Package: lande (via r-universe)

October 16, 2026

Type Package

Title Tools for Evolutionary Selection Analysis

Version 0.1.0

Description Estimates selection differentials and linear, quadratic and correlational selection gradients on relative fitness following Lande and Arnold (1983) <doi:10.1111/j.1558-5646.1983.tb00236.x>, with p-values from the logistic model for survival and from a Poisson or negative binomial model for counts. Fits cubic spline fitness functions as in Schluter (1988) <doi:10.1111/j.1558-5646.1988.tb02507.x> and two-trait fitness surfaces drawn only where there are data, and computes adaptive landscapes of mean fitness against the population mean for one or two traits, by period and for several groups on one surface. Includes bootstrap intervals, a combined results table and a Shiny app.

License MIT + file LICENSE

URL `https://github.com/Human-Augment-Analytics/lande`, `https://human-augment-analytics.github.io/lande/`

BugReports `https://github.com/Human-Augment-Analytics/lande/issues`

Encoding UTF-8

LazyData true

Depends R (>= 4.0.0)

Imports mgcv, MASS, Matrix, ggplot2 (>= 3.4.0), dplyr, fields, grDevices, stats, utils

Suggests akima, car, ggrepel, patchwork, performance, plotly, shiny, viridis, knitr, RhpcBLASctl, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr, rmarkdown

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Vanessa Zhou [aut] (affiliation: College of Computing, Georgia Institute of Technology, Atlanta, GA, United States), Ali Sean Khan [aut] (affiliation: College of Computing, Georgia Institute of Technology, Atlanta, GA, United States), Calvin Yu [aut] (affiliation: College of Computing, Georgia Institute of Technology, Atlanta, GA, United States), Taylor Papke [aut] (affiliation: College of Computing, Georgia Institute of Technology, Atlanta, GA, United States), Dima Al Nabhani [aut] (affiliation: College of Computing, Georgia Institute of Technology, Atlanta, GA, United States), Guillermo Garcia-Costoya [aut] (affiliation: School of Biological Sciences, Georgia Institute of Technology, Atlanta, GA, United States), James T. Stroud [aut, cre] (affiliation: School of Biological Sciences, Georgia Institute of Technology, Atlanta, GA, United States)

Maintainer James T. Stroud <stroud@gatech.edu>

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

Date/Publication 2026-10-16 12:40:09 UTC

RemoteUrl https://github.com/cran/lande

RemoteRef HEAD

RemoteSha 23b8e1c73a78b0c3ba6ac9eb8de53438e7d1a57f

Contents

adaptive_landscape . . . 3
analyze_disruptive_selection . . . 5
analyze_linear_selection . . . 6
analyze_nonlinear_selection . . . 7
bootstrap_selection . . . 8
bumpus . . . 9
canonical_analysis . . . 10
check_selection_assumptions . . . 12
compare_fitness_surfaces_data . . . 14
correlated_fitness_surface . . . 15
crescent_pond_pupfish . . . 18
detect_family . . . 19
extract_interaction_coefficients . . . 20
extract_linear_coefficients . . . 21
extract_quadratic_coefficients . . . 21
finch_community . . . 22
finch_yearly . . . 23
peak_difference . . . 23
plot_adaptive_landscape . . . 25
plot_adaptive_landscape_3d . . . 26
plot_canonical_axes . . . 27
plot_correlated_fitness . . . 28

plot_correlated_fitness_enhanced 29
plot_fitness_surfaces_comparison 30
plot_temporal_landscape 31
plot_univariate_fitness 32
prepare_selection_data 33
print.adaptive_landscape 34
print.selection_assumptions 35
print.selection_report 35
run_app 36
selection_coefficients 36
selection_differential 38
selection_report 39
temporal_landscape 40
univariate_spline 42

Index **45**

`adaptive_landscape` *Calculate Adaptive Landscape*

Description

Computes the adaptive landscape (mean fitness as a function of population mean phenotype) using a fitted individual-level fitness model.

Usage

```
adaptive_landscape(
  data,
  fitness_model,
  trait_cols,
  group_col = NULL,
  population_variance = NULL,
  simulation_n = 1000,
  grid_n = 50,
  custom_range = NULL,
  clamp = TRUE,
  support_warn = 0.25
)
```

Arguments

`data`	A data frame containing the original trait and fitness data.
`fitness_model`	A fitted model object (e.g., GAM or Tps) predicting individual fitness.
`trait_cols`	One or two trait column names. With one trait the model is usually the `$model` of `univariate_spline()` and the result is a curve; with two it is the `$model` of `correlated_fitness_surface()` and the result is a surface.

`group_col` Optional character string specifying a grouping variable.

`population_variance` Optional covariance matrix for the traits. Estimated from data if `NULL`.

`simulation_n` Integer specifying the number of individuals to simulate per grid point. Default is 1000.

`grid_n` Integer specifying the resolution of the population mean grid. Default is 50.

`custom_range` Optional list specifying custom ranges for the traits.

`clamp` Logical; if `TRUE` (the default) simulated fitness from a thin-plate model is held inside the range of the fitness type before averaging: 0 to 1 for survival, at least 0 for counts. Continuous fitness is left alone, and a GAM is unaffected because its link already respects the range. The run message says how many values were held.

`support_warn` Share of the population simulated at the optimum that may fall outside the data before the optimum is flagged as extrapolated. Default is 0.25.

Details

For a single trait the grid also carries `.ind_fit`, the individual fitness function evaluated at each population mean, for drawing the two curves together (see `plot_adaptive_landscape()`).

The simulated populations spread beyond the data, especially towards the edge of the grid, and the fitness of those individuals is extrapolated. `.outside` in the grid is the share of each simulated population falling outside the convex hull of the observed trait pairs, or outside the observed range for one trait, and the result's `support` gives that share over the whole grid and at the optimum.

Value

An object of class `"adaptive_landscape"`. Its `optimum_edge` is `TRUE` when the highest mean fitness lies on the edge of the grid, where the landscape may keep rising beyond it.

Examples

```
prep <- prepare_selection_data(bumpus, "survival", c("total_length", "weight"))
surf <- correlated_fitness_surface(prep, "survival", c("total_length", "weight"), grid_n = 30)
land <- adaptive_landscape(prep, surf$model, c("total_length", "weight"),
                           simulation_n = 100, grid_n = 15)
land$optimum

# one trait, from the spline fitness function
uni <- univariate_spline(prep, "survival", "total_length")
land1 <- adaptive_landscape(prep, uni$model, "total_length", simulation_n = 100, grid_n = 30)
plot_adaptive_landscape(land1, "total_length")
```

analyze_disruptive_selection

Disruptive or stabilising selection on one trait

Description

Estimates the linear (beta) and quadratic (gamma) selection gradients of a single trait. Gamma is positive under disruptive selection and negative under stabilising selection.

Usage

```
analyze_disruptive_selection(
  data,
  fitness_col,
  trait_col,
  fitness_type = c("auto", "binary", "count", "continuous"),
  standardize = TRUE,
  group = NULL,
  return_grouped = FALSE
)
```

Arguments

`data`	A data frame containing fitness and trait measurements.
`fitness_col`	A string specifying the name of the fitness column.
`trait_col`	A string specifying the name of the single trait column.
`fitness_type`	A string indicating the fitness type: `"auto"` (detect from the data), `"binary"`, or `"continuous"`. Default is `"auto"`.
`standardize`	Logical indicating whether to standardize the trait to mean 0 and SD 1. Default is `TRUE`.
`group`	Optional string specifying a grouping variable; standardisation and relative fitness are then computed within each group.
`return_grouped`	Logical; if `TRUE` and `group` is given, the gradients are estimated separately for each group and returned with a `Group` column. Default is `FALSE`.

Value

A data frame with one row per gradient (`Term`, `Type`, `Beta_Coefficient`, `Standard_Error`, `P_Value`, `Variance`).

Examples

```
analyze_disruptive_selection(bumpus, "survival", "total_length")
analyze_disruptive_selection(bumpus, "survival", "total_length",
                             group = "sex", return_grouped = TRUE)
```

analyze_linear_selection

Analyze linear selection gradients (beta)

Description

Estimates linear selection gradients by OLS on relative fitness. Binary fitness gets its p-values from a logistic GLM and count fitness from a Poisson GLM (negative binomial if overdispersed), since the OLS tests are not valid for either.

Usage

```
analyze_linear_selection(
  data,
  fitness_col,
  trait_cols,
  fitness_type,
  binary_response_col = NULL,
  group = NULL,
  se_type = c("ols", "hc3")
)
```

Arguments

`data`	A data frame containing fitness and trait measurements.
`fitness_col`	A string specifying the response column for the OLS gradient model (relative fitness).
`trait_cols`	A character vector of trait column names.
`fitness_type`	A string indicating the fitness type: `"binary"`, `"continuous"`, `"count"`, or `"proportion"`.
`binary_response_col`	Optional string naming the raw fitness column (0/1 for binary, counts for count fitness) used for the GLM that supplies p-values. If `NULL`, `fitness_col` is treated as the raw outcome and relativised internally.
`group`	Optional grouping column. With two or more groups the GLM that supplies the p-values gets an intercept for each, to match the standardising within groups; the least-squares fit has no group term.
`se_type`	Standard errors of the least-squares gradients: `"ols"` (the default) or `"hc3"`, heteroscedasticity-consistent; see `selection_coefficients()`.

Value

A list containing the fitted models, summaries, ANOVA tables, and VIFs.

Examples

```
prep <- prepare_selection_data(bumpus, "survival", c("total_length", "weight"))
fit <- analyze_linear_selection(prep, "survival", c("total_length", "weight"), "binary")
extract_linear_coefficients(c("total_length", "weight"), fit)
```

---

analyze_nonlinear_selection

Analyze nonlinear selection gradients (gamma)

Description

Estimates quadratic and correlational selection gradients by OLS on relative fitness.

Usage

```
analyze_nonlinear_selection(
  data,
  fitness_col,
  trait_cols,
  fitness_type,
  binary_response_col = NULL,
  group = NULL,
  se_type = c("ols", "hc3")
)
```

Arguments

`data`	A data frame containing fitness and trait measurements.
`fitness_col`	A string specifying the response column for the OLS gradient model (relative fitness).
`trait_cols`	A character vector of trait column names.
`fitness_type`	A string indicating the fitness type: `"binary"`, `"continuous"`, `"count"`, or `"proportion"`.
`binary_response_col`	Optional string naming the raw fitness column (0/1 for binary, counts for count fitness) used for the GLM that supplies p-values. If `NULL`, `fitness_col` is treated as the raw outcome and relativised internally.
`group`	Optional grouping column, used as in `analyze_linear_selection()`: a separate intercept for each group in the GLM that supplies the p-values.
`se_type`	Standard errors of the least-squares gradients: `"ols"` (the default) or `"hc3"`, heteroscedasticity-consistent; see `selection_coefficients()`.

Value

A list containing the fitted nonlinear models, summaries, ANOVA tables, and VIFs.

Examples

```
prep <- prepare_selection_data(bumpus, "survival", c("total_length", "weight"))
fit <- analyze_nonlinear_selection(prep, "survival", c("total_length", "weight"), "binary")
extract_quadratic_coefficients(c("total_length", "weight"), fit)
extract_interaction_coefficients(c("total_length", "weight"), fit)
```

---

bootstrap_selection *Bootstrap selection gradients*

Description

Resamples individuals with replacement and re-estimates the selection gradients on each resample to obtain bootstrap standard errors and percentile confidence intervals for every coefficient (beta, gamma, gamma_ij).

Usage

```
bootstrap_selection(
  data,
  fitness_col,
  trait_cols,
  fitness_type = c("auto", "binary", "count", "continuous"),
  standardize = TRUE,
  group = NULL,
  use_relative_for_fit = TRUE,
  n_boot = 1000,
  conf = 0.95
)
```

Arguments

`data`	A data frame containing fitness and trait measurements.
`fitness_col`	A string specifying the name of the fitness column.
`trait_cols`	A character vector of trait column names.
`fitness_type`	A string indicating the fitness type: `"auto"`, `"binary"`, `"count"`, or `"continuous"`. Binary and count fitness take their p-values from a GLM (logistic, or Poisson and negative binomial) on the raw values; the gradients always come from OLS on relative fitness.
`standardize`	Logical indicating whether to standardize traits to mean 0 and SD 1. Default is `TRUE`.
`group`	Optional string specifying a grouping variable (e.g., "year", "site"). Traits and relative fitness are standardised within each group, and for binary and count fitness the GLM that supplies the p-values gets a separate intercept for each group.

`use_relative_for_fit`
: Logical; if `TRUE` (default) the gradients are estimated on relative fitness $W/\bar{W}$ for every fitness type, as in Lande & Arnold (1983). Set `FALSE` only to reproduce coefficients on the absolute fitness scale.

`n_boot`
: Integer number of bootstrap resamples. Default is 1000.

`conf`
: Confidence level for the percentile interval. Default is 0.95.

Details

Each resample goes through the whole procedure again: with `standardize = TRUE` the traits are restandardised, and fitness is made relative to the resample's own mean, before both models are refitted, so the intervals carry the uncertainty in the standardisation as well as in the fit. Resamples whose fit fails are dropped, and `N_Boot` counts the ones that were used.

When `group` is given, individuals are resampled within each group so every resample keeps the original group sizes. A resample in which a trait has no variance within a group (so it could only be centred) is discarded rather than fitted on a degenerate value. The point estimate is the ordinary `selection_coefficients()` fit on the full data and its warnings are reported as usual.

With continuous fitness the intervals for beta can be too narrow when the traits have heavy tails (skewed or not) and the linear model leaves out curvature, though less so than the parametric ones (see `check_selection_assumptions()`). For survival and counts in the package's simulation they covered a little less than the parametric ones, 92 to 93% for beta. Refitting on a transformed trait changes the scale of the gradients and is only a check.

Value

A data frame with one row per coefficient and columns `Term`, `Type`, `Estimate` (point estimate on the full data), `Boot_SE`, `CI_lower`, `CI_upper`, `P_Value`, and `N_Boot` (usable resamples for that coefficient). `P_Value` is the parametric p-value from the point-estimate fit (OLS t-test, or the logistic-GLM Wald test for binary fitness), not a bootstrap p-value.

Examples

```
set.seed(1)
bootstrap_selection(bumpus, "survival", c("total_length", "weight"), n_boot = 50)
```

`bumpus` *Bumpus house sparrow survival data*

Description

Morphological measurements and overwinter survival of house sparrows (*Passer domesticus*) collected by Hermon Bumpus in 1898 after a severe winter storm in Providence, Rhode Island.

Usage

```
bumpus
```

Format

A data frame with 136 rows and 11 variables:

sex Sex of the bird (`male` or `female`).

survival Survival indicator: `1` if the bird survived, `0` otherwise.

total_length Total length (mm).

wingspread Alar extent / wingspread (mm).

weight Body weight (g).

head_length Length of head and beak (mm).

humerus Length of humerus (in).

femur Length of femur (in).

tibiotarsus Length of tibiotarsus (in).

skull_width Width of skull (in).

sternum Length of sternum / keel (in).

Source

Bumpus, H. C. (1899). The elimination of the unfit as illustrated by the introduced sparrow, *Passer domesticus*. *Biological Lectures, Marine Biological Laboratory, Woods Hole*, 209-226.

`canonical_analysis` *Canonical analysis of the gamma matrix*

Description

Rotates the matrix of quadratic and correlational selection gradients to its canonical axes (Phillips and Arnold 1989; Blows and Brooks 2003). Each axis is a combination of the traits with a single curvature, its eigenvalue: negative for stabilising and positive for disruptive selection along that axis. Correlational selection that is spread over many γ_{ij} shows up as curvature on a few axes.

Usage

```
canonical_analysis(
  data,
  fitness_col,
  trait_cols,
  fitness_type = c("auto", "binary", "count", "continuous"),
  standardize = TRUE,
  group = NULL,
  bootstrap = FALSE,
  n_boot = 200,
  conf = 0.95,
  test = c("permutation", "double_regression"),
  n_perm = 999
)
```

Arguments

`data`	A data frame containing fitness and trait measurements.
`fitness_col`	A string specifying the name of the fitness column.
`trait_cols`	A character vector of trait column names.
`fitness_type`	A string indicating the fitness type: `"auto"`, `"binary"`, `"count"`, or `"continuous"`. Binary and count fitness take their p-values from a GLM (logistic, or Poisson and negative binomial) on the raw values; the gradients always come from OLS on relative fitness.
`standardize`	Logical indicating whether to standardize traits to mean 0 and SD 1. Default is `TRUE`.
`group`	Optional string specifying a grouping variable (e.g., "year", "site"). Traits and relative fitness are standardised within each group, and for binary and count fitness the GLM that supplies the p-values gets a separate intercept for each group.
`bootstrap`	Logical; add percentile intervals for the eigenvalues by resampling individuals, within groups when `group` is given. Default is `FALSE`.
`n_boot`	Number of bootstrap resamples. Default is 200.
`conf`	Confidence level of the bootstrap interval. Default is 0.95.
`test`	Where the p-values come from: `"permutation"` (the default), Reynolds et al.'s (2010) permutation test, or `"double_regression"`, the double-regression tests, which are far too liberal.
`n_perm`	Number of permutations. Default is 999.

Details

The gamma matrix γ comes from `selection_coefficients()`, with the quadratic gradients doubled and the correlational gradients as they stand. Its eigenvectors are the columns of `M` and its eigenvalues the curvatures λ; directional selection along the axes is $\theta = M^T\beta$. Standard errors come from the double regression of Bisgaard and Ankenman (1996): fitness is refitted on the canonical scores and their squares, without cross-products, and twice the coefficient of a squared score is that axis's λ.

The double-regression tests treat the axes as known when they were estimated from the same data. With 136 individuals and fitness unrelated to five traits, the largest eigenvalue came out significant in about three samples in four, so by default the p-values come from the permutation test of Reynolds et al. (2010): fitness is shuffled among individuals, within groups when `group` is given, the canonical analysis and the double regression are repeated, and each axis's p-value is the share of shuffles, counting the data as one, whose F for the eigenvalue of the same rank is at least the observed one. It tests whether an axis curves more than it would with fitness unrelated to the traits; once one axis is strongly curved, the tests of the others are only rough. Every fitness type uses least squares on relative fitness, and the result depends on the random seed; call `set.seed()` first. With `test = "double_regression"` the p-values come from the double regression itself, for survival and counts from the logistic or count model on the same terms. The standard errors treat the axes as known either way.

Sampling error in γ also spreads its eigenvalues, so the largest curvatures are overestimated, especially with many traits and few individuals (Reynolds et al. 2010), and the estimated axes lean

towards directions of phenotype with little variance (Morrissey 2014). Read the eigenvalues together with the fitness surface along the same axes, which `plot_canonical_axes()` draws, and with the bootstrap intervals. With the bootstrap, each resample's axes are matched to the original ones by their largest absolute cosine, since order and sign change between resamples.

Value

An object of class `"canonical_analysis"`: `gamma`, the gamma matrix; `beta`; `M`, the loadings with one column per axis; `axes`, a data frame with `lambda`, its standard error and p-value, `theta`, and with the bootstrap `CI_lower`, `CI_upper` and `N_Boot`; `scores`, the prepared data with the canonical scores `m1`, `m2`, ... added; the fitness type, `n`, `test` and `n_perm`.

References

Bisgaard, S. and Ankenman, B. (1996) Standard errors for the eigenvalues in second-order response surface models. Technometrics 38, 238-246. Blows, M. W. and Brooks, R. (2003) Measuring nonlinear selection. The American Naturalist 162, 815-820. Morrissey, M. B. (2014) In search of the best methods for multivariate selection analysis. Methods in Ecology and Evolution 5, 1095-1109. Phillips, P. C. and Arnold, S. J. (1989) Visualizing multivariate selection. Evolution 43, 1209-1222. Reynolds, R. J., Childers, D. K. and Pajewski, N. M. (2010) The distribution and hypothesis testing of eigenvalues from the canonical analysis of the gamma matrix of quadratic and correlational selection gradients. Evolution 64, 1076-1085.

Examples

```
ca <- canonical_analysis(bumpus, "survival", c("total_length", "weight", "humerus"))
ca
```

---

check_selection_assumptions

Check the assumptions behind a selection analysis

Description

Runs the checks a Lande and Arnold analysis rests on and returns them in one table: multivariate normality of the traits, normality of each trait, collinearity, rows per model term, and for the gradient models a residual normality and heteroscedasticity test (continuous fitness), a separation check (binary fitness) or the dispersion ratio (count fitness). Nothing is enforced; the table is there to report alongside the gradients, as the protocol of Palacio et al. (2019) asks.

Usage

```
check_selection_assumptions(
  data,
  fitness_col,
  trait_cols,
  fitness_type = c("auto", "binary", "count", "continuous"),
  standardize = TRUE,
```

```
  group = NULL
)
```

Arguments

`data`	A data frame containing fitness and trait measurements.
`fitness_col`	A string specifying the name of the fitness column.
`trait_cols`	A character vector of trait column names.
`fitness_type`	A string indicating the fitness type: `"auto"`, `"binary"`, `"count"`, or `"continuous"`. Binary and count fitness take their p-values from a GLM (logistic, or Poisson and negative binomial) on the raw values; the gradients always come from OLS on relative fitness.
`standardize`	Logical indicating whether to standardize traits to mean 0 and SD 1. Default is `TRUE`.
`group`	Optional string specifying a grouping variable (e.g., "year", "site"). Traits and relative fitness are standardised within each group, and for binary and count fitness the GLM that supplies the p-values gets a separate intercept for each group.

Details

The regression gradients equal the covariance-adjusted differentials, $P^{-1}S$, by least squares whatever the trait distribution. Normal traits let them be read as the average slope and curvature of the fitness surface (Lande and Arnold 1983; Morrissey and Sakrejda 2013) and tie gamma to the change in the phenotypic covariance. Fitness doesn't need to be normal, and normality doesn't change how the gradients are computed. Mardia's (1970) skewness and kurtosis test multivariate normality, and each trait is also tested on its own with Shapiro and Wilk's test when there are 5000 rows or fewer.

In the package's simulation (validation/simulation.R), on a curved fitness surface, nominal 95% intervals for beta covered 88% of the time with normal traits, 67% with log-normal ones and 77% with symmetric heavy-tailed ones (92, 80 and 91% by bootstrap). Most of that loss comes from the curvature, and HC3 standard errors (`selection_coefficients(se_type = "hc3")`) recover most of it. Estimating the SD of a heavy-tailed trait lowers coverage too, for gamma and, with skewed traits, for beta even on a straight surface (90%), and HC3 doesn't help there. Skew also biases beta a little. Refitting with a transformed trait changes the scale selection is measured on, so report it with the untransformed result. Collinearity is the largest variance inflation factor from the linear gradient model. The rows-per-term check counts the individuals, or for binary fitness the rarer outcome, per term of the quadratic model, with ten as the working minimum. With more than 2000 individuals Mardia's statistics use a random subsample of 2000, drawn the same way every run. Like the gradient models, the logistic and count models fit an intercept per group, with unlabelled rows as one more group. A group in which every individual survived, or none did, is fitted at 0 or 1 by its intercept and is left out of the separation check. For counts the note names the model behind each set of p-values; the linear and quadratic models are checked for overdispersion separately. If the `performance` package is installed its heteroscedasticity and overdispersion tests are added next to the package's own, along with an R squared for the gradient model.

Value

A data frame of class "selection_assumptions" with one row per check: check, statistic, p_value and note.

References

Lande, R. and Arnold, S. J. (1983) The measurement of selection on correlated characters. Evolution 37, 1210-1226. Mardia, K. V. (1970) Measures of multivariate skewness and kurtosis with applications. Biometrika 57, 519-530. Morrissey, M. B. and Sakrejda, K. (2013) Unification of regression-based methods for the analysis of natural selection. Evolution 67, 2094-2100. Palacio, F. X., Ordano, M. and Benitez-Vieyra, S. (2019) Measuring natural selection on multivariate phenotypic traits: a protocol for verifiable and reproducible analyses of natural selection. Israel Journal of Ecology and Evolution 65, 130-136.

Examples

```
check_selection_assumptions(bumpus, "survival", c("total_length", "weight", "humerus"))
```

---

compare_fitness_surfaces_data

Compare Correlated Fitness Surface vs Adaptive Landscape

Description

Compares individual-level fitness (correlated surface) to population-level mean fitness (adaptive landscape).

Usage

```
compare_fitness_surfaces_data(
  correlated_surface,
  adaptive_landscape,
  trait_cols,
  calculate_correlation = TRUE
)
```

Arguments

`correlated_surface`
: Output list from `correlated_fitness_surface()`.

`adaptive_landscape`
: Output list from adaptive landscape modeling.

`trait_cols`
: A character vector of length 2 specifying the trait column names.

`calculate_correlation`
: Logical indicating whether to calculate the correlation between the two surfaces.

Value

A list containing combined grid data, summary stats, optima points, and correlation metrics.

Examples

```
prep <- prepare_selection_data(bumpus, "survival", c("total_length", "weight"))
surf <- correlated_fitness_surface(prep, "survival", c("total_length", "weight"), grid_n = 30)
land <- adaptive_landscape(prep, surf$model, c("total_length", "weight"),
                           simulation_n = 100, grid_n = 15)
comp <- compare_fitness_surfaces_data(surf, land, c("total_length", "weight"))
plot_fitness_surfaces_comparison(comp)
```

---

correlated_fitness_surface

Calculate the Correlated Fitness Surface

Description

Fits a model for individual fitness based on multiple individual phenotypes (w ~ z1 + z2 + interactions). Traits MUST be standardized BEFORE calling this function.

Usage

```
correlated_fitness_surface(
  data,
  fitness_col,
  trait_cols,
  grid_n = 60,
  method = "auto",
  scale_traits = FALSE,
  group = NULL,
  group_effect = TRUE,
  k = NULL,
  mask = TRUE,
  too_far = NULL,
  bs = c("tp", "cr", "ps"),
  smoothing = c("REML", "GCV.Cp", "ML"),
  clamp = TRUE,
  level = 0.95,
  by_group = FALSE,
  count_family = c("poisson", "quasipoisson", "nb")
)
```

Arguments

`data`	A data frame containing fitness and trait measurements.
`fitness_col`	A string specifying the name of the fitness column.

`trait_cols` A character vector of exactly length 2 specifying the trait column names.

`grid_n` Integer specifying the resolution of the prediction grid. Default is 60.

`method` A string specifying the modeling method: `"auto"`, `"gam"`, or `"tps"`.

`scale_traits` Deprecated. Logical. Set to `FALSE` to avoid double standardization.

`group` Optional string naming a grouping column, such as species or year. The result's `groups` table then gives each group's mean trait values and the highest point of the surface within that group's own convex hull, and the plot functions draw both.

`group_effect` Logical; with `TRUE` (the default) the GAM includes `group` as a fixed effect and predicts the surface at the reference level, as the gradient models do for year or site; rows with no group label are one more level. With `FALSE` one surface is fitted to everyone and the group is only used for the `groups` table and the overlay, which is what a community surface of several species needs (Beausoleil et al. 2023).

`k` Basis dimension for the GAM smooth. The default `NULL` sets it from the data as `min(30, max(10, floor(sqrt(n1 * n2))))`, where `n1` and `n2` are the numbers of distinct values of each trait; it is always capped at one less than the number of observations. Pass an integer to override.

`mask` Logical; if `TRUE` (the default) grid points outside the convex hull of the observed trait pairs get `NA` fitness, so the surface is only drawn where there are data. The grid column `.inside` records which points were kept, `.fit_all` holds the unmasked predictions, and the result's `hull` is the polygon the plot functions use to cover the outside.

`too_far` Optional distance rule for blanking the grid, on top of `mask`. Grid points farther than this from the nearest individual get `NA` fitness, with distances measured after scaling the grid to the unit square, as in `mgcv::vis.gam`. Beausoleil et al. (2023) used 0.15. The default `NULL` applies no distance rule. The distance to the nearest individual is kept in the grid column `.dist`.

`bs` Basis for the GAM smooth: `"tp"` (thin plate, the default), `"cr"` or `"ps"`.

`smoothing` How the GAM's smoothing parameter is chosen: `"REML"` (the default), `"GCV.Cp"` or `"ML"`.

`clamp` Logical; with `method = "tps"` and `TRUE` (the default), predictions are held inside the range of the fitness type, 0 to 1 for survival and at least 0 for counts, as `adaptive_landscape()` does. The GAM respects the range through its link and is unaffected.

`level` Confidence level of the band around a GAM surface. Default is 0.95.

`by_group` Logical; if `TRUE` fit a separate surface to each level of `group` and return a named list of surfaces, each with its own shape, grid and hull. The default `FALSE` fits one surface, with the group as a fixed effect or only marked on it (see `group_effect`).

`count_family` Family for count fitness in the GAM: `"poisson"` (the default); `"quasipoisson"`, which estimates the dispersion, so the standard errors and peak comparisons allow for overdispersed counts; or `"nb"`, a negative binomial. The smoothing parameter is chosen again with the dispersion estimated, so the fitted surface

changes too, usually becoming smoother, and can change a lot where there are few individuals. The result's `dispersion` is the Pearson dispersion of the fit, and a Poisson fit warns when it is above 1.5 (a rule of thumb).

Details

The family follows the fitness column: 0/1 fitness gets a binomial family, non-negative whole numbers with more than two values (lifespan in years, offspring) a Poisson family with a log link or the one set by `count_family`, and anything else a Gaussian family. With `method = "auto"` binary and count fitness use the GAM and continuous fitness the thin-plate spline. By default the GAM uses a thin-plate smooth with the smoothing parameter chosen by REML; `bs` and `smoothing` are there to match another study's smoother and do not apply to `method = "tps"`. With `"cr"` or `"ps"`, which are one-dimensional bases, the two traits enter as a tensor product smooth. Predicting a fitted surface over the full rectangle of the grid extrapolates into corners that no individual occupies; `mask = TRUE` leaves those blank rather than showing a fitted value there. The hull still fills gaps between separate clusters of individuals, such as several species on one surface; `too_far` blanks those too.

Value

A list containing the fitted model, grid predictions, and metadata. With `method = "gam"` the grid also carries `.se`, the standard error of the fitted fitness, and `.fit_lo` and `.fit_hi`, the band at `level`, worked out on the scale of the link and blanked where the fit is; the thin-plate spline gives none. `peaks` lists the local maxima of the kept surface, highest first, with `interior` `TRUE` when every neighbouring cell is kept and lower, and `FALSE` when the maximum sits against the edge of the data, which may only be where the data end. `original_data` holds the rows that were analysed. With a `group` the `groups` data frame has one row per group with its size, mean traits and the highest point of the surface within its own range, flagged `peak_interior` when that point is an interior maximum of the surface and `peak_edge` when it lies at the edge of the data; with both `FALSE` the surface keeps rising past the group's range. For count fitness `count_family` and `dispersion` record the family used and the Pearson dispersion of the fit.

Examples

```
prep <- prepare_selection_data(bumpus, "survival", c("total_length", "weight"))
surf <- correlated_fitness_surface(prep, "survival", c("total_length", "weight"), grid_n = 30)
surf$grid[which.max(surf$grid$.fit), ]

# two lakes of pupfish on one surface: cells far from any fish blank, each
# lake's mean and local peak marked, the lake kept out of the model
pup <- rbind(crescent_pond_pupfish, little_lake_pupfish)
pup <- pup[pup$density == "H", ]
prep2 <- prepare_selection_data(pup, "survival", c("nose", "noseangle"))
surf2 <- correlated_fitness_surface(prep2, "survival", c("nose", "noseangle"), grid_n = 30,
                                    too_far = 0.15, group = "lake", group_effect = FALSE)
surf2$groups
```

`crescent_pond_pupfish` *Pupfish hybrid survival in two San Salvador lakes*

Description

Laboratory-reared F2 hybrids of the three pupfish species of San Salvador Island, Bahamas, photographed, tagged and released into field enclosures in Crescent Pond and Little Lake from March to June 2011, with laboratory-reared fish of the three species measured the same way. Martin (2016) analysed the high-density enclosures, `density == "H"`, as do the examples and the app.

Usage

```
crescent_pond_pupfish

little_lake_pupfish
```

Format

Data frames with 23 variables, 993 rows for Crescent Pond and 1062 for Little Lake:

lake `"CP"` (Crescent Pond) or `"LL"` (Little Lake).

density The enclosure of an F2 hybrid, `"H"` (high density) or `"L"` (low density), or the species of a parental fish: `"norm"`, the generalist *Cyprinodon variegatus*; `"bozo"`, the molluscivore *C. brontotheroides*; `"bull"`, the scale-eater *C. desquamator*.

survival `1` if the hybrid survived the three months in the enclosure, `0` otherwise; `NA` for parental fish.

ln.growth Log growth rate of survivors over the three months. Hybrids that died are coded `0`, so drop them before analysing growth; `NA` for parental fish.

color Martin's plotting colour: orange for hybrids; blue, green and red for the generalist, molluscivore and scale-eater.

d13C, d15N Carbon and nitrogen stable isotope ratios (per mil) of muscle from surviving hybrids; `NA` for the rest.

jaw Lower jaw length, from the jaw joint to the tip of the dentary.

eye Eye diameter, the mean of the major and minor axes of the iris (orbit diameter).

eye2 Eye roundness, the ratio of the major to the minor axis of the iris.

pmx Craniofacial height, from the jaw joint to the tip of the premaxilla (upper jaw length).

snout Lateral snout length, from the tip of the premaxilla to the front of the iris.

body Dorsal to anal distance, between the first rays of the dorsal and anal fins (body depth).

caudal Caudal peduncle height.

nasal Dorsal snout length, from the front of the orbit to the tip of the maxilla in dorsal view.

mouth Buccal width in dorsal view.

width Head width across the opercula in dorsal view.

boteyeangle Ventral orbit angle, at the tip of the premaxilla between the lower edge of the iris and the quadrate.

topeyeangle Orbit angle, at the tip of the premaxilla between the upper and lower edges of the iris.

nose Maxillary head protrusion (nasal protrusion).

noseangle Maxillary head protrusion angle (nasal angle).

adduct Preopercular height.

SL Standard length.

An object of class `data.frame` with 1062 rows and 23 columns.

Details

The trait columns are Martin's scores for the 16 measurements of Martin and Wainwright (2013), taken from the photographs. He log-transformed the distances, regressed each trait on a size index (the first principal component of seven size-related distances) and kept the residuals, fitting the regression to the parental fish alone for `jaw`, `mouth`, `nasal` and `width`; `eye2`, `nose` and `noseangle` were not size-corrected. Every trait is standardised to mean 0 and standard deviation 1 within each lake, hybrids and parental fish together.

The column names are Martin's. The trait names below are the ones defined in the supplement to Martin and Wainwright (2013), matched to the columns through his size-correction script (in the Dryad archive) and their discriminant loadings, with the name Martin (2016) uses in brackets where it differs. The six functional traits of his Table 3 are `jaw`, `pmx`, `nose`, `noseangle`, `body` and `eye`. Their loadings table swaps the labels of `snout` and `nasal`; the names here follow the size-correction script.

Source

Martin, C. H. (2016) Context dependence in complex adaptive landscapes: frequency and trait-dependent selection surfaces within an adaptive radiation of Caribbean pupfishes. *Evolution* 70, 1265-1282. doi:10.1111/evo.12932. Data: doi:10.5061/dryad.n3mj3.

References

Martin, C. H. and Wainwright, P. C. (2013) Multiple fitness peaks on the adaptive landscape drive adaptive radiation in the wild. *Science* 339, 208-211. doi:10.1126/science.1227710.

`detect_family` *Automatically detect fitness type and GLM family*

Description

Heuristically determines whether a fitness vector represents binary, count, proportion, or continuous data.

Usage

```
detect_family(y)
```

Arguments

y	A numeric vector representing fitness values.

Value

A list containing `type` (string), `family` (GLM family object), and `note` (string).

Examples

```
detect_family(c(0, 1, 0, 0, 1, 1))
detect_family(bumpus$survival)$type
detect_family(rpois(50, 2))$type
```

---

extract_interaction_coefficients

Extract correlational selection coefficients

Description

Extract correlational selection coefficients

Usage

```
extract_interaction_coefficients(trait_cols, results)
```

Arguments

trait_cols	A character vector of trait column names.
results	A model results object returned by `analyze_nonlinear_selection()`.

Value

A data frame with correlational (interaction) selection coefficients and statistics.

Examples

```
prep <- prepare_selection_data(bumpus, "survival", c("total_length", "weight"))
fit <- analyze_nonlinear_selection(prep, "survival", c("total_length", "weight"), "binary")
extract_interaction_coefficients(c("total_length", "weight"), fit)
```

extract_linear_coefficients

Extract linear selection coefficients

Description

Extract linear selection coefficients

Usage

```
extract_linear_coefficients(trait_cols, results)
```

Arguments

trait_cols	A character vector of trait column names.
results	A model results object returned by analyze_linear_selection().

Value

A data frame with linear selection coefficients and statistics.

Examples

```
prep <- prepare_selection_data(bumpus, "survival", c("total_length", "weight"))
fit <- analyze_linear_selection(prep, "survival", c("total_length", "weight"), "binary")
extract_linear_coefficients(c("total_length", "weight"), fit)
```

extract_quadratic_coefficients

Extract quadratic selection coefficients

Description

Extract quadratic selection coefficients

Usage

```
extract_quadratic_coefficients(trait_cols, results)
```

Arguments

trait_cols	A character vector of trait column names.
results	A model results object returned by analyze_nonlinear_selection().

Value

A data frame with quadratic selection coefficients (doubled estimates) and statistics.

Examples

```
prep <- prepare_selection_data(bumpus, "survival", c("total_length", "weight"))
fit <- analyze_nonlinear_selection(prep, "survival", c("total_length", "weight"), "binary")
extract_quadratic_coefficients(c("total_length", "weight"), fit)
```

---

finch_community *Darwin's finch community at El Garrapatero*

Description

Four ground finch species at El Garrapatero, Santa Cruz Island, Galapagos, with *Geospiza fortis* split into its small and large beak morphs. One row per bird, with its mean beak measurements and its apparent lifespan, the fitness measure of Beausoleil et al. (2023). Birds first caught late in the study had fewer years in which to be seen again. Built by `data-raw/finch_community.R` from the file in the authors' code repository (GPL-3).

Usage

```
finch_community
```

Format

A data frame with 3428 rows and 7 variables:

band Band code of the bird.

species A factor: `"fortis small"` and `"fortis large"` (*G. fortis*), `"fuliginosa"` (*G. fuliginosa*), `"magnirostris"` (*G. magnirostris*) or `"scandens"` (*G. scandens*).

beak_length Mean beak length (mm).

beak_depth Mean beak depth (mm).

beak_width Mean beak width (mm).

lifespan Apparent lifespan in years: the last year the bird was seen minus the first.

first_year The year the bird was first caught.

Source

Beausoleil, M.-O. et al. (2023) The fitness landscape of a community of Darwin's finches. *Evolution* 77, 2533-2546. doi:10.1093/evolut/qpad160. Data: doi:10.5683/SP3/0YIWSE.

finch_yearly — *Medium ground finch survival from year to year*

Description

Medium ground finches (*Geospiza fortis*) marked and recaptured at El Garrapatero, Santa Cruz Island, Galapagos. Each row is a bird in a year it was seen, from 2004 to 2010, with whether it was seen again in any later year (apparent survival). Built by `data-raw/finch_yearly.R`.

Usage

```
finch_yearly
```

Format

A data frame with 1087 rows and 7 variables:

band Band code of the bird.

year Year the bird was seen.

survived `1` if the bird was seen again in any later year up to 2018, the last in the file, `0` otherwise. Death cannot be told from emigration.

beak_pc1 Beak size: the first principal component of the three beak measurements over all the birds, signed so that larger beaks score higher.

beak_length Median beak length (mm).

beak_width Median beak width (mm).

beak_depth Median beak depth (mm).

Source

Beausoleil, M.-O. et al. (2019) Temporally varying disruptive selection in the medium ground finch (*Geospiza fortis*). *Proceedings of the Royal Society B* 286, 20192290. doi:10.1098/rspb.2019.2290. Data: doi:10.5061/dryad.zcrjdfn6q.

peak_difference — *Compare the fitted fitness at two points of a surface*

Description

Takes two points on a GAM surface, given as group names or trait values, and returns the difference in fitted fitness on the scale of the link, with its standard error. Two peaks are separate only if each is higher than the pass between them, the lowest point on the highest route from one to the other, which `valley = TRUE` compares.

Usage

```
peak_difference(
  surface,
  from,
  to,
  valley = FALSE,
  route = c("pass", "line"),
  n_path = 50
)
```

Arguments

surface	Output of correlated_fitness_surface() fitted with method = "gam".
from, to	A group name, standing for that group's highest point in surface$groups, or a numeric vector of the two trait values.
valley	Logical; also find the valley between the two points and compare each end with it. Default is FALSE.
route	How the valley is found: "pass" (the default), the pass over the kept cells of the surface's grid, or "line", the lowest point on the straight line between the two points, which is how Beausoleil et al. (2023) measured valley depths. A straight line can cross a trough that a curving ridge goes round, so it can make two peaks look more separate than they are.
n_path	Number of points along the straight line with route = "line". Default is 50.

Details

Differences are on the scale of the link, log fitness for counts and log odds for survival. The standard error comes from the covariance of the model's coefficients, corrected for smoothing parameter uncertainty when the fit was by REML or ML. The points are taken as fixed, although the peaks and the valley were found on the same fitted surface, so the comparisons describe the fitted surface and are not planned tests. The valley is the lowest point on the route, so its differences are biased upwards and their z values too large, which is why the valley rows have no p-value. The pass is found cell by cell over the grid, joining each kept cell to its eight neighbours, from the two kept cells nearest the points in grid steps; its precision follows grid_n, and a point far from any kept cell gets a message. If the route never drops below the cell a point starts from, or the lower point is no higher than the pass, there is no dip to compare. The straight line ignores the mask. For overdispersed counts fit the surface with count_family = "quasipoisson" first, since Poisson standard errors are then too small.

Value

A data frame with one row per comparison: the fitted fitness at the two points (fit_a, fit_b), their difference on the link scale, its standard error, z and the two-sided p_value (NA for the valley rows). With valley = TRUE the attribute "valley" holds the trait values of the pass or of the lowest point on the line.

Examples

```
prep <- prepare_selection_data(bumpus, "survival", c("total_length", "weight"))
surf <- correlated_fitness_surface(prep, "survival", c("total_length", "weight"), grid_n = 30)
peak_difference(surf, c(-1, -1), c(1, 1))
```

---

plot_adaptive_landscape

Plot Adaptive Landscape

Description

Two traits give a contour map of mean fitness against the two population means. One trait gives a curve of mean fitness against the population mean, with the individual fitness function drawn alongside it for comparison.

Usage

```
plot_adaptive_landscape(
  landscape,
  trait_cols,
  original_data = NULL,
  group_col = NULL,
  bins = 12,
  show_optimum = TRUE,
  show_actual_means = TRUE,
  show_individual = TRUE,
  point_alpha = 0.8,
  show_support = FALSE,
  support_level = 0.5,
  ...
)
```

Arguments

`landscape`	Output object of class `"adaptive_landscape"`.
`trait_cols`	One or two trait column names, matching the landscape.
`original_data`	Optional data frame of original data points. Default is `NULL`.
`group_col`	Optional character string specifying a grouping variable for labels.
`bins`	Integer specifying the number of contour bins. Default is 12.
`show_optimum`	Logical indicating whether to display the optimum point, labelled as the highest point when it lies on the edge of the grid. Default is `TRUE`.
`show_actual_means`	Logical indicating whether to display actual population means. Default is `TRUE`.
`show_individual`	Logical; for a single trait, also draw the individual fitness function (dashed). Default is `TRUE`.

point_alpha	Numeric value for point transparency. Default is 0.8.
show_support	Logical; mark where the simulation leaves the data: a dashed line at support_level and a white overlay on the population means beyond it, where more than that share of the simulated population falls outside the observed traits. Default is FALSE.
support_level	Share of the simulated population outside the data at which show_support draws its line. Default is 0.5.
...	Additional arguments passed to ggplot2::labs().

Value

A ggplot object representing the adaptive landscape.

Examples

```
prep <- prepare_selection_data(bumpus, "survival", c("total_length", "weight"))
surf <- correlated_fitness_surface(prep, "survival", c("total_length", "weight"), grid_n = 30)
land <- adaptive_landscape(prep, surf$model, c("total_length", "weight"),
                           simulation_n = 100, grid_n = 15)
plot_adaptive_landscape(land, c("total_length", "weight"))
```

---

plot_adaptive_landscape_3d

Plot Adaptive Landscape (3D Perspective)

Description

Plot Adaptive Landscape (3D Perspective)

Usage

```
plot_adaptive_landscape_3d(
  landscape,
  trait_cols,
  theta = -30,
  phi = 30,
  grid_n = 200,
  color_palette = NULL,
  ...
)
```

Arguments

landscape	Output object of class "adaptive_landscape".
trait_cols	Character vector of length 2 specifying the trait column names.
theta	Numeric azimuthal viewing angle. Default is -30.

phi	Numeric colatitude viewing angle. Default is 30.
grid_n	Ignored; the landscape's own grid is used.
color_palette	Optional vector of colors for the surface. Defaults to viridis plasma.
...	Additional arguments passed to fields::drape.plot().

Value

A 3D plot produced by fields::drape.plot().

Examples

```
if (requireNamespace("fields", quietly = TRUE)) {
  prep <- prepare_selection_data(bumpus, "survival", c("total_length", "weight"))
  surf <- correlated_fitness_surface(prep, "survival", c("total_length", "weight"), grid_n = 30)
  land <- adaptive_landscape(prep, surf$model, c("total_length", "weight"),
                             simulation_n = 100, grid_n = 15)
  plot_adaptive_landscape_3d(land, c("total_length", "weight"))
}
```

---

plot_canonical_axes *Fitness along the canonical axes*

Description

Draws the fitness surface in the space of two canonical axes, or the fitness function along one, from the scores of canonical_analysis(). A negative λ is stabilising selection along an axis only when fitness along that axis has a peak inside the data, otherwise it is only curvature.

Usage

```
plot_canonical_axes(ca, which = 1:2, grid_n = 40, ...)
```

Arguments

ca	Output of canonical_analysis().
which	One or two axis numbers. Default is 1:2.
grid_n	Grid resolution of the surface. Default is 40.
...	Passed to plot_correlated_fitness() for two axes or to plot_univariate_fitness() for one.

Value

A ggplot object.

Examples

```
ca <- canonical_analysis(bumpus, "survival", c("total_length", "weight", "humerus"))
plot_canonical_axes(ca, which = c(1, 3), grid_n = 25)
```

plot_correlated_fitness

Plot Correlated Fitness Surface

Description

Plot Correlated Fitness Surface

Usage

```
plot_correlated_fitness(
  tps,
  trait_cols,
  bins = 12,
  point_alpha = 0.7,
  show_points = FALSE,
  show_optimum = TRUE,
  show_groups = TRUE,
  group_lines = TRUE,
  uncertainty = c("none", "se", "band"),
  ...
)
```

Arguments

tps	Output list from `correlated_fitness_surface()`.
trait_cols	Character vector of length 2 specifying the trait column names.
bins	Integer specifying the number of contour bins. Default is 12.
point_alpha	Numeric value for point transparency. Default is 0.7.
show_points	Logical indicating whether to show original data points. Default is `FALSE`.
show_optimum	Logical; mark the highest kept cell, a gold diamond, or an open one when it is on the edge of the data. Default is `TRUE`.
show_groups	Logical; when the surface was fitted with a `group`, draw each group's mean (open circle, labelled) and the highest point of the surface within that group's hull: a filled triangle when it is a peak of the surface, an open one when the surface keeps rising past the group's range or the edge of the data. Default is `TRUE`.
group_lines	Logical; join each group's mean to its peak with a dashed line. Default is `TRUE`.
uncertainty	What to draw of the surface's standard error, which a GAM surface carries: `"none"` (the default), `"se"` for dashed contour lines of the standard error of the fitted fitness over the surface, or `"band"` for three panels, the lower bound, the fit and the upper bound, on one fill scale.
...	Additional arguments passed to `ggplot2::labs()`.

Value

A ggplot object representing the correlated fitness surface.

Examples

```
prep <- prepare_selection_data(bumpus, "survival", c("total_length", "weight"))
surf <- correlated_fitness_surface(prep, "survival", c("total_length", "weight"), grid_n = 30)
plot_correlated_fitness(surf, c("total_length", "weight"), show_points = TRUE)
```

---

plot_correlated_fitness_enhanced

Plot Enhanced Correlated Fitness Surface

Description

Plot Enhanced Correlated Fitness Surface

Usage

```
plot_correlated_fitness_enhanced(
  tps,
  trait_cols = NULL,
  original_data = NULL,
  fitness_col = NULL,
  bins = 12,
  point_alpha = 0.7,
  show_optimum = TRUE,
  show_groups = TRUE,
  group_lines = TRUE,
  uncertainty = c("none", "se", "band"),
  ...
)
```

Arguments

tps	Output list from correlated_fitness_surface().
trait_cols	Character vector of length 2. If NULL, inferred from grid.
original_data	Optional data frame of original data.
fitness_col	Optional character string specifying the fitness column for coloring points.
bins	Integer specifying the number of contour bins. Default is 12.
point_alpha	Numeric value for point transparency. Default is 0.7.
show_optimum	Logical; mark the highest kept cell, a gold diamond, or an open one when it is on the edge of the data. Default is TRUE.

show_groups	Logical; when the surface was fitted with a group, draw each group's mean and the highest point of the surface within that group's hull, filled when it is a peak of the surface and open when it is not. Default is TRUE.
group_lines	Logical; join each group's mean to its peak with a dashed line. Default is TRUE.
uncertainty	What to draw of the surface's standard error, which a GAM surface carries: "none" (the default), "se" for dashed contour lines of the standard error of the fitted fitness over the surface, or "band" for three panels, the lower bound, the fit and the upper bound, on one fill scale.
...	Additional arguments passed to ggplot2::labs().

Value

A ggplot object with enhanced visualizations.

Examples

```
prep <- prepare_selection_data(bumpus, "survival", c("total_length", "weight"))
surf <- correlated_fitness_surface(prep, "survival", c("total_length", "weight"), grid_n = 30)
plot_correlated_fitness_enhanced(surf, c("total_length", "weight"),
                                 original_data = prep, fitness_col = "survival")
```

---

plot_fitness_surfaces_comparison

Plot Fitness Surfaces Comparison

Description

Visualizes the comparison between individual-level correlated fitness and population-level adaptive landscape.

Usage

```
plot_fitness_surfaces_comparison(
  comparison_data,
  bins = 10,
  title = NULL,
  point_alpha = 0.7,
  linewidth = 0.8,
  show_optima = TRUE
)
```

Arguments

comparison_data	Output list from compare_fitness_surfaces_data().
bins	Integer specifying the number of contour bins. Default is 10.

title	Optional character string for the overarching plot title.
point_alpha	Numeric value for optimum point transparency. Default is 0.7.
linewidth	Numeric value for contour line width. Default is 0.8.
show_optima	Logical indicating whether to show optimum points on the overlay. Default is TRUE.

Value

A list containing side_by_side (a patchwork object) and overlay (a ggplot object).

Examples

```
prep <- prepare_selection_data(bumpus, "survival", c("total_length", "weight"))
surf <- correlated_fitness_surface(prep, "survival", c("total_length", "weight"), grid_n = 30)
land <- adaptive_landscape(prep, surf$model, c("total_length", "weight"),
                           simulation_n = 100, grid_n = 15)
comp <- compare_fitness_surfaces_data(surf, land, c("total_length", "weight"))
plot_fitness_surfaces_comparison(comp)
```

---

plot_temporal_landscape

Plot fitness functions and landscapes over time

Description

Plot fitness functions and landscapes over time

Usage

```
plot_temporal_landscape(
  tl,
  type = c("panels", "heatmap"),
  show_points = TRUE,
  show_landscape = TRUE,
  show_optimum = TRUE,
  connect = FALSE,
  bins = 10,
  ncol = NULL,
  ...
)
```

Arguments

tl	Output of temporal_landscape().
type	"panels" draws one panel per period: the fitness function with its band and the individuals for one trait, the fitness surface for two. "heatmap", for one trait only, draws fitness against trait and period in one panel, with the position of the highest fitness in each period marked.

show_points	Logical; draw the individuals in each panel.
show_landscape	Logical; for one trait, also draw each period's adaptive landscape as a dashed curve.
show_optimum	Logical; mark the highest fitted fitness in each period, as a gold diamond when it lies inside the data and an open one when it sits at the edge of the range.
connect	Logical; in the heat map, join the highest fitness of successive periods with a line. Default is FALSE, since a maximum at the edge of the range is not a peak.
bins	Contour bins for two-trait panels.
ncol	Number of panel columns.
...	Additional arguments passed to ggplot2::labs().

Value

A ggplot object.

Examples

```
prep <- prepare_selection_data(finch_yearly, "survived", "beak_pc1")
years <- temporal_landscape(prep, "survived", "beak_pc1", "year", landscape = FALSE)
plot_temporal_landscape(years, ncol = 4)
plot_temporal_landscape(years, type = "heatmap")
```

---

plot_univariate_fitness

Plot Univariate Correlated Fitness Function

Description

Visualizes the univariate fitness function estimated via spline.

Usage

```
plot_univariate_fitness(
  uni,
  trait_col,
  title = NULL,
  show_points = TRUE,
  point_alpha = 0.25,
  ribbon_fill = NA,
  ribbon_linetype = "dashed",
  classic_plot = FALSE,
  ...
)
```

Arguments

uni	Output object of class "univariate_fitness" from univariate_spline().
trait_col	Character string specifying the trait column name.
title	Optional character string for the plot title.
show_points	Logical indicating whether to show the original data points. Default is TRUE.
point_alpha	Numeric value for point transparency. Default is 0.25.
ribbon_fill	Color for the confidence interval ribbon. Default is NA (transparent).
ribbon_linetype	Linetype for the confidence interval bounds. Default is "dashed".
classic_plot	Logical; if TRUE, draw the plain published style (a single dashed confidence outline, solid fit line, raw points, and a minimal theme). Default is FALSE.
...	Additional arguments passed to ggplot2::labs().

Value

A ggplot object representing the univariate fitness function.

Examples

```
prep <- prepare_selection_data(bumpus, "survival", "total_length")
uni <- univariate_spline(prep, "survival", "total_length")
plot_univariate_fitness(uni, "total_length")
plot_univariate_fitness(uni, "total_length", classic_plot = TRUE)
```

---

prepare_selection_data

Prepare data for selection analysis

Description

Cleans, standardizes, and calculates relative fitness for trait and fitness data. Standardizations and relative fitness calculations can be performed within groups.

Usage

```
prepare_selection_data(
  data,
  fitness_col,
  trait_cols,
  standardize = TRUE,
  group = NULL,
  add_relative = TRUE,
  na_action = c("warn", "drop", "none"),
  name_relative = "relative_fitness"
)
```

Arguments

data	A data frame containing fitness and trait measurements.
fitness_col	A string specifying the name of the fitness column.
trait_cols	A character vector of trait column names.
standardize	Logical indicating whether to standardize traits to mean 0 and SD 1. Default is TRUE.
group	Optional string specifying a grouping variable.
add_relative	Logical indicating whether to add a relative fitness column. Default is TRUE.
na_action	A string specifying how to handle missing values: "warn", "drop", or "none".
name_relative	A string specifying the name for the relative fitness column. Default is "relative_fitness".

Value

A modified data frame ready for selection analysis.

Examples

```
prep <- prepare_selection_data(bumpus, "survival", c("total_length", "weight"))
head(prep[, c("survival", "relative_fitness", "total_length", "weight")])

# standardised and relativised within each sex
by_sex <- prepare_selection_data(bumpus, "survival", "total_length", group = "sex")
tapply(by_sex$total_length, by_sex$sex, mean)
```

---

print.adaptive_landscape

Print method for adaptive landscape

Description

Print method for adaptive landscape

Usage

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

Arguments

x	An object of class "adaptive_landscape".
...	Additional arguments passed to print.

Value

The input object x, invisibly.

print.selection_assumptions

Print the assumption checks

Description

Print the assumption checks

Usage

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

Arguments

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

Value

The input object x, invisibly.

print.selection_report

Print a selection report

Description

Print a selection report

Usage

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

Arguments

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

Value

The input object x, invisibly.

run_app *Open the Shiny app*

Description

Starts the app that ships with the package: pick a bundled dataset or upload a CSV, choose the fitness column, the traits and an optional group, and read the gradients, fitness functions, surfaces and landscapes from the tabs. Needs the `shiny` package; `plotly` adds the rotatable 3D landscape and `fields` the static one.

Usage

```
run_app(...)
```

Arguments

`...`	Passed to `shiny::runApp()`, for example `port` or `launch.browser`.

Value

Whatever `shiny::runApp()` returns; called to open the app.

Examples

```
if (interactive()) run_app()
```

selection_coefficients *Calculate selection coefficients*

Description

Main wrapper function for selection analysis following Lande & Arnold (1983). Extracts linear (beta), quadratic (gamma), and correlational selection gradients.

Usage

```
selection_coefficients(
  data,
  fitness_col,
  trait_cols,
  fitness_type = c("auto", "binary", "count", "continuous"),
  standardize = TRUE,
  group = NULL,
  use_relative_for_fit = TRUE,
  return_grouped = FALSE,
  se_type = c("ols", "hc3")
)
```

Arguments

`data` A data frame containing fitness and trait measurements.

`fitness_col` A string specifying the name of the fitness column.

`trait_cols` A character vector of trait column names.

`fitness_type` A string indicating the fitness type: `"auto"`, `"binary"`, `"count"`, or `"continuous"`. Binary and count fitness take their p-values from a GLM (logistic, or Poisson and negative binomial) on the raw values; the gradients always come from OLS on relative fitness.

`standardize` Logical indicating whether to standardize traits to mean 0 and SD 1. Default is `TRUE`.

`group` Optional string specifying a grouping variable (e.g., "year", "site"). Traits and relative fitness are standardised within each group, and for binary and count fitness the GLM that supplies the p-values gets a separate intercept for each group.

`use_relative_for_fit` Logical; if `TRUE` (default) the gradients are estimated on relative fitness $W/\bar{W}$ for every fitness type, as in Lande & Arnold (1983). Set `FALSE` only to reproduce coefficients on the absolute fitness scale.

`return_grouped` Logical indicating whether to return results grouped if a `group` is specified.

`se_type` Standard errors of the gradients: `"ols"`, the usual least-squares ones (the default), or `"hc3"`, leave-one-out standard errors that allow for residual spread changing with the traits. Mitchell-Olds and Shaw (1987) suggested the jackknife for selection gradients when the residuals are not normal; HC3 (MacKinnon and White 1985) gets much the same in closed form, from how far the estimates move when each individual is left out. In the package's simulation it recovered most of the coverage beta loses when the model leaves out curvature, from 88 to 93% with normal traits, 67 to 86% with log-normal ones and 77 to 91% with heavy-tailed symmetric ones. It did nothing for the coverage lost to estimating a skewed trait's SD, and with survival and counts its intervals covered a little less, down to 91% (see `check_selection_assumptions()`). For continuous fitness the p-values follow the chosen errors; for survival and counts they come from the GLM with either `se_type`.

Value

A data frame containing selection coefficients (Term, Type, Beta_Coefficient, Standard_Error, P_Value, Variance), with the standard errors used in the attribute `"se_type"`.

References

MacKinnon, J. G. and White, H. (1985) Some heteroskedasticity-consistent covariance matrix estimators with improved finite sample properties. Journal of Econometrics 29, 305-325. Mitchell-Olds, T. and Shaw, R. G. (1987) Regression analysis of natural selection: statistical inference and biological interpretation. Evolution 41, 1149-1161.

Examples

```
selection_coefficients(bumpus, "survival", c("total_length", "weight"), fitness_type = "binary")

# one set of gradients per sex, each sex standardised on its own
selection_coefficients(bumpus, "survival", c("total_length", "weight"),
                       group = "sex", return_grouped = TRUE)
```

---

selection_differential

Calculate selection differential (S)

Description

Computes the selection differential S = Cov(z, w) for a single trait.

Usage

```
selection_differential(
  data,
  fitness_col,
  trait_col,
  standardized = TRUE,
  use_relative = TRUE,
  group = NULL,
  return_grouped = FALSE
)
```

Arguments

`data`	A data frame containing fitness and trait measurements.
`fitness_col`	A string specifying the name of the fitness column.
`trait_col`	A string specifying the name of the trait column.
`standardized`	Logical indicating whether the trait is already standardized. If `FALSE`, it will be standardized.
`use_relative`	Logical indicating whether to use relative fitness.
`group`	Optional grouping variable string. Calculates S within each group if provided.
`return_grouped`	Logical indicating whether to return a data frame with S per group (`TRUE`) or overall mean S (`FALSE`).

Value

A single numeric value representing overall S, or a data frame with S per group if `return_grouped = TRUE`.

Examples

```
prep <- prepare_selection_data(bumpus, "survival", "total_length")
selection_differential(prep, "survival", "total_length")
selection_differential(prep, "survival", "total_length", group = "sex", return_grouped = TRUE)
```

---

selection_report *Standardised selection analysis report*

Description

Runs a Lande-Arnold analysis and returns one table of selection differentials (S) and linear (beta), quadratic (gamma) and correlational (gamma_ij) gradients. All estimates use traits standardised to mean 0, SD 1 and relative fitness, so tables from different studies can be compared.

Usage

```
selection_report(
  data,
  fitness_col,
  trait_cols,
  fitness_type = c("auto", "binary", "count", "continuous"),
  standardize = TRUE,
  group = NULL,
  use_relative_for_fit = TRUE,
  include_differentials = TRUE,
  digits = 4,
  se_type = c("ols", "hc3")
)
```

Arguments

`data`	A data frame containing fitness and trait measurements.
`fitness_col`	A string specifying the name of the fitness column.
`trait_cols`	A character vector of trait column names.
`fitness_type`	A string indicating the fitness type: `"auto"`, `"binary"`, `"count"`, or `"continuous"`. Binary and count fitness take their p-values from a GLM (logistic, or Poisson and negative binomial) on the raw values; the gradients always come from OLS on relative fitness.
`standardize`	Logical indicating whether to standardize traits to mean 0 and SD 1. Default is `TRUE`.
`group`	Optional string specifying a grouping variable (e.g., "year", "site"). Traits and relative fitness are standardised within each group, and for binary and count fitness the GLM that supplies the p-values gets a separate intercept for each group.

`use_relative_for_fit`
: Logical; if `TRUE` (default) the gradients are estimated on relative fitness $W/\bar{W}$ for every fitness type, as in Lande & Arnold (1983). Set `FALSE` only to reproduce coefficients on the absolute fitness scale.

`include_differentials`
: Logical; if `TRUE` (default) selection differentials S are included alongside the gradients.

`digits`
: Integer number of digits used when printing. Default is 4.

`se_type`
: Standard errors of the gradients: `"ols"`, the usual least-squares ones (the default), or `"hc3"`, leave-one-out standard errors that allow for residual spread changing with the traits. Mitchell-Olds and Shaw (1987) suggested the jackknife for selection gradients when the residuals are not normal; HC3 (MacKinnon and White 1985) gets much the same in closed form, from how far the estimates move when each individual is left out. In the package's simulation it recovered most of the coverage beta loses when the model leaves out curvature, from 88 to 93% with normal traits, 67 to 86% with log-normal ones and 77 to 91% with heavy-tailed symmetric ones. It did nothing for the coverage lost to estimating a skewed trait's SD, and with survival and counts its intervals covered a little less, down to 91% (see `check_selection_assumptions()`). For continuous fitness the p-values follow the chosen errors; for survival and counts they come from the GLM with either `se_type`.

Details

Differentials and gradients are computed on the same individuals (those with complete fitness and trait values) and on the same trait and fitness scales. S is the population covariance (divides by n) while the OLS gradient on sd-standardised traits corresponds to the sample covariance (n - 1), so for a single trait the Differential and Linear rows differ by the factor (n - 1) / n.

Value

A data frame of class `"selection_report"` with columns `Term`, `Type`, `Estimate`, `Std_Error`, and `P_Value`.

Examples

```
selection_report(bumpus, "survival", c("total_length", "weight", "humerus"))
```

`temporal_landscape` *Fitness functions and adaptive landscapes over time*

Description

Fits the fitness function (one trait) or fitness surface (two traits) separately for each level of a time column, usually year, and optionally the adaptive landscape for each, to show how selection changes over time. Beausoleil et al. (2019) did this for Darwin's finches, one year at a time.

Usage

```
temporal_landscape(
  data,
  fitness_col,
  trait_cols,
  time_col,
  fitness_type = c("auto", "binary", "count", "continuous"),
  min_n = 20,
  landscape = TRUE,
  k = NULL,
  bs = NULL,
  smoothing = NULL,
  bootstrap = FALSE,
  n_boot = 200,
  grid_n = 60,
  simulation_n = 300,
  mask = TRUE,
  too_far = NULL,
  count_family = c("poisson", "quasipoisson", "nb")
)
```

Arguments

`data`	A data frame with fitness, trait and time columns. Standardise the traits once, over all periods, with `prepare_selection_data()` before calling; the periods must share one trait axis, so nothing is restandardised per period.
`fitness_col`	Name of the fitness column.
`trait_cols`	One or two trait column names.
`time_col`	Name of the column giving the period of each row.
`fitness_type`	Passed to `univariate_spline()` for one trait; the surface detects the type itself.
`min_n`	Periods with fewer complete rows than this are skipped and listed in the result's `skipped`. Default is 20.
`landscape`	Logical; also compute the adaptive landscape for each period. Default is `TRUE`.
`k`, `bs`, `smoothing`	Basis size, basis and smoothing criterion for the period fits. `NULL` uses the defaults of `univariate_spline()` or `correlated_fitness_surface()`.
`bootstrap`, `n_boot`	For one trait, a bootstrap band around each period's fitness function.
`grid_n`	Grid resolution for the surfaces and landscapes.
`simulation_n`	Simulated individuals per grid point in the landscapes.
`mask`, `too_far`	Passed to `correlated_fitness_surface()` for two traits.
`count_family`	Family for count fitness in every period's fit: `"poisson"` (the default), `"quasipoisson"` or `"nb"`, as in `univariate_spline()`.

Value

An object of class "temporal_landscape": fits and landscapes, one per period; summary, one row per period with n, mean fitness, trait means, the smooth's degrees of freedom, the position and height of the highest fitted fitness and whether it sits at the edge of the data (optimum_edge), the number of interior peaks (one trait) and the landscape optimum; grid, the fitted values of every period stacked with a time column; for one trait heat, each period's fitness function on one common trait grid, NA outside that period's data; landscape_grid, the landscapes stacked; data, the rows used; and skipped.

Examples

```
prep <- prepare_selection_data(finch_yearly, "survived", "beak_pc1")
years <- temporal_landscape(prep, "survived", "beak_pc1", "year", landscape = FALSE)
years$summary
plot_temporal_landscape(years, type = "heatmap")
```

---

univariate_spline *Estimate univariate correlated fitness function*

Description

This function calculates a univariate correlated fitness function using a GAM smooth term. The formula is w ~ f(z). Traits should be standardized BEFORE calling this function.

Usage

```
univariate_spline(
  data,
  fitness_col,
  trait_col,
  fitness_type = c("auto", "binary", "count", "continuous"),
  group = NULL,
  relative_col = NULL,
  k = 10,
  bs = c("cr", "tp", "ps"),
  smoothing = c("GCV.Cp", "REML", "ML"),
  bootstrap = FALSE,
  n_boot = 1000,
  by_group = FALSE,
  count_family = c("poisson", "quasipoisson", "nb")
)
```

Arguments

data	A data frame containing fitness and trait measurements.
fitness_col	A string specifying the name of the fitness column.

`trait_col` A string specifying the name of the trait column (must be numeric and standardized).

`fitness_type` A string indicating the fitness type: `"auto"` (detect from the data, the default), `"binary"`, `"count"`, or `"continuous"`. Binary and count fitness are fitted on the raw values with a binomial or Poisson family; continuous fitness on relative fitness with a Gaussian one.

`group` Optional string specifying a grouping variable. If provided, group fixed effects are included (none when it has only one level); rows with no group label are one more level.

`relative_col` Optional string naming a pre-computed relative fitness column to use for continuous fitness (e.g. one produced within groups by `prepare_selection_data`).

`k` Integer specifying the basis dimension for the smooth term. Default is 10.

`bs` Spline basis: `"cr"` (cubic regression spline, the default), `"tp"` (thin plate) or `"ps"` (P-spline).

`smoothing` How the smoothing parameter is chosen: `"GCV.Cp"` (generalised cross-validation, the default, which mgcv applies in its UBRE form to binary and Poisson fitness and as plain GCV to quasi-Poisson; a negative binomial fit uses REML in its place), `"REML"` or `"ML"`. The result's `spline_type` names the criterion mgcv used.

`bootstrap` Logical; if `TRUE` the 95% ribbon is obtained by resampling individuals and refitting (Schluter 1988). The default `FALSE` uses the parametric Wald interval, which is fast; the bootstrap refits the spline `n_boot` times.

`n_boot` Integer number of bootstrap resamples used when `bootstrap = TRUE`. Default is 1000.

`by_group` Logical; if `TRUE` fit each level of `group` on its own rows and return a named list of fits, one per group. The default `FALSE` fits one curve with the group as a fixed effect. Prepare the data with the same `group` first, so that each group is standardised on its own.

`count_family` Family for count fitness: `"poisson"` (the default), `"quasipoisson"` or `"nb"`, as in `correlated_fitness_surface()`; the smoothing parameter is chosen under whichever is used. The result's `dispersion` is the Pearson dispersion of the fit, and a Poisson fit warns when it is above 1.5.

Details

By default the fitness function is a penalised cubic regression spline with the smoothing parameter chosen by generalised cross-validation (UBRE for binary and Poisson fitness), following Schluter (1988). `bs` and `smoothing` set the family of curves and the criterion, to match the smoother of another study; the amount of smoothing is still estimated from the data, though the criterion can change it a good deal. If mgcv's check suggests the basis dimension was too small a warning says so; the check works on the residuals, which for survival carry too little to test, so it is skipped there. With `bootstrap = TRUE` the result depends on the random seed; call `set.seed()` first for a reproducible ribbon.

Value

A list of class "univariate_fitness" containing the fitted GAM model, a prediction grid, and metadata.

Examples

```
prep <- prepare_selection_data(bumpus, "survival", "total_length")
uni <- univariate_spline(prep, "survival", "total_length")
plot_univariate_fitness(uni, "total_length")

# a thin-plate basis with REML, to match another study's smoother
univariate_spline(prep, "survival", "total_length", bs = "tp", smoothing = "REML")$spline_type
```

Index

∗ **datasets**
bumpus, 9
crescent_pond_pupfish, 18
finch_community, 22
finch_yearly, 23

adaptive_landscape, 3
analyze_disruptive_selection, 5
analyze_linear_selection, 6
analyze_nonlinear_selection, 7

bootstrap_selection, 8
bumpus, 9

canonical_analysis, 10
check_selection_assumptions, 12
compare_fitness_surfaces_data, 14
correlated_fitness_surface, 15
crescent_pond_pupfish, 18

detect_family, 19

extract_interaction_coefficients, 20
extract_linear_coefficients, 21
extract_quadratic_coefficients, 21

finch_community, 22
finch_yearly, 23

little_lake_pupfish
(crescent_pond_pupfish), 18

peak_difference, 23
plot_adaptive_landscape, 25
plot_adaptive_landscape_3d, 26
plot_canonical_axes, 27
plot_correlated_fitness, 28
plot_correlated_fitness_enhanced, 29
plot_fitness_surfaces_comparison, 30
plot_temporal_landscape, 31
plot_univariate_fitness, 32
prepare_selection_data, 33
print.adaptive_landscape, 34
print.selection_assumptions, 35
print.selection_report, 35

run_app, 36

selection_coefficients, 36
selection_differential, 38
selection_report, 39

temporal_landscape, 40

univariate_spline, 42