Package: r4pde (via r-universe)

June 2, 2026

Type Package

Title Companion to R for Plant Disease Epidemiology Book

Version 0.1.0

Description Datasets and utility functions to support the book ``R for Plant Disease Epidemiology'' (R4PDE). It includes functions for quantifying disease, assessing spatial patterns, and modeling plant disease epidemics based on weather predictors. These tools are intended for teaching and research in plant disease epidemiology. Several functions are based on classical and contemporary methods, including those discussed in Laurence V. Madden, Gareth Hughes, and Frank van den Bosch (2007) <doi:10.1094/9780890545058>.

License MIT + file LICENSE

Depends R (>= 4.1.0)

Imports boot, car, dplyr, ggplot2, igraph, lubridate, mgcv, magrittr, nasapower, progress, purrr, rlang, stats, survival, tidyr, tibble, httr, jsonlite, terra

Suggests interval, testthat, knitr, rmarkdown, ggdendro, patchwork, cowplot, ncdf4

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

BugReports https://github.com/emdelponte/r4pde/issues

URL https://emdelponte.github.io/r4pde/, https://github.com/emdelponte/r4pde

Config/pak/sysreqs cmake libgdal-dev gdal-bin libgeos-dev libglpk-dev make libicu-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libx11-dev

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

Date/Publication 2026-05-03 23:37:02 UTC

RemoteUrl https://github.com/emdelponte/r4pde

RemoteRef HEAD

RemoteSha 26a483f821ebc568b3f7d2491dad3e9ccb2214da

Contents

AFSD *Analysis of foci structure and dynamics (AFSD)*

Description

This function performs the analysis of a simple method introduced by Nelson (1996) and expanded by Laranjeira et al. (1998). The function assumes the dataframe supplied as input has columns 'x', 'y', and 'i', where 'x' and 'y' are spatial coordinates and 'i' is a disease indicator variable (1 if diseased, otherwise 0). The function performs several steps including filtering rows where 'i' is 1, converting to an adjacency matrix, and creating foci using igraph. It then calculates various statistics about the foci and returns these in a list.

Usage

```
AFSD(df)
```

Arguments

df	A dataframe containing at least three columns: 'x', 'y', and 'i'. 'x' and 'y' represent spatial coordinates and 'i' is a disease indicator (1 if diseased, otherwise 0).

Value

A list containing: cluster_summary2: a dataframe summarizing the number and size of foci, and proportions of diseased plants. cluster_df: a dataframe containing foci information, including size and number of rows and columns in each foci. df_clustered: the original dataframe with an added 'focus_id' column, showing which foci each row belongs to.

See Also

Other Spatial analysis: BPL(), count_subareas(), count_subareas_random(), fit_gradients(), join_count(), oruns_test(), oruns_test_boustrophedon(), oruns_test_byrowcol(), plot_AFSD()

Examples

```
# Generate a sample dataframe
set.seed(123)
df <- data.frame(x = sample(1:100, 500, replace = TRUE),
                 y = sample(1:100, 500, replace = TRUE),
                 i = sample(0:1, 500, replace = TRUE, prob = c(0.7, 0.3)))

# Perform the AFSD
result <- AFSD(df)
```

---

augment_functional_pca *Augment functional PCA*

Description

Augment functional PCA

Usage

```
augment_functional_pca(x)
```

Arguments

x	An object of class "r4pde_functional_pca".

BlastWheat *BlastWheat dataset*

Description

Wheat blast dataset with severity and weather covariates.

Usage

```
BlastWheat
```

Format

A data frame with the following columns:

heading Date of heading

inc_mean Mean incidence

index_mean FHB index mean

latitude Latitude coordinate

location Experimental site name

longitude Longitude coordinate

state Brazilian state

study Study ID or code

year Crop year

yld_mean Mean yield

Source

Del Ponte Lab internal data

BPL *Binary Power Law Analysis for Spatial Disease Patterns*

Description

This function calculates the Binary Power Law (BPL) parameters for spatial disease patterns, fits a linear model, and performs a hypothesis test for the slope.

Usage

```
BPL(data)
```

Arguments

`data`	A data frame containing the following columns: • `field`: The field identifier. • `n`: The number of observations in each quadrat. • `i`: The incidence count in each quadrat.

Details

The function performs the following steps:

1. Summarizes the data by field to calculate the total number of observations (`n_total`), mean incidence (`incidence_mean`), observed variance (`V`), and binomial variance (`Vbin`).
2. Log-transforms the variances.
3. Fits a linear model to the log-transformed variances.
4. Tests the hypothesis that the slope of the linear model is equal to 1.

Value

A list containing the following elements:

- `summary`: A data frame summarizing the input data by field, including total observations (`n_total`), mean incidence (`incidence_mean`), observed variance (`V`), and binomial variance (`Vbin`).
- `model_summary`: A summary of the linear model fitted to the log-transformed variances.
- `hypothesis_test`: The result of the hypothesis test for the slope being equal to 1.
- `ln_Ap`: The intercept of the linear model, representing the natural logarithm of the parameter \(A_p \).
- `slope`: The slope of the linear model.

See Also

Other Spatial analysis: `AFSD()`, `count_subareas()`, `count_subareas_random()`, `fit_gradients()`, `join_count()`, `oruns_test()`, `oruns_test_boustrophedon()`, `oruns_test_byrowcol()`, `plot_AFSD()`

Examples

```
# Example usage with a sample data frame
result <- BPL(FHBWheat)
print(result$summary)
print(result$model_summary)
print(result$hypothesis_test)
print(paste("ln(Ap):", result$ln_Ap))
print(paste("Slope (b):", result$slope))
```

BudBlightSoybean *BudBlightSoybean dataset*

Description

Soybean bud blight incidence in experimental blocks.

Usage

```
BudBlightSoybean
```

Format

A data frame with the following columns:

block Block number

time Time point of assessment

treat Treatment name

y Incidence or severity value

Source

Del Ponte Lab internal data

`compare_curves` *Compare epidemic curves using GAM smoothing, functional distances, clustering, and optional permutation/bootstrapping*

Description

Fits a generalized additive model (GAM) to replicated disease progress curves. By default, the response is modeled with beta regression (logit link) after a Smithson–Verkuilen adjustment to handle 0 and 1 values; if beta precision (theta) estimation fails, the model falls back to a quasibinomial GAM fit on the unclipped proportion response.

From the fitted model, the function derives (i) environment-adjusted treatment mean curves on a common time grid, (ii) L2 functional distances among treatment mean trajectories followed by hierarchical clustering, and (iii) curve-level fitted trajectories (excluding unit random effects) to build a curve-by-curve distance matrix. Optionally, it performs a restricted permutation test on the curve-level distance matrix for a priori group labels, and computes bootstrap envelopes and distance uncertainty summaries from resampled predicted curves.

Usage

```
compare_curves(
  data,
  time,
  response,
  treatment,
  environment = NULL,
  block = NULL,
  unit = NULL,
  response_scale = c("proportion", "percent"),
  eps = 1e-04,
  min_points = 5,
  grid_n = 140,
  env_ref = NULL,
  k_smooth = 10,
  k_env = 4,
  k_trt = 6,
  gamma = 1.4,
  discrete = TRUE,
  family_try = c("betar", "quasibinomial"),
  cluster_k = 4,
  hc_method = "ward.D2",
  test_factor = NULL,
  n_perm = 999,
  test_mode = c("auto", "none", "global", "global_pairwise"),
```

```
  min_strata_for_pairwise = 6,
  perm_unit = NULL,
  perm_strata = NULL,
  bootstrap = FALSE,
  boot_B = 399,
  boot_seed = 1,
  boot_ci = c(0.025, 0.975),
  bootstrap_mode = c("predicted", "refit"),
  show_progress = TRUE,
  curve_level = NULL,
  ...
)
```

Arguments

data	A data.frame containing disease progress observations in long format.
time	Character string naming the time variable (numeric or coercible to numeric).
response	Character string naming the response variable (disease incidence/severity).
treatment	Character string naming the treatment/cultivar factor.
environment	Optional character string naming an environment factor (e.g., site-year).
block	Optional character string naming a blocking factor.
unit	Optional character string naming a unique experimental unit (curve) identifier. If NULL, a unit is constructed from the interaction of available factors (environment, treatment, and block).
response_scale	Character string specifying the response scale: "proportion" (0–1) or "percent" (0–100).
eps	Numeric small constant retained for API compatibility (not used for beta handling).
min_points	Minimum number of observations required per curve (unit) to be retained.
grid_n	Number of time points for evaluating predicted curves on a common grid.
env_ref	Reference environment level used for adjusted treatment mean predictions. If NULL, the first level of environment is used.
k_smooth	Basis dimension for the global smooth of time.
k_env	Basis dimension for the environment-specific smooth (if environment is provided).
k_trt	Basis dimension for the treatment-specific smooth.
gamma	Penalization parameter passed to mgcv::bam().
discrete	Logical; whether to use discrete (approximate) fitting in mgcv::bam().
family_try	Character string specifying the GAM family to try: "betar" (beta regression) or "quasibinomial". If "betar" is selected but precision estimation fails, the model falls back to "quasibinomial".
cluster_k	Number of clusters used to cut the hierarchical tree for treatments.
hc_method	Clustering linkage method passed to stats::hclust().

test_factor	Optional a priori grouping used for a distance-based permutation test. Either (i) a single character naming a column in data that maps uniquely to the permutation unit (see perm_unit), or (ii) a named vector whose names are permutation-unit identifiers and whose values are group labels.
n_perm	Number of permutations for the distance-based test.
test_mode	Character; permutation test mode. One of "auto", "none", "global", or "global_pairwise". In "auto", pairwise tests may be disabled when restricted strata support is limited.
min_strata_for_pairwise	Minimum number of strata levels (e.g., blocks) required to enable pairwise tests under test_mode = "auto".
perm_unit	Optional character naming the unit at which group labels are permuted (e.g., genotype/cultivar). If NULL, permutation is performed at the curve (unit) level.
perm_strata	Optional character naming a factor defining restricted permutation strata (e.g., block or environment). If NULL, defaults to block when provided.
bootstrap	Logical; if TRUE, compute bootstrap envelopes and distance summaries from resampled predicted curves.
boot_B	Integer number of bootstrap replicates.
boot_seed	Integer seed for bootstrap resampling.
boot_ci	Length-2 numeric vector of quantiles for bootstrap intervals (e.g., c(0.025, 0.975)).
bootstrap_mode	Character; bootstrap strategy. Currently supports "predicted" (resampling predicted curves). "refit" is not implemented in this version.
show_progress	Logical; whether to show progress bars for long-running tasks.
curve_level	Logical; whether to compute curve-level distance matrix. If NULL (default), it is automatically set to TRUE if test_factor is provided.
...	Reserved for future extensions.

Details

Functional distances among treatments are computed as an L2 norm over the predicted mean curves evaluated on a common time grid. Curve-level distances are computed analogously using fitted trajectories (excluding unit random effects), and are scaled by the median non-zero distance for numerical stability; interpret these distances as relative rather than absolute units.

The permutation test uses a PERMANOVA-like pseudo-F statistic computed from the curve-level distance matrix, with optional restricted permutations within strata. When enabled, pairwise comparisons are adjusted using Holm's method.

Value

An object of class "r4pde_compare_curves" containing:

- gam: fitted GAM object and family_used;
- pred: environment-adjusted treatment mean curves on the common grid;
- distance: treatment-by-treatment functional distance matrix and hc;

- `clusters`: treatment cluster assignments and summary scores;
- `curve_distance`: curve-by-curve distance matrix;
- `test`: optional distance-based permutation test results;
- `bootstrap`: optional bootstrap envelopes and distance summaries;
- diagnostic fields including `settings` and captured beta-fit warnings.

See Also

`plot_curves`, `plot_dendrogram`, `diagnose_curves`

`CompMuCens` *Survival analysis for quantitative ordinal scale data*

Description

Survival analysis for quantitative ordinal scale data

Usage

```
CompMuCens(dat, scale, grade = TRUE, ckData = FALSE)
```

Arguments

`dat`	Data frame containing the data to be processed.
`scale`	A numeric vector indicating the scale or order of classes.
`grade`	Logical. If TRUE, uses the class value. If FALSE, uses the NPE (Non-Parametric Estimate).
`ckData`	Logical. If TRUE, returns the input data along with the results. If FALSE, returns only the results.

Details

This function supports analysis of quantitative ordinal scale data via interval-censored methods. The approach follows the workflow described by Chiang et al. (2023) and the reference implementation provided in the CompMuCens repository.

Value

A list containing the score statistic, hypothesis tests, adjusted significance level, and conclusion based on pairwise comparisons.

References

Chiang, K.S., Chang, Y.M., Liu, H.I., Lee, J.Y., El Jarroudi, M. and Bock, C. (2023). Survival Analysis as a Basis to Test Hypotheses When Using Quantitative Ordinal Scale Disease Severity Data. Phytopathology. https://apsjournals.apsnet.org/doi/abs/10.1094/PHYTO-02-23-0055-R

See Also

Other Disease quantification: DSI(), DSI2()

Examples

```
if (requireNamespace("interval", quietly = TRUE)) {
  trAs <- c(5,4,2,5,5,4,4,2,5,2,2,3,4,3,2,2,6,2,2,4,2,4,2,4,5,3,4,2,2,3)
  trBs <- c(5,3,2,4,4,5,4,5,4,4,6,4,5,5,5,2,6,2,3,5,2,6,4,3,2,5,3,5,4,5)
  trCs <- c(2,3,1,4,1,1,4,1,1,3,2,1,4,1,1,2,5,2,1,3,1,4,2,2,2,4,2,3,2,2)
  trDs <- c(5,5,4,5,5,6,6,4,6,4,3,5,5,6,4,6,5,6,5,4,5,5,5,3,5,6,5,5,5,6)
  inputData <- data.frame(
    treatment = c(rep("A",30), rep("B",30), rep("C",30), rep("D",30)),
    x = c(trAs, trBs, trCs, trDs)
  )
  CompMuCens(dat = inputData,
             scale = c(0,3,6,12,25,50,75,88,94,97,100,100),
             ckData = TRUE)
}
```

---

count_subareas *Count the Number of Ones in Subareas of a Matrix*

Description

This function takes a binary matrix (0s and 1s) and divides it into rectangular subareas, counting the number of ones in each. Subareas are defined by the number of rows and columns specified by the user. If the matrix dimensions are not perfectly divisible by the subarea size, edge subareas may be smaller.

Usage

```
count_subareas(matrix_data, sub_rows, sub_cols)
```

Arguments

matrix_data	A matrix of 0s and 1s to analyze.
sub_rows	Number of rows in each subarea.
sub_cols	Number of columns in each subarea.

Value

A matrix where each cell corresponds to a subarea and contains the count of ones.

See Also

Other Spatial analysis: AFSD(), BPL(), count_subareas_random(), fit_gradients(), join_count(), oruns_test(), oruns_test_boustrophedon(), oruns_test_byrowcol(), plot_AFSD()

Examples

```
set.seed(123)
mat <- matrix(sample(c(0, 1), 12 * 16, replace = TRUE), nrow = 16, ncol = 12)
count_matrix <- count_subareas(mat, sub_rows = 3, sub_cols = 3)
print(count_matrix)
```

`count_subareas_random` *Random Subgrid Sampling of a Binary Matrix*

Description

Randomly samples submatrices (quadrats) of specified size from a binary matrix, and returns the positions, submatrices, and count of 1s in each sampled quadrat.

Usage

```
count_subareas_random(matrix_data, sub_rows = 3, sub_cols = 3, n_samples = 100)
```

Arguments

`matrix_data`	A binary matrix of 0s and 1s.
`sub_rows`	Number of rows in each subgrid sample.
`sub_cols`	Number of columns in each subgrid sample.
`n_samples`	Number of subgrid samples to draw.

Value

A list of sampled subgrids. Each element is a list with:

`position`	Row and column start position of the sample.
`submatrix`	The sampled subgrid matrix.
`count`	Number of 1s in the sampled submatrix.

See Also

Other Spatial analysis: `AFSD()`, `BPL()`, `count_subareas()`, `fit_gradients()`, `join_count()`, `oruns_test()`, `oruns_test_boustrophedon()`, `oruns_test_byrowcol()`, `plot_AFSD()`

diagnose_curves *Diagnostic tools for functional epidemic curve models*

Description

Computes residuals, fitted values, and model diagnostics for GAM-based epidemic curve models fitted with `compare_curves()`. Produces diagnostic plots without invoking base graphics.

Usage

```
diagnose_curves(x, grid_n = 200)
```

Arguments

`x`	An object of class `"functional_curves"` or `"r4pde_compare_curves"`.
`grid_n`	Number of points used for the diagnostic smooth curve.

Value

An object of class `"r4pde_curve_diagnostics"` containing:

- diagnostic data,
- k-index checks,
- concurvity estimates,
- ggplot-based diagnostic panels.

DidymellaWatermelon *DidymellaWatermelon dataset*

Description

Assessment of Didymella symptoms in watermelon plots.

Usage

```
DidymellaWatermelon
```

Format

A data frame with:

EW_row Row position (east–west)

NS_col Column position (north–south)

dap Days after planting

severity Disease severity

Source

Del Ponte Lab internal data

`DSI` *Calculate the Disease Severity Index (DSI) (class for each unit)*

Description

This function calculates the Disease Severity Index (DSI) based on the provided unit, class, and maximum class value. The DSI is computed by aggregating the classes, calculating weights by multiplying the frequency of each class by the class itself, and then dividing the sum of these weights by the product of the total number of entries and the maximum class value, then multiplying by 100.

Usage

```
DSI(unit, class, max)
```

Arguments

`unit`	A vector representing the units.
`class`	A vector representing the classes corresponding to the units.
`max`	A numeric value representing the maximum possible class value.

Value

Returns a single numeric value representing the DSI.

See Also

Other Disease quantification: `CompMuCens()`, `DSI2()`

Examples

```
# Example usage:
unit <- c(1, 2, 3, 4, 5, 6)
class <- c(1, 2, 1, 2, 3, 1)
max <- 3
DSI(unit, class, max)
```

DSI2 *Calculate the Disease severity Index (DSI) (frequency of each class)*

Description

This function calculates the Disease Severity Index (DSI) given a vector of classes, a vector of frequencies, and a maximum possible class value. The DSI is calculated as a weighted sum of class values, where each class is multiplied by its corresponding frequency, then divided by the product of the total frequency and maximum class value, and finally multiplied by 100 to get a percentage.

Usage

```
DSI2(class, freq, max)
```

Arguments

`class`	A numeric vector representing the classes.
`freq`	A numeric vector representing the frequency of each class. Must be the same length as 'class'.
`max`	A numeric value representing the maximum possible class value.

Value

Returns a single numeric value representing the DSI.

See Also

Other Disease quantification: `CompMuCens()`, `DSI()`

Examples

```
DSI2(c(0, 1, 2, 3, 4), c(2, 0, 5, 0, 5), 4)
```

FHBWheat *FHBWheat dataset*

Description

Fusarium head blight quadrat assessments in wheat.

Usage

```
FHBWheat
```

Format

A data frame with:

field Field identifier

i Row position

n Column position

quadrat Quadrat ID

season Crop season

Source

Del Ponte Lab internal data

`fit_gradients` *Fit Gradient Models to Data*

Description

This function fits three gradient models (exponential, power, and modified power) to given data. It then ranks the models based on their R-squared values and returns diagnostic plots for each model.

Usage

```
fit_gradients(data, C = 1)
```

Arguments

`data`	A dataframe containing the data, with columns "x" representing distances and "Y" representing the corresponding measurements or counts.
`C`	A constant to be used in the modified power model. Defaults to 1.

Value

A list containing:

`data`	The input data, which will include an additional column 'mod_x'.
`results_table`	A table of the model parameters and R-squared values.
`plot_exponential`	Diagnostic plot for the exponential model.
`plot_power`	Diagnostic plot for the power model.
`plot_modified_power`	Diagnostic plot for the modified power model.
`plot_exponential_original`	Plot of the original data with the exponential model fit.
`plot_power_original`	Plot of the original data with the power model fit.
`plot_modified_power_original`	Plot of the original data with the modified power model fit.

See Also

Other Spatial analysis: AFSD(), BPL(), count_subareas(), count_subareas_random(), join_count(), oruns_test(), oruns_test_boustrophedon(), oruns_test_byrowcol(), plot_AFSD()

Examples

```
x <- c(0.8, 1.6, 2.4, 3.2, 4, 7.2, 12, 15.2, 21.6, 28.8)
Y <- c(184.9, 113.3, 113.3, 64.1, 25, 8, 4.3, 2.5, 1, 0.8)
grad1 <- data.frame(x = x, Y = Y)
library(ggplot2)
mg <- fit_gradients(grad1, C = 0.4)
mg$plot_power_original +
  labs(title = "", x = "Distance from focus (m)", y = "Count of lesions")
```

---

functional_curves *Fit genotype-specific epidemic trajectories using GAM*

Description

Fits a generalized additive model (GAM) to replicated disease progress curves, returning adjusted treatment mean curves evaluated on a common time grid.

Usage

```
functional_curves(
  data,
  time,
  response,
  treatment,
  environment = NULL,
  block = NULL,
  unit = NULL,
  response_scale = c("proportion", "percent"),
  eps = 1e-04,
  min_points = 5,
  grid_n = 140,
  env_ref = NULL,
  k_smooth = 10,
  k_env = 4,
  k_trt = 6,
  gamma = 1.4,
  discrete = TRUE,
  family_try = c("betar", "quasibinomial"),
  show_progress = TRUE,
  covariates = NULL,
  include_covariates = FALSE,
```

```
  covariate_smooths = FALSE,
  ...
)
```

Arguments

data	A data.frame containing disease progress observations.
time	Character string naming the time variable.
response	Character string naming the response variable.
treatment	Character string naming the treatment/cultivar factor.
environment	Optional character string naming an environment factor.
block	Optional character string naming a blocking factor.
unit	Optional character string naming a unique experimental unit identifier.
response_scale	Character string specifying the response scale: "proportion" or "percent".
eps	Numeric small constant retained for API compatibility.
min_points	Minimum number of observations required per curve.
grid_n	Number of time points for evaluating predicted curves.
env_ref	Reference environment level used for adjusted predictions.
k_smooth	Basis dimension for the global smooth of time.
k_env	Basis dimension for the environment-specific smooth.
k_trt	Basis dimension for the treatment-specific smooth.
gamma	Penalization parameter passed to mgcv::bam().
discrete	Logical; whether to use discrete (approximate) fitting.
family_try	Character string specifying the GAM family to try.
show_progress	Logical; whether to show progress.
covariates	Optional character vector of genotype-level covariates (e.g., c("heading_group")).
include_covariates	Logical; whether to include covariates as fixed effects in the model.
covariate_smooths	Logical; if TRUE and include_covariates = TRUE, adds smooth interactions s(time, by=covariate) for factor covariates.
...	Additional arguments.

Details

Genotype-level covariates are descriptors that do not vary within a genotype, such as phenological groups. For example, in wheat blast studies, a cultivar's heading_group (early, intermediate, late) can be supplied. This helps distinguish between true genetic resistance and phenological escape, as cultivars with different heading dates may encounter different infection-risk windows in the same environment. When include_covariates = TRUE, the model accounts for these covariates, yielding heading-adjusted functional resistance.

Value

An object of class "functional_curves" containing:

- gam: fitted GAM object and family_used;
- curves: environment-adjusted treatment mean curves on the common grid;
- grid: the time grid used;
- observed_data: the processed input data;
- genotype_info: a tibble with one row per genotype containing covariates;
- vars: variable names used;
- settings: model settings;
- warnings_betar: any warnings caught during beta fitting;
- plot_mean: ggplot object of the mean curves.

Examples

```
## Not run:
fc <- functional_curves(
  data = my_data,
  time = "time_var",
  response = "severity",
  treatment = "cultivar",
  environment = "env",
  covariates = c("heading_group"),
  include_covariates = TRUE,
  covariate_smooths = TRUE
)
plot(fc)

## End(Not run)
```

---

functional_distances *Compute pairwise functional distances and cluster curves*

Description

Computes L2 functional distances among treatment mean trajectories evaluated on a common time grid from a functional_curves object. Also computes curve-level distances, performs hierarchical clustering, and optionally performs permutation testing.

Usage

```
functional_distances(
  object,
  cluster_k = 4,
  hc_method = "ward.D2",
  test_factor = NULL,
```

```
  n_perm = 999,
  test_mode = c("auto", "none", "global", "global_pairwise"),
  min_strata_for_pairwise = 6,
  perm_unit = NULL,
  perm_strata = NULL,
  bootstrap = FALSE,
  boot_B = 399,
  boot_seed = 1,
  boot_ci = c(0.025, 0.975),
  show_progress = TRUE,
  curve_level = NULL,
  ...
)
```

Arguments

`object`	An object of class `functional_curves`.
`cluster_k`	Number of clusters used to cut the hierarchical tree.
`hc_method`	Clustering linkage method passed to `stats::hclust()`.
`test_factor`	Optional a priori grouping used for a distance-based permutation test.
`n_perm`	Number of permutations for the distance-based test.
`test_mode`	Character; permutation test mode.
`min_strata_for_pairwise`	Minimum number of strata levels required for pairwise tests.
`perm_unit`	Optional character naming the unit at which group labels are permuted.
`perm_strata`	Optional character naming a factor defining restricted permutation strata.
`bootstrap`	Logical; if `TRUE`, compute bootstrap envelopes and distance summaries.
`boot_B`	Integer number of bootstrap replicates.
`boot_seed`	Integer seed for bootstrap resampling.
`boot_ci`	Length-2 numeric vector of quantiles for bootstrap intervals.
`show_progress`	Logical; whether to show progress.
`curve_level`	Logical; whether to compute curve-level distance matrix.
`...`	Additional arguments.

Value

An object of class `"functional_distances"` containing:

- `distance`: treatment-by-treatment functional distance matrix;
- `distance_table`: pairwise distances as a data frame;
- `hc`: `hclust` object;
- `clusters`: treatment cluster assignments;
- `curve_distance`: curve-by-curve distance matrix;
- `test`: permutation test results;
- `bootstrap`: bootstrap results if requested.

Examples

```
## Not run:
fd <- functional_distances(fc, cluster_k = 4)
print(fd)
plot_dendrogram(fd)
plot_curves(fd)

## End(Not run)
```

`functional_instability` *Normalized functional instability from compare_curves output*

Description

Computes normalized functional instability (NFI) for each treatment based on genotype-by-environment predicted curves extracted from a `compare_curves()` object. Optionally, instability can be decomposed into spatial and temporal components if the environment identifier can be split into location and year.

Usage

```
functional_instability(
  x,
  n_time = 200,
  env_sep = NULL,
  env_names = c("location", "year"),
  return_curves = FALSE
)
```

Arguments

`x`	An object returned by `functional_curves` or `compare_curves`.
`n_time`	Number of points in the prediction grid over the time domain.
`env_sep`	Optional separator used to split `env` into spatial and temporal components, for example `"_"` or `"-"`. If `NULL`, only the overall NFI is returned.
`env_names`	A character vector of length 2 giving the names of the components obtained after splitting `env`. Default is `c("location", "year")`.
`return_curves`	Logical. If `TRUE`, the predicted genotype-by-environment curves are also returned.

Details

The overall instability metric is defined as the mean integrated squared deviation of each genotype-by-environment curve from the genotype-specific mean curve across environments, normalized by the integrated squared mean curve.

Let $f_{ge}(t)$ denote the predicted epidemic trajectory of genotype g in environment e, and let $\bar{f}_g(t)$ denote the mean trajectory of genotype g across environments. Functional instability is computed as:

$$FI_g = \frac{1}{E_g} \sum_{e=1}^{E_g} \int_T \left(f_{ge}(t) - \bar{f}_g(t) \right)^2 dt$$

and normalized as:

$$nFI_g = \frac{FI_g}{\int_T \bar{f}_g(t)^2 dt}$$

Numerical integration is performed with the trapezoidal rule on a regular prediction grid over the observed time domain.

Value

If `return_curves = FALSE`, a tibble with one row per genotype and columns:

geno Treatment or genotype identifier.

n_env Number of environments used in the calculation.

FI Absolute functional instability.

mean_energy Integrated squared mean epidemic curve.

nFI Normalized functional instability. Lower values indicate greater stability.

If `env_sep` is provided, the returned tibble also includes:

FI_space, mean_energy_space, nFI_space Spatial component of instability.

FI_time, mean_energy_time, nFI_time Temporal component of instability.

If `return_curves = TRUE`, a list is returned with two elements:

metrics The tibble described above.

curves A tibble of predicted genotype-by-environment curves with columns `geno`, `env`, `time`, `mu`, and `eta`.

See Also

`functional_curves`, `compare_curves`

Examples

```
## Not run:
m1 <- r4pde::compare_curves(
  data = dat_ready,
  time = "time",
  response = "y",
  treatment = "geno",
  environment = "env",
  cluster_k = 4
)

# Overall instability
res <- functional_instability(m1)
res

# Overall + spatial and temporal components
# Assuming env has values such as "PF_2021"
res2 <- functional_instability(m1, env_sep = "_")
res2

# Return curves used in the calculations
out <- functional_instability(m1, env_sep = "_", return_curves = TRUE)
out$metrics
head(out$curves)

## End(Not run)
```

`functional_pca` *Functional principal component analysis of disease progress curves*

Description

Performs functional principal component analysis on fitted disease progress curves returned by `functional_curves`. The function decomposes variation among epidemic trajectories into orthogonal temporal components and returns curve-level scores, eigenfunctions, variance explained, and reconstructed curves.

Usage

```
functional_pca(
  object,
  n_components = NULL,
  var_explained = 0.95,
  center = TRUE,
  scale = FALSE,
  method = c("pca_on_grid"),
  ...
)
```

Arguments

object	An object returned by functional_curves.
n_components	Optional integer number of functional principal components to retain.
var_explained	Cumulative variance threshold used when n_components = NULL.
center	Logical; whether to center curves before PCA. Default TRUE.
scale	Logical; whether to scale grid columns before PCA. Default FALSE.
method	Character; method for FPCA, currently only "pca_on_grid" is supported.
...	Additional arguments for future extensions.

Details

The function uses the fitted curves from functional_curves() and does not refit the disease progress model. The first implementation uses PCA on a common prediction grid. FPC scores can be analyzed as functional epidemiological traits in downstream models.

Interpretation:

- FPC1 often captures the largest mode of variation, commonly overall epidemic intensity or speed.
- Later FPCs may capture timing of disease onset, curve crossing, late acceleration, or other shape-related deviations.
- Interpretation must be data-driven and should be based on eigenfunction plots and mean ± perturbation plots.

Value

An object of class "r4pde_functional_pca" containing:

- scores: Tibble of curve-level FPC scores.
- eigenfunctions: Tibble of eigenfunction values across the time grid.
- variance: Tibble of eigenvalues, variance explained, and cumulative variance.
- mean_curve: Tibble of the mean curve.
- reconstructed: Tibble of reconstructed curves using retained components.
- input_curves: Tibble of original fitted curves.
- pca: The underlying prcomp object.
- settings: List of settings used.
- call: The matched call.

Examples

```
## Not run:
curves <- functional_curves(...)
fpca <- functional_pca(curves, var_explained = 0.95)

print(fpca)
plot(fpca, type = "scree")
```

```
plot(fpca, type = "components")
plot(fpca, type = "scores", components = c(1, 2))

scores <- get_fpca_scores(fpca)

## End(Not run)
```

`functional_pca_clusters` *Cluster curves based on FPCA scores*

Description

Cluster curves based on FPCA scores

Usage

```
functional_pca_clusters(
  x,
  k = NULL,
  components = NULL,
  method = c("kmeans", "hclust"),
  choose_k = c("none", "silhouette", "elbow"),
  ...
)
```

Arguments

`x`	An object of class `"r4pde_functional_pca"`.
`k`	Number of clusters.
`components`	Integer vector of components to use.
`method`	Clustering method: "kmeans" or "hclust".
`choose_k`	Method for suggesting k: "none", "silhouette", or "elbow".
`...`	Additional arguments passed to clustering functions.

`functional_resistance` *Compute functional resistance and stability-adjusted functional resistance*

Description

Computes a functional resistance index (FRI) relative to a reference genotype curve. Optionally adjusts FRI by a functional instability penalty to calculate a stability-adjusted functional resistance index (SAFRI). Supports grouping genotypes into classes based on quantiles, clustering, or bootstrap-supported differences.

Usage

```
functional_resistance(
  object,
  instability = NULL,
  reference,
  lambda = 1,
  method = c("positive_area", "l2_difference"),
  scale_nfi = FALSE,
  n_groups = 4,
  group_method = c("none", "quantile", "bootstrap", "clustering"),
  time_var = NULL,
  genotype_var = NULL,
  n_boot = 1000,
  ci_level = 0.95,
  clustering_method = c("hclust", "kmeans"),
  adjust_by = NULL,
  reference_within_group = FALSE,
  group_within_adjust_by = TRUE,
  ...
)
```

Arguments

object	An object of class functional_curves.
instability	Optional output from functional_instability().
reference	Character string naming the reference genotype, or a named vector if reference_within_group = TRUE.
lambda	Numeric penalty weight for instability. Default is 1.
method	Character string for distance method.
scale_nfi	Logical; if TRUE, normalizes nFI to a 0-1 scale before applying penalty.
n_groups	Integer number of classes to group into.
group_method	Character string for the grouping method.
time_var	Optional variable name overrides.
genotype_var	Optional variable name overrides.
n_boot	Integer number of bootstrap iterations.
ci_level	Numeric confidence level for bootstrap intervals.
clustering_method	Character string for clustering method if group_method = "clustering".
adjust_by	Optional character string naming a genotype-level covariate to stratify by.
reference_within_group	Logical; if TRUE, the reference is evaluated within each adjust_by group.
group_within_adjust_by	Logical; if TRUE, resistance classes are assigned within each adjust_by group.
...	Additional arguments.

Value

A list with a `table` containing genotype, FRI, nFI, SAFRI, rank, and classes, and optionally bootstrap results.

Examples

```
## Not run:
fr <- functional_resistance(
  fc,
  reference = "Susceptible",
  group_method = "bootstrap"
)
print(fr)

## End(Not run)
```

FusariumBanana *FusariumBanana dataset*

Description

Observations of Fusarium symptoms in banana fields.

Usage

```
FusariumBanana
```

Format

A data frame with:

field Field ID

lat Latitude

lon Longitude

marker Infection marker presence

Source

Del Ponte Lab internal data

get_brdwgd *Fetch BR-DWGD Data for Multiple Locations*

Description

This function extracts daily weather data from the Brazilian Daily Weather Gridded Data (BR-DWGD) NetCDF files for specified locations and dates. It mimics the functionality of `get_nasapower` and `get_era5` by returning a daily-aggregated data frame with columns named according to `nasapower` conventions.

Usage

```
get_brdwgd(
  data,
  days_around,
  date_col,
  study_col = "study",
  path = "netcdf_files",
  vars = c("pr", "Tmax", "Tmin", "Rs", "RH"),
  direction = "both"
)
```

Arguments

`data`	A data frame containing the input data, including columns for `latitude`, `longitude`, and the date column specified by `date_col`.
`days_around`	An integer specifying the number of days before and after the date in the date column to download data.
`date_col`	A character string specifying the name of the date column in the data frame.
`study_col`	A character string specifying the name of the column containing the study identifier in the input data frame (default: "study").
`path`	A character string specifying the directory where the BR-DWGD NetCDF files are stored.
`vars`	A character vector specifying the weather variables to fetch. These are expected to be the prefix of the NetCDF files (e.g., "pr", "Tmax"). Default: `c("pr", "Tmax", "Tmin", "Rs", "RH")`.
`direction`	Character string specifying the direction of the date range relative to the reference date. Options are "both" (default), "back", or "forth". "back" retrieves data from `date - days_around` to `date`. "forth" retrieves data from `date` to `date + days_around`. "both" retrieves data from `date - days_around` to `date + days_around`.

Details

The function requires the `terra` package to read NetCDF files and the `tidyr` package for data reshaping. It expects NetCDF files to be named starting with the variable name (e.g., `pr_*.nc`) and to contain the variable with the same name.

Value

A data frame (tibble) with daily weather data. Columns are named to match nasapower conventions where possible: date, PRECTOTCORR (from pr), T2M_MAX (from Tmax), T2M_MIN (from Tmin), ALLSKY_SFC_SW_DWN (from Rs), RH2M (from RH), longitude, latitude, and study.

See Also

Other Disease modeling: get_era5(), get_nasapower(), windowpane()

---

get_era5 *Fetch ERA5 Data from Open-Meteo for Multiple Locations*

Description

This function downloads ERA5 historical weather data from the Open-Meteo API for specified locations and dates. It mimics the functionality of get_nasapower by fetching weather variables and returning a daily-aggregated data frame. It uses the Open-Meteo Archive API and aggregates hourly data to daily values to ensure all variables (including relative humidity and dew point) are available.

Usage

```
get_era5(
  data,
  days_around,
  date_col,
  study_col = "study",
 pars = c("temperature_2m", "relative_humidity_2m", "precipitation", "dewpoint_2m"),
  models = NULL,
  direction = "both"
)
```

Arguments

data	A data frame containing the input data, including columns for latitude, longitude, and the date column specified by date_col.
days_around	An integer specifying the number of days before and after the date in the date column to download data.
date_col	A character string specifying the name of the date column in the data frame.
study_col	A character string specifying the name of the column containing the study identifier in the input data frame (default: "study").
pars	A character vector specifying the weather variables to fetch from Open-Meteo. These are mapped to nasapower equivalents: "temperature_2m" (T2M), "relative_humidity_2m" (RH2M), "precipitation" (PRECTOTCORR), etc.
models	A character string specifying the reanalysis model(s) to use. If NULL (default), Open-Meteo uses its "best_match" logic (ERA5, ERA5-Land, etc.).

direction — Character string specifying the direction of the date range relative to the reference date. Options are "both" (default), "back", or "forth". "back" retrieves data from date - days_around to date. "forth" retrieves data from date to date + days_around. "both" retrieves data from date - days_around to date + days_around.

Details

Since the Open-Meteo Archive API daily endpoint does not provide all variables (like mean relative humidity) directly, this function fetches hourly data and performs the daily aggregation manually:

- T2M: Mean of hourly temperature_2m
- T2M_MAX: Max of hourly temperature_2m
- T2M_MIN: Min of hourly temperature_2m
- RH2M: Mean of hourly relative_humidity_2m
- PRECTOTCORR: Sum of hourly precipitation
- T2MDEW: Mean of hourly dewpoint_2m

Value

A data frame (tibble) with daily weather data. Columns are named to match nasapower conventions: date, T2M, T2M_MAX, T2M_MIN, RH2M, PRECTOTCORR, T2MDEW, longitude, latitude, and study.

See Also

Other Disease modeling: get_brdwgd(), get_nasapower(), windowpane()

---

get_fpca_eigenfunctions

Get FPCA eigenfunctions

Description

Get FPCA eigenfunctions

Usage

```
get_fpca_eigenfunctions(x, components = NULL)
```

Arguments

x — An object of class "r4pde_functional_pca".

components — Optional integer vector of components to return.

get_fpca_scores *Get FPCA scores*

Description

Get FPCA scores

Usage

```
get_fpca_scores(x, components = NULL)
```

Arguments

x	An object of class "r4pde_functional_pca".
components	Optional integer vector of components to return.

get_fpca_variance *Get FPCA variance explained*

Description

Get FPCA variance explained

Usage

```
get_fpca_variance(x)
```

Arguments

x	An object of class "r4pde_functional_pca".

get_nasapower *Fetch NASA POWER Data for Multiple Locations with a Progress Bar*

Description

This function downloads daily NASA POWER data for specified weather variables over a specified number of days around a given date column for multiple locations. It includes a progress bar to show the download progress.

Usage

```
get_nasapower(
  data,
  days_around,
  date_col,
  study_col = "study",
  pars = c("T2M", "RH2M", "PRECTOTCORR", "T2M_MAX", "T2M_MIN", "T2MDEW"),
  direction = "both"
)
```

Arguments

`data`	A data frame containing the input data, including columns for latitude, longitude, and the date column.
`days_around`	An integer specifying the number of days before and after the date in the date column to download data.
`date_col`	A character string specifying the name of the date column in the data frame.
`study_col`	A character string specifying the name of the column containing the study identifier in the input data frame (default: "study").
`pars`	A character vector specifying the weather variables to fetch from NASA POWER (default: c("T2M", "RH2M", "PRECTOTCORR", "T2M_MAX", "T2M_MIN", "T2MDEW")).
`direction`	Character string specifying the direction of the date range relative to the reference date. Options are "both" (default), "back", or "forth". "back" retrieves data from `date - days_around` to `date`. "forth" retrieves data from `date` to `date + days_around`. "both" retrieves data from `date - days_around` to `date + days_around`.

Details

The function uses the `get_power` function from the `nasapower` package to fetch weather data for a range of dates around the specified date column for each location. A progress bar is shown during the data download process, and the results are combined into a single data frame.

Value

A data frame with the downloaded weather data from NASA POWER, combined for all specified locations. Includes a variable `study` indicating the study identifier from the input data. Returns an empty data frame if no data is retrieved.

See Also

Other Disease modeling: `get_brdwgd()`, `get_era5()`, `windowpane()`

`join_count` *Test for Spatial Join Count Statistics*

Description

The function `join_count` calculates spatial join count statistics for a binary matrix, identifying patterns of aggregation or randomness.

Usage

```
join_count(matrix_data, verbose = TRUE)
```

Arguments

`matrix_data`	A binary matrix (with elements 0 and 1) representing the spatial distribution of two types of points: 0 for healthy plants (H) and 1 for diseased plants (D). This matrix reflects the geographical distribution or layout of plants in the studied area.
`verbose`	Logical. If TRUE (default), prints a formatted message to the console.

Details

The function conducts an analysis by first counting the occurrence of specific sequences ("01 or 10" and "11" - equivalent to HD and DD) in the binary matrix. It then calculates expected values, standard deviations, and Z-scores to determine the spatial randomness or aggregation. The analysis considers both horizontal and vertical adjacency (rook case) in the matrix.

Value

A comprehensive, rich-text formatted string of results that includes:

- Statistical counts of specific binary sequences (e.g., "01 or 10", "11")
- Expected counts under the assumption of Complete Spatial Randomness (CSR)
- Standard deviations and Z-scores (ZHD for "01 or 10" sequences, ZDD for "11" sequences)
- Interpretation of whether the spatial distribution for each sequence type is "Aggregated" or "Not Aggregated" based on Z-scores
- A summary explaining the implications of these statistics and patterns

The return value aims to provide a clear understanding of the spatial arrangement's characteristics, aiding in further spatial analysis or research.

References

Madden, L. V., Hughes, G., & van den Bosch, F. (2007). The Study of Plant Disease Epidemics. The American Phytopathological Society.

See Also

Other Spatial analysis: AFSD(), BPL(), count_subareas(), count_subareas_random(), fit_gradients(), oruns_test(), oruns_test_boustrophedon(), oruns_test_byrowcol(), plot_AFSD()

`oruns_test` *Runs Test*

Description

Perform a runs test on the input data to test for clustering or randomness.

Usage

```
oruns_test(x)
```

Arguments

x	A numeric vector representing the input data

Value

an `r4pde.oruns` object.

An `r4pde.oruns` object is a `list` containing:

- U, number of runs,
- EU, expected number of runs,
- sU, standard deviation of the expected number of runs
- Z, Z-score of the observed number of runs,
- pvalue, the p-value of the Z-score, and
- result, the test result of either "aggregation or clustering" or "randomness"

See Also

Other Spatial analysis: AFSD(), BPL(), count_subareas(), count_subareas_random(), fit_gradients(), join_count(), oruns_test_boustrophedon(), oruns_test_byrowcol(), plot_AFSD()

Examples

```
oruns_test(c(1, 0, 1, 1, 0, 1, 0, 0, 1, 1))
```

oruns_test_boustrophedon *Boustrophedon Run Test for Binary Matrix*

Description

Applies the ordinary runs test to a binary matrix using boustrophedon-style traversal. The function supports two modes: row-wise and column-wise boustrophedon. Each traversal flattens the matrix into a 1D sequence which is then tested using `oruns_test`.

Usage

```
oruns_test_boustrophedon(mat)
```

Arguments

`mat`	A binary matrix (containing 0s and 1s, and possibly NAs).

Value

A list with two elements:

`rowwise_boustrophedon`
: List containing the sequence and result of `oruns_test` for row-wise traversal.

`colwise_boustrophedon`
: List containing the sequence and result of `oruns_test` for column-wise traversal.

See Also

`oruns_test`

Other Spatial analysis: `AFSD()`, `BPL()`, `count_subareas()`, `count_subareas_random()`, `fit_gradients()`, `join_count()`, `oruns_test()`, `oruns_test_byrowcol()`, `plot_AFSD()`

oruns_test_byrowcol *Runs Test for Each Row and Column of a Binary Matrix*

Description

Applies the ordinary runs test to each row and column of a binary matrix individually.

Usage

```
oruns_test_byrowcol(mat)
```

Arguments

mat	A binary matrix (containing 0s and 1s, and possibly NAs).

Value

A list with four elements:

row_results	Data frame with test results for each row.
col_results	Data frame with test results for each column.
row_summary	Percentage summary of interpretation for rows.
col_summary	Percentage summary of interpretation for columns.

See Also

oruns_test

Other Spatial analysis: AFSD(), BPL(), count_subareas(), count_subareas_random(), fit_gradients(), join_count(), oruns_test(), oruns_test_boustrophedon(), plot_AFSD()

---

plot.functional_curves

Plot functional_curves

Description

Plot functional_curves

Usage

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

---

plot.r4pde_compare_curves

Plot method for compare_curves objects

Description

Produces a paired visualization of environment-adjusted mean curves and the corresponding functional dendrogram.

Usage

```
## S3 method for class 'r4pde_compare_curves'
plot(x, label_fun = NULL, palette = NULL, ...)
```

Arguments

x	An object of class "r4pde_compare_curves".
label_fun	Optional function to modify treatment labels.
palette	Optional named vector of colors for clusters.
...	Additional arguments (currently ignored).

Value

A patchwork object combining curves and dendrogram.

---

plot.r4pde_functional_pca

Plot functional PCA results

Description

Plot functional PCA results

Usage

```
## S3 method for class 'r4pde_functional_pca'
plot(
  x,
  type = c("scree", "components", "scores", "reconstruction", "mean"),
  components = c(1, 2),
  curve_id = NULL,
  ...
)
```

Arguments

x	An object of class "r4pde_functional_pca".
type	Type of plot: "scree", "components", "scores", "reconstruction", or "mean".
components	Integer vector of length 2 indicating which components to plot for scores and mean perturbation.
curve_id	Optional vector of curve IDs to include in reconstruction plot.
...	Additional arguments.

plot_AFSD *Plot ASFD*

Description

This function creates a tile plot of the foci (cluster) identified by the AFSD function. It colors each cell in a foci and labels the centroid of each cluster with the foci ID. The 'ggplot2' package is used for the plot, and will be automatically installed if not already present.

Usage

```
plot_AFSD(df)
```

Arguments

df	A dataframe containing at least three columns: 'x', 'y', and 'cluster_id'. 'x' and 'y' are spatial coordinates and 'cluster_id' is the cluster identifier to which each cell belongs.

Value

A ggplot object with the scatter plot of foci (clusters).

See Also

Other Spatial analysis: AFSD(), BPL(), count_subareas(), count_subareas_random(), fit_gradients(), join_count(), oruns_test(), oruns_test_boustrophedon(), oruns_test_byrowcol()

Examples

```
df <- data.frame(x = sample(1:100, 500, replace = TRUE),
                 y = sample(1:100, 500, replace = TRUE),
                 i = sample(0:1, 500, replace = TRUE, prob = c(0.7, 0.3)))

# Perform the AFSD
result <- AFSD(df)
# Plot the foci
plot_AFSD(result[[3]])
```

plot_curves *Plot environment-adjusted epidemic curves by cluster*

Description

Plots environment-adjusted mean epidemic curves for each treatment, colored according to functional cluster membership.

The returned object is a ggplot and can be further modified using standard ggplot2 layers.

Usage

```
plot_curves(x, label_fun = NULL, palette = NULL, alpha = 0.9, linewidth = 1.1)
```

Arguments

x	An object of class "r4pde_compare_curves" or "functional_distances".
label_fun	Optional function to modify treatment labels.
palette	Optional named vector of colors for clusters.
alpha	Line transparency.
linewidth	Line width.

Value

A ggplot object.

plot_dendrogram *Plot functional dendrogram of epidemic curves*

Description

Visualizes the hierarchical clustering of treatments based on functional distances among epidemic curves.

Usage

```
plot_dendrogram(x, label_fun = NULL, palette = NULL, show_cut = TRUE)
```

Arguments

x	An object of class "r4pde_compare_curves" or "functional_distances".
label_fun	Optional function to modify treatment labels.
palette	Optional named vector of colors for clusters.
show_cut	Logical; whether to display the cluster cut height.

Value

A ggplot object.

---

plot_diagnostics *Plot diagnostics for functional epidemic curve models*

Description

Combines multiple diagnostic plots from diagnose_curves into a multi-panel layout.

Usage

```
plot_diagnostics(
  diag,
  layout = c("2x2", "vertical", "horizontal"),
  show_curve = FALSE,
  rel_heights = c(1, 0.7)
)
```

Arguments

diag	An object of class "r4pde_curve_diagnostics".
layout	Layout of diagnostic panels: "2x2", "vertical", or "horizontal".
show_curve	Logical; whether to include the example adjusted curve.
rel_heights	Relative heights for panels when stacking.

Value

A composite plot object generated using **cowplot**.

---

plot_functional_instability *Plot method for functional instability results*

Description

Plot method for functional instability results

Usage

```
plot_functional_instability(object, type = c("overall", "space", "time"), ...)
```

Arguments

object	Output from functional_instability().
type	One of "overall", "space", or "time".
...	Further arguments passed to methods.

Value

A ggplot object.

---

print.functional_curves *Print functional_curves*

Description

Print functional_curves

Usage

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

---

print.functional_distances *Print functional_distances*

Description

Print functional_distances

Usage

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

print.functional_resistance *Print functional_resistance*

Description

Print functional_resistance

Usage

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

print.r4pde_curve_diagnostics *Print method for curve diagnostics*

Description

Print method for curve diagnostics

Usage

```
## S3 method for class 'r4pde_curve_diagnostics'
print(x, ...)
```

Arguments

x	An object of class "r4pde_curve_diagnostics".
...	Additional arguments (ignored).

Value

Invisibly returns x.

print.suggest_k *Print method for suggest_k*

Description

Print method for suggest_k

Usage

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

Arguments

x	An object of class "suggest_k".
...	Additional arguments (ignored).

Value

Invisibly returns x.

reconstruct_curves *Reconstruct curves using specified FPCA components*

Description

Reconstruct curves using specified FPCA components

Usage

```
reconstruct_curves(x, components = NULL)
```

Arguments

x	An object of class "r4pde_functional_pca".
components	Integer vector of components to use for reconstruction. If NULL, uses all retained components.

RustSoybean *RustSoybean dataset*

Description

Soybean rust severity and field metadata.

Usage

```
RustSoybean
```

Format

A data frame with:

detection Detection score or date

epidemia Epidemic phase

latitude Latitude

local Location name

longitude Longitude

planting Planting date or stage

severity Disease severity

Source

Del Ponte Lab internal data

SpatialAggregated *SpatialAggregated dataset*

Description

Simulated aggregated spatial binary disease pattern.

Usage

```
SpatialAggregated
```

Format

A data frame with:

x x-coordinate

y y-coordinate

Source

Simulated example

`SpatialRandom` *SpatialRandom dataset*

Description

Simulated random spatial binary disease pattern.

Usage

```
SpatialRandom
```

Format

A data frame with:

x x-coordinate

y y-coordinate

Source

Simulated example

`suggest_k` *Suggest GAM smoothing parameters for epidemic curve models*

Description

A helper function to guide selection of the `k_smooth`, `k_trt`, `k_env`, and `gamma` parameters used by `functional_curves`. Can infer the effective replication from a data frame or accept scalar values directly when the data are not yet available.

For sparse epidemic data it is strongly recommended to use `rule = "minimum"` so that k values are based on the least-replicated treatment-by-environment combination, guarding against over-fitting.

Usage

```
suggest_k(
  data = NULL,
  time = NULL,
  treatment = NULL,
  environment = NULL,
  n_time = NULL,
  n_env = NULL,
  rule = c("minimum", "median"),
  smoothness = c("conservative", "moderate", "flexible")
)
```

Arguments

data	Optional data frame. If supplied, time, treatment, and optionally environment are used to compute summaries of the number of unique time points and environments per treatment.
time	Unquoted column name for the time variable, or a character string naming the column.
treatment	Unquoted column name for the treatment / cultivar variable, or a character string naming the column.
environment	Optional unquoted column name for the environment variable, or a character string naming the column.
n_time	Integer. Number of unique time points to use directly (ignored when data is supplied).
n_env	Integer. Number of unique environments to use directly (ignored when data is supplied, or when there is no environment variable).
rule	Character string. How to summarise the distribution of replication counts across treatment-by-environment combinations: "minimum" (default, conservative) or "median".
smoothness	Character string controlling how liberal the recommendations are: "conservative" (default), "moderate", or "flexible".

Details

The function computes, for every treatment-by-environment combination, the number of unique time points at which observations are available. It then summarises these counts using the chosen rule to obtain effective_n_time. Similarly it computes, per treatment, the number of unique environments, and summarises using the same rule to obtain effective_n_env.

Recommended k values follow these heuristics:

- k_smooth: global smooth over time; capped below effective_n_time - 1.
- k_trt: treatment-specific smooth; capped below k_smooth.
- k_env: environment random effect; capped below effective_n_env.
- gamma: penalty multiplier; higher values encourage smoother fits.

Value

A named list with the following elements:

time_summary Named numeric vector with minimum, median, and maximum unique time points per treatment-by-environment combination.

environment_summary Named numeric vector with minimum, median, and maximum unique environments per treatment (or NULL when no environment variable is given).

effective_n_time The effective number of time points chosen by rule.

effective_n_env The effective number of environments chosen by rule, or NULL.

k_smooth Recommended basis dimension for the global time smooth.

`k_trt` Recommended basis dimension for the treatment-specific smooth.

`k_env` Recommended basis dimension for the environment random effect.

`gamma` Recommended penalisation multiplier.

`message` A short interpretation message.

Examples

```
# Using explicit values
suggest_k(n_time = 5, n_env = 8)

# Inferring from a data frame (unquoted column names)
df <- data.frame(
  time      = rep(1:6, times = 6),
  cultivar  = rep(c("A", "B", "C"), each = 12),
  env       = rep(rep(c("E1", "E2"), each = 6), times = 3),
  severity  = runif(36, 0, 0.5)
)
suggest_k(
  data        = df,
  time        = time,
  treatment   = cultivar,
  environment = env,
  rule        = "minimum",
  smoothness  = "conservative"
)
```

`theme_r4pde` *Custom ggplot2 theme based on cowplot::theme_half_open*

Description

This function creates a new ggplot2 theme by modifying the cowplot::theme_half_open theme. It sets a custom font size and changes the panel background color to gray96.

Usage

```
theme_r4pde(font_size = 16)
```

Arguments

`font_size` The base font size. Default is 16.

Value

A ggplot2 theme object.

`WhiteMoldSoybean` *WhiteMoldSoybean dataset*

Description

National dataset of white mold severity and yield.

Usage

```
WhiteMoldSoybean
```

Format

A data frame with:

country Country name

elevation Field elevation

elevation_class Elevation class

harvest_year Year of harvest

inc Incidence

inc_check Check plot incidence

inc_class Incidence class

location Location name

region Geographical region

scl Soybean canopy layer

season Crop season

state State name

study Study identifier

treat Treatment applied

yld Yield

yld_check Yield of untreated check

yld_class Yield class

Source

Del Ponte Lab internal data

windowpane *Window Pane for Epidemiological Analysis*

Description

This function calculates summary statistics within specified windows around a given end date in a dataset, facilitating epidemiological analysis. It allows backward, forward, or both directions of window calculations based on a user-defined variable and window lengths.

Usage

```
windowpane(
  data,
  end_date_col,
  date_col,
  variable,
  summary_type,
  threshold = NULL,
  window_lengths,
  direction = "backward",
  group_by_cols = NULL,
  date_format = "%Y-%m-%d"
)
```

Arguments

data	A data frame containing the input data.
end_date_col	A string specifying the name of the column representing the end date.
date_col	A string specifying the name of the column representing the date variable.
variable	A string specifying the name of the column for which summary statistics are calculated.
summary_type	A string specifying the type of summary to calculate. Options are "mean", "sum", "above_threshold", or "below_threshold".
threshold	Optional numeric value used when summary_type is "above_threshold" or "below_threshold".
window_lengths	A numeric vector specifying the window lengths (in days) for the calculations.
direction	A string specifying the direction of the window. Options are "backward" (default), "forward", or "both".
group_by_cols	Optional vector of strings specifying column names for grouping the data.
date_format	A string specifying the format of the date columns. Default is "%Y-%m-%d".

Value

A data frame with the calculated summary values for each window.

See Also

Other Disease modeling: `get_brdwgd()`, `get_era5()`, `get_nasapower()`

`windowpane_tests` *Windowpane Tests for Correlation Analysis*

Description

This function performs bootstrapped correlation analysis for multiple predictors against a response variable. It applies the Simes method for global significance testing and calculates individual correlations, p-values, and bootstrap statistics.

Usage

```
windowpane_tests(
  data,
  response_var,
  corr_type = "spearman",
  R = 1000,
  global_alpha = 0.05,
  individual_alpha = 0.005
)
```

Arguments

`data`	A data frame containing the predictors and the response variable.
`response_var`	A string representing the name of the response variable in the data frame.
`corr_type`	A string specifying the correlation method to use; options are "spearman" (default), "pearson", or "kendall".
`R`	An integer indicating the number of bootstrap replications. Default is 1000.
`global_alpha`	A numeric value representing the global alpha level for the Simes correction. Default is 0.05.
`individual_alpha`	A numeric value for the individual alpha threshold for testing individual predictors. Default is 0.005.

Details

The function calculates correlations between the response variable and each predictor in the data frame, using bootstrapping to generate mean, standard deviation, and median estimates of the correlation. The Simes correction is applied to control for multiple testing, providing a global p-value (Pg). The function also returns the maximum observed correlation.

Value

A list containing the following elements:

`results` A data frame with columns: `variable`, `correlation`, `p_value`, `mean_corr`, `sd_corr`, `median_corr`, `rank`, `simes_threshold`, `significant_simes`, and `individual_significant`.

`summary_table` A data frame summarizing the global p-value (Pg) and maximum correlation.

`global_significant` A logical value indicating whether the global test is significant.

Index