = c("auto", "normal.equations", "svd", "qr"),
  normal.equations.max.condition = 10000,
  solver = c("genlasso", "admm", "auto"),
  row.scaling = c("none", "l2"),
  admm.rho = NULL,
  admm.maxiter = 2000L,
  admm.abstol = 1e-06,
  admm.reltol = 1e-05,
  maxsteps = 2000L,
  minlam = 0,
  approx = FALSE,
  rtol = 1e-07,
  btol = 1e-07,
  eps = 1e-04,
  support.selection = c("rule", "cv"),
  support.grid = NULL,
  support.cv.max.candidates = 8L,
  return.local.diagnostics = FALSE,
  return.timing = FALSE,
  verbose = FALSE,
  admm.adaptive.rho = TRUE
)
```

Arguments

X	Numeric matrix with one observation per row. Distances for the default neighborhood search are Euclidean distances in these coordinates.
y	Numeric response vector of length nrow(X). Matrix responses are not yet supported for the ℓ_1 path.
k	Integer number of nearest neighbors per point, including the point itself, used when neighborhood.type = "knn". This follows the SSRHE Matlab convention. For adaptive-radius neighborhoods, k may be omitted when adaptive.k.scale is supplied.
tangent.dim	Integer tangent dimension used for the local PCA chart. Required when tangent.dim.rule = "fixed". If NULL and tangent.dim.rule = "eigen.cumulative", the dimension is selected locally from the PCA variance ratio.
lambda.grid	Optional nonnegative lambda grid. For lambda.selection = "fixed", this must contain exactly one value. For lambda.selection = "cv", NULL builds a default grid from the fitted full-data generalized-lasso path.
lambda.selection	"cv" for observed-label K-fold cross-validation or "fixed" for a supplied fixed lambda.
weights	Optional nonnegative observation weights. Missing responses automatically receive zero weight.

`n.lambda` Number of default lambda candidates when `lambda.grid = NULL` and `lambda.selection = "cv"`.

`nfolds` Number of validation folds over observed positive-weight labels. Ignored when `fold.id` is supplied.

`fold.id` Optional integer fold assignments of length `nrow(X)`.

`loss` Validation loss, currently `"mse"` or `"mae"`.

`selection` Selection rule. `"min"` chooses the smallest mean validation loss. `"one.se"` chooses the largest lambda within one standard error of the minimum.

`nn.index` Optional integer matrix of neighbor indices, with `nrow(X)` rows and k columns. Each row must contain its center vertex. If `NULL`, Euclidean k-NN including self is computed in C++.

`neighborhood.type`
Local support rule. `"knn"` uses the original rectangular self-including kNN neighborhoods. `"adaptive.radius"` builds variable-size supports from `dgraphs::create.rknn.graph` `"supplied"` uses `support.index` directly.

`support.index` Optional list of integer vectors, one per row of `X`. Each element gives a variable-size local support and must contain its center vertex.

`adaptive.k.scale`
Integer local-scale k used by `dgraphs::create.rknn.graph()` when `neighborhood.type = "adaptive.radius"`.

`radius.rule`, `radius.factor`
Adaptive-radius graph parameters passed to `dgraphs::create.rknn.graph()`.

`min.support` Optional minimum local support size for adaptive-radius supports. Defaults to the local quadratic design size plus `support.buffer`, clamped to `nrow(X)`.

`max.support` Optional maximum local support size. Oversized adaptive supports are trimmed to the closest `max.support` vertices while keeping the center vertex.

`support.buffer` Nonnegative integer added to the local quadratic design size when choosing the default `min.support`.

`support.topup` How undersized adaptive-radius supports are enlarged. `"nearest"` appends ambient nearest neighbors. `"none"` leaves supports unchanged and lets the C++ operator reject undersized supports.

`tangent.dim.rule`
Either `"fixed"` or `"eigen.cumulative"`.

`eigen.tolerance`
Cumulative local PCA variance threshold used when `tangent.dim.rule = "eigen.cumulative"` and `tangent.dim = NULL`.

`derivative.order`
Integer derivative order of the local SSRHE-style operator. `2L` is the original local Hessian energy. `3L` is an experimental third-derivative energy whose rows estimate unique symmetric third-derivative tensor components with factorial and tensor-multiplicity scaling.

`pinv.tol` Nonnegative tolerance multiplier for local pseudoinverses.

`local.solver` Local least-squares backend used to map local function values to derivative coefficients. `"auto"` is the default: it uses normal equations when the local design is full rank and has condition number no larger than `normal.equations.max.condition`, and otherwise falls back to SVD. `"normal.equations"` requests the normal-equation solve for full-rank local designs and falls back only on hard numerical failures. `"svd"` is the most stable reference path. `"qr"` uses pivoted QR and falls back to SVD for rank-deficient local designs.

`normal.equations.max.condition`
Positive condition-number guard used by `local.solver = "auto"` before accepting the normal-equation backend. The default is deliberately conservative because normal-equation solves square the effective condition number and order-3 near-minimum local supports can be numerically fragile.

`solver` Solver backend. `"genlasso"` uses the generalized-lasso path to propose a grid when needed, then refines every candidate with ADMM on the original objective. `"admm"` skips the path and requires an explicit grid. `"auto"` also proceeds when path construction fails, provided an explicit grid is supplied.

`row.scaling` Optional row scaling for the penalty matrix before solving. `"l2"` scales nonzero rows of A to unit Euclidean norm.

`admm.rho`, `admm.maxiter`, `admm.abstol`, `admm.reltol`
ADMM controls used by all backends, including refinement of generalized-lasso candidates. `admm.rho = NULL` chooses an initial algorithm penalty of `1 / max(1, mean(rowSums(D^2)))`, where `D` is the penalty operator after any requested row scaling. A positive numeric value sets the initial penalty explicitly. This is separate from statistical regularization `lambda.grid`. The iteration limit is a hard cap.

`maxsteps`, `minlam`, `approx`, `rtol`, `btol`, `eps`, `verbose`
Controls passed to **genlasso**.

`support.selection`
Support-profile selection rule. `"rule"` uses the supplied `adaptive.k.scale`, `min.support`, and `max.support`. `"cv"` is currently supported for `neighborhood.type = "adaptive.radius"` with `lambda.selection = "cv"` and chooses among rows of `support.grid` by an outer response cross-validation loop.

`support.grid` Optional data frame of support profiles with columns `adaptive.k.scale`, `min.support`, and optional `max.support`. If `NULL`, `ssrhe.support.grid` builds a compact default grid.

`support.cv.max.candidates`
Maximum number of support profiles to try when `support.selection = "cv"`.

`return.local.diagnostics`
Logical. If `TRUE`, compute additional R-side local chart diagnostics such as chart distortion and boundary asymmetry. These diagnostics are useful for operator audits, but can be skipped in fitting and cross-validation paths for speed.

`return.timing` Logical. If `TRUE`, attach a phase-level elapsed time table to the returned operator. The timings separate R-side validation, neighborhood/support construction, native local operator construction, optional local diagnostics, sparse matrix assembly, and output finalization. For adaptive-radius neighborhoods, subphase timings are also attached in `neighborhoods$timing`.

admm.adaptive.rho
: Logical; adapt the algorithm penalty during the first 1000 iterations by balancing tolerance-normalized primal and dual residuals. Set FALSE to hold the initial penalty fixed.

Details

This function exposes the SSRHE local polynomial derivative estimator as an ℓ_1 penalty, rather than as the original SSRHE ℓ_2 Hessian-energy penalty used by fit.ssrhe.hessian.regression. With derivative.order = 2L, A contains local quadratic/Hessian rows. With derivative.order = 3L, A contains local cubic third-derivative tensor rows with the same symmetric component scaling used by ssrhe.hessian.operator. The existing ℓ_2 fit solves a linear system involving $B = A^\top A$. This function instead keeps the rows of A and penalizes their absolute values. It is therefore closer in spirit to graph trend filtering, while retaining SSRHE's local-PCA derivative construction.

The default solver backend is **genlasso**. For larger or numerically fragile third-order operators, solver = "admm" fits the requested lambda values directly without computing a generalized-lasso path, and all backends use ADMM for the returned coefficients. solver = "auto" can additionally recover from path construction errors with an explicit grid. Cross-validation removes held-out labels from the data-fit term by setting their weights to zero, then scores predictions on those held-out labels.

ADMM uses proximal numerical stabilization, which does not add a ridge penalty to the requested objective. Convergence requires primal feasibility, the dual stopping residual, and stationarity of the original weighted L1 objective to meet their reported tolerances. A finite fixed-lambda iterate that reaches its cap is returned with a warning and solver$converged = FALSE; it is not a completed fit. Inspect solver$admm before using it, and consider a larger iteration limit or an adaptive penalty. Non-finite selected fits are errors.

CV candidates are eligible only when every fold has an acceptable solve. ADMM iterates that do not converge are not scored. The CV result retains fold.status, fold.converged, fold.diagnostics, n.successful.folds, and eligible. If no candidate qualifies, an error of class ssrhe_l1_cv_error retains fold diagnostics for inspection with tryCatch. A selected full-data ADMM fit that fails to converge is also an error in CV mode.

A generalized-lasso path alone can be inaccurate for nearly rank-deficient operators. It is retained in path for diagnostics, but its coefficients are not the returned fit. solver$backend = "genlasso_admm_refined" identifies this route; solver$path.objective records the unrefined objectives at the grid values. Backend warnings, including its numerical ridge for underdetermined designs, are retained in solver$warnings. CV uses the original objective without that additional ridge. The default ADMM tolerances are absolute 1e-6 and relative 1e-5; convergence describes numerical stopping at those tolerances, not predictive accuracy. With row.scaling = "l2", the reported objective uses the scaled penalty; energies$penalty.l1 is that penalty norm, while energies$hessian.l1 retains the unscaled operator norm.

With support.selection = "cv", each adaptive-radius support candidate builds a fresh SSRHE operator and runs the usual lambda CV with shared fold assignments. The selected fit is the candidate with the smallest selected CV error. This is useful when the default support-size rule is too rigid, but it can be much slower than tuning lambda for one fixed operator.

Value

A list of class `"ssrhe.hessian.l1.fit"` containing fitted values, residuals, selected lambda, the SSRHE operator, generalized-lasso path metadata, lambda-grid fitted values, and CV diagnostics when requested.

Choosing controls

First fit: specify `X`, `y`, `k`, `tangent.dim`, and `lambda.grid`. For a single supplied penalty set `lambda.selection = "fixed"` and `solver = "admm"` to avoid constructing a path.

Selection: `fold.id`, `nfolds`, `loss`, and `selection` control label CV. A grid requires one solve per penalty per fold; support CV also rebuilds geometry. Incomplete candidates are excluded, not silently scored.

Geometry: neighborhood, tangent and derivative controls define `A`. `row.scaling` changes the penalty itself and thus the statistical objective.

Numerics: `admm.*` controls stopping and algorithmic penalties for all backends. Path controls (`maxsteps`, `rtol`, `btol`, `eps`) affect the optional proposal path. They do not certify the final original-objective solve.

Diagnostics: `solver$admm`, candidate/fold status and objective components describe numerical acceptance. `path` and `beta.grid` can use substantial memory; use a compact explicit grid with ADMM when no full path is needed.

References

Kim, K. I., Steinke, F., and Hein, M. (2009). Semi-supervised regression using Hessian energy with an application to semi-supervised dimensionality reduction. *Advances in Neural Information Processing Systems 22*. https://papers.nips.cc/paper_files/paper/2009/hash/f4552671f8909587cf485ea990207f3b-Abstract.html

Examples

```
X <- matrix(seq(0, 1, length.out = 12), ncol = 1)
fit.ssrhe.hessian.l1.regression(
  X, sin(2 * pi * X[, 1]), k = 6L, tangent.dim = 1L,
  lambda.grid = 0.05, lambda.selection = "fixed", solver = "admm"
)
```

---

fit.ssrhe.hessian.regression

Fit SSRHE-Style Hessian-Energy Regression

Description

Fits a regression function on a point cloud by least squares with a quadratic (ℓ_2) penalty on the Hessian energy computed by `ssrhe.hessian.operator`, following the semi-supervised regression with Hessian energy (SSRHE) of Kim, Steinke, and Hein (2009). The penalties can be fixed or selected by cross-validation or GCV. This method differs from graph trend filtering (`fit.graph.trend.filtering`), which uses an ℓ_1 penalty on graph differences.

Usage

```
fit.ssrhe.hessian.regression(
  X,
  y,
  k = NULL,
  tangent.dim,
  lambda1,
  lambda2 = 0,
  weights = NULL,
  nn.index = NULL,
  neighborhood.type = c("knn", "adaptive.radius", "supplied"),
  support.index = NULL,
  adaptive.k.scale = NULL,
  radius.rule = c("geomean", "max", "min"),
  radius.factor = 1.25,
  min.support = NULL,
  max.support = NULL,
  support.buffer = 2L,
  support.topup = c("nearest", "none"),
  tangent.dim.rule = c("fixed", "eigen.cumulative"),
  eigen.tolerance = 0.95,
  derivative.order = 2L,
  stabilizer = lambda2 > 0,
  pinv.tol = sqrt(.Machine$double.eps),
  local.solver = c("auto", "normal.equations", "svd", "qr"),
  normal.equations.max.condition = 10000,
  ridge = 0,
  return.A = TRUE,
  return.local.diagnostics = FALSE,
  return.timing = FALSE,
  verbose = FALSE,
  lambda.selection = c("fixed", "cv", "gcv"),
  lambda1.grid = NULL,
  lambda2.grid = NULL,
  cv.control = list(),
  gcv.control = list()
)
```

Arguments

X	Numeric matrix with one observation per row. Distances for the default neighborhood search are Euclidean distances in these coordinates.
y	Numeric response vector of length nrow(X) or numeric matrix with nrow(X) rows. Matrix columns are fit as separate responses. NA values are allowed and are treated as unobserved by setting the corresponding observation weight to zero.
k	Integer number of nearest neighbors per point, including the point itself, used when neighborhood.type = "knn". This follows the SSRHE Matlab conven-

tion. For adaptive-radius neighborhoods, k may be omitted when adaptive.k.scale is supplied.

tangent.dim Integer tangent dimension used for the local PCA chart. Required when tangent.dim.rule = "fixed". If NULL and tangent.dim.rule = "eigen.cumulative", the dimension is selected locally from the PCA variance ratio.

lambda1 Nonnegative Hessian-energy penalty multiplier for $f^\top Bf = \|Af\|_2^2$.

lambda2 Nonnegative supplemental-stabilizer multiplier for $f^\top B_S f$. If positive, stabilizer must be TRUE. Currently supported only for derivative.order = 2L.

weights Optional nonnegative observation weights. May be NULL, a vector of length nrow(X), or a matrix with the same dimensions as y.

nn.index Optional integer matrix of neighbor indices, with nrow(X) rows and k columns. Each row must contain its center vertex. If NULL, Euclidean k-NN including self is computed in C++.

neighborhood.type
Local support rule. "knn" uses the original rectangular self-including kNN neighborhoods. "adaptive.radius" builds variable-size supports from dgraphs::create.rknn.graph "supplied" uses support.index directly.

support.index Optional list of integer vectors, one per row of X. Each element gives a variable-size local support and must contain its center vertex.

adaptive.k.scale
Integer local-scale k used by dgraphs::create.rknn.graph() when neighborhood.type = "adaptive.radius".

radius.rule, radius.factor
Adaptive-radius graph parameters passed to dgraphs::create.rknn.graph().

min.support Optional minimum local support size for adaptive-radius supports. Defaults to the local quadratic design size plus support.buffer, clamped to nrow(X).

max.support Optional maximum local support size. Oversized adaptive supports are trimmed to the closest max.support vertices while keeping the center vertex.

support.buffer Nonnegative integer added to the local quadratic design size when choosing the default min.support.

support.topup How undersized adaptive-radius supports are enlarged. "nearest" appends ambient nearest neighbors. "none" leaves supports unchanged and lets the C++ operator reject undersized supports.

tangent.dim.rule
Either "fixed" or "eigen.cumulative".

eigen.tolerance
Cumulative local PCA variance threshold used when tangent.dim.rule = "eigen.cumulative" and tangent.dim = NULL.

derivative.order
Integer derivative order of the local SSRHE-style operator. 2L is the original local Hessian energy. 3L is an experimental third-derivative energy whose rows estimate unique symmetric third-derivative tensor components with factorial and tensor-multiplicity scaling.

stabilizer Logical. If TRUE, also construct the supplemental stabilizer matrix described in the SSRHE supplement. Currently supported only for derivative.order = 2L.

`pinv.tol` Nonnegative tolerance multiplier for local pseudoinverses.

`local.solver` Local least-squares backend used to map local function values to derivative coefficients. `"auto"` is the default: it uses normal equations when the local design is full rank and has condition number no larger than `normal.equations.max.condition`, and otherwise falls back to SVD. `"normal.equations"` requests the normal-equation solve for full-rank local designs and falls back only on hard numerical failures. `"svd"` is the most stable reference path. `"qr"` uses pivoted QR and falls back to SVD for rank-deficient local designs.

`normal.equations.max.condition`
Positive condition-number guard used by `local.solver = "auto"` before accepting the normal-equation backend. The default is deliberately conservative because normal-equation solves square the effective condition number and order-3 near-minimum local supports can be numerically fragile.

`ridge` Nonnegative diagonal ridge added to the linear system for numerical stabilization.

`return.A` Logical. If `TRUE`, return A as a sparse matrix.

`return.local.diagnostics`
Logical. If `TRUE`, compute additional R-side local chart diagnostics such as chart distortion and boundary asymmetry. These diagnostics are useful for operator audits, but can be skipped in fitting and cross-validation paths for speed.

`return.timing` Logical. If `TRUE`, attach a phase-level elapsed time table to the returned operator. The timings separate R-side validation, neighborhood/support construction, native local operator construction, optional local diagnostics, sparse matrix assembly, and output finalization. For adaptive-radius neighborhoods, subphase timings are also attached in `neighborhoods$timing`.

`verbose` Logical. If `TRUE`, print a short native construction message.

`lambda.selection`
Penalty selection: `"fixed"` (the default) uses `lambda1` and `lambda2`; `"cv"` holds out observed labels; `"gcv"` uses generalized cross-validation. Grid inputs require an explicit `"cv"` or `"gcv"` selection mode.

`lambda1.grid, lambda2.grid`
Nonnegative penalty candidates for CV or GCV. `lambda1.grid` is required in either selection mode; `lambda2.grid = NULL` means `0`. Do not also supply scalar `lambda1` or `lambda2`. Unless explicitly set, `stabilizer` is enabled when any supplemental penalty candidate is positive.

`cv.control` Named list used only with `lambda.selection = "cv"`. Entries: `foldid = NULL` (one fold label per row of `X`; nonpositive or missing labels are ignored), `cv.folds = 5L`, `loss = "mse"` (or `"mae"`), and `selection = "min"` (or `"one.se"`, choosing the largest total penalty within one standard error of the minimum). Support-search entries are `support.selection = "rule"` (or `"cv"`), `support.grid = NULL`, and `support.max.candidates = 8L`. Fold count is ignored when `foldid` is supplied.

`gcv.control` Named list used only with `lambda.selection = "gcv"`. Entries: `trace.method = "exact"` (or `"hutchinson"`), `trace.n.probes = 50L`, `trace.seed = NULL`, `support.selection = "rule"` (or `"gcv"`), `support.grid = NULL`, and `support.max.candidates = 8L`. Unknown, unnamed, or duplicated control entries are errors. Controls for a different selection mode are rejected rather than ignored.

Details

For each response column, this function solves

$$(W + \lambda_1 B + \lambda_2 B_S + \epsilon I)\hat{f} = Wy,$$

where W is the diagonal matrix of observation weights and ϵ is `ridge`. Fully observed data with `lambda1 = lambda2 = ridge = 0` therefore reproduce the observed response exactly. Missing responses are excluded from the data-fit term by setting their weights to zero. The Matlab SSRHE semi-supervised convention is represented by a `0/1` labeled-indicator `weights` vector, or equivalently by setting unlabeled responses to `NA`: labeled vertices contribute unit diagonal data-fit terms and unlabeled vertices contribute no data-fit term.

The lower-level operator is matched to the Kim–Steinke–Hein SSRHE Matlab construction: self-including kNN neighborhoods, local PCA charts, a fixed-intercept local quadratic fit, doubled diagonal Hessian components, and optional supplemental stabilizer. The package exposes both A and $B = A^\top A$; the fitted ℓ_2 estimator uses B.

Value

A list of class `"ssrhe.hessian.fit"` containing fitted values, residuals, input response, weights, lambda parameters, objective/energy diagnostics, the reused `operator`, solver metadata, the call, and `lambda.selection`. CV adds class `"ssrhe.hessian.cv.fit"`, `cv.table`, `fold.id`, and `selection`; GCV adds class `"ssrhe.hessian.gcv.fit"`, `gcv.table`, `gcv.trace`, and `selection`. Support searches retain `selected.support` and the corresponding `support.cv.table` or `support.gcv.table`.

Penalty selection

Fixed fitting supports response vectors or matrices. CV and GCV currently select penalties for one response vector at a time. Label CV removes each validation fold from the data-fit term and scores its held-out observed positive-weight labels. It supports missing responses. GCV requires a fully observed response and strictly positive observation weights, and minimizes

$$\frac{n^{-1}\sum_i w_i(y_i - \hat{f}_i)^2}{(1 - \mathrm{tr}(S_\lambda)/n)^2},$$

where $S_\lambda = (W+\lambda_1 B+\lambda_2 B_S+\epsilon I)^{-1}W$. The exact trace is deterministic. The Hutchinson option uses Rademacher probes, reports trace standard errors, and is reproducible with `gcv.control = list(trace.method = "hutchinson", trace.seed = ...)`.

Both selection modes can also search adaptive-radius support profiles: set `support.selection` to the same mode in its control list. Each row of `support.grid` contains `adaptive.k.scale`, `min.support`, and optionally `max.support`. A `NULL` grid is constructed by `ssrhe.support.grid`. Each profile requires a new operator and a full penalty search, so this can substantially increase runtime.

Migration

The former `fit.ssrhe.hessian.regression.cv()` and `fit.ssrhe.hessian.regression.gcv()` entry points have been retired. Use this function with `lambda.selection = "cv"` or `"gcv"`. Move `fold.id` to `cv.control$foldid`, `nfolds` to `cv.control$cv.folds`, and `loss`/`selection` to `cv.control`. Move `gcv.trace.method`, `gcv.trace.n.probes`, and `gcv.trace.seed` to `trace.method`, `trace.n.probes`,

and `trace.seed` in `gcv.control`. Support search settings also belong in the active control list; both former support candidate limits are now named `support.max.candidates`.

Choosing controls

First fit: supply `X`, `y`, neighborhood size `k`, `tangent.dim`, and `lambda1`; use `lambda.selection = "fixed"` for a specified penalty.

Selection: `lambda.selection`, `lambda1.grid`, and `lambda2.grid` select penalties. Fold settings belong in `cv.control`; trace settings belong in `gcv.control`. Label CV and GCV are different criteria. Support selection rebuilds the operator for each candidate and multiplies this work.

Geometry: neighborhood, adaptive-radius, tangent and derivative controls determine the operator. Reuse it when fitting multiple responses.

Numerics: ridge and local conditioning controls stabilize solves. Exact GCV traces require more solves than stochastic trace estimates; the latter report sampling uncertainty and need a reproducible probe seed.

Diagnostics: inspect the selection table, energies, observed-label mask, and solver metadata. `fitted()` and `residuals()` preserve response matrices.

References

Kim, K. I., Steinke, F., and Hein, M. (2009). Semi-supervised regression using Hessian energy with an application to semi-supervised dimensionality reduction. *Advances in Neural Information Processing Systems 22*. https://papers.nips.cc/paper_files/paper/2009/hash/f4552671f8909587cf485ea990207f3b-Abstract.html

Examples

```
X <- matrix(seq(0, 1, length.out = 12), ncol = 1)
fit.ssrhe.hessian.regression(
  X, sin(2 * pi * X[, 1]), k = 6L, tangent.dim = 1L, lambda1 = 0.1
)
fit.ssrhe.hessian.regression(
  X, sin(2 * pi * X[, 1]), k = 6L, tangent.dim = 1L,
  lambda.selection = "cv", lambda1.grid = c(0.01, 0.1),
  cv.control = list(cv.folds = 2L)
)
fit.ssrhe.hessian.regression(
  X, sin(2 * pi * X[, 1]), k = 6L, tangent.dim = 1L,
  lambda.selection = "gcv", lambda1.grid = c(0.01, 0.1)
)
```

`fit.subject.od` *Fit Subject-Occupation Density*

Description

Estimates the occupation density of one subject: the distribution of the subject's visits over a fixed set of support points. The visits are given as indices of support points, with repeats, and are converted to counts. Methods that estimate a density directly are passed on to `fit.density`. The methods based on LPS and PS-LPS first fit the corresponding regression smoother to the counts and then convert the fitted values to probability mass with `normalize.density`.

Usage

```
fit.subject.od(
  X,
  subject.index,
  method = c("empirical", "graph_random_walk", "lps_count", "ps_lps_count",
    "lps_logistic_binary", "chart_kernel", "local_likelihood_density",
    "local_likelihood_bernoulli"),
  graph = NULL,
  graph.control = list(),
  od.control = list(),
  return.details = TRUE,
  od.cv = c("none", "visit"),
  visit.foldid = NULL,
  visit.cv.folds = 5L,
  visit.cv.seed = 1L,
  visit.cv.epsilon = 1e-15,
  ...
)
```

Arguments

`X`	Numeric matrix with one row per support point.
`subject.index`	Integer row indices of subject visits in `X`. Repeated indices are allowed.
`method`	Density method identifier.
`graph`	Optional precomputed graph object.
`graph.control`	Named list for random walks: `walk.steps` (or `walk.step`), `affinity.method`, `affinity.scale`, `affinity.epsilon`, and logical `normalize`. Underscore aliases are accepted, but conflicting aliases and unknown names are rejected. Must be empty for the empirical estimator.
`od.control`	OD-facing alias for `density.control`.
`return.details`	Logical; if `TRUE`, keep diagnostic details in the result.
`od.cv`	OD-level selection mode. `"none"` preserves the direct fit. `"visit"` holds out subject visits, fits candidates on the remaining visits, and selects the candidate minimizing held-out negative log occupation mass.
`visit.foldid`	Optional positive integer vector assigning subject visits to OD-level cross-validation folds. Its length must equal `length(subject.index)`.
`visit.cv.folds`	Number of visit folds when `visit.foldid` is not supplied.

visit.cv.seed Random seed for generated visit folds.

visit.cv.epsilon
Positive floor used in held-out -log(rho[visit]) scoring.

... Reserved for future methods; unused arguments are errors.

Details

The method name "lps_logistic_binary" is kept from earlier versions and is somewhat misleading. It calls fit.lps(..., outcome.family = "bernoulli"), which fits probabilities by local least squares and clips them to the unit interval, and then converts the result with normalize.density. It does not use the local logistic regression of outcome.family = "binomial".

When od.cv = "visit", graph random-walk candidates may pass OD-level grids inside graph.control: walk.step.grid, affinity.method.grid, affinity.scale.grid, affinity.epsilon.grid, and normalize.grid. Missing affinity.scale values mean the usual data-derived affinity scale.

When od.cv = "visit", chart-based and LPS-family methods may pass chart.dim.grid through .... For "chart_kernel", "local_likelihood_density", "local_likelihood_bernoulli", "lps_count", "lps_logistic_binary", and "ps_lps_count", this grid compares fixed integer chart dimensions with optional "auto" and "local.auto" chart-dimension policies under the same held-out-visit negative-log-occupation score. For LPS-family OD visit CV, each outer candidate is passed to the source smoother as a scalar local model configuration; this avoids nested row-level multi-candidate selection inside each held-out-visit fold.

Value

A list of class "density_fit". Its rho component is the estimated probability mass over the rows of X, and empirical.rho is the normalized visit-count mass. The subject component summarizes the number of visits, the number of distinct visited support points, the largest visit multiplicity, and the fraction of repeatedly visited support points. Other components describe the fitted method, normalization accounting, smoothing diagnostics, and warnings as documented in fit.density. With od.cv = "visit" and return.details = TRUE, the result also contains the visit-level cross-validation table, fold assignments, and held-out predicted masses.

Examples

```
X <- matrix(seq(0, 1, length.out = 6), ncol = 1)
fit.subject.od(X, subject.index = c(1L, 3L, 3L, 6L),
               method = "empirical")
```

---

get.region.boundary *Get Boundary Vertices of a Region in a Graph*

Description

Identifies the boundary vertices of a specified region in a graph, defined as vertices in the region that have at least one neighbor outside the region.

Usage

```
get.region.boundary(adj.list, region)
```

Arguments

adj.list	A list of integer vectors, where each vector contains the indices of vertices adjacent to the corresponding vertex. Indices must be 1-based.
region	An integer vector of vertex indices (1-based) defining the region for which to find boundary vertices.

Details

This function determines which vertices in a specified region are positioned at the boundary - meaning they have at least one neighbor that is not part of the region. These boundary vertices are often treated differently in graph-based algorithms, such as harmonic smoothing, where their values are typically fixed as constraints.

The boundary definition used matches the one implemented in the C++ harmonic smoothing functions, focusing on vertices that have connections to the outside of the region.

Value

An integer vector containing the indices of the boundary vertices, which is a subset of the input `region` vector. Returns an empty integer vector if no boundary vertices exist.

See Also

perform.harmonic.smoothing, harmonic.smoother

Examples

```
# Create a simple grid graph adjacency list (5x5 grid)
create_grid_adj_list <- function(n_rows, n_cols) {
  n <- n_rows * n_cols
  adj_list <- vector("list", n)
  for (i in 1:n_rows) {
    for (j in 1:n_cols) {
      v <- (i-1) * n_cols + j
      neighbors <- numeric(0)

      # Add neighbors (up, down, left, right)
      if (i > 1) neighbors <- c(neighbors, (i-2) * n_cols + j)  # up
      if (i < n_rows) neighbors <- c(neighbors, i * n_cols + j)  # down
      if (j > 1) neighbors <- c(neighbors, (i-1) * n_cols + (j-1))  # left
      if (j < n_cols) neighbors <- c(neighbors, (i-1) * n_cols + (j+1))  # right

      adj_list[[v]] <- neighbors
    }
  }
  return(adj_list)
}
```

```
# Create a 5x5 grid graph
grid_adj_list <- create_grid_adj_list(5, 5)

# Define a region (central 3x3 subgrid)
central_region <- c(7:9, 12:14, 17:19)

# Find boundary vertices of the central region
boundary <- get.region.boundary(grid_adj_list, central_region)
print(boundary)
# Expected output: c(7, 8, 9, 12, 14, 17, 18, 19)
# (all except the center vertex 13)
```

`graph.trend.filtering.operator`
Construct a Graph Trend-Filtering Operator

Description

Builds the weighted graph difference operators used by graph trend filtering. The default `operator.family = "graph.laplacian.recursive"` uses recursive graph operators

$$k = 0: \quad \Delta_w^{(1)} = D_w,$$

$$k = 1: \quad \Delta_w^{(2)} = L_w = D_w^\top D_w,$$

and

$$k = 2: \quad \Delta_w^{(3)} = D_w L_w.$$

The experimental `operator.family = "path.divided.difference"` builds canonical path rows and computes position-aware finite differences along graph paths using `weight.list` as metric edge lengths.

Usage

```
graph.trend.filtering.operator(
  adj.list,
  weight.list = NULL,
  order = 0L,
  weight.rule = c("conductance", "sqrt.conductance", "unit"),
  operator.family = c("graph.laplacian.recursive", "path.divided.difference"),
  path.family = c("branch.continuation", "all.simple"),
  path.weighting = c("unit", "anchor.mean"),
  return.sparse = TRUE
)
```

Arguments

adj.list	List of integer neighbor vectors using 1-based vertex indices. The graph must be undirected: every edge must appear in both endpoint adjacency lists.
weight.list	Optional list of positive edge weights parallel to adj.list. For operator.family = "graph.laplacian.recursive", weights are graph trend-filtering weights or conductances. For operator.family = "path.divided.difference", weights are metric edge lengths used to define path coordinates. If NULL, all values are one.
order	Integer trend-filtering order. Supported values are 0L, 1L, and 2L.
weight.rule	Character scalar. "conductance" uses weight.list directly as ω_e; "sqrt.conductance" uses $\sqrt{w_e}$; "unit" ignores weight.list. This applies to operator.family = "graph.laplacian.recursive".
operator.family	Character scalar selecting the penalty operator family. The default "graph.laplacian.recursive" preserves the existing graph trend-filtering semantics. "path.divided.difference" uses local pathwise finite differences and is designed to reproduce classical one-dimensional trend filtering on path graphs.
path.family	Character scalar used for operator.family = "path.divided.difference". "branch.continuation" keeps paths whose internal vertices do not pass through branch points. "all.simple" keeps all simple paths of the required length.
path.weighting	Character scalar used for operator.family = "path.divided.difference". "unit" assigns every canonical path row weight one. "anchor.mean" divides rows by the number of retained canonical paths sharing the same canonical start vertex.
return.sparse	Logical. If TRUE, attach **Matrix** sparse incidence, Laplacian, and penalty matrices.

Value

A list of class "graph.trend.filtering.operator" containing the canonical graph, edge table, trend-filtering weights, incidence, weighted Laplacian, selected penalty operator, and nullity estimate.

Examples

```
adj <- list(2L, c(1L, 3L), c(2L, 4L), 3L)
graph.trend.filtering.operator(adj, order = 1L, weight.rule = "unit")
```

---

harmonic.smoother *Perform Harmonic Smoothing with Topology Tracking*

Description

Relaxes graph values while recording changes in local extrema. Boundary vertices have neighbors outside the region and retain their original values.

Usage

```
harmonic.smoother(
  adj.list,
  weight.list,
  values,
  region.vertices,
  max.iterations = 100,
  tolerance = 1e-06,
  record.frequency = 1,
  stability.window = 3,
  stability.threshold = 0.05
)
```

Arguments

`adj.list`	A list of integer vectors, where each vector contains indices of vertices adjacent to the corresponding vertex. Indices must be 1-based.
`weight.list`	A list of numeric vectors containing weights of edges corresponding to adjacencies in `adj.list`.
`values`	A numeric vector of function values defined at each vertex.
`region.vertices`	An integer vector of vertex indices (1-based) defining the region to be smoothed. Boundary vertices will have fixed values.
`max.iterations`	Integer scalar, the maximum number of relaxation iterations to perform. Default is 100.
`tolerance`	Numeric scalar, the convergence threshold for value changes. Default is 1e-6.
`record.frequency`	Integer scalar, how often to record states (every N iterations). Default is 1 (record every iteration).
`stability.window`	Integer scalar, number of consecutive recorded differences to check for extrema stability. Default is 3.
`stability.threshold`	Numeric scalar in $[0, 1]$, maximum allowed difference in topology to consider stable. Default is 0.05.

Details

Uses the same inverse-length neighbor averages and boundary definition as `perform.harmonic.smoothing`. Unlike the basic function, it also relaxes regions with no boundary. Such a problem is not uniquely anchored; synchronous updates can oscillate on bipartite graphs, including paths.

The recorded diagnostic compares sets of (vertex, minimum/maximum) pairs. Its value is one minus twice the number of shared pairs divided by the sum of the two set sizes (zero for two empty sets). A detection means that the last `stability.window` recorded differences are at most `stability.threshold`. It does not establish numerical convergence, optimal smoothing, noise removal, or stability of full attraction basins. In particular, recording every two iterations can conceal an alternating oscillation. Always inspect `converged` and `max_residual` separately. Stability

detection does not stop relaxation or select the returned fit. The initial and final states are always recorded, including a final partial recording interval. Disconnected components without fixed boundary values have the same lack of anchoring as a region without a boundary.

Value

A list of class `"harmonic_smoother"` containing:

`harmonic_predictions`
: Numeric vector of smoothed function values

`i_harmonic_predictions`
: Matrix of function values at each recorded iteration (columns are iterations)

`i_basins`
: List of matrices representing extrema at each iteration

`stable_iteration`
: First iteration with a detected extrema-stability window, or `NA_integer_` if none was detected

`stability_detected`
: Whether an extrema-stability window was detected

`num_iterations`, `recorded_iterations`
: Number of relaxation updates and their actual indices for the recorded columns, starting at zero

`max_change`, `max_residual`
: Per-update maximum change and harmonic residual. Both must fall below `tolerance` for convergence

`status`
: `converged`, `iteration_limit`, or `no_interior`

`topology_differences`
: Numeric vector of differences between consecutive recorded iterations

`basin_cx_differences`
: Alias of `topology_differences`

`converged`
: Logical indicator of whether the relaxation converged

`num_region`, `num_boundary`, `num_interior`
: Vertex counts for the requested region and its boundary/interior split

See Also

`perform.harmonic.smoothing` for basic smoothing, `plot.harmonic_smoother` for visualization methods, `summary.harmonic_smoother` for summary statistics

Examples

```
adj.list <- list(2L, c(1L, 3L), c(2L, 4L), 3L)
weight.list <- list(1, c(1, 1), c(1, 1), 1)
values <- c(0, 2, -1, 1)
result <- harmonic.smoother(
  adj.list, weight.list, values, region.vertices = seq_along(values),
  max.iterations = 20, record.frequency = 2
)
smoothed.values <- result$harmonic_predictions
smoothed.values
```

`lpl.tf.operator` *Construct a Local Polynomial Lifting Trend-Filtering Operator*

Description

Builds the Local Polynomial Lifting Trend Filtering (LPL-TF) analysis operator from local polynomial prediction residuals. For each observed target point i, the operator predicts f_i from nearby values f_j using a local polynomial model and stores the residual row

$$r_i(f) = f_i - \sum_{j \in S_i} h_{ij} f_j.$$

Usage

```
lpl.tf.operator(
  X,
  adj.list = NULL,
  weight.list = NULL,
  graph = NULL,
  graph.stage = "final",
  anchor.index = NULL,
  anchor.coordinates = NULL,
  degree = 2L,
  support.type = c("adaptive.radius", "knn", "fixed.radius"),
  support.size = NULL,
  radius = NULL,
  min.support = NULL,
  support.buffer = 3L,
  kernel = c("epanechnikov", "triangular", "gaussian", "tricube"),
  coordinate.method = c("coordinates", "local.pca"),
  chart.dim = NULL,
  support.metric = c("auto", "coordinates", "graph.geodesic"),
  auto.chart.support.metric = c("coordinates", "operator", "both"),
  auto.chart.selection.metric = c("coordinates", "operator"),
  exclude.self = TRUE,
  row.normalize = c("l2", "none", "l1"),
  local.solver = c("auto", "normal.equations", "qr", "svd"),
  normal.equations.max.condition = 1e+08,
  duplicate.action = c("keep", "error"),
  drop.rank.deficient = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

`X`	Numeric coordinate matrix with one row per observation.

adj.list, weight.list
: Optional supplied undirected graph adjacency and positive edge-length lists using 1-based vertex indices.

graph
: Optional graph object returned by dgraphs::create.rknn.graph() or a list containing adj.list/weight.list or adj_list/weight_list.

graph.stage
: Graph stage to extract from graph: "final", "raw", or "pruned".

anchor.index
: Optional integer vector of observed anchor/target indices. The current implementation supports observed anchors only.

anchor.coordinates
: Reserved for later off-observation anchors; this implementation requires NULL.

degree
: Polynomial degree. The current implementation supports 0L, 1L, and 2L.

support.type
: Coordinate support rule. "knn" uses the nearest support.size candidates including the target before self exclusion. "adaptive.radius" uses the nearest min.support positive predictors after self exclusion. "fixed.radius" uses all points within radius.

support.size
: Positive integer for support.type = "knn".

radius
: Positive coordinate radius for support.type = "fixed.radius".

min.support
: Minimum number of positive-weight predictors after self exclusion. If NULL, uses choose(ncol(X) + degree, degree) + support.buffer.

support.buffer
: Nonnegative integer added to the local polynomial design size when choosing default supports.

kernel
: Kernel used to weight local predictors by distance.

coordinate.method
: Coordinate chart used for the local polynomial design. "coordinates" uses centered ambient coordinates; "local.pca" uses a deterministic local PCA chart on the selected support.

chart.dim
: Chart dimension. For "coordinates", this must be NULL or ncol(X). For "local.pca", NULL defaults to ncol(X). The special value "auto" estimates a single chart dimension from observed-coordinate local PCA spectra only, without using responses, truth values, latent coordinates, or labels.

support.metric
: Support-distance rule. "coordinates" uses Euclidean coordinate distances; "graph.geodesic" uses shortest-path distances in the supplied graph. "auto" uses graph geodesics when a graph is supplied and coordinate distances otherwise.

auto.chart.support.metric
: Support system used when chart.dim = "auto". "coordinates" uses Euclidean coordinate neighborhoods, "operator" uses the resolved operator support metric, and "both" computes both diagnostics side by side.

auto.chart.selection.metric
: Which auto chart-dimension diagnostic to use for the fitted operator when auto.chart.support.metric = "both". The default "coordinates" preserves historical behavior.

exclude.self
: Logical. LPL-TF residual rows require TRUE.

row.normalize
: Complete-row normalization rule applied after setting target coefficient +1 and predictor coefficients -h_ij.

`local.solver` Local linear algebra rule.

`normal.equations.max.condition`
Maximum condition number for `local.solver = "auto"` to use normal equations.

`duplicate.action`
Duplicate coordinate policy.

`drop.rank.deficient`
Logical. The current implementation requires `TRUE`.

`verbose` Logical. Reserved for diagnostic messages.

`...` Reserved.

Details

The operator uses observed anchors, excludes the target point from every predictor support, and drops requested-degree rank-deficient rows with explicit metadata. There is no silent degree downgrade. Graph support distances affect support selection and kernel bandwidths only; local polynomial coordinates are controlled separately by `coordinate.method`. Graph construction is supplied by **dgraphs** or by explicit `adj.list`/`weight.list` payloads. Coordinate support paths and graph-geodesic payload validation are package-local; shortest-path distances are computed through **dgraphs**.

If a_i is the complete assembled residual row, then `row.normalize = "l2"` uses $a_i/\|a_i\|_2$, and `row.normalize = "l1"` uses $a_i/\|a_i\|_1$. Normalization is applied to the whole row, not only to predictor coefficients.

Value

A list of class `"lpl_tf_operator"` containing the sparse operator matrix `A`, row metadata, supports, local design summaries, diagnostics, settings, and the matched call.

Examples

```
X <- matrix(seq(0, 1, length.out = 18), ncol = 1)
lpl.tf.operator(X, degree = 1L, support.type = "knn",
                support.size = 7L, kernel = "gaussian")
```

---

lps.backend.diagnostics

Report LPS Backend and Chart-Dimension Diagnostics

Description

Returns a one-row table that describes how a local polynomial smoother was fitted. The table gives the backend that was requested and the backend that was used, the rule requested for the chart dimension and the dimension that resulted, the selected tuning parameters, and whether the fit used the recommended settings for local PCA charts described below.

Usage

```
lps.backend.diagnostics(object)
```

Arguments

object	A fitted "lps" object.

Details

The choice of backend is conservative. backend = "auto" uses the C++ backend when the smoother works in the supplied coordinates and the R reference backend when coordinate.method = "local.pca". The C++ backend for local PCA, "cpp.local.pca", is used only when requested explicitly. This function reports which backend was used, so that it can be recorded with the results of an analysis.

For real-data local-PCA runs, the deployable chart-dimension contract is the ordinary local.chart.method = "pca" path with chart.dim = "auto" or "local.auto" and observed-covariate auto-dimension diagnostics. In P7-style experiments this is paired with auto.chart.support.metric = "both" and auto.chart.selection.metric = "operator". The experimental "second.order.svd" chart path is reported explicitly but is not certified by this deployable local-PCA contract. For LPS itself, which uses coordinate supports, the operator-support diagnostic is currently equivalent to the coordinate-support diagnostic; the fields are still recorded so the same manifest schema can be shared with LPL-TF and S-LPL-TF experiments.

Value

A one-row data.frame with backend, chart-dimension, selection, candidate-count, and policy fields.

Examples

```
X <- matrix(seq(0, 1, length.out = 16), ncol = 1)
fit <- fit.lps(X, X[, 1]^2, support.grid = 6L, degree.grid = 1L,
               kernel.grid = "gaussian", cv.folds = 2L, backend = "R")
lps.backend.diagnostics(fit)
```

---

lps.grouped.foldid *Build a Cluster-Respecting Fold Assignment*

Description

Assigns whole clusters to cross-validation folds so that no cluster is split across folds: every observation of a cluster receives the same fold id, and therefore no training set ever shares a cluster with the fold held out against it. Folds are built by a deterministic size-balanced greedy rule: clusters are taken largest first (ties by first appearance in cluster.id) and each is placed into the currently smallest fold (ties by lowest fold index). With v equal to the number of clusters this reduces to leave-cluster-out.

Usage

```
lps.grouped.foldid(cluster.id, v = 5L, shuffle.seed = NULL)
```

Arguments

cluster.id	Vector of cluster labels (factor, character, or integer-like), one per observation, no missing values.
v	Number of folds; an integer between 2 and the number of distinct clusters.
shuffle.seed	Optional integer seed for a randomized cluster order; NULL (default) keeps the deterministic order.

Details

The assignment is fully deterministic when shuffle.seed is NULL. Supplying shuffle.seed applies one seeded permutation to the cluster order before the greedy pass (the seed is consumed immediately via set.seed), giving a reproducible randomized variant; callers following the LPS evidence conventions should record the seed they pass.

Value

An integer fold-id vector of length length(cluster.id) with values in 1:v; every fold is nonempty and every cluster maps to exactly one fold.

See Also

lps.nested.cv() for nested cross-validation that can consume grouped folds at both the outer and inner level.

Examples

```
cluster <- rep(letters[1:6], each = 2)
lps.grouped.foldid(cluster, v = 3L)
```

---

lps.nested.cv *Nested Cross-Validation for LPS with Explicit, Recorded Folds*

Description

Runs outer-fold nested cross-validation around fit.lps(): for each outer fold, candidate selection is one ordinary fit.lps call on the inner-training rows only, with an explicit inner fold id and X.eval set to the held-out outer-test rows, so the held-out fold never participates in inner selection. The pooled outer-test error is the nested generalization estimate. The same call also computes the *selected-min* arm — an ordinary fit.lps on all rows using the *same* outer.foldid — so optimism comparisons are paired on one fold assignment by construction.

Usage

```
lps.nested.cv(
  X,
  y,
  outer.foldid,
  fit.args = list(),
  inner.folds = 5L,
  cluster.id = NULL,
  inner.foldid.method = c("round.robin", "grouped"),
  inner.shuffle.seed = NULL
)
```

Arguments

X	Numeric matrix of observations (rows) used for training.
y	Numeric response vector with length(y) == nrow(X).
outer.foldid	Positive integer vector of length nrow(X) assigning rows to outer folds (at least two distinct folds).
fit.args	Named list of additional arguments passed to every fit.lps call (grids, backend, design settings, ...). Must not contain X, y, foldid, or X.eval: the fold plumbing is owned by this function so both arms see the same folds.
inner.folds	Number of inner selection folds (integer >= 2).
cluster.id	Optional cluster labels (length nrow(X)), required for inner.foldid.method = "grouped".
inner.foldid.method	Inner fold construction: "round.robin" (default) or "grouped" (cluster-respecting).
inner.shuffle.seed	Optional integer; per-fold offset seed for the randomized variants described above. Recorded in the return value.

Details

The existing fit.lps behavior is untouched: this utility consumes the public API with explicit foldid only. It currently supports outcome.family = "gaussian" (the default) and scores with RMSE.

Inner folds are built per outer fold and fully recorded in the return value: "round.robin" assigns rep_len(1:inner.folds, n) over the inner-training rows in row order (deterministic; if inner.shuffle.seed is supplied, fold k — by position — uses set.seed(inner.shuffle.seed + k) and permutes the balanced assignment, giving a reproducible randomized variant); "grouped" builds cluster-respecting inner folds with lps.grouped.foldid() on the inner-training rows of cluster.id (passing the same per-fold offset seed when inner.shuffle.seed is supplied).

Value

A list of class "lps_nested_cv":

nested.rmse pooled RMSE of the outer-test predictions over all rows with finite predictions.

`n.missing.predictions` count of non-finite outer-test predictions (e.g. `unstable.action = "na"` fallbacks).

`folds` one row per outer fold: fold label, test size, selected `support.size` / `degree` / `kernel` / `bandwidth.multiplier`, the inner selected-min CV score, and the fold's outer-test RMSE.

`predictions` length-`nrow(X)` vector of outer-test predictions (each row predicted by the fold that held it out).

`selected.min` the paired selected-min arm on the same `outer.foldid`: `selected` (the `fit.lps` selected row), `cv.score` (its observed CV RMSE), `foldid`, and `fit` (the full-data `fit.lps` object, usable for deployment on an external test set).

`outer.foldid`, `test.index`, `train.index`, `inner.foldid`, `inner.foldid.used`, `inner.cv.table` complete fold/index telemetry: the realized inner fold ids as constructed and as recorded by each inner `fit.lps` (`$foldid`), per-fold index sets, and each fold's full inner CV candidate table.

`outer.cluster.whole` `TRUE`/`FALSE` whether every cluster lies wholly inside one outer fold (`NA` when `cluster.id` is absent).

`inner.folds`, `inner.foldid.method`, `inner.shuffle.seed`, `fit.args`, `call` the recorded configuration.

See Also

`lps.grouped.foldid()` for the grouped fold constructor.

Examples

```
X <- matrix(seq(0, 1, length.out = 18), ncol = 1)
y <- sin(2 * pi * X[, 1])
nested <- lps.nested.cv(
  X, y, outer.foldid = rep(1:3, length.out = 18), inner.folds = 2L,
  fit.args = list(support.grid = 6L, degree.grid = 1L,
                  kernel.grid = "gaussian", backend = "R")
)
nested$rmse
```

---

lps.pointwise.band *Pointwise Variance and Confidence Band for a Fixed-Configuration LPS Fit*

Description

Computes, from the analytically extracted linear-smoother matrix `S` of a fixed-configuration LPS fit (see `smoother.matrix()`), the pointwise variance `Var(fitted_i) = sigma^2 * sum_j S_ij^2`, the effective degrees of freedom `df = tr(S)`, the plug-in noise estimate `sigma.hat^2 = RSS / (n - 2*tr(S) + sum(S^2))`, and the pointwise confidence band `fitted_i +/- z * sigma * ||S_i.||_2` with `z = qnorm(1 - (1 - level) / 2)`.

Usage

```
lps.pointwise.band(
  object,
  sigma = NULL,
  level = 0.95,
  check.tol = 1e-10,
  variance.method = c("residual", "legacy")
)
```

Arguments

object	A fitted "lps" object from fit.lps() at a fixed configuration, with X.eval identical to X.
sigma	Known noise standard deviation (positive scalar), or NULL (default) to use the plug-in sigma.hat.
level	Confidence level of the band, a single number strictly between 0 and 1. Default 0.95. For independent errors with common variance and a fixed smoother, the expected residual sum of squares is sigma^2 * sum((I - S)^2) + ||(I - S) mu||^2. The default denominator is therefore the residual noise degrees of freedom, n - 2*tr(S) + sum(S^2), computed as sum((I - S)^2) for stability. This corrects the variance contribution; it cannot remove signal bias. Normal bands target the smoothed mean S mu. Their interpretation as bands for mu requires negligible smoothing bias. Plug-in bands are approximate, pointwise, and conditional on the specified configuration; they do not account for tuning on the same responses, dependent errors, or simultaneous coverage.
check.tol	Passed to smoother.matrix()'s self-guard.
variance.method	Noise-variance denominator: "residual" (default) uses sum((I - S)^2); "legacy" reproduces the former n - tr(S) rule. Neither removes smoothing bias. Added after check.tol to preserve positional calls. The choice affects sigma.hat even when a known sigma is supplied.

Details

With sigma supplied (known noise standard deviation), the band uses it directly (sigma.source = "known"); with sigma = NULL the plug-in sigma.hat is used (sigma.source = "plug.in"). sigma.hat is reported in both modes. The variance is purely the linear-smoother sampling variance: the band makes no bias correction, so where bias dominates (boundary, high curvature) undercoverage is expected and must be reported, not masked.

Requires X.eval identical to X (the square training-fit smoother, on which tr(S) and RSS are defined), in addition to all restrictions of smoother.matrix(). If any evaluation point's local fit returned NA, its variance/band entries are NA, and df, rss, and sigma.hat are NA as well; supplying a known sigma still yields bands at the unaffected points.

Value

A list of class "lps.pointwise.band" with named fields: fitted (the fit's raw fitted values), se, variance, lower, upper, level, z, sigma (the supplied known sigma, or NA in plug-in mode),

`sigma.hat`, `sigma.source` (`"known"` or `"plug.in"`), `df` (`tr(S)`), `residual.df`, `variance.method`, `rss`, `n.train`, `smoother.row.norm` (`||S_i.||_2`), and `configuration` (the pinned fit configuration).

See Also

`smoother.matrix()`

Examples

```
set.seed(1)
n <- 30
X <- matrix(runif(2 * n, -1, 1), ncol = 2)
y <- sin(pi * X[, 1]) + 0.1 * rnorm(n)
fit <- fit.lps(X, y, foldid = rep(1:2, length.out = n),
               support.grid = 12L, degree.grid = 1L,
               kernel.grid = "tricube", backend = "R",
               design.basis = "orthogonal.polynomial.drop",
               ridge.multiplier.grid = 0, ridge.condition.max = Inf,
               unstable.action = "na")
band.known <- lps.pointwise.band(fit, sigma = 0.1)
band.plugin <- lps.pointwise.band(fit)
band.known$df
band.plugin$sigma.hat
```

---

malps.gcv *Compute MALPS Linear-Smoother Diagnostics*

Description

Computes effective degrees of freedom, generalized cross-validation (GCV), and analytic leave-one-out residual diagnostics from a MALPS smoother matrix. These diagnostics are exact for fixed-support, fixed-weight linear MALPS fits. For cross-validated fits they are conditional on the selected support profile. Robust fits are rejected by default for the same reason described in `smoother.matrix`.

Usage

```
malps.gcv(
  object,
  y = NULL,
  smoother.matrix = NULL,
  include.loocv = TRUE,
  max.n = 1000L,
  allow.robust = FALSE,
  ...
)
```

Arguments

object	A "malps" object from fit.malps or refit.
y	Optional response vector. Defaults to object$y.
smoother.matrix	Optional precomputed matrix from smoother.matrix.
include.loocv	Logical; include analytic leave-one-out residuals and mean squared error.
max.n	Maximum dense smoother size passed to smoother.matrix when smoother.matrix = NULL.
allow.robust	Logical; passed to smoother.matrix.
...	Reserved for future extensions.

Details

The GCV score is computed as

$$\mathrm{GCV} = \frac{n^{-1}\|y - Sy\|_2^2}{(1 - \mathrm{tr}(S)/n)^2}.$$

The analytic leave-one-out residuals are computed as

$$e_i^{\mathrm{loo}} = \frac{y_i - \hat{y}_i}{1 - S_{ii}}.$$

Value

A list with residual, fitted-value, EDF, GCV, and optional LOOCV diagnostics.

Examples

```
X <- matrix(seq(0, 1, length.out = 20), ncol = 1)
fit <- fit.malps(X, X[, 1]^2, degree = 1L,
                 support.type = "knn", support.size = 8L)
malps.gcv(fit)$gcv
```

---

materialize.synthetic *Materialize a synthetic-data specification*

Description

Materialize a synthetic-data specification

Usage

```
materialize.synthetic(
  spec,
  n = NULL,
  seed,
  rng.policy = c("named.stream.v1", "legacy"),
  validate = TRUE
)
```

Arguments

spec	A synthetic_spec.
n	Sample size, or NULL for fixed-size sampling.
seed	Scalar whole-number seed.
rng.policy	Versioned RNG policy.
validate	Whether to validate the returned dataset.

Value

A synthetic_dataset.

Examples

```
spec <- synthetic.registry.spec("G1")
materialize.synthetic(spec, n = 20L, seed = 1L)
```

---

materialize.synthetic.instance

Materialize a frozen synthetic instance

Description

Materialize a frozen synthetic instance

Usage

```
materialize.synthetic.instance(instance.id, validate = TRUE)
```

Arguments

instance.id	Frozen instance ID.
validate	Whether to validate and checksum the result.

Value

A synthetic_dataset whose dataset.id is instance.id.

Examples

```
materialize.synthetic.instance("geosmooth.g1.default.v1")
```

`metric.graph.heat.eta.grid` *Construct a Graph Heat-Time Grid*

Description

Creates a positive heat-time grid from a `"metric.graph.lowpass.basis"` object.

Usage

```
metric.graph.heat.eta.grid(
  basis,
  rule = c("spectral_guarded", "w1_inverse_spectrum"),
  n.initial = 40L,
  include.zero = FALSE,
  truncation.tol = 1e-04,
  equilibrium.tol = 1e-04
)
```

Arguments

`basis`	A `"metric.graph.lowpass.basis"` object.
`rule`	Grid rule. `"w1_inverse_spectrum"` requires a complete basis and reproduces the W1 inverse-spectrum grid. `"spectral_guarded"` raises the lower endpoint for a truncated basis until the conservative omitted-mode attenuation bound is no larger than `truncation.tol`, and extends the upper endpoint until the slowest retained positive mode is attenuated to `equilibrium.tol`.
`n.initial`	Number of positive grid values.
`include.zero`	Logical. Include the exact no-smoothing endpoint.
`truncation.tol`	Positive tolerance smaller than one for the conservative omitted-mode attenuation bound.
`equilibrium.tol`	Positive tolerance smaller than one for the slowest positive retained mode at the upper endpoint.

Value

A numeric vector with grid-construction metadata stored as attributes. Exact null modes alone are excluded; uniformly rescaling conductances inversely rescales heat times. Unrepresentable endpoints cause an error.

Examples

```
adj <- list(2L, c(1L, 3L), c(2L, 4L), 3L)
lengths <- list(1, c(1, 1), c(1, 1), 1)
basis <- metric.graph.lowpass.basis(adj, lengths, n.eigenpairs = 4L,
                                    eigen.solver = "dense")
metric.graph.heat.eta.grid(basis, n.initial = 6L)
```

`metric.graph.heat.extend.lower` *Propose a Guarded Lower Graph Heat Time*

Description

Proposes one lower positive heat time after an external search controller has classified the current lower endpoint. The helper enforces a geometric expansion step, an identity-departure floor, a maximum number of expansion rounds, and the truncated-basis resolution certificate. It does not decide whether an endpoint is competitive; fold-level or cohort-level search logic remains the caller's responsibility.

Usage

```
metric.graph.heat.extend.lower(
  basis,
  eta.grid,
  endpoint.status = c("active", "inactive", "unresolved"),
  expansion.factor = 3,
  max.expansions = 3L,
  expansions.completed = 0L,
  identity.departure = 0.01,
  truncation.tol = 1e-04,
  unresolved.action = c("error", "mark")
)
```

Arguments

`basis`	A `"metric.graph.lowpass.basis"` object.
`eta.grid`	Current finite positive heat-time grid. The exact identity time `eta = 0` is intentionally excluded from competitive search.
`endpoint.status`	External lower-endpoint classification: `"active"` proposes one extension, while `"inactive"` and `"unresolved"` return the grid unchanged.
`expansion.factor`	Finite numeric factor greater than one. An active proposal is `min(eta.grid) / expansion.factor` before applying the identity floor.
`max.expansions`	Nonnegative integer maximum number of lower-extension rounds.
`expansions.completed`	Nonnegative integer number of extension rounds already admitted under the same search contract.
`identity.departure`	Positive number smaller than one. The proposed time cannot be smaller than `-log(1 - identity.departure) / lambda.max`, where `lambda.max` is the largest retained eigenvalue. For a complete basis this is the exact largest graph-Laplacian eigenvalue.

`truncation.tol` Positive tolerance smaller than one for the conservative omitted-mode attenuation bound.

`unresolved.action` Action when an active proposal is not certified by a truncated basis: `"error"` or `"mark"`.

Value

A list of class `"metric.graph.heat.lower.extension"` with the augmented or unchanged grid and proposal telemetry. A proposal is added only when `admitted` is `TRUE`.

Examples

```
adj <- list(2L, c(1L, 3L), c(2L, 4L), 3L)
lengths <- list(1, c(1, 1), c(1, 1), 1)
basis <- metric.graph.lowpass.basis(adj, lengths, n.eigenpairs = 4L,
                                    eigen.solver = "dense")
eta <- metric.graph.heat.eta.grid(basis, n.initial = 6L)
metric.graph.heat.extend.lower(basis, eta, endpoint.status = "active")
```

---

metric.graph.lowpass.basis

Construct a Reusable Metric Graph Low-Pass Spectral Basis

Description

Builds the weighted graph Laplacian and computes its eigenvalues and eigenvectors of lowest frequency, which graph low-pass filters use. The result does not depend on any response, so it can be reused to smooth many responses with many filter parameters.

Usage

```
metric.graph.lowpass.basis(
  adj.list,
  weight.list,
 conductance.rule = c("inverse.length.power", "exp.length", "exp.length.squared",
    "self.tuned.gaussian"),
  conductance.epsilon = 1e-08,
  conductance.alpha = 1,
  conductance.sigma = NULL,
  conductance.sigma.rule = c("edge.quantile", "median", "local.k"),
  conductance.sigma.quantile = 0.75,
  conductance.local.k = 5L,
  laplacian.type = c("unnormalized", "symmetric.normalized"),
  n.eigenpairs = 50L,
  eigen.solver = c("auto", "sparse", "dense"),
  dense.eigen.threshold = 200L,
  dense.fallback.threshold = 5000L,
```

```
  dense.fallback = c("auto", "never", "always"),
  verbose = FALSE,
  weight.type = c("length", "conductance"),
  spectrum = c("partial", "full"),
  numerical.policy = c("conservative", "verified.fallback")
)
```

Arguments

`adj.list`	List of integer neighbor vectors using 1-based vertex indices.
`weight.list`	List of weights parallel to `adj.list`: positive metric lengths or nonnegative conductances, according to `weight.type`.
`conductance.rule`	Character scalar. One of `"inverse.length.power"`, `"exp.length"`, `"exp.length.squared"`, or `"self.tuned.gaussian"`.
`conductance.epsilon`	Positive numeric regularizer used by inverse-power conductances and as a local-scale floor.
`conductance.alpha`	Positive numeric exponent for `"inverse.length.power"`.
`conductance.sigma`	Optional positive global scale for exponential rules. If `NULL`, it is selected by `conductance.sigma.rule`.
`conductance.sigma.rule`	Rule for selecting a global scale when needed.
`conductance.sigma.quantile`	Quantile used when `conductance.sigma.rule = "edge.quantile"`.
`conductance.local.k`	Positive integer local incident-edge order statistic for `"self.tuned.gaussian"`.
`laplacian.type`	Laplacian operator. `"unnormalized"` uses the weighted graph Laplacian $L = D - C$. `"symmetric.normalized"` uses $L_{\mathrm{sym}} = I - D^{-1/2} C D^{-1/2}$.
`n.eigenpairs`	Positive integer number of eigenpairs to compute.
`eigen.solver`	`"auto"`, `"sparse"`, or `"dense"`.
`dense.eigen.threshold`	Exact dense threshold for auto mode.
`dense.fallback.threshold`	Maximum graph size for emergency dense fallback when sparse decomposition fails and fallback is allowed.
`dense.fallback`	`"auto"`, `"never"`, or `"always"`.
`verbose`	Logical. Reserved for future diagnostic messages.
`weight.type`	Weight interpretation. `"length"` transforms lengths; `"conductance"` uses positive weights unchanged and removes zero edges. Reciprocal conductances must agree; self-loops must be removed by the caller. Conductance-transform settings are unused for direct conductances.

`spectrum` `"partial"` retains the requested number of modes. `"full"` requests and verifies all vertex eigenpairs with a dense solver, overriding `n.eigenpairs`; sparse-only requests are rejected.

`numerical.policy`
Numerical acceptance policy. The default `"conservative"` retains the heat-time and missing-block condition guards. Opt-in `"verified.fallback"` requires a full spectrum, unnormalized Laplacian, heat filter, and fixed tuning grid. It compares every requested positive time and response mask with a separate LAPACK eigensystem and a bordered pseudoinverse calculation. Failed residual or agreement checks stop the entire path. This double-precision agreement check is not a rigorous forward-error certificate. Structurally flat paths and overlapping score-resolution bands cannot select a time automatically; they remain available from the path function or at a prespecified time. Endpoint and near-minimum flags are retained in `tuning`. See `vignette("verified-heat-fallback")` for equations and limits.

Details

Numerical verification anchors the known component nullspace (square-root degree for the normalized operator), and rejects unresolved positive modes rather than clamping them to zero. Heat paths, fits and refits additionally check spectral error at the requested times. Missing-response blocks must pass a conditioning check as well as Cholesky and residual checks; failure returns an error without deleting positive edges or adding a ridge. These are conservative double-precision diagnostics, not rigorous interval error bounds. A complete basis alone does not certify arbitrary long times.

The basis is complete only when it contains one eigenvector per graph vertex. A truncated basis represents only the retained low-frequency subspace. The largest retained eigenvalue supplies a conservative proxy for bounding the contribution of omitted modes because all omitted eigenvalues are at least as large.

Value

A list of class `"metric.graph.lowpass.basis"` containing the graph operator, eigenvalues, eigenvectors, solver metadata, and spectral completeness diagnostics.

Examples

```
adj <- list(2L, c(1L, 3L), c(2L, 4L), 3L)
lengths <- list(1, c(1, 1), c(1, 1), 1)
metric.graph.lowpass.basis(adj, lengths, n.eigenpairs = 4L,
                           eigen.solver = "dense")
```

---

metric.graph.lowpass.operator

Construct a Metric-Conductance Graph Low-Pass Operator

Description

Builds the weighted graph Laplacian used by graph low-pass filtering. The edge weights of the Laplacian are conductances, which are either supplied directly or computed from edge lengths.

Usage

```
metric.graph.lowpass.operator(
  adj.list,
  weight.list,
 conductance.rule = c("inverse.length.power", "exp.length", "exp.length.squared",
    "self.tuned.gaussian"),
  conductance.epsilon = 1e-08,
  conductance.alpha = 1,
  conductance.sigma = NULL,
  conductance.sigma.rule = c("edge.quantile", "median", "local.k"),
  conductance.sigma.quantile = 0.75,
  conductance.local.k = 5L,
  laplacian.type = c("unnormalized", "symmetric.normalized"),
  return.sparse = TRUE,
  verbose = FALSE,
  weight.type = c("length", "conductance")
)
```

Arguments

`adj.list`	List of integer neighbor vectors using 1-based vertex indices.
`weight.list`	List of weights parallel to `adj.list`: positive metric lengths or nonnegative conductances, according to `weight.type`.
`conductance.rule`	Character scalar. One of `"inverse.length.power"`, `"exp.length"`, `"exp.length.squared"`, or `"self.tuned.gaussian"`.
`conductance.epsilon`	Positive numeric regularizer used by inverse-power conductances and as a local-scale floor.
`conductance.alpha`	Positive numeric exponent for `"inverse.length.power"`.
`conductance.sigma`	Optional positive global scale for exponential rules. If `NULL`, it is selected by `conductance.sigma.rule`.
`conductance.sigma.rule`	Rule for selecting a global scale when needed.
`conductance.sigma.quantile`	Quantile used when `conductance.sigma.rule = "edge.quantile"`.
`conductance.local.k`	Positive integer local incident-edge order statistic for `"self.tuned.gaussian"`.
`laplacian.type`	Laplacian operator. `"unnormalized"` uses the weighted graph Laplacian $L = D - C$. `"symmetric.normalized"` uses $L_{\mathrm{sym}} = I - D^{-1/2} C D^{-1/2}$.

`return.sparse`	Logical. If `TRUE`, attach a **Matrix** sparse Laplacian when **Matrix** is available.
`verbose`	Logical. Reserved for future diagnostic messages.
`weight.type`	Weight interpretation. `"length"` transforms lengths; `"conductance"` uses positive weights unchanged and removes zero edges. Reciprocal conductances must agree; self-loops must be removed by the caller. Conductance-transform settings are unused for direct conductances.

Details

The legacy rdgraph regression precomputed-graph path uses supplied `weight.list` values as edge lengths for neighborhood ordering. The spectral conductance in that smoother is overlap-density based:

$$c_e^\rho = 1/\max(\rho_1(e), 10^{-10}),$$

where $\rho_1(e)$ is computed by the Riemannian-complex density machinery.

This function instead constructs conductances directly from metric lengths:

$$c_{ij} = \phi(\ell_{ij}).$$

Supported transforms are

$$c_{ij} = (\ell_{ij} + \epsilon)^{-\alpha},$$

$$c_{ij} = \exp(-\ell_{ij}/\sigma),$$

$$c_{ij} = \exp(-\ell_{ij}^2/\sigma^2),$$

and

$$c_{ij} = \exp(-\ell_{ij}^2/(\sigma_i\sigma_j)).$$

For `laplacian.type = "symmetric.normalized"`, smoothing is performed in the Euclidean eigenbasis of L_{sym}. The null vector is proportional to $\sqrt{d}$, not the constant vector, so this mode does not preserve constant responses in the same way as the unnormalized Laplacian.

Value

A list of class `"metric.graph.lowpass.operator"` containing the edge table, conductances, degree vector, Laplacian triplets, summaries, and optionally a sparse Laplacian matrix.

Examples

```
adj <- list(2L, c(1L, 3L), c(2L, 4L), 3L)
lengths <- list(1, c(1, 1), c(1, 1), 1)
metric.graph.lowpass.operator(adj, lengths)
```

model-summary *Summarize a Geometric Smoother or Probability-Mass Fit*

Description

Summaries expose stored settings and diagnostics without recomputing geometry, fitting, or selection. Console output is bounded; the summary's structured fields retain selected controls, criterion tables, local diagnostic counts, solver information, and density accounting for closer inspection.

Usage

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

## S3 method for class 'lps'
summary(object, ...)

## S3 method for class 'malps'
summary(object, ...)

## S3 method for class 'ps_lps'
summary(object, ...)

## S3 method for class 'chart_kernel'
summary(object, ...)

## S3 method for class 'local_likelihood'
summary(object, ...)

## S3 method for class 'lpl_tf'
summary(object, ...)

## S3 method for class 'slpl_tf'
summary(object, ...)

## S3 method for class 'metric.graph.lowpass.fit'
summary(object, ...)

## S3 method for class 'metric.graph.lowpass.refit'
summary(object, ...)

## S3 method for class 'ssrhe.hessian.fit'
summary(object, ...)

## S3 method for class 'ssrhe.hessian.refit'
summary(object, ...)
```

```
## S3 method for class 'ssrhe.hessian.l1.fit'
summary(object, ...)

## S3 method for class 'graph.trend.filtering.fit'
summary(object, ...)

## S3 method for class 'pttf.trend.filtering.fit'
summary(object, ...)

## S3 method for class 'density_fit'
summary(object, ...)

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

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

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

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

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

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

Arguments

`...`	Reserved; unused arguments are rejected.
`object`, `x`	A fitted object, or a summary for its print method.

Details

A finite result is not a convergence certificate. Where the algorithm records no stopping certificate, convergence is explicitly unavailable. Local fallback counts and failed evaluations are reported separately. Selection criteria are conditional tuning scores; they are not an independent assessment of prediction error. GCV, CV MSE, RMSE, and likelihood losses have different interpretations.

Value

`summary()` returns a `summary.geosmooth_fit` list with `model`, `n.training`, `n.evaluation`, `n.responses`, `n.observed`, `nonfinite`, `status`, `convergence`, `prediction.domain`, `selected`, `criteria`, `diagnostics`, `solver`, `mass`, `accounting`, and `warnings` fields. Print methods return their input invisibly.

Examples

```
X <- matrix(seq(0, 1, length.out = 20), ncol = 1)
fit <- fit.chart.kernel(X, X[, 1]^2, support.size = 8L)
fit
summary(fit)
mass <- normalize.density(c(1, 2, 3))
mass
summary(mass)$accounting
```

`model-values` *Extract Stored Fitted Values and Response Residuals*

Description

Use `fitted(object)` for the stored response-scale fitted values, preserving vector or matrix shape, row order, names, and missing values. It does not recompute a fit or imply prediction at new points. For `harmonic.smoother()` results it returns `harmonic_predictions`. The basic `perform.harmonic.smoothing()` result remains an unclassed list, accessed through `$harmonic_predictions`; density probability masses remain `$rho`.

Usage

```
## S3 method for class 'lps'
fitted(object, ...)

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

## S3 method for class 'malps'
fitted(object, ...)

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

## S3 method for class 'ps_lps'
fitted(object, ...)

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

## S3 method for class 'chart_kernel'
fitted(object, ...)

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

## S3 method for class 'local_likelihood'
```

```
fitted(object, ...)

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

## S3 method for class 'lpl_tf'
fitted(object, ...)

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

## S3 method for class 'slpl_tf'
fitted(object, ...)

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

## S3 method for class 'metric.graph.lowpass.fit'
fitted(object, ...)

## S3 method for class 'metric.graph.lowpass.fit'
residuals(object, ...)

## S3 method for class 'metric.graph.lowpass.refit'
fitted(object, ...)

## S3 method for class 'metric.graph.lowpass.refit'
residuals(object, ...)

## S3 method for class 'ssrhe.hessian.fit'
fitted(object, ...)

## S3 method for class 'ssrhe.hessian.fit'
residuals(object, ...)

## S3 method for class 'ssrhe.hessian.refit'
fitted(object, ...)

## S3 method for class 'ssrhe.hessian.refit'
residuals(object, ...)

## S3 method for class 'ssrhe.hessian.l1.fit'
fitted(object, ...)

## S3 method for class 'ssrhe.hessian.l1.fit'
residuals(object, ...)

## S3 method for class 'graph.trend.filtering.fit'
```

```
fitted(object, ...)

## S3 method for class 'graph.trend.filtering.fit'
residuals(object, ...)

## S3 method for class 'pttf.trend.filtering.fit'
fitted(object, ...)

## S3 method for class 'pttf.trend.filtering.fit'
residuals(object, ...)

## S3 method for class 'harmonic_smoother'
fitted(object, ...)
```

Arguments

`object`	A supported fitted model; see Methods below.
`...`	Reserved; unused arguments are rejected.

Details

`residuals(object)` returns stored response residuals where available, including missing-label masks. Otherwise it computes `y - fitted.values` only when the evaluation coordinates equal the training coordinates. Residuals at different evaluation points are undefined and cause an error, even when training and evaluation have the same number of rows. Binomial residuals are on the response (probability) scale, not Pearson or deviance residuals. These accessors do not change which refit results are reusable.

Value

The stored vector or matrix, or response residuals of the same shape.

See Also

`refit`, `smoother.matrix`

Examples

```
X <- matrix(seq(0, 1, length.out = 20), ncol = 1)
model <- fit.malps(X, X[, 1]^2, degree = 1L,
                   support.type = "knn", support.size = 8L)
head(fitted(model))
head(residuals(model))
```

`normalize.density` *Normalize Fitted Values Into a Density*

Description

Converts a numeric vector, or the fitted values of a smoother, to a vector of probability masses over the evaluation points. Use it to obtain a density or an occupation density from the output of a regression smoother such as `fit.lps`, `fit.ps.lps`, or `fit.metric.graph.lowpass`.

Usage

```
normalize.density(x, ...)

## S3 method for class 'numeric'
normalize.density(
  x,
  X = NULL,
  density.control = list(),
  method.id = "normalized_numeric",
  keep.source.fit = FALSE,
  adj.list = NULL,
  empirical.rho = NULL,
  return.details = TRUE,
  ...
)

## Default S3 method:
normalize.density(
  x,
  X = NULL,
  density.control = list(),
  method.id = NULL,
  keep.source.fit = TRUE,
  adj.list = NULL,
  empirical.rho = NULL,
  return.details = TRUE,
  ...
)

## S3 method for class 'lps'
normalize.density(
  x,
  X = NULL,
  density.control = list(),
  method.id = NULL,
  keep.source.fit = TRUE,
  adj.list = NULL,
```

```
  empirical.rho = NULL,
  return.details = TRUE,
  ...
)

## S3 method for class 'ps_lps'
normalize.density(
  x,
  X = NULL,
  density.control = list(),
  method.id = NULL,
  keep.source.fit = TRUE,
  adj.list = NULL,
  empirical.rho = NULL,
  return.details = TRUE,
  ...
)

## S3 method for class 'metric.graph.lowpass.fit'
normalize.density(
  x,
  X = NULL,
  density.control = list(),
  method.id = NULL,
  keep.source.fit = TRUE,
  adj.list = NULL,
  empirical.rho = NULL,
  return.details = TRUE,
  ...
)

## S3 method for class 'metric.graph.lowpass.refit'
normalize.density(
  x,
  X = NULL,
  density.control = list(),
  method.id = NULL,
  keep.source.fit = TRUE,
  adj.list = NULL,
  empirical.rho = NULL,
  return.details = TRUE,
  ...
)
```

Arguments

x	Numeric vector or fitted object with a `fitted.values` field.
...	Additional arguments passed to methods.

X	Optional support/evaluation matrix. If omitted, methods use the fitted object's stored X.eval or X field when available; for a bare numeric vector, a one-dimensional index support is used.
density.control	List controlling clipping, normalization, and accounting checks. See fit.density.
method.id	Character method identifier recorded in the returned object.
keep.source.fit	Logical; if TRUE, retain the source fit in diagnostics for object methods.
adj.list	Optional adjacency list used to compute graph-local smoothness diagnostics for the normalized density.
empirical.rho	Optional empirical probability mass vector used for accounting diagnostics.
return.details	Logical; if TRUE, keep diagnostic details in the result.

Value

A list of class "density_fit".

Examples

```
normalize.density(c(0.5, -0.1, 0.6, 0), X = matrix(1:4, ncol = 1))
```

---

perform.harmonic.smoothing

Perform Harmonic Smoothing on Graph Function Values

Description

Applies harmonic smoothing to function values defined on vertices of a graph, preserving values at the boundary of a specified region while smoothly interpolating interior values. This function implements a discrete Laplace equation solution using weighted averaging.

Usage

```
perform.harmonic.smoothing(
  adj.list,
  weight.list,
  values,
  region.vertices,
  max.iterations = 100,
  tolerance = 1e-06
)
```

Arguments

adj.list	A list of integer vectors, where each vector contains indices of vertices adjacent to the corresponding vertex. Indices must be 1-based.
weight.list	A list of numeric vectors containing weights of edges corresponding to adjacencies in adj.list.
values	A numeric vector of function values defined at each vertex.
region.vertices	An integer vector of vertex indices (1-based) defining the region to be smoothed. Boundary vertices will have fixed values.
max.iterations	Integer scalar, the maximum number of relaxation iterations to perform. Default is 100.
tolerance	Numeric scalar, the convergence threshold for value changes. Default is 1e-6.

Details

Boundary vertices are exactly the region vertices with a neighbor outside the region. Degree-one vertices are not automatically boundary vertices. Boundary values and values outside the region remain fixed. Each synchronous relaxation step replaces interior values by neighbor averages with conductance $1/(w + 10^{-10})$, where w is the supplied edge length.

With no boundary, this function returns the input unchanged, with status "no_boundary" and converged = FALSE. With no interior there is nothing to solve: status is "no_interior" and convergence is true. On a disconnected region, components without a connection to fixed boundary values are not uniquely anchored; relaxation can oscillate. Inspect convergence before using the result as a harmonic solution. Unlike this basic function, harmonic.smoother() also runs relaxation when the whole region has no boundary, retaining its historical behavior.

Value

A list containing:

- harmonic_predictions: the numeric vector of smoothed values;
- converged: whether both the maximum update and discrete harmonic residual fell below tolerance;
- num_region, num_boundary, and num_interior: vertex counts for the requested region and its boundary/interior split;
- num_iterations: the number of relaxation iterations run;
- max_change and max_residual: per-iteration convergence diagnostics;
- status: converged, iteration_limit, no_boundary, or no_interior.

See Also

harmonic.smoother for smoothing with topology tracking, get.region.boundary for boundary vertex identification

Examples

```
adj.list <- list(2L, c(1L, 3L), c(2L, 4L), 3L)
weight.list <- list(1, c(1, 1), c(1, 1), 1)
values <- c(0, 2, -1, 1)
result <- perform.harmonic.smoothing(
  adj.list, weight.list, values, region.vertices = 1:3
)
result$harmonic_predictions
```

---

plot.harmonic_smoother

Plot Method for Harmonic Smoother Results

Description

Creates various plots to visualize the results of harmonic smoothing with topology tracking.

Usage

```
## S3 method for class 'harmonic_smoother'
plot(x, y = NULL, ..., type = c("topology", "extrema", "values"))
```

Arguments

x	An object of class "harmonic_smoother".
y	Ignored (included for S3 method consistency).
...	Additional graphical parameters passed to plot().
type	Character string specifying the type of plot. Options are: **"topology"** Evolution of topology differences (default) **"extrema"** Evolution of extrema counts **"values"** Original vs smoothed values

Value

Invisibly returns the input object.

See Also

harmonic.smoother

Examples

```
adj.list <- list(2L, c(1L, 3L), c(2L, 4L), 3L)
weight.list <- list(1, c(1, 1), c(1, 1), 1)
values <- c(0, 2, -1, 1)
result <- harmonic.smoother(
  adj.list, weight.list, values, region.vertices = seq_along(values),
  max.iterations = 20, record.frequency = 2
)
plot(result, type = "values")
```

`plot.synthetic_dataset` *Plot a canonical synthetic dataset*

Description

The default `view = "auto"` preserves the dimension-dependent display: one latent coordinate shows response and truth against that coordinate; otherwise the first two predictor coordinates show a geometry projection. Select `"geometry"` explicitly to see a curve's embedding, or `"response"` to see responses against the first latent coordinate (observation index when latent coordinates are absent). A two-coordinate projection can hide separation in higher ambient dimensions. Plotting does not draw random values.

Usage

```
## S3 method for class 'synthetic_dataset'
plot(
  x,
  color = c("response", "truth", "region"),
  ...,
  view = c("auto", "geometry", "response"),
  legend = TRUE,
  col = NULL
)
```

Arguments

`x`	A `synthetic_dataset`.
`color`	Color mapping for points: response, truth, or region. Region mapping requires stored region labels. Numeric keys display the actual range.
`...`	Additional arguments passed to `graphics::plot()`, including overrides for axis labels, point shape, or color.
`view`	`"auto"`, `"geometry"`, or `"response"`. Geometry requires at least two predictor coordinates. Named-only, after `...`, for compatibility.
`legend`	Logical; show point-color and response/truth keys. A user-supplied `col` suppresses the automatic color key.
`col`	Optional explicit point color, overriding the color mapping.

Value

x, invisibly.

Examples

```
x <- materialize.synthetic(synthetic.registry.spec("G1"), n = 20L, seed = 1L)
plot(x)
plot(x, view = "geometry", color = "truth")
```

`predict.gaussian.trend.filtering.fit`

Predict from Gaussian Trend Filtering

Description

Predict from Gaussian Trend Filtering

Usage

```
## S3 method for class 'gaussian.trend.filtering.fit'
predict(
  object,
  newdata = NULL,
  covariates = NULL,
  offset = NULL,
  type = c("response", "link"),
  lambda.index = NULL,
  allow.unconverged = FALSE,
  ...
)

## S3 method for class 'gaussian.trend.filtering.fit'
fitted(object, ...)

## S3 method for class 'gaussian.trend.filtering.fit'
residuals(object, ...)

## S3 method for class 'gaussian.trend.filtering.fit'
print(x, ...)
```

Arguments

`object`	A `gaussian.trend.filtering.fit` object.
`newdata`	Finite numeric predictor locations; default training locations.
`covariates`	Additional predictor matrix, with the training column order.
`offset`	Offset vector, required for new locations if used during fitting.

type	"response" or "link"; both return fitted means because the Gaussian model uses the identity link.
lambda.index	Path indices; defaults to the CV selection if available, otherwise all supplied penalties.
allow.unconverged	Explicitly allow predictions from failed fits.
...	Reserved for compatibility with generic methods.
x	A fitted Gaussian trend-filtering object.

Value

A numeric vector for one path index, otherwise a matrix. Attributes outside.range and lambda record boundary predictions and penalties.

Examples

```
x <- seq(0, 1, length.out = 20)
fit <- fit.gaussian.trend.filtering(x, 1+x-x^2, lambda = Inf)
predict(fit, c(0.25, 0.75))
head(fitted(fit))
head(residuals(fit))
print(fit)
```

---

predict.logistic.trend.filtering.fit

Predict from Logistic Trend Filtering

Description

Predict from Logistic Trend Filtering

Usage

```
## S3 method for class 'logistic.trend.filtering.fit'
predict(
  object,
  newdata = NULL,
  covariates = NULL,
  offset = NULL,
  type = c("response", "link"),
  lambda.index = NULL,
  allow.unconverged = FALSE,
  ...
)

## S3 method for class 'logistic.trend.filtering.fit'
fitted(object, ...)
```

```
## S3 method for class 'logistic.trend.filtering.fit'
residuals(object, ...)

## S3 method for class 'logistic.trend.filtering.fit'
print(x, ...)
```

Arguments

object	A logistic.trend.filtering.fit object.
newdata	Finite numeric predictor locations; default training locations.
covariates	Additional predictor matrix, with the training column order.
offset	Offset vector, required for new locations if used during fitting.
type	"response" for probabilities or "link" for logits.
lambda.index	Path indices; defaults to the CV selection if available, otherwise all supplied penalties.
allow.unconverged	Explicitly allow predictions from failed fits.
...	Reserved for compatibility with generic methods.
x	A fitted logistic trend-filtering object.

Value

A numeric vector for one path index, otherwise a matrix. Attributes outside.range and lambda identify boundary predictions and penalties.

Examples

```
x <- seq(0, 1, length.out = 24)
fit <- fit.logistic.trend.filtering(x, rep(c(0, 1), 12), lambda = Inf)
predict(fit, c(0.25, 0.75))
head(fitted(fit))
head(residuals(fit))
print(fit)
```

---

predict.lpl_tf *Predict From A Local Polynomial Lifting Trend Filter*

Description

Predict From A Local Polynomial Lifting Trend Filter

Usage

```
## S3 method for class 'lpl_tf'
predict(
  object,
  newdata = NULL,
  type = c("response"),
  allow.incomplete = FALSE,
  ...
)
```

Arguments

object	A "lpl_tf" object.
newdata	Must be NULL; prediction is currently available only at the training points.
type	Prediction type. The current implementation supports only "response".
allow.incomplete	Reserved.
...	Reserved.

Value

Fitted values at the training points.

Examples

```
if (requireNamespace("genlasso", quietly = TRUE)) {
  X <- matrix(seq(0, 1, length.out = 18), ncol = 1)
  fit <- fit.lpl.tf(X, sin(2 * pi * X[, 1]), degree = 1L,
                    support.type = "knn", support.size = 7L,
                    lambda = 0.1, lambda.selection = "fixed")
  predict(fit)
}
```

---

predict.lps *Predict from an LPS Fit*

Description

Predict from an LPS Fit

Usage

```
## S3 method for class 'lps'
predict(object, newdata = NULL, type = c("response", "raw"), ...)
```

Arguments

object	A fitted "lps" object.
newdata	Optional prediction matrix. Defaults to the fitted object's stored X.eval.
type	Prediction scale. "response" returns response-scale predictions; for outcome.family = "bernoulli" or "binomial" these are probabilities in [0,1]. "raw" returns the unmodified local least-squares predictions for "bernoulli" and the fitted probabilities for "binomial".
...	Unused.

Value

A numeric vector of predictions.

Examples

```
X <- matrix(seq(0, 1, length.out = 12), ncol = 1)
fit <- fit.lps(X, sin(2 * pi * X[, 1]), support.grid = 6L)
predict(fit)
```

---

predict.malps *Predict From A MALPS Fit*

Description

Predict From A MALPS Fit

Usage

```
## S3 method for class 'malps'
predict(
  object,
  newdata = NULL,
  type = c("response"),
  allow.incomplete = FALSE,
  ...
)
```

Arguments

object	A "malps" object from fit.malps.
newdata	Optional numeric coordinate matrix with the same number of columns as object$X. If NULL, returns stored training-point fitted values. New-point prediction is currently implemented for coordinate-support fits only; it fails informatively for objects fitted with support.metric = "graph.geodesic" because new targets need a graph-attachment/geodesic-distance contract.
type	Prediction type. Currently only "response".

`allow.incomplete`
: Logical; if `FALSE`, new-point prediction fails when any target has no positive averaging coverage. If `TRUE`, those targets receive `NA_real_`.

`...`
: Reserved for future extensions.

Value

Numeric vector of fitted values.

Examples

```
X <- matrix(seq(0, 1, length.out = 12), ncol = 1)
fit <- fit.malps(X, sin(2 * pi * X[, 1]), support.size = 6L)
predict(fit)
```

---

predict.neighbor.probability.fit

Predict Nearest-Person Binary Probabilities

Description

Predict Nearest-Person Binary Probabilities

Usage

```
## S3 method for class 'neighbor.probability.fit'
predict(
  object,
  newdata = NULL,
  distances = NULL,
  query.group = NULL,
  k = object$k,
  type = c("response", "diagnostics"),
  ...
)
```

Arguments

`object`
: A fitted `fit.neighbor.probability` object.

`newdata`
: Query coordinate matrix. With no query data, predict at the stored training rows while excluding each query's whole participant.

`distances`
: Query-by-training distance matrix for a precomputed fit. Columns must retain the original training record order.

`query.group`
: Optional participant IDs for query rows. Matching training participants are excluded. Omit only for genuinely new participants.

`k`
: One or more requested sizes; defaults to the fitted size.

type	Return a probability matrix or a list including matrices of neighbor counts, case counts, radii and boundary-tie diagnostics.
...	Reserved; additional arguments are rejected.

Value

A query-by-size matrix, or a diagnostic list with that matrix as probability. Unsupported estimates remain NA. boundary.ties counts distances within 1e-12 of the cutoff; ordering itself uses exact distance ties. Diagnoses count distinct people. With positive eps, boundary counts cover retrieved participants only (boundary.complete = FALSE). search.candidates gives the final number of candidate records, including visit refinement.

Examples

```
X <- matrix(0:5, ncol = 1)
f <- fit.neighbor.probability(X, c(1, 0, 1, 0, 0, 1), k = 2)
predict(f) # leave-participant-out, not resubstitution
predict(f, matrix(2.5, ncol = 1), type = "diagnostics")
```

---

predict.projection.trend.filtering.fit

Predict from Projection Trend Filtering

Description

Predict from Projection Trend Filtering

Usage

```
## S3 method for class 'projection.trend.filtering.fit'
predict(
  object,
  newdata = NULL,
  offset = NULL,
  type = c("response", "link", "terms"),
  allow.unconverged = FALSE,
  ...
)

## S3 method for class 'projection.trend.filtering.fit'
fitted(object, ...)

## S3 method for class 'projection.trend.filtering.fit'
residuals(object, ...)

## S3 method for class 'projection.trend.filtering.fit'
print(x, ...)
```

```
## S3 method for class 'projection.trend.filtering.fit'
summary(object, ...)

## S3 method for class 'projection.trend.filtering.cv'
predict(object, newdata = NULL, ...)
```

Arguments

object	A projection.trend.filtering.fit or a completed projection.trend.filtering.cv.
newdata	Predictor matrix or numeric data frame; defaults to training rows. Named columns are aligned to training names when both are supplied.
offset	Known offset, required for new rows if used in fitting.
type	"response", "link", or "terms". Terms exclude the common intercept and offset, which are returned as attributes.
allow.unconverged	Explicitly allow predictions from a failed fit.
...	Reserved for generic compatibility.
x	A fitted projection trend-filtering model.

Value

A vector, or a matrix for type="terms". The outside.range attribute is a row-by-component matrix identifying constant-boundary extension of each univariate curve.

Examples

```
X <- matrix(seq(0, 1, length.out = 12), ncol = 1)
y <- sin(3*X[,1])
fit <- fit.projection.trend.filtering(X, y, lambda=0.001,
    direction.search="fixed")
predict(fit, matrix(c(0.2, 0.8), ncol=1))
head(fitted(fit))
head(residuals(fit))
print(fit)
summary(fit)
cv <- cv.projection.trend.filtering(X, y, foldid=rep(1:3, 4),
    lambda.grid=Inf, nterms.grid=1, direction.search="fixed")
predict(cv, X[1:2, , drop=FALSE])
```

---

predict.slpl_tf *Predict From A Fixed-Operator Synchronized LPL-TF Model*

Description

Predict From A Fixed-Operator Synchronized LPL-TF Model

Usage

```
## S3 method for class 'slpl_tf'
predict(object, newdata = NULL, type = c("response"), ...)
```

Arguments

object	A "slpl_tf" fit.
newdata	Must be NULL; prediction is currently available only at the training points.
type	Prediction type. The current implementation supports only "response".
...	Reserved.

Value

Fitted values at the training points.

Examples

```
if (requireNamespace("genlasso", quietly = TRUE)) {
  X <- matrix(seq(0, 1, length.out = 18), ncol = 1)
  fit <- fit.slpl.tf(X, sin(2 * pi * X[, 1]), degree = 1L,
                     support.type = "knn", support.size = 7L,
                     lambda1 = 0.1, lambda2 = 0,
                     lambda.selection = "fixed")
  predict(fit)
}
```

---

print.graph.trend.filtering.fit
Print Geosmooth S3 Objects

Description

Print concise summaries for fitted geosmooth models and constructed operators.

Usage

```
## S3 method for class 'graph.trend.filtering.fit'
print(x, ...)

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

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

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

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

## S3 method for class 'metric.graph.lowpass.fit'
print(x, ...)

## S3 method for class 'metric.graph.lowpass.refit'
print(x, ...)

## S3 method for class 'pttf.trend.filtering.fit'
print(x, ...)

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

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

## S3 method for class 'ssrhe.hessian.operator'
print(x, ...)

## S3 method for class 'ssrhe.hessian.l1.fit'
print(x, ...)

## S3 method for class 'ssrhe.hessian.fit'
print(x, ...)

## S3 method for class 'ssrhe.hessian.refit'
print(x, ...)

## S3 method for class 'ssrhe.hessian.cv.fit'
print(x, ...)

## S3 method for class 'ssrhe.hessian.gcv.fit'
print(x, ...)

## S3 method for class 'transported.graph.hessian.operator'
print(x, ...)
```

Arguments

x	A geosmooth S3 object.
...	Additional arguments, currently ignored.

Value

Invisibly returns x.

Examples

```
X <- matrix(seq(0, 1, length.out = 12), ncol = 1)
fit <- fit.lps(X, sin(2 * pi * X[, 1]), support.grid = 6L)
print(fit)
```

`print.harmonic_smoother` *Print Method for Harmonic Smoother Results*

Description

Prints a concise summary of harmonic smoother results.

Usage

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

Arguments

`x`	An object of class `"harmonic_smoother"`.
`...`	Further arguments passed to or from other methods.

Value

Invisibly returns the input object.

See Also

`harmonic.smoother`, `summary.harmonic_smoother`

Examples

```
fit <- structure(
  list(
    stable_iteration = 2L,
    i_harmonic_predictions = matrix(1:4, nrow = 2),
    i_basins = list(matrix(c(1, 1), ncol = 2)),
    topology_differences = numeric()
  ),
  class = "harmonic_smoother"
)
print(fit)
```

print.summary.harmonic_smoother *Print Method for Summary of Harmonic Smoother Results*

Description

Prints the summary of harmonic smoother results in a formatted manner.

Usage

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

Arguments

x	An object of class "summary.harmonic_smoother".
...	Further arguments passed to or from other methods.

Value

Invisibly returns the input object.

See Also

summary.harmonic_smoother

Examples

```
object <- structure(
  list(
    stable_iteration = 2L,
    iterations_recorded = 1L,
    initial_extrema = 1L,
    initial_maxima = 1L,
    initial_minima = 0L,
    final_extrema = 1L,
    final_maxima = 1L,
    final_minima = 0L,
    extrema_reduction = 0L,
    topology_diff_summary = NULL
  ),
  class = "summary.harmonic_smoother"
)
print(object)
```

print.synthetic_dataset *Print a canonical synthetic dataset*

Description

Print a canonical synthetic dataset

Usage

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

Arguments

x	A synthetic_dataset.
...	Ignored.

Value

x, invisibly.

Examples

```
x <- materialize.synthetic(synthetic.registry.spec("G1"), n = 20L, seed = 1L)
print(x)
```

pttf.geometry *Construct PTTF Local Geometry*

Description

Builds the geometry layer for the experimental parallel-transport trend filtering (PTTF) route. The returned object contains graph supports, local PCA tangent frames, edge directions, orthogonal edge transports, sampling density metadata, and diagnostics. It does not assemble transported difference operators and it does not fit a response.

Usage

```
pttf.geometry(
  X,
  adj.list = NULL,
  weight.list = NULL,
  graph = c("rknn", "supplied"),
  tangent.dim,
  local.support = c("graph.disk", "neighbors"),
```

```
  min.support = NULL,
  max.hops = 3L,
  support.k = NULL,
  graph.k.scale = 1L,
  graph.radius.factor = 1,
  graph.radius.rule = "geomean",
  transport.rule = c("procrustes"),
  synchronize.orientation = TRUE,
  density.method = c("support.radius"),
  alpha = 1,
  diagnostics = TRUE
)
```

Arguments

`X`	Numeric data matrix with one row per vertex.
`adj.list`	Optional supplied graph adjacency list using 1-based vertex indices.
`weight.list`	Optional positive edge-length list parallel to `adj.list`.
`graph`	Graph source. `"rknn"` builds an adaptive-radius graph from `X`; `"supplied"` uses `adj.list` and `weight.list`.
`tangent.dim`	Fixed global tangent dimension for local geometry.
`local.support`	Local support construction rule. `"graph.disk"` uses graph-hop disks; `"neighbors"` starts from the closed one-hop neighborhood and tops up by graph-hop expansion.
`min.support`	Minimum support size. Defaults to $\max(2m+2, m+3)$, where $m =$ `tangent.dim`.
`max.hops`	Maximum hop radius used when topping up local supports.
`support.k`	Optional cap on local support size after top-up. The center vertex is always retained and remaining vertices are selected by graph metric distance, with vertex-index tie breaking.
`graph.k.scale`, `graph.radius.factor`, `graph.radius.rule`	Adaptive-radius graph controls passed to `dgraphs::create.rknn.graph()` when `graph = "rknn"`.
`transport.rule`	Edge transport rule. The current implementation uses `"procrustes"`.
`synchronize.orientation`	Logical. If `TRUE`, return deterministic spanning-tree orientation-synchronization metadata. This metadata does not mutate the returned frames.
`density.method`	Sampling-density metadata rule. The current implementation uses `"support.radius"`.
`alpha`	Numeric exponent used only to report proposed future density-normalization weights.
`diagnostics`	Logical. If `TRUE`, compute cycle diagnostics.

Details

The stored transport convention is column-vector based. For an oriented edge $i \leftarrow j$, `Oij` maps tangent coordinates from frame j into frame i:

$$v_j \mapsto O_{ij} v_j.$$

For shared-support Procrustes, row-coordinate matrices Z_i and Z_j are formed on the same vertex set. The row problem $\min_Q \|Z_j Q - Z_i\|_F^2$ gives $Q = UV^\top$ when $Z_j^\top Z_i = U\Sigma V^\top$. The stored column map is $O_{ij} = Q^\top = VU^\top$; the reported residual is $\|Z_j O_{ij}^\top - Z_i\|_F / \max(\|Z_i\|_F, \epsilon)$.

Value

A list of class `"pttf_geometry"`.

Examples

```
n <- 10L
X <- matrix(seq(0, 1, length.out = n), ncol = 1)
adj <- lapply(seq_len(n), function(i) intersect(c(i - 1L, i + 1L), 1:n))
lengths <- Map(function(i, j) abs(X[j, 1] - X[i, 1]), seq_len(n), adj)
pttf.geometry(X, adj, lengths, graph = "supplied", tangent.dim = 1L)
```

---

pttf.operator *Construct PTTF Transported-Difference Operators*

Description

Assembles experimental parallel-transport trend-filtering (PTTF) transported-difference operators from a `pttf.geometry` object. This is an operator-construction function only: it does not fit a response, tune a penalty parameter, or run an ℓ_1/ℓ_2 smoother.

Usage

```
pttf.operator(
  geometry,
  derivative.order = 3L,
  edge.status.policy = c("ok.only", "frame.fallback"),
 regression.weight.rule = c("inverse.length.squared", "inverse.length", "unit"),
  edge.length.epsilon = 1e-08,
  tensor.scaling = c("hs", "raw"),
  row.mass.rule = c("node.mass", "none"),
  row.normalize = c("none", "l2"),
  min.operator.rank.tol = 1e-10,
  max.operator.condition = 1e+08,
  return.intermediate = TRUE,
  return.A = TRUE,
  return.B = TRUE,
  diagnostics = TRUE
)
```

Arguments

`geometry`	A `"pttf_geometry"` object from `pttf.geometry`.
`derivative.order`	Integer derivative order, one of `1L`, `2L`, or `3L`.
`edge.status.policy`	Edge transport policy. `"ok.only"` uses only transports with status `"ok"`. `"frame.fallback"` also allows direct-frame fallback transports for non-OK shared-support statuses.
`regression.weight.rule`	Local regression edge-weight rule.
`edge.length.epsilon`	Positive numeric edge-length floor.
`tensor.scaling`	Symmetric tensor coordinate scaling. `"hs"` uses square-root multiplicity scaling so coordinate Euclidean norms match Hilbert–Schmidt norms. `"raw"` stores unscaled unique symmetric components.
`row.mass.rule`	Final row-mass multiplier. `"node.mass"` multiplies rows for vertex i by $\sqrt{\mu_i}$ from the density metadata. `"none"` leaves rows unweighted.
`row.normalize`	Optional final row normalization.
`min.operator.rank.tol`	Relative singular-value tolerance for local derivative regression rank.
`max.operator.condition`	Maximum allowed weighted local-design condition number.
`return.intermediate`	Logical. If `TRUE`, include full logical intermediate derivative matrices.
`return.A`	Logical. If `TRUE`, include the compact sparse row operator.
`return.B`	Logical. If `TRUE`, include $A^\top A$.
`diagnostics`	Logical. If `TRUE`, include assembly diagnostics.

Details

The operator keeps a full logical vertex/component layout internally. For order r, the logical field has one block of $q_r = \binom{m+r-1}{r}$ components at each vertex. The returned `A` is compact: it includes rows only for accepted final-order vertex/component blocks, and `row.table$full.row` records the original full logical row.

For derivative level r, the local regression uses blocks $\Phi_r(u_{ij}) \in \mathbb{R}^{q_{r-1} \times q_r}$, lifted weights $W_i^{(r)} = \mathrm{diag}(w_{ij}) \otimes I_{q_{r-1}}$, and transported predecessor differences $K_{r-1}(O_{ij})z_j - z_i$. Unavailable predecessor blocks are never filled with zero; affected neighbors are dropped and recorded.

Value

A list of class `"pttf_operator"` containing sparse operators, row/vertex provenance, tensor-basis metadata, and diagnostics.

Examples

```
n <- 10L
X <- matrix(seq(0, 1, length.out = n), ncol = 1)
adj <- lapply(seq_len(n), function(i) intersect(c(i - 1L, i + 1L), 1:n))
lengths <- Map(function(i, j) abs(X[j, 1] - X[i, 1]), seq_len(n), adj)
geometry <- pttf.geometry(
  X, adj, lengths, graph = "supplied", tangent.dim = 1L
)
pttf.operator(geometry, derivative.order = 2L)
```

---

pttf.operator.filter.rows

Filter Rows Of A PTTF Operator

Description

Creates a new "pttf_operator" object with a subset of compact operator rows. The sparse triplet payload is rebuilt from the filtered matrix so triplet row indices always refer to the filtered fit rows, while original compact and full logical row identities remain in row.table.

Usage

```
pttf.operator.filter.rows(
  operator,
  rows,
  reason = NULL,
  preserve.original = TRUE
)
```

Arguments

operator	A "pttf_operator" object from pttf.operator.
rows	Integer, logical, or character row selector. Character selectors are matched against row.table$status.
reason	Optional character reason stored in row.filter.
preserve.original	Logical. If TRUE, preserve original row identities in row.table$compact.row and row.table$full.row.

Value

A filtered "pttf_operator" object.

Examples

```
n <- 10L
X <- matrix(seq(0, 1, length.out = n), ncol = 1)
adj <- lapply(seq_len(n), function(i) intersect(c(i - 1L, i + 1L), 1:n))
lengths <- Map(function(i, j) abs(X[j, 1] - X[i, 1]), seq_len(n), adj)
geometry <- pttf.geometry(
  X, adj, lengths, graph = "supplied", tangent.dim = 1L
)
operator <- pttf.operator(geometry, derivative.order = 2L)
pttf.operator.filter.rows(operator, seq_len(max(1L, nrow(operator$A) - 2L)))
```

`refit` *Refit A Model Using Its Stored Geometry*

Description

Fit a new response while reusing the geometry or operators in a supported fitted object. R selects the method from the class of `object`.

Usage

```
refit(object, y, ...)

## Default S3 method:
refit(object, y, ...)
```

Arguments

`object`	A fitted MALPS, local polynomial lifting trend filter, synchronized lifting trend filter, metric graph low-pass, quadratic Hessian-energy, or Hessian L1 model.
`y`	New response. MALPS and the two lifting trend filters accept a vector; Hessian L1 accepts one response; metric graph low-pass and quadratic Hessian methods also accept a matrix with one response per column. MALPS and both Hessian methods reuse the stored response when `y` is omitted or `NULL`. The other methods require `y`.
`...`	Method-specific controls, described in the linked method help. Unrecognized arguments are errors.

Details

Refitting does not automatically repeat the original model selection:

`malps` `refit.malps` reuses supports and averaging weights; accepts observation `weights`. Robust local residual weights, if used, are recomputed for the new response.

`lpl_tf` `refit.lpl_tf` reuses the residual operator and penalty, or accepts a new fixed `lambda`.

`slpl_tf` `refit.slpl_tf` reuses both operators and penalties, or accepts fixed `lambda1` and `lambda2`.

`metric.graph.lowpass.fit` `refit.metric.graph.lowpass.fit` reuses the spectral basis and smoothing parameter. Set `per.column.gcv = TRUE` to select a parameter for each response column.

`ssrhe.hessian.fit` `refit.ssrhe.hessian.fit` reuses the Hessian operator and fixed penalties, with optional `lambda1`, `lambda2`, `weights`, and `ridge`. This also handles CV- and GCV-selected fits, retaining their selected penalties by default.

`ssrhe.hessian.l1.fit` `refit.ssrhe.hessian.l1.fit` reuses the derivative operator with fixed or CV-selected penalties; accepts `lambda.grid`, `lambda.selection`, and solver controls.

Use the original fitted object for repeated metric graph low-pass or quadratic Hessian refits: their refit results are summaries, not reusable input fits. Other supported families return reusable fitted objects.

Value

A refitted object or refit summary, as documented by the method.

Migration

The six former exported `refit.*()` functions are replaced by `refit(object, y, ...)`. Replace named `fitted.model` and `y.new` arguments with `object` and `y`, respectively. Method-specific controls and return classes are preserved. Call the generic rather than an individual S3 method; the former help names remain available for navigation.

See Also

`smoother.matrix`

Examples

```
X <- matrix(seq(0, 1, length.out = 20), ncol = 1)
fit <- fit.malps(X, X[, 1]^2, degree = 1L,
                 support.type = "knn", support.size = 8L)
updated <- refit(fit, y = X[, 1]^3)
head(predict(updated))
```

`refit.lpl_tf` *Refit A Local Polynomial Lifting Trend Filter*

Description

Reuses the fixed operator and selected lambda from an existing LPL-TF fit.

Usage

```
## S3 method for class 'lpl_tf'
refit(object, y, lambda = NULL, reuse.lambda = TRUE, verbose = FALSE, ...)
```

Arguments

`object`	A `"lpl_tf"` object.
`y`	New numeric response vector.
`lambda`	Optional fixed lambda.
`reuse.lambda`	Logical. If `TRUE` and `lambda = NULL`, reuse `object$lambda`.
`verbose`	Logical.
`...`	Reserved.

Value

A refitted `"lpl_tf"` object.

See Also

`refit` for supported fitted objects and migration.

Examples

```
if (requireNamespace("genlasso", quietly = TRUE)) {
  X <- matrix(seq(0, 1, length.out = 18), ncol = 1)
  fit <- fit.lpl.tf(X, X[, 1]^2, degree = 1L,
                    support.type = "knn", support.size = 7L,
                    lambda = 0.1, lambda.selection = "fixed")
  refit(fit, y = X[, 1]^3)
}
```

`refit.malps` *Refit A MALPS Smoother With A New Response*

Description

Reuses the anchors, selected support profile, local supports, prediction supports, coordinate method, kernel, degree, and local-solver controls from a previous `fit.malps` result, then recomputes local polynomial coefficients for a new response vector. Observation weights, when supplied, multiply the stored geometric fitting weights for local coefficient estimation. Supports and model-averaging weights remain fixed. A weighted refit requires every stored local support to retain at least one positive effective fitting weight; sparse bootstrap-style weights can therefore fail even when the global weight vector is not all zero. If the original fit used robust local residual reweighting, robust weights are recomputed for the new response because they are response-dependent.

Usage

```
## S3 method for class 'malps'
refit(
  object,
  y = NULL,
  weights = NULL,
  reuse.selection = TRUE,
  refit.local.coefficients = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

object	A "malps" object from fit.malps.
y	Optional new numeric response vector with length nrow(object$X). If NULL, the original response is reused.
weights	Optional nonnegative numeric observation weights with length nrow(object$X). Zero weights remove observations from local coefficient estimation but do not change the stored supports or averaging weights. The refit fails if any local support has no positive effective fitting weight after multiplying by these observation weights.
reuse.selection	Currently requires TRUE.
refit.local.coefficients	Currently requires TRUE.
verbose	Logical; reserved for future progress messages.
...	Reserved for future extensions.

Value

A "malps" object with the same support profile and new fitted values.

See Also

refit for supported fitted objects and migration.

Examples

```
X <- matrix(seq(0, 1, length.out = 20), ncol = 1)
fit <- fit.malps(X, X[, 1]^2, degree = 1L,
                 support.type = "knn", support.size = 8L)
refit(fit, y = X[, 1]^3)
```

refit.metric.graph.lowpass.fit
Refit Metric-Conductance Graph Low-Pass Regression

Description

Reuses a fitted metric graph low-pass eigensystem to smooth new responses.

Usage

```
## S3 method for class 'metric.graph.lowpass.fit'
refit(
  object,
  y,
  per.column.gcv = FALSE,
  eta.grid = NULL,
  n.candidates = 40L,
  n.cores = 1L,
  block.size = NULL,
  verbose = FALSE,
  numerical.policy = NULL,
  ...
)
```

Arguments

object	A "metric.graph.lowpass.fit" object.
y	Numeric vector or matrix with one row per graph vertex. Each column may have its own NA mask for complete-basis heat filtering.
per.column.gcv	Logical. If TRUE, select eta independently for each response column using the cached eigenbasis.
eta.grid	Optional positive numeric eta grid for per-column GCV.
n.candidates	Number of eta candidates when eta.grid = NULL.
n.cores	Positive integer retained for compatibility. Response columns are always processed sequentially; values greater than one do not enable parallel execution. Per-column GCV results report n.cores.used = 1L.
block.size	Optional block size for fixed-eta multi-column refits.
verbose	Logical progress flag.
numerical.policy	NULL inherits the fitted object's policy; otherwise "conservative" or "verified.fallback".
...	Additional arguments are not currently supported.

Value

A list of class "metric.graph.lowpass.refit".

See Also

refit for supported fitted objects and migration.

Examples

```
adj <- list(2L, c(1L, 3L), c(2L, 4L), 3L)
lengths <- list(1, c(1, 1), c(1, 1), 1)
fit <- fit.metric.graph.lowpass(
  adj, lengths, y = 1:4, n.eigenpairs = 4L,
  eta.grid = c(0.1, 1), eigen.solver = "dense"
)
refit(fit, y = 4:1)
```

---

refit.slpl_tf *Refit A Fixed-Operator Synchronized LPL-TF Model*

Description

Refit A Fixed-Operator Synchronized LPL-TF Model

Usage

```
## S3 method for class 'slpl_tf'
refit(
  object,
  y,
  lambda1 = NULL,
  lambda2 = NULL,
  reuse.lambda = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

object	A "slpl_tf" fit.
y	New numeric response vector.
lambda1, lambda2	Optional fixed penalties. Defaults reuse object$lambda1 and object$lambda2.
reuse.lambda	Logical. If TRUE, reuse omitted penalties.
verbose	Logical.
...	Reserved.

Value

A refitted "slpl_tf" object.

See Also

refit for supported fitted objects and migration.

Examples

```
if (requireNamespace("genlasso", quietly = TRUE)) {
  X <- matrix(seq(0, 1, length.out = 18), ncol = 1)
  fit <- fit.slpl.tf(X, X[, 1]^2, degree = 1L,
                     support.type = "knn", support.size = 7L,
                     lambda1 = 0.1, lambda2 = 0,
                     lambda.selection = "fixed")
  refit(fit, y = X[, 1]^3)
}
```

---

refit.ssrhe.hessian.fit

Refit SSRHE-Style Hessian-Energy Regression

Description

Reuses the operator from fit.ssrhe.hessian.regression to fit new responses or new fixed penalty weights without rebuilding local PCA neighborhoods or Hessian-energy matrices.

Usage

```
## S3 method for class 'ssrhe.hessian.fit'
refit(
  object,
  y = NULL,
  lambda1 = object$lambda$lambda1,
  lambda2 = object$lambda$lambda2,
  weights = NULL,
  ridge = object$lambda$ridge,
  verbose = FALSE,
  ...
)
```

Arguments

object	A "ssrhe.hessian.fit" object.
y	New numeric response vector or matrix with one row per vertex. If NULL, the original response is reused.
lambda1	Nonnegative Hessian-energy penalty multiplier for $f^\top Bf = \|Af\|_2^2$.
lambda2	Nonnegative supplemental-stabilizer multiplier for $f^\top B_S f$. If positive, stabilizer must be TRUE. Currently supported only for derivative.order = 2L.

weights	Optional nonnegative observation weights. May be NULL, a vector of length nrow(X), or a matrix with the same dimensions as y.
ridge	Nonnegative diagonal ridge added to the linear system for numerical stabilization.
verbose	Logical. If TRUE, print a short native construction message.
...	Additional arguments are not currently supported.

Value

A list of class "ssrhe.hessian.refit".

See Also

refit for supported fitted objects and migration.

Examples

```
X <- matrix(seq(0, 1, length.out = 12), ncol = 1)
fit <- fit.ssrhe.hessian.regression(
  X, X[, 1]^2, k = 6L, tangent.dim = 1L, lambda1 = 0.1
)
refit(fit, y = X[, 1]^3)
```

---

refit.ssrhe.hessian.l1.fit

Refit SSRHE-Style Hessian L1 Regression

Description

Reuses the A operator from fit.ssrhe.hessian.l1.regression to fit a new response or a new lambda grid without rebuilding local neighborhoods or local PCA Hessian rows.

Usage

```
## S3 method for class 'ssrhe.hessian.l1.fit'
refit(
  object,
  y = NULL,
  lambda.grid = object$lambda,
  lambda.selection = c("fixed", "cv"),
  weights = NULL,
  n.lambda = 40L,
  nfolds = 5L,
  fold.id = NULL,
  loss = c("mse", "mae"),
  selection = c("min", "one.se"),
  solver = c("genlasso", "admm", "auto"),
```

```
  row.scaling = c("none", "l2"),
  admm.rho = NULL,
  admm.maxiter = 2000L,
  admm.abstol = 1e-06,
  admm.reltol = 1e-05,
  maxsteps = 2000L,
  minlam = 0,
  approx = FALSE,
  rtol = 1e-07,
  btol = 1e-07,
  eps = 1e-04,
  verbose = FALSE,
  admm.adaptive.rho = TRUE,
  ...
)
```

Arguments

object	A "ssrhe.hessian.l1.fit" object.
y	Optional new numeric response vector. If NULL, the original response is reused.
lambda.grid	Optional nonnegative lambda grid. For lambda.selection = "fixed", this must contain exactly one value. For lambda.selection = "cv", NULL builds a default grid from the fitted full-data generalized-lasso path.
lambda.selection	"cv" for observed-label K-fold cross-validation or "fixed" for a supplied fixed lambda.
weights	Optional nonnegative observation weights. Missing responses automatically receive zero weight.
n.lambda	Number of default lambda candidates when lambda.grid = NULL and lambda.selection = "cv".
nfolds	Number of validation folds over observed positive-weight labels. Ignored when fold.id is supplied.
fold.id	Optional integer fold assignments of length nrow(X).
loss	Validation loss, currently "mse" or "mae".
selection	Selection rule. "min" chooses the smallest mean validation loss. "one.se" chooses the largest lambda within one standard error of the minimum.
solver	Solver backend. "genlasso" uses the generalized-lasso path to propose a grid when needed, then refines every candidate with ADMM on the original objective. "admm" skips the path and requires an explicit grid. "auto" also proceeds when path construction fails, provided an explicit grid is supplied.
row.scaling	Optional row scaling for the penalty matrix before solving. "l2" scales nonzero rows of A to unit Euclidean norm.
admm.rho, admm.maxiter, admm.abstol, admm.reltol	ADMM controls used by all backends, including refinement of generalized-lasso candidates. admm.rho = NULL chooses an initial algorithm penalty of 1 / max(1,

mean(rowSums(D^2))), where D is the penalty operator after any requested row scaling. A positive numeric value sets the initial penalty explicitly. This is separate from statistical regularization lambda.grid. The iteration limit is a hard cap.

maxsteps, minlam, approx, rtol, btol, eps, verbose
: Controls passed to **genlasso**.

admm.adaptive.rho
: Logical; adapt the algorithm penalty during the first 1000 iterations by balancing tolerance-normalized primal and dual residuals. Set FALSE to hold the initial penalty fixed.

...
: Additional arguments are not currently supported.

Value

A list of class "ssrhe.hessian.l1.refit".

See Also

refit for supported fitted objects and migration.

Examples

```
X <- matrix(seq(0, 1, length.out = 12), ncol = 1)
fit <- fit.ssrhe.hessian.l1.regression(
  X, X[, 1]^2, k = 6L, tangent.dim = 1L,
  lambda.grid = 0.05, lambda.selection = "fixed", solver = "admm"
)
refit(
  fit, y = X[, 1]^3, solver = "admm"
)
```

---

select.masked.graph.field

Evaluate Nested Selection Among Candidate Graphs

Description

Chooses the graph using inner-fold loss only, then assembles its outer-fold predictions. Only graphs with full inner-fold prediction coverage compete; unsupported outer predictions remain NA. Candidate vertex identities and ordering must agree. Final graph selection uses full-data inner CV.

Usage

```
select.masked.graph.field(comparison, study)
```

Arguments

comparison	Output from compare.masked.graph.fields.
study	Name of the study used to choose graphs (for example HMP).

Value

Selected graph per outer fold, outer predictions and coverage, participant-balanced MSE, final graph name, and its final fitted field.

Examples

```
op <- metric.graph.lowpass.operator(list(2L, c(1L, 3L), 2L),
                                    list(1, c(1, 1), 1))
studies <- list(example = list(vertex = rep(c(1, 3), 6), y = 1:12,
                               group = letters[1:12]))
comparison <- compare.masked.graph.fields(list(path = op), studies,
    lambdas = c(0.01, 1), outer.folds = 3, inner.folds = 2)
selected <- select.masked.graph.field(comparison, "example")
selected$final.graph
selected$coverage
```

---

slpl.tf.operator *Construct A Synchronized LPL-TF Fixed Operator*

Description

Builds the two operators of synchronized local polynomial lifting trend filtering (S-LPL-TF). The first, A_{LPL}, is the residual operator of `lpl.tf.operator`, in which each point is predicted from its neighbors without using its own value. The second, C_{sync}, compares the local polynomial predictions of overlapping neighborhoods, each of which includes its own center point.

Usage

```
slpl.tf.operator(
  X,
  adj.list = NULL,
  weight.list = NULL,
  graph = NULL,
  graph.stage = "final",
  anchor.index = NULL,
  anchor.coordinates = NULL,
  degree = 2L,
  support.type = c("adaptive.radius", "knn", "fixed.radius"),
  support.size = NULL,
  radius = NULL,
  min.support = NULL,
  support.buffer = 3L,
  kernel = c("epanechnikov", "triangular", "gaussian", "tricube"),
  coordinate.method = c("coordinates", "local.pca"),
  chart.dim = NULL,
  support.metric = c("auto", "coordinates", "graph.geodesic"),
  auto.chart.support.metric = c("coordinates", "operator", "both"),
```

```
  auto.chart.selection.metric = c("coordinates", "operator"),
  row.normalize = c("l2", "none", "l1"),
  local.solver = c("auto", "normal.equations", "qr", "svd"),
  normal.equations.max.condition = 1e+08,
  duplicate.action = c("keep", "error"),
  drop.rank.deficient = TRUE,
  sync.row.normalize = c("l2"),
  sync.min.norm = 1e-12,
  verbose = FALSE,
  ...
)
```

Arguments

`X`	Numeric coordinate matrix with one row per observation.
`adj.list, weight.list`	Optional supplied undirected graph adjacency and positive edge-length lists using 1-based vertex indices.
`graph`	Optional graph object returned by `dgraphs::create.rknn.graph()` or a list containing `adj.list`/`weight.list` or `adj_list`/`weight_list`.
`graph.stage`	Graph stage to extract from `graph`: `"final"`, `"raw"`, or `"pruned"`.
`anchor.index`	Optional integer vector of observed anchor/target indices. The current implementation supports observed anchors only.
`anchor.coordinates`	Reserved for later off-observation anchors; this implementation requires `NULL`.
`degree`	Polynomial degree. The current implementation supports `0L`, `1L`, and `2L`.
`support.type`	Coordinate support rule. `"knn"` uses the nearest `support.size` candidates including the target before self exclusion. `"adaptive.radius"` uses the nearest `min.support` positive predictors after self exclusion. `"fixed.radius"` uses all points within `radius`.
`support.size`	Positive integer for `support.type = "knn"`.
`radius`	Positive coordinate radius for `support.type = "fixed.radius"`.
`min.support`	Minimum number of positive-weight predictors after self exclusion. If `NULL`, uses `choose(ncol(X) + degree, degree) + support.buffer`.
`support.buffer`	Nonnegative integer added to the local polynomial design size when choosing default supports.
`kernel`	Kernel used to weight local predictors by distance.
`coordinate.method`	Coordinate chart used for the local polynomial design. `"coordinates"` uses centered ambient coordinates; `"local.pca"` uses a deterministic local PCA chart on the selected support.
`chart.dim`	Chart dimension. For `"coordinates"`, this must be `NULL` or `ncol(X)`. For `"local.pca"`, `NULL` defaults to `ncol(X)`. The special value `"auto"` estimates a single chart dimension from observed-coordinate local PCA spectra only, without using responses, truth values, latent coordinates, or labels.

`support.metric` Support-distance rule. `"coordinates"` uses Euclidean coordinate distances; `"graph.geodesic"` uses shortest-path distances in the supplied graph. `"auto"` uses graph geodesics when a graph is supplied and coordinate distances otherwise.

`auto.chart.support.metric`
Support system used when `chart.dim = "auto"`. `"coordinates"` uses Euclidean coordinate neighborhoods, `"operator"` uses the resolved operator support metric, and `"both"` computes both diagnostics side by side.

`auto.chart.selection.metric`
Which auto chart-dimension diagnostic to use for the fitted operator when `auto.chart.support.metric = "both"`. The default `"coordinates"` preserves historical behavior.

`row.normalize` Complete-row normalization rule applied after setting target coefficient `+1` and predictor coefficients `-h_ij`.

`local.solver` Local linear algebra rule.

`normal.equations.max.condition`
Maximum condition number for `local.solver = "auto"` to use normal equations.

`duplicate.action`
Duplicate coordinate policy.

`drop.rank.deficient`
Logical. The current implementation requires `TRUE`.

`sync.row.normalize`
Synchronization-row normalization. The current implementation uses `"l2"` by default.

`sync.min.norm` Minimum raw synchronization-row norm. Rows below this threshold are dropped with metadata.

`verbose` Logical. Reserved for diagnostic messages.

`...` Reserved.

Value

A list of class `"slpl_tf_operator"` containing `A_LPL`, `C_sync`, row metadata, diagnostics, settings, and the embedded `"lpl_tf_operator"`.

Examples

```
X <- matrix(seq(0, 1, length.out = 18), ncol = 1)
slpl.tf.operator(X, degree = 1L, support.type = "knn",
                 support.size = 7L, kernel = "gaussian")
```

`smoother.matrix` *Extract A Conditional Smoother Matrix*

Description

Extract the linear map from responses to predictions for a supported fitted model. R selects the method from the class of `object`.

Usage

```
smoother.matrix(object, ...)

## Default S3 method:
smoother.matrix(object, ...)
```

Arguments

`object`	An `"lps"` or `"malps"` fitted object.
`...`	Method-specific controls: `check.tol` for LPS; `max.n` and `allow.robust` for MALPS. Unrecognized arguments are errors.

Details

`smoother.matrix.lps` requires a fixed-configuration Gaussian LPS fit using the R backend, the orthogonal polynomial drop basis, singleton tuning grids, and fixed chart dimension. It returns one row per evaluation point and checks that the matrix reproduces the stored raw fitted values.

`smoother.matrix.malps` returns a training-point matrix conditional on stored supports, charts, and weights. Its default size limit is `max.n = 1000L`. Robust fits are rejected unless `allow.robust = TRUE`; that option freezes the final robust weights and does not represent how those weights change with the response.

Neither method represents uncertainty from repeating model selection.

Value

A numeric matrix mapping training responses to predictions. See the method help for dimensions and attributes.

Migration

Replace `lps.smoother.matrix(fit, ...)` and `malps.smoother.matrix(fit, ...)` with `smoother.matrix(fit, ...)`. Existing method-specific arguments and restrictions are preserved. The former functions are no longer exported; their help aliases lead to the corresponding methods.

See Also

`refit`, `lps.pointwise.band`, `malps.gcv`

Examples

```
X <- matrix(seq(0, 1, length.out = 20), ncol = 1)
fit <- fit.malps(X, X[, 1]^2, degree = 1L,
                 support.type = "knn", support.size = 8L)
S <- smoother.matrix(fit)
max(abs(as.vector(S %*% fit$y) - predict(fit)))
```

`smoother.matrix.lps` *Extract the Linear-Smoother Matrix of a Fixed-Configuration LPS Fit*

Description

For a fixed configuration (singleton `support.grid`, `degree.grid`, and `kernel.grid`, with an explicit numeric chart dimension) the LPS fitted vector is linear in the response: `fitted = S %*% y` with `S` depending on `X`, the kernel weights, and the configuration, but not on `y`. This function reconstructs `S` analytically, one evaluation row at a time, by rebuilding each local fit's support, kernel weights, local chart, and design through the same internal routines `fit.lps()` used, and reading off the influence row of the local weighted least-squares solve.

Usage

```
## S3 method for class 'lps'
smoother.matrix(object, check.tol = 1e-10, ...)
```

Arguments

`object`	A fitted `"lps"` object from `fit.lps()` at a fixed configuration (see Details).
`check.tol`	Positive scalar: maximum allowed absolute discrepancy of the self-guard identity `S %*% y == fitted.values.raw`. Default `1e-10`, the program's algebraic tolerance.
`...`	Additional arguments are not currently supported.

Details

The extraction refuses configurations where the linearity premise fails or is unsupported: it requires `outcome.family = "gaussian"`, the R backend, `design.basis = "orthogonal.polynomial.drop"`, singleton grids (no data-driven selection: the CV-selected pipeline is *not* a linear smoother), and `coordinate.method = "coordinates"` or `"local.pca"` with an explicit numeric `chart.dim` (never `"auto"` or `"local.auto"`).

Self-guard: before returning, the function verifies `max(abs(S %*% y - fitted.values.raw)) <= check.tol` against the fit it was given (and that the `NA` patterns agree), so any divergence between the reconstruction and the estimator is a hard error, never a silent drift.

Local fits that fell back are represented honestly: a weighted-mean fallback (`unstable.action = "mean"`) contributes its exact linear row `w / sum(w)`; an `unstable.action = "na"` non-fit contributes an all-`NA` row.

Value

A numeric matrix `S` with `nrow(object$X.eval)` rows and `nrow(object$X)` columns: row `i` holds the weights through which the training responses enter the prediction at evaluation point `i`.

See Also

`lps.pointwise.band()` for pointwise variances and confidence bands derived from `S`.

`smoother.matrix` for supported fitted objects and migration.

Examples

```
set.seed(1)
n <- 30
X <- matrix(runif(2 * n, -1, 1), ncol = 2)
y <- sin(pi * X[, 1]) + 0.1 * rnorm(n)
fit <- fit.lps(X, y, foldid = rep(1:2, length.out = n),
               support.grid = 12L, degree.grid = 1L,
               kernel.grid = "tricube", backend = "R",
               design.basis = "orthogonal.polynomial.drop",
               ridge.multiplier.grid = 0, ridge.condition.max = Inf,
               unstable.action = "na")
S <- smoother.matrix(fit)
max(abs(S %*% y - fit$fitted.values))   # ~1e-15: the linear identity
sum(diag(S))                            # effective degrees of freedom
```

---

smoother.matrix.malps *Construct The Conditional MALPS Smoother Matrix*

Description

Constructs the exact training-point smoother matrix for a fitted `fit.malps` object, conditional on the stored supports, local charts, fitting weights, and averaging weights. For a non-robust MALPS fit or weighted refit with fixed supports, the returned matrix S satisfies $\hat{y} = Sy$. If the original fit used cross-validation, this matrix is conditional on the selected support profile; it does not account for the response-dependence of the model-selection step.

Usage

```
## S3 method for class 'malps'
smoother.matrix(object, max.n = 1000L, allow.robust = FALSE, ...)
```

Arguments

`object`	A `"malps"` object from `fit.malps` or `refit`.
`max.n`	Maximum number of training observations for which a dense smoother matrix may be constructed. Use `Inf` to disable this guard.
`allow.robust`	Logical; allow fixed-final-weight linearization for fits with `robust.iterations > 0L`.
`...`	Reserved for future extensions.

Details

Robust residual reweighting makes the effective fitting weights depend on the response. By default this function therefore rejects robust MALPS fits. Setting `allow.robust = TRUE` constructs the fixed-final-weight linearization, which reproduces the stored fit but should not be interpreted as the exact response-to-fit map.

Value

A dense numeric $n \times n$ matrix with attributes describing whether the matrix is conditional on support selection and whether it used a robust fixed-weight linearization.

See Also

`smoother.matrix` for supported fitted objects and migration.

Examples

```
X <- matrix(seq(0, 1, length.out = 20), ncol = 1)
fit <- fit.malps(X, X[, 1]^2, degree = 1L,
                 support.type = "knn", support.size = 8L)
S <- smoother.matrix(fit)
max(abs(S %*% fit$y - fit$fitted.values))
```

---

ssrhe.hessian.operator

Construct an SSRHE-Style Local Hessian Energy Operator

Description

Constructs the discrete local Hessian operator of the semi-supervised regression method of Kim, Steinke, and Hein (2009). For each point, the second derivatives of a function are estimated from its values at nearby points, in local coordinates computed by PCA. The function returns the sparse operator A whose rows compute these second derivatives, the Hessian-energy matrix $B = A^\top A$, and optionally the additional stabilizing matrix B_S.

Usage

```
ssrhe.hessian.operator(
  X,
  k = NULL,
  tangent.dim,
  nn.index = NULL,
  neighborhood.type = c("knn", "adaptive.radius", "supplied"),
  support.index = NULL,
  adaptive.k.scale = NULL,
  radius.rule = c("geomean", "max", "min"),
  radius.factor = 1.25,
```

```
  min.support = NULL,
  max.support = NULL,
  support.buffer = 2L,
  support.topup = c("nearest", "none"),
  tangent.dim.rule = c("fixed", "eigen.cumulative"),
  eigen.tolerance = 0.95,
  derivative.order = 2L,
  stabilizer = FALSE,
  pinv.tol = sqrt(.Machine$double.eps),
  local.solver = c("auto", "normal.equations", "svd", "qr"),
  normal.equations.max.condition = 10000,
  return.A = TRUE,
  return.B = TRUE,
  return.BS = stabilizer,
  return.sparse = TRUE,
  return.local.diagnostics = TRUE,
  return.timing = FALSE,
  verbose = FALSE
)
```

Arguments

`X`	Numeric matrix with one observation per row. Distances for the default neighborhood search are Euclidean distances in these coordinates.
`k`	Integer number of nearest neighbors per point, including the point itself, used when `neighborhood.type = "knn"`. This follows the SSRHE Matlab convention. For adaptive-radius neighborhoods, `k` may be omitted when `adaptive.k.scale` is supplied.
`tangent.dim`	Integer tangent dimension used for the local PCA chart. Required when `tangent.dim.rule = "fixed"`. If `NULL` and `tangent.dim.rule = "eigen.cumulative"`, the dimension is selected locally from the PCA variance ratio.
`nn.index`	Optional integer matrix of neighbor indices, with `nrow(X)` rows and `k` columns. Each row must contain its center vertex. If `NULL`, Euclidean k-NN including self is computed in C++.
`neighborhood.type`	Local support rule. `"knn"` uses the original rectangular self-including kNN neighborhoods. `"adaptive.radius"` builds variable-size supports from `dgraphs::create.rknn.graph` `"supplied"` uses `support.index` directly.
`support.index`	Optional list of integer vectors, one per row of `X`. Each element gives a variable-size local support and must contain its center vertex.
`adaptive.k.scale`	Integer local-scale k used by `dgraphs::create.rknn.graph()` when `neighborhood.type = "adaptive.radius"`.
`radius.rule, radius.factor`	Adaptive-radius graph parameters passed to `dgraphs::create.rknn.graph()`.
`min.support`	Optional minimum local support size for adaptive-radius supports. Defaults to the local quadratic design size plus `support.buffer`, clamped to `nrow(X)`.

`max.support` Optional maximum local support size. Oversized adaptive supports are trimmed to the closest `max.support` vertices while keeping the center vertex.

`support.buffer` Nonnegative integer added to the local quadratic design size when choosing the default `min.support`.

`support.topup` How undersized adaptive-radius supports are enlarged. `"nearest"` appends ambient nearest neighbors. `"none"` leaves supports unchanged and lets the C++ operator reject undersized supports.

`tangent.dim.rule`
Either `"fixed"` or `"eigen.cumulative"`.

`eigen.tolerance`
Cumulative local PCA variance threshold used when `tangent.dim.rule = "eigen.cumulative"` and `tangent.dim = NULL`.

`derivative.order`
Integer derivative order of the local SSRHE-style operator. `2L` is the original local Hessian energy. `3L` is an experimental third-derivative energy whose rows estimate unique symmetric third-derivative tensor components with factorial and tensor-multiplicity scaling.

`stabilizer` Logical. If `TRUE`, also construct the supplemental stabilizer matrix described in the SSRHE supplement. Currently supported only for `derivative.order = 2L`.

`pinv.tol` Nonnegative tolerance multiplier for local pseudoinverses.

`local.solver` Local least-squares backend used to map local function values to derivative coefficients. `"auto"` is the default: it uses normal equations when the local design is full rank and has condition number no larger than `normal.equations.max.condition`, and otherwise falls back to SVD. `"normal.equations"` requests the normal-equation solve for full-rank local designs and falls back only on hard numerical failures. `"svd"` is the most stable reference path. `"qr"` uses pivoted QR and falls back to SVD for rank-deficient local designs.

`normal.equations.max.condition`
Positive condition-number guard used by `local.solver = "auto"` before accepting the normal-equation backend. The default is deliberately conservative because normal-equation solves square the effective condition number and order-3 near-minimum local supports can be numerically fragile.

`return.A` Logical. If `TRUE`, return A as a sparse matrix.

`return.B` Logical. If `TRUE`, return $B = A^\top A$.

`return.BS` Logical. If `TRUE` and `stabilizer = TRUE`, return the supplemental stabilizer B_S.

`return.sparse` Logical. If `TRUE`, attach **Matrix** sparse matrices in addition to raw triplets.

`return.local.diagnostics`
Logical. If `TRUE`, compute additional R-side local chart diagnostics such as chart distortion and boundary asymmetry. These diagnostics are useful for operator audits, but can be skipped in fitting and cross-validation paths for speed.

`return.timing` Logical. If `TRUE`, attach a phase-level elapsed time table to the returned operator. The timings separate R-side validation, neighborhood/support construction, native local operator construction, optional local diagnostics, sparse matrix assembly, and output finalization. For adaptive-radius neighborhoods, subphase timings are also attached in `neighborhoods$timing`.

`verbose` Logical. If `TRUE`, print a short native construction message.

Details

For each point x_i, the operator constructs a local PCA chart from a self-including local support, projects the support into $\mathbb{R}^m$, and centers the local coordinates at x_i. A local quadratic model is then fitted with the intercept fixed at f_i:

$$f(x_j) - f(x_i) \approx \sum_{a \le b} h_{ab} z_{ja} z_{jb} + \sum_a g_a z_{ja}.$$

The local linear map from function values to the quadratic coefficients is the Matlab-compatible

$$\mathrm{RegMat} = X^+ - X^+ \mathrm{IndMat},$$

where X is the reduced quadratic-plus-linear design and IndMat copies the center value into all local rows. Diagonal Hessian components are scaled by $\sqrt{2}$ and off-diagonal components by 1, so that the row energy matches the reference Hessian energy. With `derivative.order = 3L`, the reduced local design uses cubic monomials first, followed by quadratic and linear monomials. The returned rows estimate unique symmetric third-derivative components $\partial_{abc} f$ with scale

$$s_{abc} = m_{abc} \sqrt{\mu_{abc}},$$

where m_{abc} is the factorial monomial-to-derivative multiplier (6 for aaa, 2 for aab, and 1 for three distinct indices) and μ_{abc} is the ordered-tensor multiplicity (1, 3, or 6). Thus $\|Af\|_2^2$ approximates the squared Frobenius norm of the full symmetric third-derivative tensor.

The returned A is the stacked sparse matrix of these local derivative coefficient rows. The ℓ_2 SSRHE penalty is

$$f^\top B f = \|Af\|_2^2, \quad B = A^\top A.$$

Exposing A is useful for auditability and for future ℓ_1-style variants based on $\|Af\|_1$.

Adaptive-radius graph construction uses **dgraphs**; fixed-k and supplied neighborhoods are package-local geosmooth paths.

Value

A list of class `"ssrhe.hessian.operator"` containing:

- `A`: sparse local Hessian row operator, if requested;
- `B`: sparse quadratic energy matrix $A^\top A$, if requested;
- `BS`: optional supplemental stabilizer matrix;
- `A.triplet`, `BS.triplet`: raw sparse triplets;
- `nn.index`: neighbor index matrix for rectangular kNN neighborhoods, if used;
- `support.index`: list of actual local supports used by the backend;
- `neighborhoods`: support-construction diagnostics;
- `row.table`: one row per Hessian component row of A;
- `diagnostics`: local design rank, condition, and PCA summaries;
- `local.diagnostics`: if requested, one row per center vertex with support size, design rank/condition, PCA variance, chart-distortion, boundary-asymmetry, and curvature-bias proxy diagnostics;
- `timing`: if requested, a phase-level elapsed-time table;
- `native.timing`: if requested, a C++ backend subphase timing table for native operator construction;
- `parity`: notes documenting reference-behavior conventions.

References

Kim, K. I., Steinke, F., and Hein, M. (2009). Semi-supervised regression using Hessian energy with an application to semi-supervised dimensionality reduction. *Advances in Neural Information Processing Systems 22*. https://papers.nips.cc/paper_files/paper/2009/hash/f4552671f8909587cf485ea990207f3b-Abstract.html

Examples

```
X <- matrix(seq(0, 1, length.out = 12), ncol = 1)
ssrhe.hessian.operator(X, k = 6L, tangent.dim = 1L)
```

`ssrhe.support.grid` *Build Candidate Support Profiles for SSRHE Adaptive-Radius Tuning*

Description

Constructs a compact grid of adaptive-radius support profiles for use with `support.selection = "cv"` in SSRHE fitting functions. Each row gives an `adaptive.k.scale` value and a requested `min.support`. The grid is intentionally small by default because support selection nests operator construction inside response cross-validation.

Usage

```
ssrhe.support.grid(
  n,
  tangent.dim,
  derivative.order = 2L,
  support.buffer = 2L,
  max.candidates = 8L
)
```

Arguments

`n`	Number of observations.
`tangent.dim`	Tangent dimension used by the SSRHE local polynomial design.
`derivative.order`	SSRHE derivative order, currently `2L` or `3L`.
`support.buffer`	Nonnegative integer added to the local design size when constructing the smallest candidate support.
`max.candidates`	Maximum number of candidate support profiles to return.

Value

A data frame with columns `adaptive.k.scale`, `min.support`, and `max.support`. `max.support` is `NA` by default, meaning no truncation.

Examples

```
ssrhe.support.grid(n = 50L, tangent.dim = 2L, max.candidates = 4L)
```

`summary.harmonic_smoother`

Summary Method for Harmonic Smoother Results

Description

Provides detailed summary statistics for harmonic smoother results, including the evolution of extrema counts and basin structure differences.

Usage

```
## S3 method for class 'harmonic_smoother'
summary(object, ...)
```

Arguments

`object`	An object of class `"harmonic_smoother"`.
`...`	Further arguments passed to or from other methods.

Value

An object of class `"summary.harmonic_smoother"` containing:

`stable_iteration`, `stability_detected`
: Extrema-stability detection, separate from numerical convergence; iteration is NA if undetected

`converged`, `status`, `num_iterations`
: Numerical relaxation status

`iterations_recorded`
: Total number of recorded iterations

`initial_extrema`
: Number of extrema at the first iteration

`initial_maxima`
: Number of maxima at the first iteration

`initial_minima`
: Number of minima at the first iteration

`final_extrema`
: Number of extrema at the final iteration

`final_maxima`
: Number of maxima at the final iteration

`final_minima`
: Number of minima at the final iteration

`extrema_reduction`
: Reduction in number of extrema

`topology_diff_summary`
: Summary statistics of topology differences

See Also

harmonic.smoother, print.summary.harmonic_smoother

Examples

```
fit <- structure(
  list(
    stable_iteration = 2L,
    i_basins = list(matrix(c(1, 1), ncol = 2)),
    topology_differences = numeric()
  ),
  class = "harmonic_smoother"
)
summary(fit)
```

---

support.masked.graph.field

Summarize Measurement Support for a Graph Field

Description

Reports graph-hop distance to the nearest observed vertex and the number of distinct measured participants at each vertex and its immediate neighbors. These are descriptive support measures, not uncertainty intervals or a guarantee of transportability. Vertex order follows the supplied operator.

Usage

```
support.masked.graph.field(operator, vertex, group)
```

Arguments

operator	An unnormalized metric.graph.lowpass.operator.
vertex	Integer vertex indices of observed responses.
group	Nonmissing participant identifiers for the observations.

Value

A data frame with nearest.observation.hops (Inf if unreachable), local.participants and component.supported for each vertex.

Examples

```
op <- metric.graph.lowpass.operator(list(2L, c(1L, 3L), 2L),
                                    list(1, c(1, 1), 1))
support.masked.graph.field(op, vertex = c(1, 3), group = c("a", "b"))
```

synthetic.dataset.checksum

Compute the canonical content checksum of a synthetic dataset

Description

Compute the canonical content checksum of a synthetic dataset

Usage

```
synthetic.dataset.checksum(x)
```

Arguments

x	A synthetic_dataset.

Value

A lowercase SHA-256 string.

Examples

```
x <- materialize.synthetic(synthetic.registry.spec("G1"), n = 20L, seed = 1L)
synthetic.dataset.checksum(x)
```

synthetic.registry.ids

List maintained synthetic recipe IDs

Description

List maintained synthetic recipe IDs

Usage

```
synthetic.registry.ids()
```

Value

Character recipe IDs.

Examples

```
head(synthetic.registry.ids())
```

synthetic.registry.seed

Resolve the legacy seed for an SSRHE registry recipe

Description

Resolve the legacy seed for an SSRHE registry recipe

Usage

```
synthetic.registry.seed(
  recipe.id,
  replicate = 1L,
  n = NULL,
  base.seed = 273001L
)
```

Arguments

recipe.id	A one-dimensional, flat, or quadform SSRHE recipe ID.
replicate	Positive replicate number.
n	Sample size. Required for flat and quadform recipes.
base.seed	One-dimensional suite base seed.

Value

A validated integer seed.

Examples

```
synthetic.registry.seed("S16.V1", replicate = 2L)
```

synthetic.registry.spec

Resolve a maintained synthetic recipe

Description

Default recipes are reconstructed from normalized component foreign keys. Parameter overrides are accepted only for the legacy G-family compatibility surface and remain content-bound, non-frozen specifications.

Usage

```
synthetic.registry.spec(recipe.id, parameters = list())
```

Arguments

recipe.id	One of synthetic.registry.ids().
parameters	Optional named parameter overrides.

Value

A validated synthetic_spec.

Examples

```
synthetic.registry.spec("G1")
```

---

synthetic.response.bernoulli

Bernoulli response specification

Description

Bernoulli response specification

Usage

```
synthetic.response.bernoulli(minimum.positive = 0L, maximum.attempts = 1L)
```

Arguments

minimum.positive	Minimum accepted positive responses.
maximum.attempts	Maximum complete redraw attempts.

Value

A synthetic response component.

Examples

```
synthetic.response.bernoulli(minimum.positive = 1L, maximum.attempts = 5L)
```

synthetic.response.clustered.gaussian
Clustered Gaussian response specification

Description

Clustered Gaussian response specification

Usage

```
synthetic.response.clustered.gaussian(residual.sd, intraclass.correlation)
```

Arguments

residual.sd	Residual standard deviation.
intraclass.correlation	Intraclass correlation in [0,1).

Value

A synthetic response component.

Examples

```
synthetic.response.clustered.gaussian(0.2, 0.3)
```

synthetic.response.gaussian
Gaussian response specification

Description

Gaussian response specification

Usage

```
synthetic.response.gaussian(sd)
```

Arguments

sd	Response standard deviation.

Value

A synthetic response component.

Examples

```
synthetic.response.gaussian(sd = 0.2)
```

---

synthetic.response.heteroskedastic.gaussian

Heteroskedastic Gaussian response specification

Description

Heteroskedastic Gaussian response specification

Usage

```
synthetic.response.heteroskedastic.gaussian(base.sd, truth.multiplier)
```

Arguments

base.sd	Base standard deviation.
truth.multiplier	Multiplier applied to truth.

Value

A synthetic response component.

Examples

```
synthetic.response.heteroskedastic.gaussian(0.1, 0.2)
```

---

synthetic.response.laplace.outlier

Laplace response with Gaussian contamination

Description

Laplace response with Gaussian contamination

Usage

```
synthetic.response.laplace.outlier(
  laplace.scale,
  outlier.fraction,
  outlier.sd,
  minimum.outliers = 0L,
  count.rounding = "round",
  laplace.algorithm = "uniform.inverse.v1"
)
```

Arguments

`laplace.scale`	Laplace scale.
`outlier.fraction`	Fraction contaminated.
`outlier.sd`	Contamination standard deviation.
`minimum.outliers`	Minimum contamination count.
`count.rounding`	Count policy.
`laplace.algorithm`	Versioned Laplace algorithm.

Value

A synthetic response component.

Examples

```
synthetic.response.laplace.outlier(0.1, 0.05, 1)
```

`synthetic.sampling.stratified` *Legacy G4 stratified sampling specification*

Description

Legacy G4 stratified sampling specification

Usage

```
synthetic.sampling.stratified(
  fraction.a = 0.5,
  algorithm = c("stratum.sequential.v1", "legacy.g4.stratum.sequential.v1")
)
```

Arguments

`fraction.a`	Fraction allocated to stratum A.
`algorithm`	Versioned traversal algorithm.

Value

A synthetic sampling component.

Examples

```
synthetic.sampling.stratified(fraction.a = 0.6)
```

synthetic.spec *Assemble a synthetic-data specification*

Description

Combines draw-free geometry, sampling, truth, and response components into one canonically hashable specification.

Usage

```
synthetic.spec(
  geometry,
  sampling,
  truth,
  response,
  recipe.id = NULL,
  registry.tag = NULL,
  compatibility = NULL,
  metadata = list()
)
```

Arguments

geometry	A synthetic geometry component.
sampling	A synthetic sampling component.
truth	A synthetic truth component.
response	A synthetic response component.
recipe.id	Optional readable recipe label.
registry.tag	Optional stable experimental-plan tag.
compatibility	Optional serializable legacy compatibility metadata.
metadata	Optional serializable scientific metadata.

Value

A synthetic_spec.

Examples

```
spec <- synthetic.spec(
  dgraphs::synthetic.circle(), dgraphs::synthetic.sampling.uniform.interval(0, 2 * pi),
  synthetic.truth.named("helix.sin.v1"), synthetic.response.gaussian(0.1)
)
spec$specification.sha256
```

`synthetic.truth.gaussian.mixture` *Specify a Gaussian-mixture truth*

Description

Specify a Gaussian-mixture truth

Usage

```
synthetic.truth.gaussian.mixture(
  centers,
  scales,
  weights,
  normalize = c("none", "sample.max"),
  evaluation.coordinates = "latent"
)
```

Arguments

`centers`	Component centers, one per row.
`scales`	Positive isotropic standard deviations or covariance matrices.
`weights`	Nonnegative mixture weights.
`normalize`	Normalization scope.
`evaluation.coordinates`	Coordinates on which to evaluate.

Value

A synthetic truth component.

Examples

```
synthetic.truth.gaussian.mixture(
  centers = rbind(-1, 1), scales = c(0.5, 0.5), weights = c(1, 1)
)
```

synthetic.truth.logit *Specify a sinusoidal finite-design logit truth*

Description

Specify a sinusoidal finite-design logit truth

Usage

```
synthetic.truth.logit(
  amplitude = 1.5,
  target.prevalence = 0.5,
  clip = c(0.05, 0.95)
)
```

Arguments

amplitude	Sinusoidal log-odds amplitude.
target.prevalence	Target mean pre-clipping probability.
clip	Probability bounds.

Value

A synthetic truth component.

Examples

```
synthetic.truth.logit(amplitude = 1, target.prevalence = 0.4)
```

synthetic.truth.named *Specify a versioned named truth evaluator*

Description

Specify a versioned named truth evaluator

Usage

```
synthetic.truth.named(id, parameters = list())
```

Arguments

id	Evaluator ID.
parameters	Serializable evaluator parameters.

Value

A synthetic truth component.

Examples

```
synthetic.truth.named("intrinsic.linear.first.v1")
```

`synthetic.truth.occupation.mixture` *Specify a finite-design occupation-mixture truth*

Description

The analytic Gaussian-mixture density is transformed to Bernoulli occupation probabilities by `probability.maximum * density^gamma / max(density^gamma)`.

Usage

```
synthetic.truth.occupation.mixture(
  centers,
  covariances,
  weights,
  gamma = 1,
  probability.maximum = 0.65,
  normalization = "design.maximum"
)
```

Arguments

`centers`	Component centers, one per row.
`covariances`	Positive-definite covariance matrices.
`weights`	Nonnegative mixture weights.
`gamma`	Positive density-shape exponent.
`probability.maximum`	Maximum realized occupation probability.
`normalization`	Currently `"design.maximum"`.

Value

A finite-design synthetic truth component.

Examples

```
synthetic.truth.occupation.mixture(
  centers = rbind(c(-1, 0), c(1, 0)),
  covariances = list(diag(2), diag(2)), weights = c(1, 1)
)
```

synthetic.truth.polynomial *Specify a polynomial truth*

Description

Specify a polynomial truth

Usage

```
synthetic.truth.polynomial(coefficients, evaluation.coordinates = "latent")
```

Arguments

coefficients	Named polynomial coefficients.
evaluation.coordinates	Coordinates on which to evaluate.

Value

A synthetic truth component.

Examples

```
synthetic.truth.polynomial(c(b0 = 1, b1 = 2, b11 = -0.5))
```

transported.graph.hessian.operator *Construct a Transported Graph Hessian Operator*

Description

Builds an experimental operator that computes second differences (or third differences) of a function on the vertices of a graph. The operator takes first differences along directed edges, matches each direction at one end of an edge with a direction at the other end, and subtracts the matched first differences. No response is fitted. The function is meant for studying the operator, and it returns diagnostics that show how the directions were matched.

Usage

```
transported.graph.hessian.operator(
  adj.list,
  weight.list = NULL,
  transport.order = 2L,
  transport.rule = c("exact.coordinate", "local.embedding.soft", "edge.angle.hard",
    "edge.angle.soft", "regression.gradient"),
  coordinates = NULL,
  direction.labels = NULL,
  polynomial.probes = NULL,
  local.embedding.method = c("auto", "coordinates", "grip.edge.kk", "mds.edge.kk",
    "cmdscale"),
  local.embedding.dim = 2L,
  local.disk.hops = 1L,
  local.max.vertices = 50L,
  return.sparse = TRUE,
  tol = 1e-08,
  soft.angle.scale = NULL,
  soft.length.scale = NULL,
  soft.bandwidth = 0.25,
  max.match.angle = NULL,
  max.length.relative.error = NULL,
  min.match.margin = NULL,
  max.effective.matches = NULL,
  match.threshold.rule = c("none", "fixed", "local.quantile", "local.robust.z"),
  match.score.quantile = 0.25,
  match.margin.quantile = 0.5,
  min.best.score.z = 1,
  min.margin.z = 0,
  max.effective.match.fraction = NULL,
  edge.angle.scale = NULL,
  edge.length.scale = NULL,
  edge.angle.bandwidth = 0.35,
  edge.angle.max.angle.difference = NULL,
  edge.angle.max.length.relative.error = NULL,
  gradient.coordinate.method = c("coordinates", "local.embedding"),
  gradient.embedding.method = c("grip.edge.kk", "mds.edge.kk", "cmdscale"),
  gradient.embedding.dim = NULL,
  gradient.disk.hops = local.disk.hops,
  gradient.max.vertices = local.max.vertices,
  gradient.disk.rule = c("hops", "metric.diameter.fraction", "metric.local.scale"),
  gradient.disk.radius.fraction = 0.1,
  gradient.disk.local.scale.method = c("knn.distance", "median.incident.length",
    "quantile.incident.length"),
  gradient.disk.local.scale.k = 8L,
  gradient.disk.local.scale.quantile = 0.75,
  gradient.disk.local.scale.multiplier = 2,
  gradient.disk.min.vertices = 0L,
```

```
  gradient.chart.selection = c("fixed", "adaptive"),
  gradient.embedding.candidates = c("cmdscale", "mds.edge.kk"),
  gradient.disk.rule.candidates = c("hops", "metric.diameter.fraction",
    "metric.local.scale"),
  gradient.disk.hops.candidates = 1:5,
  gradient.disk.radius.fraction.candidates = c(0.05, 0.075, 0.1, 0.15, 0.2),
  gradient.disk.local.scale.multiplier.candidates = c(1, 1.5, 2, 3, 4),
  gradient.ridge = 1e-08,
  gradient.quadratic.disk.hops = 2L,
  gradient.quadratic.max.vertices = gradient.max.vertices
)
```

Arguments

`adj.list`	List of integer neighbor vectors using 1-based vertex indices. The graph must be undirected.
`weight.list`	Optional list of positive edge lengths parallel to `adj.list`. If `NULL`, all edge lengths are one.
`transport.order`	Integer scalar, either `2L` or `3L`. `2L` constructs the transported Hessian-like second-difference operator. `3L` currently constructs a supplied-coordinate regression-quadratic third-difference operator and is supported only for `transport.rule = "regression.gradient"` with `gradient.coordinate.method = "coordinates"`.
`transport.rule`	Character scalar. `"exact.coordinate"` uses exact direction labels. `"local.embedding.soft"` uses coordinate direction vectors and softmax transport weights. `"edge.angle.hard"` and `"edge.angle.soft"` use coordinate-derived edge angles relative to the base dart. `"regression.gradient"` compares locally estimated gradient components.
`coordinates`	Optional numeric matrix with one row per vertex. For `transport.rule = "exact.coordinate"`, coordinates must define axis-aligned graph edges so direction labels can be inferred. Coordinates are required by `"edge.angle.hard"` and `"edge.angle.soft"`.
`direction.labels`	Optional list parallel to `adj.list`; each entry gives the exact direction label for the corresponding outgoing dart. When supplied, these labels override labels inferred from `coordinates`.
`polynomial.probes`	Optional numeric vector or matrix with one row per vertex. If supplied, residual diagnostics $\|AP\|_F/\|P\|_F$ are reported for the transported Hessian matrix A.
`local.embedding.method`	Character scalar controlling the coordinates used by `transport.rule = "local.embedding.soft"`. `"auto"` uses supplied `coordinates` when available and otherwise tries `"grip.edge.kk"`. `"coordinates"` uses supplied coordinates directly. `"grip.edge.kk"` builds a local graph disk and prefers weighted GRIP through `grip::grip(metric = "edge_length")`, followed by `grip::edge.kk()` optimization. Both functions must provide the dot-delimited argument interface available in **grip** 0.3.0 or later; incompatible older interfaces produce an error. `"mds.edge.kk"` uses classical MDS followed by the same edge-KK optimizer when available. `"cmdscale"` uses classical MDS only.

`local.embedding.dim`
: Positive integer embedding dimension for local graph-disk embeddings.

`local.disk.hops`
: Non-negative integer hop radius around each base dart used by graph-derived local embedding methods.

`local.max.vertices`
: Positive integer cap on local disk size. If a disk is larger, the nearest vertices by hop distance are retained.

`return.sparse`
: Logical. If `TRUE`, attach **Matrix** sparse matrices to the returned payloads.

`tol`
: Positive numeric tolerance for coordinate-axis direction inference.

`soft.angle.scale`, `soft.length.scale`
: Optional positive numeric scales for the soft transport angular and length score components. If `NULL`, robust global defaults are estimated from candidate dart comparisons.

`soft.bandwidth`
: Positive numeric softmax bandwidth τ. Smaller values make the soft rule closer to hard nearest-direction matching.

`max.match.angle`
: Optional non-negative numeric angle threshold in radians. For soft transport, candidate Hessian rows whose best match has angle larger than this threshold are dropped.

`max.length.relative.error`
: Optional non-negative numeric threshold for the best match's relative edge-length error, $|\ell_{\mathrm{matched}} - \ell_{\mathrm{direction}}|/\ell_{\mathrm{direction}}$. Candidate rows above the threshold are dropped.

`min.match.margin`
: Optional non-negative numeric threshold for the difference between the second-best and best soft-match scores. Candidate rows with smaller margins are dropped.

`max.effective.matches`
: Optional positive numeric threshold for $\exp(H)$, where H is the softmax transport entropy. Candidate rows with more diffuse matches are dropped.

`match.threshold.rule`
: Character scalar. `"none"` and `"fixed"` apply only the explicit fixed gates above. `"local.quantile"` also gates rows by local candidate-score and margin quantiles. `"local.robust.z"` also gates rows by robust local z-scores for best-match score and margin.

`match.score.quantile`, `match.margin.quantile`
: Quantile thresholds used by `match.threshold.rule = "local.quantile"`. The best score must lie at or below `match.score.quantile`; the best-vs-second margin must lie at or above `match.margin.quantile`.

`min.best.score.z`, `min.margin.z`
: Non-negative robust-z thresholds used by `match.threshold.rule = "local.robust.z"`. A larger best-score z-score means the best candidate is unusually low-scoring relative to local alternatives. A larger margin z-score means the best-vs-second separation is unusually large.

`max.effective.match.fraction`
: Optional positive numeric threshold for $\exp(H)/n_{\mathrm{candidate}}$. Candidate rows whose softmax mass is too diffuse across the local target directions are dropped.

edge.angle.scale, edge.length.scale
: Optional positive numeric scales for edge-angle matching. If NULL, robust global medians are estimated from candidate angle differences and length differences.

edge.angle.bandwidth
: Positive numeric softmax bandwidth used only by transport.rule = "edge.angle.soft".

edge.angle.max.angle.difference
: Optional non-negative threshold, in radians, for accepting an edge-angle match.

edge.angle.max.length.relative.error
: Optional non-negative threshold for accepting an edge-angle match based on relative edge-length error.

gradient.coordinate.method
: Character scalar used only by transport.rule = "regression.gradient". "coordinates" uses the supplied global coordinates. "local.embedding" estimates the two endpoint gradients of each base dart in one shared graph-derived local chart.

gradient.embedding.method
: Character scalar used by gradient.coordinate.method = "local.embedding". The choices match the graph-derived local embedding backends used by local.embedding.soft.

gradient.embedding.dim
: Optional positive integer dimension for graph-derived regression-gradient local charts. If NULL, the supplied coordinate dimension is used when available, otherwise local.embedding.dim.

gradient.disk.hops
: Non-negative integer hop radius for graph-derived regression-gradient local charts.

gradient.max.vertices
: Positive integer cap on graph-derived regression-gradient local chart size.

gradient.disk.rule
: Character scalar controlling graph-derived regression-gradient disk construction. "hops" uses gradient.disk.hops. "metric.diameter.fraction" uses a two-center graph-geodesic disk whose metric radius is gradient.disk.radius.fraction times the graph diameter. "metric.local.scale" uses a two-center graph-geodesic disk whose metric radius is gradient.disk.local.scale.multiplier times the larger endpoint local scale.

gradient.disk.radius.fraction
: Positive numeric scalar used by gradient.disk.rule = "metric.diameter.fraction".

gradient.disk.local.scale.method
: Character scalar controlling the per-vertex metric scale used by gradient.disk.rule = "metric.local.scale". "knn.distance" uses the weighted graph distance to the gradient.disk.local.scale.k-th nearest vertex. "median.incident.length" and "quantile.incident.length" use incident edge-length summaries.

gradient.disk.local.scale.k
: Positive integer neighborhood index used by gradient.disk.local.scale.method = "knn.distance".

gradient.disk.local.scale.quantile
: Numeric probability in [0, 1] used by gradient.disk.local.scale.method = "quantile.incident.length".

gradient.disk.local.scale.multiplier
: Positive numeric scalar multiplier for gradient.disk.rule = "metric.local.scale".

`gradient.disk.min.vertices`
: Non-negative integer lower target for metric disk coverage. Metric disks whose radius selects too few vertices are expanded to include at least this many nearest vertices, subject to `gradient.max.vertices`.

`gradient.chart.selection`
: Character scalar used only by graph-derived regression-gradient transport. `"fixed"` uses `gradient.embedding.method` and `gradient.disk.hops`. `"adaptive"` tries `gradient.embedding.candidates` crossed with `gradient.disk.hops.candidates` for each base dart and chooses the chart with the lowest graph-only diagnostic score.

`gradient.embedding.candidates`
: Character vector of graph-derived local embedding backends considered by `gradient.chart.selection = "adaptive"`.

`gradient.disk.rule.candidates`
: Character vector of disk construction rules considered by `gradient.chart.selection = "adaptive"`.

`gradient.disk.hops.candidates`
: Non-negative integer vector of hop radii considered by `gradient.chart.selection = "adaptive"`.

`gradient.disk.radius.fraction.candidates`
: Positive numeric vector of graph-diameter fractions considered by `gradient.chart.selection = "adaptive"`.

`gradient.disk.local.scale.multiplier.candidates`
: Positive numeric vector of local-scale multipliers considered by `gradient.chart.selection = "adaptive"`.

`gradient.ridge`
: Non-negative ridge parameter used by `transport.rule = "regression.gradient"` when inverting local weighted least-squares normal equations.

`gradient.quadratic.disk.hops`
: Non-negative integer hop radius for the supplied-coordinate local quadratic regressions used by `transport.order = 3L`.

`gradient.quadratic.max.vertices`
: Positive integer cap on local quadratic-regression disk size for `transport.order = 3L`.

Details

A directed edge is called a dart. For a dart $u \to v$ with length ℓ_{uv}, the first difference of a function f is

$$\delta_{u\to v} f = \frac{f(v) - f(u)}{\ell_{uv}}.$$

Take a base dart $u \to u'$, and suppose the direction $u \to v$ at u has been matched with the direction $u' \to v'$ at u'. The corresponding row of the transported Hessian operator computes

$$\delta_{u'\to v'} f - \delta_{u\to v} f.$$

The rules for matching directions are selected by `transport.rule`:

- "exact.coordinate", the reference rule, matches directions that have the same label. The labels can be supplied in direction.labels or inferred from coordinates when all edges are parallel to coordinate axes.
- "local.embedding.soft" uses coordinates to match directions by angle and edge length, with weights spread over several candidate directions.
- "edge.angle.hard" and "edge.angle.soft" match directions by their angles relative to the base edge.
- "regression.gradient" estimates a gradient at each end of the base dart by local least squares and compares the components of the two gradients. The gradients are computed in the supplied coordinates, or in a local embedding of the part of the graph around the base dart.

Value

A list of class "transported.graph.hessian.operator". It contains the validated graph, the table of directed edges, the first-difference matrix, the transported Hessian matrix, information about each of its rows, the candidate rows that were dropped, and diagnostics. When transport.rule = "regression.gradient" uses a local embedding of the graph and coordinates are also supplied, the diagnostics include a comparison of the local embeddings with those coordinates.

Examples

```
adj <- list(2L, c(1L, 3L), c(2L, 4L), 3L)
weights <- list(1, c(1, 1), c(1, 1), 1)
x <- 0:3
transported.graph.hessian.operator(
  adj, weights, coordinates = matrix(x, ncol = 1),
  polynomial.probes = cbind(1, x, x^2)
)
```

---

unfold.simplex.chart *Unfold a Local Neighborhood of the L-infinity Simplex*

Description

Rotate one adjacent face into the query's face before constructing a local chart. This function certifies a conservative one- or two-face chart; it does not force a chart through a corner or through three or more faces.

Usage

```
unfold.simplex.chart(X, query, face.tol = 1e-12, distance.tol = 1e-08)
```

Arguments

X	Nonnegative numeric matrix, with support points in rows. Each row must have a positive maximum; rows are max-normalized internally.
query	Nonnegative query vector with length ncol(X) and positive maximum.
face.tol	Nonnegative tolerance for identifying tied maxima.
distance.tol	Positive absolute tolerance for agreement between unfolded Euclidean chords and intrinsic distances.

Details

On reference face i, an adjacent-face j point is mapped to `v[i]=1`, `v[j]=2-u[i]`, retaining the other coordinates. The query is included when identifying faces. Tied maxima or more than two faces yield an uncertified chart. For two faces, the intrinsic support radius must be less than the query's distance to every other face, and all support/query pair distances must agree with the unfolded chords. Distances use `dgraphs::linf.simplex.distances()`, including intermediate faces. Certification costs quadratic work and memory in the local support size. A failed certificate does not prove that no useful chart exists.

Value

A list with certified, status, coordinates, query, faces, radius, clearance and distance.error. Coordinates and query are NULL when the chart is uncertified. Coordinates retain the constant reference column.

Examples

```
X <- rbind(c(1, .9, .2), c(.9, 1, .2), c(1, .8, .3))
unfold.simplex.chart(X, c(1, .95, .2))
```

---

validate.synthetic.dataset

Validate a canonical synthetic dataset

Description

Validate a canonical synthetic dataset

Usage

```
validate.synthetic.dataset(x)
```

Arguments

x	A `synthetic_dataset`.

Value

x, invisibly.

Examples

```
x <- materialize.synthetic(synthetic.registry.spec("G1"), n = 20L, seed = 1L)
validate.synthetic.dataset(x)
```

Index

apply.metric.graph.lowpass.path, 6, *49*
as.data.frame.synthetic_dataset, 7
assess.gaussian.trend.filtering, *5*, 8, *25, 26*
assess.logistic.trend.filtering, *5*, *10*, 10, *35*

bootstrap.malps, 14, *46*

compare.masked.graph.fields, 15
compare.synthetic.dataset, 16
cv.masked.graph.field, 17
cv.projection.trend.filtering, 18

fit.chart.kernel, 20
fit.density, *5*, 23, *76, 77*, *108*
fit.gaussian.trend.filtering, *5*, *10*, 24, *55*
fit.graph.trend.filtering, *5*, 26, *70*
fit.local.likelihood, 29
fit.logistic.trend.filtering, *5*, *13*, *24*, *26*, 32, *55*
fit.lpl.tf, *5*, 35
fit.lps, *5*, *21, 22*, *31*, 37, *58–60*, *106*
fit.lps(), *87*, *90*, *143*
fit.malps, *5*, *14*, 42, *92*, *116*, *131, 132*, *144*
fit.masked.graph.field, 47
fit.metric.graph.lowpass, *5*, 48, *106*
fit.neighbor.probability, 51, *117*
fit.projection.trend.filtering, 53
fit.ps.lps, 56, *106*
fit.pttf.trend.filtering, 60
fit.slpl.tf, *5*, 63
fit.ssrhe.hessian.l1.regression, *5*, *62*, 65, *136*
fit.ssrhe.hessian.regression, *5*, *69*, 70, *135*
fit.subject.od, *5*, *23*, 75
fitted(), *75*
fitted.chart_kernel *(*model-values*)*, 103
fitted.gaussian.trend.filtering.fit *(*predict.gaussian.trend.filtering.fit*)*, 112
fitted.graph.trend.filtering.fit *(*model-values*)*, 103
fitted.harmonic_smoother *(*model-values*)*, 103
fitted.local_likelihood *(*model-values*)*, 103
fitted.logistic.trend.filtering.fit *(*predict.logistic.trend.filtering.fit*)*, 113
fitted.lpl_tf *(*model-values*)*, 103
fitted.lps *(*model-values*)*, 103
fitted.malps *(*model-values*)*, 103
fitted.metric.graph.lowpass.fit *(*model-values*)*, 103
fitted.metric.graph.lowpass.refit *(*model-values*)*, 103
fitted.projection.trend.filtering.fit *(*predict.projection.trend.filtering.fit*)*, 118
fitted.ps_lps *(*model-values*)*, 103
fitted.pttf.trend.filtering.fit *(*model-values*)*, 103
fitted.slpl_tf *(*model-values*)*, 103
fitted.ssrhe.hessian.fit *(*model-values*)*, 103
fitted.ssrhe.hessian.l1.fit *(*model-values*)*, 103
fitted.ssrhe.hessian.refit *(*model-values*)*, 103

geosmooth *(*geosmooth-package*)*, 4
geosmooth-package, 4
get.region.boundary, 77, *109*
graph.trend.filtering.operator, 79
graphics::plot(), *111*

harmonic.smoother, *78*, 80, *109, 110*, *122*,

151

lpl.tf.operator, *36*, *37*, 83, *139*
lps.backend.diagnostics, 85
lps.backend.diagnostics(), *42*
lps.grouped.foldid, 86
lps.grouped.foldid(), *88*, *89*
lps.nested.cv, 87
lps.nested.cv(), *42*, *87*
lps.pointwise.band, 89, *142*
lps.pointwise.band(), *144*
lps.smoother.matrix *(*smoother.matrix.lps*)*, 143

malps.gcv, *46*, 91, *142*
malps.smoother.matrix *(*smoother.matrix.malps*)*, 144
materialize.synthetic, *5*, 92
materialize.synthetic.instance, 93
metric.graph.heat.eta.grid, 94
metric.graph.heat.extend.lower, 95
metric.graph.lowpass.basis, 96
metric.graph.lowpass.operator, 98
model-summary, 101
model-values, 103

normalize.density, *5*, *20*, *29*, *76*, *77*, 106

perform.harmonic.smoothing, *78*, *81*, *82*, 108
plot.harmonic_smoother, *82*, 110
plot.synthetic_dataset, 111
predict.gaussian.trend.filtering.fit, 112
predict.logistic.trend.filtering.fit, 113
predict.lpl_tf, 114
predict.lps, 115
predict.malps, 116
predict.neighbor.probability.fit, 117
predict.projection.trend.filtering.cv *(*predict.projection.trend.filtering.fit*)*, 118
predict.projection.trend.filtering.fit, 118
predict.slpl_tf, 119
print.chart_kernel *(*model-summary*)*, 101
print.density_fit *(*model-summary*)*, 101
print.gaussian.trend.filtering.assessment *(*assess.gaussian.trend.filtering*)*, 8
print.gaussian.trend.filtering.fit *(*predict.gaussian.trend.filtering.fit*)*, 112
print.graph.trend.filtering.fit, 120
print.harmonic_smoother, 122
print.local_likelihood *(*model-summary*)*, 101
print.logistic.trend.filtering.assessment *(*assess.logistic.trend.filtering*)*, 10
print.logistic.trend.filtering.fit *(*predict.logistic.trend.filtering.fit*)*, 113
print.lpl_tf *(*model-summary*)*, 101
print.lpl_tf_operator *(*print.graph.trend.filtering.fit*)*, 120
print.lps *(*print.graph.trend.filtering.fit*)*, 120
print.malps *(*print.graph.trend.filtering.fit*)*, 120
print.malps_bootstrap *(*print.graph.trend.filtering.fit*)*, 120
print.metric.graph.lowpass.fit *(*print.graph.trend.filtering.fit*)*, 120
print.metric.graph.lowpass.refit *(*print.graph.trend.filtering.fit*)*, 120
print.projection.trend.filtering.fit *(*predict.projection.trend.filtering.fit*)*, 118
print.ps_lps *(*model-summary*)*, 101
print.pttf.trend.filtering.fit *(*print.graph.trend.filtering.fit*)*, 120
print.pttf_operator *(*print.graph.trend.filtering.fit*)*, 120
print.slpl_tf *(*model-summary*)*, 101
print.slpl_tf_operator *(*print.graph.trend.filtering.fit*)*,

120
print.ssrhe.hessian.cv.fit
(print.graph.trend.filtering.fit), 120
print.ssrhe.hessian.fit
(print.graph.trend.filtering.fit), 120
print.ssrhe.hessian.gcv.fit
(print.graph.trend.filtering.fit), 120
print.ssrhe.hessian.l1.fit
(print.graph.trend.filtering.fit), 120
print.ssrhe.hessian.operator
(print.graph.trend.filtering.fit), 120
print.ssrhe.hessian.refit
(print.graph.trend.filtering.fit), 120
print.summary.geosmooth_fit
(model-summary), 101
print.summary.harmonic_smoother, 123, *151*
print.synthetic_dataset, 124
print.transported.graph.hessian.operator
(print.graph.trend.filtering.fit), 120
pttf.geometry, *61, 62*, 124, *126, 127*
pttf.operator, *60*, *62*, 126, *128*
pttf.operator.filter.rows, 128

refit, *14*, *46*, *92*, *105*, 129, *131, 132*, *134–136*, *138*, *142*, *144*
refit.lpl.tf (refit.lpl_tf), 130
refit.lpl_tf, *129*, 130
refit.malps, *129*, 131
refit.metric.graph.lowpass
(refit.metric.graph.lowpass.fit), 133
refit.metric.graph.lowpass.fit, *130*, 133
refit.slpl.tf (refit.slpl_tf), 134
refit.slpl_tf, *129*, 134
refit.ssrhe.hessian.fit, *130*, 135
refit.ssrhe.hessian.l1.fit, *130*, 136
refit.ssrhe.hessian.l1.regression
(refit.ssrhe.hessian.l1.fit), 136
refit.ssrhe.hessian.regression
(refit.ssrhe.hessian.fit), 135
residuals(), *75*
residuals.chart_kernel (model-values), 103
residuals.gaussian.trend.filtering.fit
(predict.gaussian.trend.filtering.fit), 112
residuals.graph.trend.filtering.fit
(model-values), 103
residuals.local_likelihood
(model-values), 103
residuals.logistic.trend.filtering.fit
(predict.logistic.trend.filtering.fit), 113
residuals.lpl_tf (model-values), 103
residuals.lps (model-values), 103
residuals.malps (model-values), 103
residuals.metric.graph.lowpass.fit
(model-values), 103
residuals.metric.graph.lowpass.refit
(model-values), 103
residuals.projection.trend.filtering.fit
(predict.projection.trend.filtering.fit), 118
residuals.ps_lps (model-values), 103
residuals.pttf.trend.filtering.fit
(model-values), 103
residuals.slpl_tf (model-values), 103
residuals.ssrhe.hessian.fit
(model-values), 103
residuals.ssrhe.hessian.l1.fit
(model-values), 103
residuals.ssrhe.hessian.refit
(model-values), 103

select.masked.graph.field, 138
slpl.tf.operator, *64*, 139
smoother.matrix, *42*, *46, 47*, *91, 92*, *105*, *130*, 142, *144, 145*
smoother.matrix(), *89–91*
smoother.matrix.lps, *142*, 143
smoother.matrix.malps, *142*, 144
ssrhe.hessian.operator, *65*, *69, 70*, 145
ssrhe.support.grid, *68*, *74*, 149
summary.chart_kernel (model-summary), 101
summary.density_fit (model-summary), 101

summary.graph.trend.filtering.fit
(model-summary), 101
summary.harmonic_smoother, *82*, *122, 123*, 150
summary.local_likelihood
(model-summary), 101
summary.lpl_tf (model-summary), 101
summary.lps (model-summary), 101
summary.malps (model-summary), 101
summary.metric.graph.lowpass.fit
(model-summary), 101
summary.metric.graph.lowpass.refit
(model-summary), 101
summary.projection.trend.filtering.fit
(predict.projection.trend.filtering.fit), 118
summary.ps_lps (model-summary), 101
summary.pttf.trend.filtering.fit
(model-summary), 101
summary.slpl_tf (model-summary), 101
summary.ssrhe.hessian.fit
(model-summary), 101
summary.ssrhe.hessian.l1.fit
(model-summary), 101
summary.ssrhe.hessian.refit
(model-summary), 101
support.masked.graph.field, 151
synthetic.dataset.checksum, 152
synthetic.registry.ids, 152
synthetic.registry.seed, 153
synthetic.registry.spec, 153
synthetic.response.bernoulli, 154
synthetic.response.clustered.gaussian, 155
synthetic.response.gaussian, 155
synthetic.response.heteroskedastic.gaussian, 156
synthetic.response.laplace.outlier, 156
synthetic.sampling.stratified, 157
synthetic.spec, *5*, 158
synthetic.truth.gaussian.mixture, 159
synthetic.truth.logit, 160
synthetic.truth.named, 160
synthetic.truth.occupation.mixture, 161
synthetic.truth.polynomial, 162

transported.graph.hessian.operator, 162

unfold.simplex.chart, *32*, 168

validate.synthetic.dataset, 169