Package: invasimapr (via r-universe)

May 31, 2026

Type Package

Title Workflow to visualise trait dispersion and assess invasibility

Version 0.1.0

Description Utilities for assembling invader–resident interaction tensors, computing trait-based competition kernels from distance matrices, deriving environmental suitability kernels, calculating invasion fitness, and summarising trait dispersion metrics. Includes helpers to predict site × species responses for residents and simulated invaders.

License MIT + file LICENSE

URL https://github.com/b-cubed-eu/invasimapr, https://b-cubed-eu.github.io/invasimapr/

BugReports https://github.com/b-cubed-eu/invasimapr/issues

Encoding UTF-8

LazyData false

Depends R (>= 4.5)

Imports stats, graphics, cluster, MASS, viridisLite, ggplot2, factoextra, dplyr, tidyr, tibble, purrr, magrittr, glmmTMB, rlang, matrixStats, vegan, fuzzyjoin, forcats, Matrix, httr, jsonlite, magick, rvest, data.table (>= 1.15.0)

Suggests dissmapr, pheatmap, scico, sf, terra, ClustGeo, devtools, remotes, scales, viridis, stringr, fastDummies, performance, lattice, RColorBrewer, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/Needs/website false

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/pak/sysreqs cmake make libmagick++-dev gsfonts libicu-dev libuv1-dev libxml2-dev libssl-dev

Repository https://b-cubed-eu.r-universe.dev

Date/Publication 2026-01-30 13:25:28 UTC

RemoteUrl https://github.com/b-cubed-eu/invasimapr

RemoteRef HEAD

RemoteSha 47df006fa26dcf98d6c581c4412c717c60ec208c

Contents

`assemble_matrices` *Assemble community matrices for invasion-fitness workflows*

Description

`assemble_matrices()` creates the standard inputs used throughout the invasion-fitness pipeline from either a single long-format table or a set of pre-assembled tables. It returns aligned objects including:

- `site_df`: site by x,y coordinates

- `env_df`: site by environment numeric matrix
- `comm_res`: site by resident abundance matrix
- `pa_res`: site by resident presence-absence matrix
- `traits_res`: resident by trait data frame (mixed types allowed)

Usage

```
assemble_matrices(
  long_df = NULL,
  site_df = NULL,
  env_df = NULL,
  comm_res = NULL,
  traits_res = NULL,
  comm_long = c("auto", "long", "wide"),
  site_col = "site",
  x_col = "x",
  y_col = "y",
  species_col = "species",
  count_col = "count",
  env_cols = NULL,
  env_prefix = "^env",
  trait_cols = NULL,
  trait_prefix = "^trait",
  drop_empty_sites = TRUE,
  drop_empty_species = TRUE,
  return_diversity = TRUE,
  make_plots = FALSE
)
```

Arguments

`long_df`	Optional long-format data frame with at least the columns `site`, `x`, `y`, `species`, and `count`.
`site_df`	Optional data frame with columns `site`, `x`, and `y`, used when `long_df` is `NULL`.
`env_df`	Optional site by environment numeric data frame or matrix. Row names must correspond to site identifiers.
`comm_res`	Optional site by resident numeric matrix or data frame (wide), or a long-format table with columns `site`, `species`, and `count`.
`traits_res`	Optional resident by trait data frame with row names corresponding to species identifiers.
`comm_long`	Character string indicating how to interpret a separately supplied `comm_res`; one of `"auto"`, `"long"`, or `"wide"`.
`site_col`, `x_col`, `y_col`, `species_col`, `count_col`	Column names used when building from `long_df`.
`env_cols`	Character vector of environment column names to extract.

`env_prefix` Regular expression used to select environment columns.

`trait_cols` Character vector of trait column names to extract.

`trait_prefix` Regular expression used to select trait columns.

`drop_empty_sites`
Logical. If `TRUE`, drop sites with zero total abundance.

`drop_empty_species`
Logical. If `TRUE`, drop resident species with zero total abundance across all sites.

`return_diversity`
Logical. If `TRUE`, compute and return site-level diversity summaries.

`make_plots` Logical. If `TRUE`, generate quick diagnostic plots.

`build_invader_predictors`
Build standardized invader predictors r_is_z, C_is_z, S_is_z (expects env/traits frames already projected to model columns)

Description

Build standardized invader predictors r_is_z, C_is_z, S_is_z (expects env/traits frames already projected to model columns)

Usage

```
build_invader_predictors(
  fit_r,
  env_df_model,
  traits_inv_model,
  sites,
  inv_ids,
  r_mu_s,
  r_sd_s,
  W_site,
  gower_all,
  res_ids,
  sigma_alpha,
  C_mu_s,
  C_sd_s,
  S_s_z,
  verbose = TRUE
)
```

Arguments

`fit_r`	Fitted residents-only `glmmTMB` model.
`env_df_model`	Data frame of environment predictors used by the model (rows = sites). Either `ENV_PC*` (if PCA was used) or standardized `env_df_z`.
`traits_inv_model`	Data frame of invader traits used by the model (rows = invaders). Includes `TR_PC*` (if PCA was used) and any raw factor traits present in the model formula.
`sites`	Character vector of site identifiers (must match row names of `env_df_model` and `W_site`).
`inv_ids`	Character vector of invader identifiers (must match row names of `traits_inv_model`).
`r_mu_s`	Numeric vector of site-wise means used to standardize abiotic suitability (`r_js`).
`r_sd_s`	Numeric vector of site-wise standard deviations used to standardize abiotic suitability (`r_js`).
`W_site`	Site × resident weighting matrix (e.g. relative abundance or presence–absence weights).
`gower_all`	Square matrix or `dist` object of Gower distances among all residents and invaders.
`res_ids`	Character vector of resident species identifiers.
`sigma_alpha`	Numeric kernel bandwidth for crowding effects. If `NULL` or non-positive, a data-driven default is used.
`C_mu_s`	Numeric vector of site-wise means used to standardize crowding (`C_is`).
`C_sd_s`	Numeric vector of site-wise standard deviations used to standardize crowding (`C_is`).
`S_s_z`	Numeric vector of standardized site saturation values.
`verbose`	Logical; if `TRUE`, emit diagnostic messages.

`build_model_formula` *Flexible formula constructor for residents-only trait–environment models*

Description

`build_model_formula()` assembles a right-hand side (RHS) for a GLMM (or LM/GLM) from environment terms, trait terms, and (optionally) all pairwise environment-by-trait interactions. It also appends random-effects structures, such as `(1 | site) + (1 | species)` and optional zero-correlation random slopes like `(0 + r_z || site)`.

You can pass the terms directly as character vectors, or let the function derive them from `env_df` and/or `trait_df` column names.

Usage

```
build_model_formula(
  response = "abundance",
  env_terms = NULL,
  trait_terms = NULL,
  env_df = NULL,
  trait_df = NULL,
  include_intercept = TRUE,
  include_env_main = TRUE,
  include_trait_main = TRUE,
  include_env_trait_interactions = TRUE,
  extra_fixed = NULL,
  random_intercepts = c("site", "species"),
  random_slopes = NULL,
  backend = c("glmmTMB", "lme4"),
  verbose = FALSE
)
```

Arguments

`response`	Character scalar. Name of the response on the LHS (default `"abundance"`).
`env_terms`	Optional character vector of environment term names to include as fixed effects. If `NULL`, they can be derived from `env_df`.
`trait_terms`	Optional character vector of trait term names to include as fixed effects. If `NULL`, they can be derived from `trait_df`.
`env_df`	Optional data frame containing environment predictors; used only to infer `env_terms` when `env_terms` is `NULL`.
`trait_df`	Optional data frame containing trait predictors; used only to infer `trait_terms` when `trait_terms` is `NULL`.
`include_intercept`	Logical. Include the fixed-effect intercept? If `FALSE`, the intercept is removed via `0` (equivalent to `-1`). Default `TRUE`.
`include_env_main`	Logical. Include environment main effects? Default `TRUE`.
`include_trait_main`	Logical. Include trait main effects? Default `TRUE`.
`include_env_trait_interactions`	Logical. Include all pairwise environment-by-trait interactions? Implemented as `(E1 + E2 + ...):(T1 + T2 + ...)`. Default `TRUE`.
`extra_fixed`	Optional character vector of additional fixed-effect terms to append verbatim (e.g., `"poly(temp,2)"`, `"I(pH^2)"`).
`random_intercepts`	Character vector of grouping factors for random intercepts (e.g., `c("site","species")`). Use `NULL` to omit. Default `c("site","species")`.
`random_slopes`	Named list of the form `list(site = c("r_z","C_z"), species = "r_z")` to add zero-correlation slopes `(0 + term || group)`. Use `NULL` (default) for none.

`backend`	Character flag used only for messaging; both **lme4** and **glmmTMB** accept the same syntax here. Default `c("glmmTMB","lme4")`.
`verbose`	Logical. If `TRUE`, prints the assembled formula string.

Details

Build a GLMM-ready model formula from trait and environment terms

Value

An object of class `formula`, e.g.: `abundance ~ env1 + env2 + tr1 + tr2 + (env1 + env2):(tr1 + tr2) + (1 | site) + (1 | species) + (0 + r_z || site)`

Examples

```
## Not run:
# Toy data
set.seed(1)
env_df_z        = data.frame(env1 = rnorm(10), env2 = rnorm(10))
traits_res_glmm = data.frame(tr1  = rnorm(5),  tr2  = rnorm(5))

fml = build_model_formula(
  response   = "abundance",
  env_df     = env_df_z,
  trait_df   = traits_res_glmm,
  random_intercepts = c("site","species"),
  random_slopes     = list(site = c("r_z","C_z"))
)
fml

# Fit a GLMM (example; requires your long residents×sites table `dat_r`)
# library(glmmTMB)
# fit = glmmTMB::glmmTMB(fml, family = glmmTMB::tweedie(link="log"), data = dat_r)
# summary(fit)

## End(Not run)
```

`compute_centrality_hull` *Compute trait-space centrality (robust Mahalanobis) and hull status, with visuals*

Description

Given 2D trait coordinates for **residents** and **invaders** (typically PCoA scores), this function:

- estimates a robust centre and covariance for residents (MCD via `MASS::cov.rob`, fallback to classical),

- computes Mahalanobis distances for residents and invaders,
- converts distances to **centrality** using the resident distance CDF (`centrality = 1 - F(d)`; 1 = core, 0 = peripheral),
- determines whether invaders sit **inside** the resident convex hull (familiar) or **outside** (novel),
- returns a tidy table and three ggplot figures.

Usage

```
compute_centrality_hull(
  Q_res,
  Q_inv,
  ellipse_level = 0.5,
  point_size = 2.8,
  alpha = 0.95,
  stroke = 0.7,
  rank_by = c("centrality", "distance"),
  peripheral_first = TRUE,
  palette = "viridis"
)
```

Arguments

`Q_res`	Data frame with resident coordinates; must contain columns `tr1`, `tr2`. Row names are treated as resident IDs.
`Q_inv`	Data frame with invader coordinates; must contain `tr1`, `tr2`. Row names are treated as invader IDs. Can be 0-row.
`ellipse_level`	Numeric in (0,1); confidence level for the resident normal ellipse in the trait-plane plot. Default `0.50`.
`point_size`	Numeric; point size in the trait-plane plot. Default `2.8`.
`alpha`	Numeric in (0,1]; point alpha. Default `0.95`.
`stroke`	Numeric; outline stroke width for points. Default `0.7`.
`rank_by`	Character; one of `"centrality"` or `"distance"`. Controls the invader ranking plot. Default `"centrality"`.
`peripheral_first`	Logical; if `TRUE` and `rank_by = "centrality"`, sort by increasing centrality (peripheral first). If ranking by distance, `TRUE` sorts by decreasing distance (far first). Default `TRUE`.
`palette`	Centrality fill palette; passed to `scale_fill_viridis_c`. Default `"viridis"`.

Details

Centrality and convex-hull membership in trait space

Mahalanobis distances may fail if the covariance is nearly singular; a tiny ridge is added internally if needed. Hull membership is computed with **sp** when available, otherwise with **sf** if available; if neither is installed and residents < 3, hull membership is returned as `NA`.

Value

A list with:

- `df` — tidy table with columns: `id`, `grp` (`"resident"|"invader"`), `tr1`, `tr2`, `d_md` (Mahalanobis), `d_eu` (Euclidean), `centrality` (0–1), `in_hull` (logical for invaders; residents = TRUE).
- `center`, `cov` — robust centre and covariance used.
- `hull_df` — closed ring (tr1,tr2) of resident convex hull, or `NULL` if <3 residents.
- `plots` — list with: `p_trait` (trait-plane scatter), `p_dist` (distance distributions), `p_rank` (invader ranking with hull flags; `NULL` if no invaders).

Examples

```
## Not run:
# Assume you already computed PCoA coordinates:
# Q_res, Q_inv each with columns tr1, tr2 and rownames as IDs.
out = compute_centrality_hull(Q_res, Q_inv)
head(out$df)
out$plots$p_trait
out$plots$p_dist
out$plots$p_rank

## End(Not run)
```

`compute_establishment_probability` *Probabilistic establishment from invasion fitness*

Description

Invasion fitness λ_{is} integrates trait-space geometry (distances, overlaps, convex hulls, centroids) with abiotic suitability (alignment of invader traits to the local environment), niche crowding (overlap with resident trait space weighted by composition), and resident competition (site saturation).

`compute_establishment_probability()` maps λ_{is} to probabilities of establishment using a unified interface:

- **Probit**: $P = \Phi(\lambda/\sigma)$, where σ is a scalar, the residual standard deviation from a fitted auxiliary model, or a cell-wise predictive standard deviation.
- **Logistic**: $P = \text{logit}^{-1}(\lambda/\tau)$, where τ is a scale parameter.
- **Hard rule**: $P = I(\lambda > 0)$, yielding a binary map.

If `lambda_is` is not supplied, the function builds it from standardized predictors using $\lambda_{is} = \gamma r_{is}^{(z)} - \alpha C_{is}^{(z)} - \beta S_{is}^{(z)} + \kappa$.

Usage

```
compute_establishment_probability(
  lambda_is = NULL,
  r_is_z = NULL,
  C_is_z = NULL,
  S_is_z = NULL,
  gamma = 1,
  alpha = NULL,
  beta = NULL,
  kappa = 0,
  method = c("probit", "logit", "hard"),
  sigma = NULL,
  tau = 1,
  fit = NULL,
  predictive = FALSE,
  sigma_mat = NULL,
  use_vcov = FALSE,
  Q_inv = NULL,
  site_df = NULL,
  return_long = TRUE,
  make_plots = TRUE,
  option_label = NULL
)
```

Arguments

`lambda_is`	Optional matrix of invasion fitness values with dimensions S by I (rows are sites, columns are invaders). If `NULL`, fitness is computed from the supplied components.
`r_is_z`, `C_is_z`, `S_is_z`	Optional matrices of standardized abiotic suitability, niche crowding, and saturation. Used only when `lambda_is = NULL`.
`gamma`	Optional vector of length I or matrix of dimension S by I giving the abiotic slope.
`alpha`	Optional vector or matrix of crowding penalties. By convention, these values are constrained to be non-negative.
`beta`	Optional vector of saturation penalties. Signed values allow facilitation.
`kappa`	Optional scalar offset added to invasion fitness (default 0).
`method`	Character string; one of `"probit"`, `"logit"`, or `"hard"`.
`sigma`	Numeric scalar standard deviation for the probit transform.
`tau`	Numeric scalar scale parameter for the logistic transform.
`fit`	Optional fitted model object used to obtain a residual standard deviation when `method = "probit"`.
`predictive`	Logical; if `TRUE`, a predictive standard deviation matrix is used for the probit transform.

`sigma_mat`	Optional matrix of predictive standard deviations with the same dimensions as `lambda_is`.
`use_vcov`	Logical; if `TRUE`, compute predictive standard deviations from the variance-covariance matrix of `fit`.
`Q_inv`	Optional data frame of invader trait scores with columns `tr1` and `tr2`.
`site_df`	Optional site metadata table with columns `site`, `x`, and `y`.
`return_long`	Logical; if `TRUE`, include a long-format table in the output.
`make_plots`	Logical; if `TRUE`, return diagnostic plots.
`option_label`	Optional label attached to the output.

Details

Convert invasion fitness to probabilistic establishment (probit, logit, hard)

For the probit method, probabilities are computed as $P_{is} = \Phi(\lambda_{is}/\sigma)$. A scalar or cell-wise standard deviation may be used.

For the logistic method, probabilities are computed as $P_{is} = \text{logit}^{-1}(\lambda_{is}/\tau)$.

The hard rule returns a binary indicator equal to one when $\lambda_{is} > 0$.

Value

A list with components:

- `p_is`: matrix of establishment probabilities
- `lambda_is`: invasion fitness matrix
- `sigma_used`: standard deviation used by the probit transform
- `method`: transformation method
- `option_label`: label used for summaries
- `prob_long`: long-format table (optional)
- `plots`: list of plots (optional)

Examples

```
## Minimal example (toy shapes)
set.seed(1)
S = 6; I = 4
sites   = paste0("s", 1:S)
inv     = paste0("i", 1:I)
r_is_z  = matrix(rnorm(S*I), S, I, dimnames=list(sites, inv))
C_is_z  = matrix(rnorm(S*I), S, I, dimnames=dimnames(r_is_z))
S_is_z  = matrix(rep(scale(rnorm(S)), each=I), S, I, dimnames=dimnames(r_is_z))
gamma   = setNames(runif(I, 0.5, 1.2), inv)
alpha   = setNames(runif(I, 0.2, 1.0), inv)
beta    = setNames(runif(I, 0.1, 0.6), inv)

# Build lambda internally, then get logistic probabilities
out_logit = compute_establishment_probability(
```

```
  r_is_z=r_is_z, C_is_z=C_is_z, S_is_z=S_is_z,
  gamma=gamma, alpha=alpha, beta=beta,
  method="logit", tau=1, return_long=FALSE, make_plots=FALSE
)
str(out_logit$p_is)

# Probit with a scalar sigma
out_probit = compute_establishment_probability(
  r_is_z=r_is_z, C_is_z=C_is_z, S_is_z=S_is_z,
  gamma=gamma, alpha=alpha, beta=beta,
  method="probit", sigma=1
)
# View site-mean probability map (requires ggplot2)
if (requireNamespace("ggplot2", quietly=TRUE)) print(out_probit$plots$site_mean)

# Hard rule (lambda>0)
out_hard = compute_establishment_probability(
  r_is_z=r_is_z, C_is_z=C_is_z, S_is_z=S_is_z,
  gamma=gamma, alpha=alpha, beta=beta,
  method="hard", return_long=TRUE, make_plots=FALSE
)
table(out_hard$prob_long$val)  # 0/1 counts
```

`compute_invasion_fitness`
Compute invasion fitness λ_is for multiple model options

Description

`compute_invasion_fitness()` evaluates the invasion-fitness surface λ_{is} over sites (s) and invaders (i) using three standardized predictors:

- Abiotic suitability $r_{is}^{(z)}$ (alignment between invader traits and the local environment),
- Niche crowding $C_{is}^{(z)}$ (overlap with resident trait space, weighted by composition),
- Resident competition $S_{is}^{(z)}$ (site-level saturation).

Five model variants are supported:

- Option A: $\gamma = 1$
- Option B: global slope θ_0
- Option C: invader-specific slopes θ_i
- Option D: site-varying slopes Γ_{is} and penalties α_{is}
- Option E: signed saturation effect

Optionally, an offset κ can be calibrated so that the mean resident invasion fitness is approximately zero when resident standardized matrices are supplied.

Usage

```
compute_invasion_fitness(
  r_is_z,
  C_is_z,
  S_is_z,
  option = c("A", "B", "C", "D", "E"),
  alpha_i = NULL,
  beta_i = NULL,
  theta0 = 1,
  theta_i = NULL,
  Gamma_is = NULL,
  alpha_is = NULL,
  beta_signed_i = NULL,
  calibrate_kappa = FALSE,
  r_js_z = NULL,
  C_js_z = NULL,
  S_js_z = NULL,
  Q_res = NULL,
  a0 = NULL,
  a1 = NULL,
  a2 = NULL,
  b0 = NULL,
  b1 = NULL,
  b2 = NULL,
  site_df = NULL,
  return_long = TRUE,
  label = NULL
)
```

Arguments

`r_is_z`	Matrix of standardized abiotic suitability with dimensions S by I.
`C_is_z`	Matrix of standardized niche crowding with dimensions S by I.
`S_is_z`	Matrix of standardized site saturation with dimensions S by I.
`option`	Character string specifying the model option. One of `"A"`, `"B"`, `"C"`, `"D"`, or `"E"`.
`alpha_i`	Optional named vector of invader-level crowding sensitivities.
`beta_i`	Optional named vector of invader-level saturation sensitivities.
`theta0`	Global abiotic slope used in options A, B, and E.
`theta_i`	Optional invader-specific abiotic slopes used in option C.
`Gamma_is`	Optional site by invader matrix of abiotic slopes used in option D.
`alpha_is`	Optional site by invader matrix of crowding penalties used in option D.
`beta_signed_i`	Optional signed saturation sensitivities used in option E.
`calibrate_kappa`	Logical; if `TRUE`, compute a calibration offset using resident data.

`r_js_z`, `C_js_z`, `S_js_z`
: Optional resident standardized matrices required when `calibrate_kappa = TRUE`.

`Q_res`
: Optional data frame of resident trait-plane scores.

`a0`, `a1`, `a2`, `b0`, `b1`, `b2`
: Optional numeric coefficients used to derive resident analog slopes when calibrating κ.

`site_df`
: Optional site metadata table with columns `site`, `x`, and `y`.

`return_long`
: Logical; if `TRUE`, return a tidy long-format table.

`label`
: Optional character label attached to the output.

Details

Compute invasion fitness with multiple model options

The invasion fitness is computed as:

- Option A: $\lambda_{is} = r_{is}^{(z)} - \alpha_i C_{is}^{(z)} - \beta_i S_{is}^{(z)} + \kappa$
- Option B: $\lambda_{is} = \theta_0 r_{is}^{(z)} - \alpha_i C_{is}^{(z)} - \beta_i S_{is}^{(z)} + \kappa$
- Option C: $\lambda_{is} = \theta_i r_{is}^{(z)} - \alpha_i C_{is}^{(z)} - \beta_i S_{is}^{(z)} + \kappa$
- Option D: $\lambda_{is} = \Gamma_{is} r_{is}^{(z)} - \alpha_{is} C_{is}^{(z)} - \beta_i S_{is}^{(z)} + \kappa$
- Option E: $\lambda_{is} = \theta_0 r_{is}^{(z)} - \alpha_i C_{is}^{(z)} + \beta_i^{(signed)} S_{is}^{(z)} + \kappa$

When `calibrate_kappa = TRUE`, the offset is chosen so that the mean resident invasion fitness equals zero.

Value

A list containing:

- `lambda_is`: invasion fitness matrix
- `GI`: abiotic slope used
- `AI`: crowding penalties used
- `BI`: saturation penalties used
- `kappa`: calibration offset
- `option`: option label
- `lambda_long`: tidy table (optional)

Examples

```
## ------------------------------------------------------------
## Minimal reproducible example (toy dimensions, no real ecology)
## ------------------------------------------------------------

## S = number of sites, I = number of invaders
S = 5
```

```
I = 3
set.seed(1)

## r_is_z : intrinsic growth component
## rows = sites (s), columns = invaders (i)
r_is_z = matrix(
  rnorm(S * I),
  nrow = S,
  ncol = I,
  dimnames = list(paste0("s", 1:S), paste0("i", 1:I))
)

## C_is_z : crowding / competition component (same shape as r_is_z)
C_is_z = matrix(
  rnorm(S * I),
  nrow = S,
  ncol = I,
  dimnames = dimnames(r_is_z)
)

## S_is_z : site-only environmental gradient
## generated per-site, then broadcast across invaders
S_is_z = matrix(
  rep(scale(rnorm(S)), each = I),
  nrow = S,
  ncol = I,
  dimnames = dimnames(r_is_z)
)

## Invader-specific baseline parameters
alpha_i = setNames(runif(I, 0.2, 1.0), colnames(r_is_z))  # baseline crowding sensitivity
beta_i  = setNames(runif(I, 0.1, 0.5), colnames(r_is_z))  # strength of S effect

## ------------------------------------------------------------
## Option A: fixed gamma = 1, no S effect (kappa = 0)
## ------------------------------------------------------------
outA = compute_invasion_fitness(
  r_is_z, C_is_z, S_is_z,
  option      = "A",
  alpha_i     = alpha_i,
  beta_i      = beta_i,
  theta0      = 1,
  return_long = FALSE
)

## ------------------------------------------------------------
## Option B: gamma shared across invaders (gamma = theta_0)
## ------------------------------------------------------------
outB = compute_invasion_fitness(
  r_is_z, C_is_z, S_is_z,
  option      = "B",
  alpha_i     = alpha_i,
  beta_i      = beta_i,
```

```
  theta0      = 0.8,
  return_long = FALSE
)

## ----------------------------------------------------------------
## Option C: invader-specific gamma_i
## ----------------------------------------------------------------
theta_i = setNames(
  runif(I, 0.5, 1.2),
  colnames(r_is_z)
)

outC = compute_invasion_fitness(
  r_is_z, C_is_z, S_is_z,
  option      = "C",
  alpha_i     = alpha_i,
  beta_i      = beta_i,
  theta_i     = theta_i,
  return_long = FALSE
)

## ----------------------------------------------------------------
## Option D: fully site-varying Gamma_is and alpha_is
## ----------------------------------------------------------------

## gamma_is : site x invader matrix of density-dependence scalars
## constructed by repeating gamma_i across sites
Gamma_is = matrix(
  rep(theta_i, each = nrow(r_is_z)),
  nrow = nrow(r_is_z),
  ncol = ncol(r_is_z),
  dimnames = dimnames(r_is_z)
)

## alpha_is : site x invader crowding sensitivity
## start from invader-specific alpha_i and allow small site-level variation
alpha_is = matrix(
  NA_real_,
  nrow = nrow(r_is_z),
  ncol = ncol(r_is_z),
  dimnames = dimnames(r_is_z)
)

for (j in seq_len(ncol(r_is_z))) {
  alpha_is[, j] = alpha_i[j]
}

## Add site-level noise and enforce positivity.
## pmax() drops matrix attributes, so we re-wrap explicitly.
alpha_is = {
  tmp = pmax(
    0,
    alpha_is + matrix(
```

```
      rnorm(length(alpha_is), 0, 0.1),
      nrow = nrow(r_is_z),
      ncol = ncol(r_is_z)
    )
  )
  matrix(
    tmp,
    nrow = nrow(r_is_z),
    ncol = ncol(r_is_z),
    dimnames = dimnames(r_is_z)
  )
}

outD = compute_invasion_fitness(
  r_is_z, C_is_z, S_is_z,
  option      = "D",
  alpha_is    = alpha_is,
  beta_i      = beta_i,
  Gamma_is    = Gamma_is,
  return_long = FALSE
)

## ----------------------------------------------------------------
## Option E: signed S effect (positive or negative beta_i)
## ----------------------------------------------------------------
beta_signed_i = setNames(
  rnorm(I, 0, 0.3),
  colnames(r_is_z)
)

outE = compute_invasion_fitness(
  r_is_z, C_is_z, S_is_z,
  option          = "E",
  alpha_i         = alpha_i,
  beta_signed_i   = beta_signed_i,
  theta0          = 1,
  return_long     = FALSE
)
```

`compute_resident_crowding` *Resident crowding C_{js} via Hellinger composition and Gower–Gaussian trait kernel*

Description

Given a site × resident community matrix and a resident traits table, this function:

1. normalizes site composition with Hellinger transformation ($W_{s\cdot}$),

2. computes Gower distances among residents (mixed types supported),
3. converts distances to a Gaussian similarity kernel K with bandwidth σ_α,
4. returns the resident crowding matrix $C = W \times K^\top$ (sites $\times$ residents),
5. optionally row-z-scores C for within-site contrasts,
6. optionally draws a heatmap of C (row-z) and a geographic map of site-level summaries.

Usage

```
compute_resident_crowding(
  comm_res,
  traits_res,
  site_df = NULL,
  overlay_sf = NULL,
  method_comp = "hellinger",
  distance_metric = "gower",
  sigma_alpha = NULL,
  row_z = TRUE,
  quantile_probs = 0.95,
  do_heatmap = TRUE,
  show_names = TRUE,
  cluster_rows = TRUE,
  cluster_cols = TRUE,
  heatmap_palette = viridisLite::viridis,
  do_map = TRUE
)
```

Arguments

`comm_res`	A numeric matrix or data.frame of **sites × resident species** (non-negative abundances or transformed counts). Row names must be site IDs.
`traits_res`	A data.frame of **resident species × traits**. Row names must be resident IDs that match the column names of `comm_res`. Mixed types allowed.
`site_df`	Optional data.frame with columns `site`, `x`, `y` for mapping. If provided and `do_map=TRUE`, a site-level map is produced.
`overlay_sf`	Optional `sf` object to outline on the map (e.g., study boundary).
`method_comp`	Character method for `vegan::decostand`. Default `"hellinger"`.
`distance_metric`	Character distance name for `cluster::daisy`. Default `"gower"`. You usually want `"gower"` for mixed traits.
`sigma_alpha`	Optional numeric bandwidth for the Gaussian kernel. If `NULL` (default), uses the median of positive pairwise Gower distances. If that is missing or zero, falls back to `0.5`.
`row_z`	Logical; if `TRUE` (default) return a row-z-scored version `C_js_z` for within-site contrasts.
`quantile_probs`	Numeric vector of probabilities to summarize site-level crowding quantiles (e.g., `c(0.95)`). Use `NULL` to skip quantiles. Default `0.95`.

`do_heatmap` Logical; if `TRUE` (default) draw a clustered heatmap of `C_js_z` using **pheatmap** if available.

`show_names` Logical; show row/column names on the heatmap. Default `TRUE`.

`cluster_rows` Logical; cluster sites on the heatmap. Default `TRUE`.

`cluster_cols` Logical; cluster residents on the heatmap. Default `TRUE`.

`heatmap_palette` Function returning colors (e.g., `viridisLite::viridis`). Default `viridisLite::viridis`.

`do_map` Logical; if `TRUE`, draw a ggplot map of mean crowding per site when `site_df` is provided. Default `TRUE`.

Details

Compute resident crowding from community composition and trait similarity

The Hellinger transformation places compositions in a Euclidean-friendly space. Gower distance handles mixed trait types. Distances are converted to similarities with $K_{ij} = \exp(-D_{ij}^2/(2\sigma_\alpha^2))$, and the diagonal is set to 0 to remove self-crowding. Crowding for resident j at site s is the weighted sum of trait-similar resident composition at that site: $C_{sj} = \sum_r W_{sr} K_{jr}$.

Value

A named list with:

- `W_site` — Hellinger composition matrix (sites × residents).
- `D_res` — resident × resident Gower distance matrix.
- `sigma_alpha` — Gaussian bandwidth used.
- `K_res_res` — Gaussian similarity kernel with zero diagonal.
- `C_js` — resident crowding matrix (sites × residents).
- `C_js_z` — row-z-scored version of `C_js` (if `row_z=TRUE`).
- `site_summary` — data.frame with per-site summaries (mean and requested quantiles).
- `heatmap` — `pheatmap` object or `NULL` if not drawn.
- `map` — `ggplot` object or `NULL` if not drawn.

Examples

```
## Not run:
# Fake data: 6 sites × 5 residents
set.seed(42)
comm_res = matrix(rpois(30, lambda = 5), nrow = 6,
                  dimnames = list(paste0("s", 1:6), paste0("sp", 1:5)))
traits_res = data.frame(
  size = rnorm(5, 10, 2),
  diet = factor(sample(c("A","B"), 5, TRUE)),
  row.names = paste0("sp", 1:5)
)
site_df = data.frame(site = paste0("s", 1:6),
```

```
                      x = rep(1:3, each=2),
                      y = rep(1:2, 3))

out = compute_resident_crowding(
  comm_res   = comm_res,
  traits_res = traits_res,
  site_df    = site_df,
  quantile_probs = 0.95,
  do_heatmap = TRUE,
  do_map     = TRUE
)

str(out, max.level = 1)
out$heatmap  # pheatmap object
out$map      # ggplot map of mean C per site

## End(Not run)
```

`compute_site_saturation` *Compute site-only saturation S_s and global z-score S_s^(z)*

Description

Implements three site-only saturation definitions and returns (i) the raw site statistic S_s, (ii) its global mean/SD, (iii) the global z-scored site statistic $S_s^{(z)}$, and (iv) a **broadcast** site-resident matrix $S_{js}^{(z)}$ for downstream modelling.

Usage

```
compute_site_saturation(mode, comm_res, res_ids, mu_js = NULL)
```

Arguments

`mode`	One of `c("opportunity_penalty","modelled_dominance","evenness_deficit")`.
`comm_res`	Site x resident matrix (or frame) of observed counts. Will be coerced to a numeric matrix; non-numeric entries become `NA`.
`res_ids`	Character vector of resident IDs (column names to consider). Columns not found in `comm_res` are ignored with a warning.
`mu_js`	Optional site x resident matrix (or frame) of expected abundances, required for `mode = "modelled_dominance"`. Will be coerced to numeric matrix.

Value

A list with components:

`S_s` Numeric vector (sites) of saturation values.

`S_mu` Global mean of `S_s`.

`S_sd` Global SD of `S_s` (set to 1 if 0/NA).

`S_s_z` Global z-scores of `S_s`.

`S_js_z` Site x resident matrix broadcasting `S_s_z` across `res_ids`.

Definitions

Let X_{js} be observed resident counts and μ_{js} expected counts.

`"opportunity_penalty"` Abundance penalty: $S_s = \log(1 + \sum_j X_{js})$.

`"modelled_dominance"` Model-based crowding: $S_s = \log(1 + \sum_j \mu_{js})$. Requires `mu_js`.

`"evenness_deficit"` Pielou's evenness deficit: $J_s = H_s / \log S_s^{(\#)}$ with Shannon H_s and richness $S_s^{(\#)}$; then $S_s = 1 - J_s$.

After computing S_s, a global z-score is applied: $S_s^{(z)} = (S_s - \bar{S})/\mathrm{sd}(S_s)$, and this is broadcast across residents to form $S_{js}^{(z)}$ (same value for all j within a site s).

See Also

Package reference: compute_site_saturation()

Examples

```
## Not run:
# Evenness deficit from observed community
sat = compute_site_saturation("evenness_deficit", comm_res, colnames(comm_res))
head(sat$S_s_z)

# Modelled dominance using expected abundances mu_js
sat2 = compute_site_saturation("modelled_dominance", comm_res, colnames(comm_res), mu_js = mu_js)
dim(sat2$S_js_z)

## End(Not run)
```

`compute_trait_space` *Shared trait-space construction (Gower + PCoA), resident hull, centroid, and density plot*

Description

Builds a unified trait space for resident and invader species by: (1) computing a Gower distance on the stacked trait table, (2) projecting to 2D via classical MDS/PCoA (`stats::cmdscale`), (3) deriving the resident convex hull (realised niche region), (4) estimating a kernel density over the 2D space, and (5) optionally rendering a base R `filled.contour` map that overlays the hull, all points (residents black, invaders red), and the cloud centroid (white square with black outline). Optionally it also computes a functional dendrogram (`hclust`) and a pretty dendrogram plot (`factoextra::fviz_dend`) when available.

Usage

```
compute_trait_space(
  traits_res,
  traits_inv,
  k = 2,
  kde_n = 100,
  pad_prop = 0.1,
  main_title = "Trait space density with convex hull and centroid",
  legend_line =
    "Hull = realised niche; white square = centroid; black dots = residents; red dots = invader
  cex_main = 1,
  cex_sub = 0.72,
  cex_lab = 0.85,
  cex_axis = 0.75,
  highlight_level = 0.5,
  do_plot = TRUE,
  do_dend = FALSE
)
```

Arguments

`traits_res`	A data.frame of resident species (rows) by traits (columns). Mixed types are supported (numeric, integer, factor, ordered) via Gower distance. Row names are used as resident IDs; if absent, sequential IDs are created.
`traits_inv`	A data.frame of invader species (rows) by traits (columns). Must share the same trait columns (names and types) as `traits_res`. Row names are used as invader IDs; if absent, sequential IDs are created. If you have no invaders, pass a 0-row data.frame with matching columns.
`k`	Integer embedding dimension for PCoA; default 2.
`kde_n`	Integer grid size for 2D kernel density estimation (per axis). Default `100`.

`pad_prop` Numeric padding proportion added to the plotting range on each axis. Default `0.10` (10%).

`main_title` Character main title for the contour plot.

`legend_line` Character subtitle used as an inline legend in the title area.

`cex_main`, `cex_sub`, `cex_lab`, `cex_axis` Numeric text scaling for main title, subtitle, axis labels, and axis tick labels, respectively. Defaults `1, 0.72, 0.85, 0.75`.

`highlight_level` Numeric in (0,1]; draws a bold contour at this proportion of the maximum density (e.g., `0.5`=50% of max). Set `NA` to skip.

`do_plot` Logical; if `TRUE` renders the `filled.contour` figure. Default `TRUE`.

`do_dend` Logical; if `TRUE`, computes `hclust` on Gower and attempts to produce a dendrogram plot with `factoextra::fviz_dend` when available. Default `FALSE`.

Details

Compute and plot shared trait space for residents and invaders

This function assumes that the trait columns in `traits_inv` are compatible with those in `traits_res`. Gower distance (`cluster::daisy`) supports mixed data types and handles factors/ordered factors appropriately. The PCoA (`cmdscale`) is computed on the Gower distances and the first two axes are returned by default for visualisation. The resident convex hull is computed on the resident scores only and is drawn as the "realised niche region".

Value

A list with:

- `gower` - Gower distance matrix (all species).
- `scores` - Data frame of 2D PCoA scores (`Q_all`) with columns `tr1,tr2`.
- `Q_res`, `Q_inv` - Subsets of `scores` for residents and invaders.
- `hull_res` - Data frame (tr1,tr2) of the resident convex hull ring (closed), or `NULL` if $<$ 3 residents.
- `centroid` - Numeric vector of the overall (res+inv) centroid `c(tr1, tr2)`.
- `density` - List `list(x, y, z)` from `MASS::kde2d`.
- `colors` - Named vector with color per species ID (residents black, invaders red).
- `hc` - `hclust` object (if `do_dend=TRUE`), else `NULL`.
- `dend_plot` - A ggplot object from `factoextra::fviz_dend` when available, else `NULL`.
- `xlim`, `ylim` - Numeric length-2 ranges used for plotting with padding.

Examples

```
## Not run:
# Minimal reproducible example with numeric traits
set.seed(1)
traits_res = data.frame(
  trait_1 = rnorm(20),
  trait_2 = rnorm(20, 2),
  row.names = paste0("sp", 1:20)
)
traits_inv = data.frame(
  trait_1 = rnorm(5, 0.5),
  trait_2 = rnorm(5, 2.5),
  row.names = paste0("inv", 1:5)
)

out = compute_trait_space(
  traits_res = traits_res,
  traits_inv = traits_inv,
  do_plot    = TRUE,
  do_dend    = TRUE,
  main_title = "Trait space density with hull and centroid",
  legend_line = "Hull = realised niche region;
  white square = centroid; black = residents; red = invaders"
)

# Access returned objects
head(out$scores)
out$centroid
if (!is.null(out$dend_plot)) print(out$dend_plot)

## End(Not run)
```

---

derive_sensitivities *Derive trait-varying sensitivities (,) and abiotic slope (, Γ)*

Description

From an auxiliary GLMM on standardized predictors, extract fixed-effect systems for `r_z`, `C_z`, and `S_z`, evaluate them at invader trait coordinates, and (optionally) test whether the abiotic slope on `r_z` should vary with traits (`tr1`, `tr2`). Biotic terms are enforced as nonnegative penalties on invasion fitness.

Usage

```
derive_sensitivities(fit_coeffs, Q_inv, inv_ids, lrt = TRUE)
```

Arguments

`fit_coeffs`	A fitted `glmmTMB` model on `(r_z + C_z + S_z) × (tr1 + tr2)`.
`Q_inv`	Data frame with invader trait coordinates `tr1`, `tr2` (rownames = invader IDs).
`inv_ids`	Character vector of invader IDs specifying the output order.
`lrt`	Logical **or** character. If logical: • `TRUE` → Wald test; `FALSE` → no test. If character, one of `"wald"`, `"lrt"`, or `"none"`. Default is equivalent to `"wald"` for `TRUE` and `"none"` for `FALSE`.

Details

Biotic penalties (prior/constraint). Invasion fitness is modeled as

$$\lambda_{is} = \Gamma_{is}\, r_{is}^{(z)} \;-\; \alpha_i\, C_{is}^{(z)} \;-\; \beta_i\, S_{is}^{(z)}.$$

We therefore set $\alpha_i = \max(0, -\text{slope}_{C,i})$ and $\beta_i = \max(0, -\text{slope}_{S,i})$, where $\text{slope}_{C,i}$ and $\text{slope}_{S,i}$ are the fitted (possibly trait-varying) slopes for `C_z` and `S_z`. Signed versions (before clamping) are returned for diagnostics.

Testing trait-varying abiotic slope. The argument `lrt` controls whether and how to test the joint null $H_0 : r_z : tr1 = r_z : tr2 = 0$.

- For backward compatibility, `lrt = TRUE` performs a **Wald** joint ^{2} test using the fixed-effects variance–covariance (no refit).
- `lrt = FALSE` performs no test.
- Alternatively, pass a character string: `"wald"`, `"lrt"`, or `"none"`.
  - `"lrt"` refits a reduced model (dropping both interactions) and runs a likelihood-ratio test (`anova(..., test = "Chisq")`).

If the chosen test is significant at $\alpha = 0.05$, the function uses the trait-varying `theta_i`; otherwise a constant `theta0` is used for `gamma_i`.

Value

A list with components:

alpha_i, beta_i Nonnegative penalties for `C_z` and `S_z`.

alpha_signed_i, beta_signed_i Signed (pre-clamp) slopes.

theta0 Intercept for `r_z` slope system.

theta_i Trait-varying `r_z` slope evaluated at invader traits.

gamma_i Either `theta_i` (if test is significant) or a constant vector equal to `theta0` (if not).

a0,a1,a2; b0,b1,b2 Coefficient components for `C_z` and `S_z` systems: main effect and interactions with `tr1`,`tr2`.

df Per-invader tidy data frame with traits, signed slopes, clamping flags, and `theta_i`/`gamma_i`.

clamp_summary Counts and proportions of clamped α and β.

use_theta Logical; `TRUE` if the chosen test was significant.

lrt **Back-compat alias** of `test_tab` (data frame with test result).

test_tab Data frame with test statistic, df, and p-value (or `NULL`).

test_type One of `"wald"`, `"lrt"`, or `"none"`.

prior_note Reminder of the nonnegative-penalty prior.

See Also

`glmmTMB::glmmTMB()`, `stats::anova()`

`fit_auxiliary_residents_glmm` *Auxiliary GLMM for trait-varying and site-varying slopes*

Description

`fit_auxiliary_residents_glmm()` assembles a long table of resident site by species cells with standardized predictors $r_{js}^{(z)}$, $C_{js}^{(z)}$, and $S_{js}^{(z)}$, together with two-dimensional trait scores (`tr1`, `tr2`). It then fits a Gaussian generalized linear mixed model to estimate how slopes on abiotic suitability, niche crowding, and site saturation vary across the trait plane.

Optional site-level random slopes can be included for `r_z` and `C_z`. Random slopes for `S_z` are intentionally excluded because `S_z` is site-only and has no within-site variation across residents.

This function is intentionally forgiving: non-matrix inputs are coerced using `as.matrix()`, missing dimension names are repaired when possible, and inputs are aligned to `comm_res`.

Usage

```
fit_auxiliary_residents_glmm(
  comm_res,
  r_js_z,
  C_js_z,
  S_js_z,
  Q_res,
  use_site_random_slopes = TRUE,
  family = gaussian(),
  control = glmmTMB::glmmTMBControl(optCtrl = list(iter.max = 1e+05, eval.max = 1e+05)),
  na_action = c("drop", "error"),
  verbose = TRUE
)
```

Arguments

`comm_res`	Numeric matrix of site by resident abundances. Row names are site identifiers and column names are resident identifiers.
`r_js_z`	Numeric matrix of standardized abiotic suitability with the same dimensions and names as `comm_res`.
`C_js_z`	Numeric matrix of standardized niche crowding with the same dimensions and names as `comm_res`.
`S_js_z`	Numeric matrix of standardized site saturation broadcast across residents, with the same dimensions and names as `comm_res`.
`Q_res`	Data frame of resident trait scores. Must contain numeric columns `tr1` and `tr2`, with row names matching `colnames(comm_res)`.
`use_site_random_slopes`	Logical; if `TRUE`, include random slopes for `r_z` and `C_z` at the site level.
`family`	GLM family for the conditional model. Default is `gaussian()` applied to `log1p(abundance)`.
`control`	Control object passed to `glmmTMB::glmmTMB()`.
`na_action`	How to handle rows with missing values. Either `"drop"` (default) or `"error"`.
`verbose`	Logical; if `TRUE`, print a short model summary line.

Details

Fit an auxiliary residents-only GLMM on standardized predictors

The fixed-effects component of the model is

$$\log(1 + \text{abundance}) \sim (r_z + C_z + S_z) \times (tr1 + tr2),$$

with random intercepts for species and site always included.

When `use_site_random_slopes = TRUE`, random slopes $(0 + r_z|site)$ and $(0 + C_z|site)$ are added. Random slopes for `S_z` are omitted because `S_z` is constant within each site and therefore not identifiable.

Value

A list with components:

- `fit`: the fitted `glmmTMB` model
- `data`: the long-format data frame used for fitting
- `formula`: the model formula
- `args`: resolved arguments used in the fit

Role in invasion fitness

This auxiliary model estimates trait-dependent slope systems that can be mapped to invader-level parameters such as θ_i, α_i, and β_i, and optionally to site-varying counterparts via random slopes. These parameters enter directly into the linear invasion-fitness decomposition

$$\lambda_{is} = \Gamma_{is} r_{is}^{(z)} - \alpha_{is} C_{is}^{(z)} - \beta_i S_{is}^{(z)},$$

linking trait position to abiotic suitability, niche crowding, and resident competition.

Examples

```
## Not run:
# Toy dimensions
set.seed(1)
S = 8  # sites
J = 6  # residents
sites   = paste0("s", 1:S)
res_ids = paste0("sp", 1:J)

# Site x resident abundance
comm_res = matrix(rpois(S*J, lambda = 2), S, J,
                  dimnames = list(sites, res_ids))

# Standardized predictors (row-wise z by site for r and C; site-only for S)
r_raw = matrix(rnorm(S*J), S, J, dimnames = dimnames(comm_res))
C_raw = matrix(rnorm(S*J), S, J, dimnames = dimnames(comm_res))
S_raw = matrix(rep(scale(rnorm(S))[,1], each = J), S, J,
               dimnames = dimnames(comm_res))
# Simple per-site z for demo
r_js_z = t(scale(t(r_raw))); r_js_z[!is.finite(r_js_z)] = 0
C_js_z = t(scale(t(C_raw))); C_js_z[!is.finite(C_js_z)] = 0
S_js_z = S_raw

# Resident trait-plane scores (PCoA on Gower in real workflow)
Q_res = data.frame(tr1 = rnorm(J), tr2 = rnorm(J))
rownames(Q_res) = res_ids

aux = fit_auxiliary_residents_glmm(
  comm_res   = comm_res,
  r_js_z     = r_js_z,
  C_js_z     = C_js_z,
  S_js_z     = S_js_z,
  Q_res      = Q_res,
  use_site_random_slopes = TRUE
)

summary(aux$fit)
head(aux$data)
aux$formula

## End(Not run)
```

`get_trait_data` *Scrape and Analyze Wikipedia & Trait Data for a Species*

Description

Given a binomial species name, this function retrieves optional metadata from Wikipedia (taxonomic summary, taxonomy, image, and a color palette) and joins relevant plant/trait data from a TRY-style or user-provided trait table. Fuzzy matching is used for both TRY and local tables to handle minor spelling or naming mismatches.

Usage

```
get_trait_data(
  species,
  remove_bg = FALSE,
  do_palette = TRUE,
  do_taxonomy = TRUE,
  do_summary = TRUE,
  do_image = TRUE,
  bg_thresh = 80,
  green_delta = 20,
  n_palette = 5,
  preview = FALSE,
  save_folder = NULL,
  use_try = FALSE,
  try_data = NULL,
  trait_species_col = "AccSpeciesName",
  local_trait_df = NULL,
  local_species_col = "species",
  max_dist = 1
)
```

Arguments

`species`	Character. Species name (binomial, e.g. `"Acacia karroo"`).
`remove_bg`	Logical. If `TRUE`, call remove.bg via `remove_bg_and_save()` to remove the background from the Wikipedia infobox image and use the processed PNG for preview/palette. Default: `FALSE`.
`do_palette`, `do_taxonomy`, `do_summary`, `do_image`	Logical. Control which metadata to scrape. All default to `TRUE`.
`bg_thresh`	Integer. Deprecated/ignored when `remove_bg = TRUE`. Kept for signature compatibility. Default: `80`.
`green_delta`	Integer. Deprecated/ignored when `remove_bg = TRUE`. Kept for signature compatibility. Default: `20`.
`n_palette`	Integer. Number of colors to extract for the palette. Default: `5`.

`preview` Logical. Print the processed image (magick) in the console. Default: `FALSE`.

`save_folder` Character or `NULL`. If non-`NULL`, write the PNG used for palette/preview into this folder. When `remove_bg = TRUE`, the background-removed PNG is written here; otherwise the original image is written. If `NULL` and `remove_bg = TRUE`, a temporary directory is used.

`use_try` Logical. If `TRUE`, join plant traits using a TRY-format database/table. Default: `FALSE`.

`try_data` Character (path) or `data.frame`. Path to a TRY file, or a data frame containing TRY-style trait data (must include `trait_species_col`, `TraitName`, and `OrigValueStr`).

`trait_species_col`
Name of the species column in the TRY trait table. Default: `"AccSpeciesName"`.

`local_trait_df`
Optional. `data.frame` of local trait data (any wide table). All columns except the species column are returned.

`local_species_col`
Name of the species column in the local trait table. Default: `"species"`.

`max_dist` Numeric. Maximum distance for fuzzy matching (Jaro–Winkler via `fuzzyjoin::stringdist_le`
Default: `1`.

Details

When `remove_bg = TRUE`, the infobox image background is removed using the remove.bg API via an internal helper (`remove_bg_and_save()`), the resulting PNG is re-read with **magick**, and the palette is extracted from that processed image. Set the environment variable `REMOVE_BG_API_KEY` to a valid remove.bg API key before calling.

- **Wikipedia:** summary via REST API; taxonomy parsed from the infobox.
- **Image:** first infobox image is used; when `remove_bg = TRUE` the function calls the remove.bg API. Set `Sys.setenv(REMOVE_BG_API_KEY = "…")`.
- **Palette:** simple k-means on non-transparent pixels of the (processed) PNG.
- **Traits (TRY):** wide table produced from `TraitName` and numeric `OrigValueStr`.
- **Traits (local):** returns all columns except the species column.
- **Dependencies:** `dplyr`, `purrr`, `tibble`, `fuzzyjoin`, `rvest`, `httr`, `stringr`, `jsonlite`, `magick`, `abind`.

Value

A tibble (one row) with columns: `species`, optional metadata (`summary`, taxonomy ranks, `img_url`, `palette`, `image_file`), and all available trait columns found via TRY/local joins. `image_file` contains the normalized path to the PNG used for palette/preview (or `NA` if none).

Examples

```
## Not run:
# Using TRY table
get_trait_data("Acacia karroo",
               use_try = TRUE,
               try_data = try_traits,
               trait_species_col = "SpeciesName")

# Using a local trait table
get_trait_data("Acraea horta", local_trait_df = traits, local_species_col = "species")

# Metadata only, with background removal and saving to a folder
Sys.setenv(REMOVE_BG_API_KEY = "<your-removebg-key>")
get_trait_data("Acacia karroo", use_try = FALSE, remove_bg = TRUE, save_folder = "out/")

## End(Not run)
```

---

learn_sensitivities *Learn sensitivities (alpha_i, beta_i, theta_i/gamma_i) and optional site-varying alpha_is, Gamma_is*

Description

Fits an auxiliary GLMM on resident data to estimate invader-level sensitivities to crowding and saturation (α_i, β_i), and abiotic conversion slopes (θ_i or γ_i). When supported by the data, optional site-level random slopes yield site-varying adjustments (α_{is}, Γ_{is}).

Results are written into the `fit$sensitivities` slot of an new_invasimapr_fit object for downstream invasion-fitness and establishment calculations.

Usage

```
learn_sensitivities(fit, use_site_random_slopes = TRUE, lrt = TRUE)
```

Arguments

`fit`	An object produced by prepare_inputs and assemble_matrices, containing resident predictor matrices (`r_js_z`, `C_js_z`, `S_js_z`), trait-space structures (`Q_res`, `Q_inv`), and the resident community layout (`inputs$comm_res`).
`use_site_random_slopes`	Logical; if `TRUE`, the auxiliary model includes site-level random slopes for abiotic and crowding terms, enabling estimation of site-varying α_{is} and Γ_{is} when supported by the data. Defaults to `TRUE`.
`lrt`	Logical; if `TRUE`, compute Wald or likelihood-ratio tests for key contrasts (e.g., trait-varying versus global slopes). Defaults to `TRUE`.

Details

Fits an auxiliary GLMM on **resident** data to estimate invader-level sensitivities to crowding and saturation (alpha_i, beta_i), and abiotic conversion slopes (theta_i or gamma_i), with optional **site-varying** random slopes that yield per-site adjustments (alpha_is, Gamma_is). Results are written into the `fit$sensitivities` slot of an `invasimapr_fit` object for downstream invasion-fitness and establishment steps.

Workflow

1. Fit an auxiliary GLMM on resident responses using fit_auxiliary_residents_glmm, optionally including site-level random slopes for r_z and C_z.
2. Convert GLMM coefficients to sensitivities (α_i, β_i, θ_i or γ_i) using derive_sensitivities, returning signed and unsigned variants plus inference summaries.
3. When supported, extract site-varying effects (α_{is}, Γ_{is}) via site_varying_alpha_beta_gamma.

The resulting components are stored in `fit$sensitivities`, including:

- global and trait-varying sensitivities;
- inference diagnostics and clamping summaries;
- optional site-varying matrices and compact decomposition tables.

Value

The input `fit` object (invisibly), with an updated `fit$sensitivities` list.

See Also

prepare_inputs, assemble_matrices, fit_auxiliary_residents_glmm, derive_sensitivities, site_varying_alpha_beta_gamma

Examples

```
## Not run:
fit <- prepare_inputs(
  sites = site_df,
  residents = resident_df,
  invaders = invader_df,
  traits = trait_df
)

fit <- learn_sensitivities(fit, use_site_random_slopes = TRUE)
names(fit$sensitivities)

## End(Not run)
```

`model_residents` *Resident GLMM and construction of standardized predictors*

Description

`model_residents()` fits a residents-only generalized linear mixed model and constructs site-standardized predictors used for learning sensitivities and projecting invaders. Specifically, the function:

1. builds a fixed-effects design from environment and resident traits,
2. optionally expands factors and applies PCA compression with stored centering and scaling metadata,
3. fits a `glmmTMB` model to resident abundance,
4. predicts resident linear predictors r_{js} and applies row-wise z-standardisation to obtain $r_{js}^{(z)}$,
5. standardises resident crowding C_{js} and computes site saturation summaries S_s, $S_s^{(z)}$, and $S_{js}^{(z)}$.

Usage

```
model_residents(
  fit,
  family = glmmTMB::tweedie(link = "log"),
  include_env_trait_interactions = TRUE,
  saturation_mode = c("evenness_deficit", "opportunity_penalty", "modelled_dominance"),
  robust_r = TRUE,
  robust_c = TRUE,
  fit_model = TRUE,
  max_dense_gb = 8,
  reduce_strategy = c("auto", "none", "no_interactions", "pca"),
  pca_env_k = 5,
  pca_trait_k = 5,
  verbose = TRUE
)
```

Arguments

`fit`	An object of class `invasimapr_fit` produced by `prepare_trait_space()`. Must contain resident inputs, environment covariates, and raw crowding matrices.
`family`	A `glmmTMB` family. Default is `glmmTMB::tweedie(link = "log")`.
`include_env_trait_interactions`	Logical; whether to include environment by trait interactions when not reduced.

`saturation_mode`
: Character string controlling site saturation computation. Passed to `compute_site_saturation()`

`robust_r`, `robust_c`
: Logical; whether to use robust row-wise z-standardisation for r_{js} and C_{js}.

`fit_model`
: Logical; if `FALSE`, perform preflight only and return without fitting the model.

`max_dense_gb`
: Numeric; maximum allowed dense fixed-effects size (GB) in automatic reduction mode.

`reduce_strategy`
: One of `"auto"`, `"none"`, `"no_interactions"`, or `"pca"`.

`pca_env_k`, `pca_trait_k`
: Integers giving the number of retained principal components for environment and trait blocks.

`verbose`
: Logical; if `TRUE`, emit progress messages.

Details

Model residents and build standardized predictors

Design construction. Environment and trait tables are standardised prior to model fitting. Character variables are coerced to factors to ensure stable dummy expansion. Under PCA reduction, each block is one-hot encoded, column-standardised, and passed to `prcomp()`. All centering, scaling, and rotation metadata are stored to enable reproducible projection of invaders.

Preflight memory checks. In automatic reduction mode, the dense fixed-effects design size is estimated before fitting. If the design exceeds `max_dense_gb`, interactions are removed and/or PCA compression is applied until the constraint is satisfied.

Z-standardisation. Row-wise z-standardisation stores site-level means and standard deviations for later use in invasion fitness and probability mapping.

Value

The input `fit` object with updated components:

`model` Standardised inputs, PCA objects, and metadata required for invader projection.

`residents` Resident GLMM fit, prediction grid, raw and standardised predictor matrices, and site saturation summaries.

`model$pca_used` Flags indicating whether PCA was applied and the retained dimensionality.

Reduction strategies

The argument `reduce_strategy` controls fixed-effects complexity.

- `"auto"`: estimate dense design size and iteratively reduce complexity until the memory budget is met.
- `"no_interactions"`: drop all environment by trait interactions.
- `"pca"`: dummy-expand, standardise, and apply `prcomp()` to environment and trait blocks, retaining the first components.
- `"none"`: retain the full requested fixed-effects structure.

See Also

- `build_model_formula()`
- `prep_resident_glmm()`
- `compute_site_saturation()`
- `prepare_trait_space()`
- `predict_invaders()`

Examples

```
## Not run:
fit <- prepare_inputs(sites = site_df, residents = resident_df,
                      invaders = invader_df, traits = trait_df)
fit <- prepare_trait_space(fit, traits_inv)
fit <- model_residents(fit, reduce_strategy = "auto", verbose = TRUE)

# Inspect z-standardised predictors for residents
dim(fit$residents$r_js_z); dim(fit$residents$C_js_z)
fit$residents$messages

## End(Not run)
```

---

predict_establishment

Compute invasion fitness (λ) and optional establishment probability (P)

Description

Wraps `compute_invasion_fitness()` to generate invader-by-site invasion fitness λ_{is} under model Options **A-E**, and (optionally) maps λ to probabilities via `compute_establishment_probability()`. Supports calibration of κ so that **resident** mean λ is approximately zero, and can overlay an `sf` boundary on map-like plots.

Usage

```
predict_establishment(
  fit,
  option = c("A", "B", "C", "D", "E"),
  calibrate_kappa = FALSE,
  prob_method = c(NULL, "probit", "logit", "hard"),
  prob_args = list(),
  method = NULL,
  prob_scale = NULL,
  boundary_sf = NULL,
  boundary_params = list(inherit.aes = FALSE, fill = NA, color = "black", size = 0.3)
)
```

Arguments

`fit`	An `invasimapr_fit` from `predict_invaders()`, containing `invaders$r_is_z/C_is_z/S_is_z`, resident summaries, and learned sensitivities.
`option`	Character, one of `c("A","B","C","D","E")` selecting the invasion-fitness specification: **A** Baseline: $\lambda = r^{(z)} - \alpha C^{(z)} - \beta S^{(z)}$. **B** Global abiotic scaling $\theta_0 \cdot r^{(z)}$. **C** Trait-varying abiotic scaling $\theta_i \cdot r^{(z)}$. **D** Site-varying abiotic and crowding Γ_{is}, α_{is}. **E** Signed saturation effect (facilitation allowed via signed β_i).
`calibrate_kappa`	Logical; if `TRUE`, set κ so mean **resident** $\lambda \approx 0$ (scale alignment for communication/comparison).
`prob_method`	(legacy) `NULL` or one of `c("probit","logit","hard")`; preserved for backward compatibility.
`prob_args`	(legacy) List of arguments passed to `compute_establishment_probability()` (e.g., `sigma`, `tau`, `predictive`, `sigma_mat`, `site_df`, `return_long`, `make_plots`).
`method`	Alias of `prob_method` (preferred in user code).
`prob_scale`	Alias of `prob_args` (preferred in user code).
`boundary_sf`	Optional **sf** object to overlay on map-like probability plots.
`boundary_params`	Named list for `ggplot2::geom_sf()` aesthetics; default `list(inherit.aes=FALSE, fill=NA, color="black", size=0.3)`.

Details

What this wrapper does

1. Chooses the appropriate sensitivity inputs for the requested `option` (e.g., `theta_i` for C, `Gamma_is`/`alpha_is` for D).
2. Calls ... with site-standardised inputs $(r^{(z)}, C^{(z)}, S^{(z)})$ held in `fit`.
3. (Optional) Converts λ to probability P via probit, logit, or a hard threshold, forwarding `prob_*` settings.
4. (Optional) If `boundary_sf` is supplied and plots are available, overlays the boundary on map-like plots using `geom_sf()`.

Calibration (`calibrate_kappa = TRUE`) Aligns invader-resident scales by shifting λ so that the **resident** mean is ~0, using resident moments and trait-plane slopes stored in `fit`.

Value

The input `fit` with:

`$fitness` List returned by `compute_invasion_fitness()` (including `lambda_is`, `lambda_long`, `option`, and any plots/maps).

`$prob` If requested, list returned by `compute_establishment_probability()` with probability tables and plots (with boundary overlay applied when provided).

See Also

- Fitness: `compute_invasion_fitness()`
- Probabilities: `compute_establishment_probability()`
- Sensitivities: `learn_sensitivities()`
- Invader predictors: `predict_invaders()`

Examples

```
## Not run:
fit = fit |>
  predict_invaders(traits_inv) |>
  predict_establishment(option = "C", calibrate_kappa = TRUE,
                        method = "probit", prob_scale = list(sigma = 1))

# Overlay a boundary on probability maps
fit = predict_establishment(fit, option = "D", method = "logit",
                            prob_scale = list(tau = 1),
                            boundary_sf = rsa_boundary)

## End(Not run)
```

`predict_invaders` *Build standardized invader predictors (r_is_z, C_is_z, S_is_z)*

Description

Uses resident-side moments and the residents-only model to construct invader-level predictors on the **site-standardised** scale: $r_{is}^{(z)}$, $C_{is}^{(z)}$, $S_{is}^{(z)}$. Handles PCA projection with "dummy" factor expansion so that invader covariates exactly match the **training design** used in `model_residents()` / the stored resident GLMM.

Usage

```
predict_invaders(fit, traits_inv)
```

Arguments

`fit`	`invasimapr_fit` after `prepare_trait_space()` / `learn_sensitivities()` and resident model fitting (i.e., it contains resident GLMM artefacts, PCA objects/metadata, and resident moments such as `r_mu_s`, `r_sd_s`, `C_mu_s`, `C_sd_s`).
`traits_inv`	**data.frame** of *raw* invader trait values; columns must match the resident trait names used at training (numerics and/or factors).

Details

Pipeline

1. **Environment**: if an environment PCA was used at training, rebuild the environment design matrix with the saved factor map and project using the stored rotation; otherwise reuse the stored `env_df_z`.
2. **Traits**: coerce invader raw trait factors to training levels (redirect unseen levels to `_other_` if present); rebuild the dummy matrix and standardise with stored means/SDs; project via trait PCA if present; then append any raw factor terms that remained in the fixed-effects formula.
3. **Predictors**: call `build_invader_predictors()` to compute $r_{is}^{(z)}$, $C_{is}^{(z)}$, $S_{is}^{(z)}$ using resident moments, crowding kernels, and similarity structures stored in `fit`.

Assumptions & safeguards

- Requires a trained resident model stored in `fit$residents$fit_r` and PCA metadata in `fit$model$*_pca*` if PCA was used.
- Uses resident moments (`r_mu_s`, `r_sd_s`, `C_mu_s`, `C_sd_s`) and similarity/crowding information (`W_site`, `gower`).
- Site ordering is taken from `fit$meta$sites`; inputs are conformed to it.

Value

The input `fit` with an updated `fit$invaders` list containing:

`traits_inv_raw` Raw (unmodified) invader trait table (as supplied).

`traits_inv_glmm` Design-scale trait data passed to the resident model (post dummy-expansion, standardisation, PCA, and factor alignment).

`r_is_z`, `C_is_z`, `S_is_z` Site-standardised matrices for invaders.

`df` Tidy table used/returned by `build_invader_predictors()` for joins.

Rationale

Many downstream steps (e.g. invasion fitness, establishment probability) assume invader predictors live on the **same centred/scaled basis** as the resident training data. This function guarantees alignment by: (i) coercing raw trait factor levels to training levels (with `_other_` fallbacks), (ii) rebuilding the design matrix with the training dummy map, (iii) applying the stored centring/scaling, and (iv) projecting through the stored PCA rotations before calling `build_invader_predictors()`.

See Also

- Input assembly: `assemble_matrices()`
- Predictor builder: `build_invader_predictors()`
- Residents GLMM: `fit_auxiliary_residents_glmm()`
- Sensitivities: `learn_sensitivities()`

Examples

```
## Not run:
fit <- prepare_inputs(sites = site_df, residents = resident_df,
                      invaders = invader_df, traits = trait_df)
fit <- learn_sensitivities(fit)

# New invader trait table (raw scale, same columns as residents' traits)
inv_traits <- data.frame(height = c(1.3, 0.9), SLA = c(12, 18),
                         life_form = c("shrub","forb"),
                         row.names = c("spA","spB"))

fit <- predict_invaders(fit, inv_traits)
str(fit$invaders, 1)
dim(fit$invaders$r_is_z)  # |sites| × |invaders|

## End(Not run)
```

---

prep_resident_glmm *Build residents-by-sites long table and fit the residents-only GLMM*

Description

Creates a long data set with one row per (site, resident) pair, attaches standardized environment and resident traits, then fits a GLMM for residents. Also returns fixed-effect predictions on the link scale as a site×resident matrix (r_js) and the response-scale mean (mu_js).

Usage

```
prep_resident_glmm(
  comm_res,
  env_df_z,
  traits_res_glmm,
  fml,
  family = glmmTMB::tweedie(link = "log"),
  seed = 123,
  fit_model = TRUE
)
```

Arguments

`comm_res`	Numeric matrix (sites × residents) of observed counts or abundance. Row names are site IDs; column names are resident species IDs.
`env_df_z`	Data frame of standardized site-level environment. Row names are site IDs and must match `rownames(comm_res)`.

`traits_res_glmm`
: Data frame of standardized resident traits used in the GLMM. Row names are resident IDs and must match `colnames(comm_res)`.

`fml`
: A model formula for `glmmTMB::glmmTMB` (e.g., from `build_model_formula()`).

`family`
: A `glmmTMB` family object. Default is `glmmTMB::tweedie(link="log")`.

`seed`
: Integer. Random seed.

`fit_model`
: logical; if FALSE, return dat_r/fml without fitting.

Value

list(dat_r, fit_r, grid_res, r_js, mu_js, sites, res_ids)

`prepare_inputs` *Prepare inputs (assemble and align core tables)*

Description

Thin wrapper around assemble_matrices that standardises and stores assembled inputs in a new_invasimapr_fit object for downstream modelling. Use this function to run the input-assembly step once and pass a single structured container through subsequent pipelines.

Usage

```
prepare_inputs(..., make_plots = FALSE)
```

Arguments

`...`
: Arguments passed on to `assemble_matrices`

`long_df` Optional long-format data frame with at least the columns `site`, `x`, `y`, `species`, and `count`.

`site_df` Optional data frame with columns `site`, `x`, and `y`, used when `long_df` is `NULL`.

`env_df` Optional site by environment numeric data frame or matrix. Row names must correspond to site identifiers.

`comm_res` Optional site by resident numeric matrix or data frame (wide), or a long-format table with columns `site`, `species`, and `count`.

`traits_res` Optional resident by trait data frame with row names corresponding to species identifiers.

`comm_long` Character string indicating how to interpret a separately supplied `comm_res`; one of `"auto"`, `"long"`, or `"wide"`.

`site_col,x_col,y_col,species_col,count_col` Column names used when building from `long_df`.

`env_cols` Character vector of environment column names to extract.

`env_prefix` Regular expression used to select environment columns.

`trait_cols` Character vector of trait column names to extract.

`trait_prefix` Regular expression used to select trait columns.

`drop_empty_sites` Logical. If `TRUE`, drop sites with zero total abundance.

`drop_empty_species` Logical. If `TRUE`, drop resident species with zero total abundance across all sites.

`return_diversity` Logical. If `TRUE`, compute and return site-level diversity summaries.

`make_plots` Logical; if `TRUE`, propagate `make_plots` to assemble_matrices so that diagnostic plots are produced during assembly. Defaults to `FALSE`.

Details

What this does

1. Calls assemble_matrices to build core site-by-species and site-by-trait matrices, perform consistency checks, and harmonise keys and factor levels.
2. Wraps the returned list into a lightweight S3 container of class `invasimapr_fit`, with a dedicated `inputs` slot and a small `meta` block containing basic counts used by later steps.

Object layout

```
invasimapr_fit
  inputs : list   (output from assemble_matrices)
  meta   : list   (n_sites, n_residents, n_traits)
```

Notes

- No mutation of the assembled inputs occurs here; the function only packages and annotates them.
- To inspect the assembled data, use `fit$inputs` on the returned object.

Value

An object of class `invasimapr_fit` with components:

inputs The full list returned by assemble_matrices.

meta A list with elements `n_sites`, `n_residents`, and `n_traits`.

See Also

assemble_matrices, new_invasimapr_fit

Examples

```
## Not run:
fit <- prepare_inputs(
  sites      = site_df,
  residents  = resident_df,
  invaders   = invader_df,
```

```
  traits     = trait_df,
  make_plots = TRUE
)

str(fit, 1)
str(fit$inputs, 1)
fit$meta

## End(Not run)
```

`prepare_trait_space` *Prepare trait space and resident crowding (no site-z)*

Description

Orchestrates the *traits → space → centrality/hull → crowding* pipeline for residents and invaders. Optionally standardises trait inputs, constructs a joint trait space, computes convex hull / centroid / centrality, and derives **raw** resident crowding C_{js} using a Gaussian kernel (no per-site standardisation here). Row-wise z-scoring is deferred to `model_residents()`. Plot objects are *always* created and returned; they are only displayed when `show_plots = TRUE`.

Usage

```
prepare_trait_space(
  fit,
  traits_inv,
  crowding_sigma = NULL,
  show_plots = TRUE,
  do_standardise = TRUE,
  row_z = FALSE
)
```

Arguments

`fit`	An `invasimapr_fit` produced by `prepare_inputs()`/`assemble_matrices()`. Must contain `fit$inputs$traits_res`, `fit$inputs$comm_res`, and (optionally) `fit$inputs$site_df`.
`traits_inv`	Data frame (or matrix) of **raw** invader traits (rows = invaders; columns aligned to resident trait columns).
`crowding_sigma`	Optional numeric bandwidth for the resident crowding kernel. If `NULL`, a default/optimised value is chosen inside `compute_resident_crowding()`.
`show_plots`	Logical. If `TRUE`, display plots as they are created. If `FALSE`, plots are still created/returned but not shown. Default `TRUE`.

`do_standardise`
: Logical. If `TRUE` and `standardise_model_inputs()` exists, standardise resident *and* invader trait inputs (environment handled later). Default `TRUE`.

`row_z`
: Logical. Whether to perform **row-wise** (site) z-scoring inside `compute_resident_crowding()`. Leave `FALSE` here to keep **raw** `C_js`; row-z is typically applied in `model_residents()`.

Details

Pipeline

1. *(Optional)* Standardise `traits_res/traits_inv` via `standardise_model_inputs()` when available; otherwise pass through raw traits.
2. Build a **joint trait space** with `compute_trait_space()` (returns Gower distances, ordination scores for residents `Q_res` and invaders `Q_inv`, a density surface, dendrograms, etc.). Plots are requested but may be hidden depending on `show_plots`.
3. Compute **centrality & hull** (resident convex hull, centroid, centrality scores) using `compute_centrality_hull()`.
4. Compute **resident crowding** with `compute_resident_crowding()`, returning kernel weights `K_res_res`, distances `D_res`, site kernels `W_site`, chosen `sigma_alpha`, and **raw** `C_js` (unless `row_z = TRUE` is requested).

Display control To ensure reproducible plot objects without cluttering the console, the function temporarily opens a null graphics device when `show_plots = FALSE`.

Assumptions & safeguards

- Requires resident community and trait tables in `fit$inputs`.
- If standardisation fails or is unavailable, the function proceeds with raw traits.
- Site-wise z-scoring is intentionally **not** applied here by default.

Value

The input `invasimapr_fit` with updated components:

`$traits` List with `gower`, `Q_res`, `Q_inv`, `hull`, `centroid`, `density`, stored plots (`plots_ts`, `plots_ch`), centrality table, and hull vertices.

`$crowding` List with `W_site`, `D_res`, `sigma_alpha`, `K_res_res`, and **raw** `C_js` (unless `row_z = TRUE`).

`$meta` Lightweight cache of `residents`, `sites`, `invaders`, and `n_invaders`.

See Also

- Trait space: `compute_trait_space()`
- Centrality & hull: `compute_centrality_hull()`
- Resident crowding: `compute_resident_crowding()`
- Standardisation: `standardise_model_inputs()`
- Downstream modelling: `model_residents()`

Examples

```
## Not run:
# Compute all plots but do not display them
fit2 <- prepare_trait_space(fit, traits_inv, show_plots = FALSE)

# Display plots during construction
fit3 <- prepare_trait_space(fit, traits_inv, show_plots = TRUE)

# Use a fixed kernel bandwidth and request row-z inside the crowding step
fit4 <- prepare_trait_space(fit, traits_inv, crowding_sigma = 0.35, row_z = TRUE)

## End(Not run)
```

---

simulate_invaders *Simulate hypothetical invader trait profiles from a resident trait pool*

Description

Generate n_{inv} rows of trait values for hypothetical invaders by resampling the empirical distribution of resident traits. By default, traits are sampled independently by column, creating novel trait combinations. Alternatively, entire rows can be resampled to preserve the covariance structure of the resident trait space.

Row names of the returned object are set to the invader identifiers.

Usage

```
simulate_invaders(
  resident_traits,
  n_inv = 10,
  species_col = "species",
  trait_cols = NULL,
  mode = c("columnwise", "rowwise"),
  numeric_method = c("bootstrap", "normal", "uniform"),
  keep_bounds = TRUE,
  inv_prefix = "inv",
  keep_species_column = TRUE,
  seed = NULL
)
```

Arguments

`resident_traits`
: A data frame containing either a species identifier column (specified by `species_col`) or species identifiers as row names, plus one or more trait columns.

`n_inv`	Integer; number of invaders to simulate (default `10`).
`species_col`	`NULL` or character; name of the species identifier column in `resident_traits`. If `NULL`, species identifiers are taken from row names.
`trait_cols`	`NULL` or character vector specifying which trait columns to use. Defaults to all columns except `species_col` when present.
`mode`	Character; either `"columnwise"` or `"rowwise"`.
`numeric_method`	Character; for numeric traits in columnwise mode, one of `"bootstrap"`, `"normal"`, or `"uniform"`.
`keep_bounds`	Logical; if `TRUE`, normal or uniform draws are constrained to the observed minimum and maximum values.
`inv_prefix`	Character; prefix used to construct invader identifiers.
`keep_species_column`	Logical; if `TRUE` and `species_col` is not `NULL`, retain the species identifier column after setting row names.
`seed`	`NULL` or integer; optional random seed for reproducibility.

Details

Species identifiers Species identifiers can be supplied via a dedicated column specified by `species_col`, or taken from the row names of `resident_traits` when `species_col = NULL`. Newly simulated invaders receive fresh, unique identifiers constructed from `inv_prefix`, which become the row names of the returned data frame. When `species_col` is not `NULL`, the same identifiers are also stored in that column unless `keep_species_column = FALSE`.

Trait selection If `trait_cols` is `NULL`, all columns except `species_col` (when present) are treated as traits. Otherwise, only the intersection of `trait_cols` and existing column names is used.

Sampling modes

- `mode = "columnwise"`: Each trait is generated independently. Numeric traits can be drawn by bootstrap sampling, from a normal distribution parameterised by the empirical mean and standard deviation, or from a uniform distribution on the observed range. Factor and character traits are sampled from observed values.
- `mode = "rowwise"`: Entire rows are resampled with replacement from the resident trait table, preserving joint structure across traits.

Identifier collisions If proposed invader identifiers collide with existing resident identifiers, they are made unique using make.unique.

Value

A data frame of simulated invader traits. Row names correspond to invader identifiers. If `species_col` is not `NULL` and `keep_species_column = TRUE`, the species identifier column contains the same identifiers.

See Also

sample, make.unique, rnorm, runif, sd

Examples

```
## Not run:
set.seed(1)
residents <- data.frame(
  species  = paste0("sp", 1:5),
  height   = c(10.2, 9.8, 11.1, 10.5, 9.9),
  SLA      = c(15.0, 15.2, 14.7, 15.5, 15.1),
  lifeform = factor(c("tree", "shrub", "shrub", "tree", "herb"))
)

inv <- simulate_invaders(
  residents,
  n_inv = 4,
  species_col = "species",
  mode = "columnwise",
  numeric_method = "bootstrap"
)
head(inv)

## End(Not run)
```

---

site_varying_alpha_beta_gamma

Site-varying alpha and gamma with trait-dependent beta

Description

Constructs site-by-invader crowding penalties and density-dependence scalars by combining trait-derived slopes with site-level random effects.

Site-level random slopes for C_z and r_z are added to the trait-dependent systems to produce site-varying crowding penalties α_{is} and density-dependence scalars Γ_{is}. Saturation effects β_i are trait-varying only; no site-varying β_{is} is constructed because S_z is site-only.

The function also exposes signed slopes and clamping diagnostics returned by `derive_sensitivities()`.

Usage

```
site_varying_alpha_beta_gamma(
  fit_coeffs,
  Q_inv,
  sites,
  inv_ids,
  lrt = TRUE,
  quiet = FALSE
)
```

Arguments

`fit_coeffs`	Fitted residents-only GLMM (e.g. `glmmTMB`) used to extract fixed effects and site-level random slopes.
`Q_inv`	Data frame of invader trait scores with columns `tr1` and `tr2`; row names must correspond to `inv_ids`.
`sites`	Character vector of site identifiers (rows of the output matrices).
`inv_ids`	Character vector of invader identifiers (columns of the output matrices).
`lrt`	Logical; passed to `derive_sensitivities()` to control likelihood-ratio testing for trait effects.
`quiet`	Logical; if `TRUE`, suppress warnings about missing or negligible site-level random effects.

Value

A list with the following elements:

- `alpha_is`: matrix of site-by-invader crowding penalties
- `Gamma_is`: matrix of site-by-invader density-dependence scalars
- `beta_i`, `beta_signed_i`: invader-level saturation effects
- `alpha_signed_i`: signed crowding slopes prior to clamping
- `theta_i`, `gamma_i`: invader-level abiotic sensitivities
- `slope_C_i`: trait-derived crowding slopes
- `delta_C_s`, `delta_r_s`: site-level random effects
- `notes`: character vector of diagnostics
- `df`: tidy long-format data frame

`standardise_by_site` *Standardize a site-by-species matrix by site (row-wise z) with optional robust SD*

Description

For each site (row), subtract the row-mean and divide by a row-scale. The scale can be classical SD or a robust max(MAD, SD) with a small floor.

Usage

```
standardise_by_site(X, robust = FALSE)
```

Arguments

`X`	Numeric matrix (sites × species).
`robust`	Logical. If TRUE, uses max(MAD, SD, 1e-8) per row; else classical SD with a safe 1 fallback.

Value

List with `z` (z-scored matrix), `mu` (row means), `sd` (row scales).

Examples

```
## Not run:
std = standardise_by_site(C_js, robust = TRUE)
C_js_z = std$z

## End(Not run)
```

---

standardise_model_inputs

Standardise model inputs (no leakage) for residents and invaders

---

Description

Column-wise z-scores **environment** and **resident trait** numerics, then scales **invader trait** numerics **using resident moments only** (to avoid information leakage). Invader factor/character columns are coerced to the resident levels; unseen levels become `NA`. Optionally drops invader-only columns so the resident/invader trait schemas match.

Usage

```
standardise_model_inputs(
  env_df = NULL,
  traits_res,
  traits_inv = NULL,
  drop_extra_invader_cols = FALSE,
  verbose = TRUE
)
```

Arguments

`env_df`	Optional `data.frame` (sites × environment). Numeric columns are z-scored; non-numeric are preserved.
`traits_res`	`data.frame` (residents × traits). Mixed types allowed; numeric columns are z-scored.
`traits_inv`	Optional `data.frame` (invaders × traits). Must contain at least the trait columns present in `traits_res`. Numeric columns are scaled using **resident** means/SDs; factors are coerced to resident levels.
`drop_extra_invader_cols`	Logical; if `TRUE`, invader-only columns are dropped (not used downstream). If `FALSE`, they are still dropped for alignment but flagged in `info$notes`.
`verbose`	Logical; print messages about what was standardised/coerced.

Details

What gets standardised and how

- **Environment (`env_df`)**: numeric columns are z-scored (mean 0, sd 1); non-numeric columns are kept as-is. Zero variance is guarded by setting sd=1.
- **Resident traits (`traits_res`)**: numeric columns are z-scored; mixed types allowed—non-numeric columns are kept.
- **Invader traits (`traits_inv`)**: numeric columns are scaled using the **resident** trait means/SDs only (never computed from invaders). Factor/ character columns are coerced to resident levels; unseen levels become `NA`. Extra invader columns are dropped (with a note).

Returned objects

- `env_df_z`: environment table with numeric columns standardised (or `NULL`)
- `traits_res_glmm`: resident traits with numeric columns standardised
- `traits_inv_glmm`: invader traits, scaled **like residents** + factor levels matched (or `NULL`)
- `moments`: resident/reference moments used for scaling (`env_*`, `trait_*`)
- `info$notes`: human-readable notes on coercions/drops

Where this is used in the workflow

- Called explicitly prior to GLMM fitting and when harmonising invaders, and implicitly by wrappers such as `prepare_trait_space()` (if available).

Value

A named `list` with components:

env_df_z Environment table with numeric columns z-scored (or `NULL`).

traits_res_glmm Resident trait table with numeric columns z-scored.

traits_inv_glmm Invader trait table scaled to resident moments and factor levels matched (or `NULL`).

moments `list(env_means, env_sds, trait_means_res, trait_sds_res)` used for scaling.

info `list(notes=character())` with human-readable notes.

Invariants and guards

- Column names and row names are preserved.
- Zero-variance numeric columns use `sd=1` so z-scores stay defined.
- Invader trait numerics are always scaled by **resident** moments (no leakage).
- Invader extra columns are dropped for alignment; missing required columns error.

See Also

prepare_inputs(), prepare_trait_space(), build_invader_predictors(), compute_trait_space(), compute_resident_crowding()

Examples

```
# Minimal reproducible example ---------------------------------------------
set.seed(1)
env_df = data.frame(elev = rnorm(5), temp = rnorm(5), zone = factor(sample(c("A","B"), 5, TRUE)))
rownames(env_df) = paste0("s", 1:5)

traits_res = data.frame(
  size = rlnorm(4), leaf = factor(c("broad","needle","broad","needle")),
  stringsAsFactors = FALSE
)
rownames(traits_res) = paste0("sp", 1:4)

traits_inv = data.frame(
  size = c(10, 1), leaf = factor(c("broad","unknown"))  # 'unknown' -> NA after coercion
)
rownames(traits_inv) = c("inv1","inv2")

std = standardise_model_inputs(env_df, traits_res, traits_inv, verbose = FALSE)
str(std, 1)
head(std$traits_res_glmm)
head(std$traits_inv_glmm)   # note: 'leaf' for 'unknown' becomes NA

# Workflow sketch ----------------------------------------------------------
# fit = prepare_inputs(long_df = longDF, ...)         # gives fit$inputs$env_df, $traits_res
# inv  = simulate_invaders(fit$inputs$traits_res, n_inv = 10)
# std  = standardise_model_inputs(fit$inputs$env_df, fit$inputs$traits_res, inv)
# std$traits_res_glmm; std$traits_inv_glmm              # pass to GLMM / trait-space steps
```

---

summarise_invasiveness_invasibility

Summaries of invasion fitness: species invasiveness, trait effects, and site invasibility

---

Description

Invasion fitness λ_{is} integrates structure in trait space (distances, overlaps, convex hull, cloud centroid) with abiotic suitability (alignment to environment), niche crowding (overlap with residents weighted by composition), and resident competition (site saturation). summarise_invasiveness_invasibility() collapses the site-by-invader surface into species-, trait-, and site-level summaries using either a probabilistic measure $P(\lambda > 0)$ or a hard rule $I\{\lambda > 0\}$.

Species invasiveness (per invader) summarises the breadth of establishment across sites:

$$V_i = |S|^{-1} \sum_s I\{\lambda_{is} > 0\}$$

or

$$\tilde{V}_i = |S|^{-1} \sum_s P(\lambda > 0 \mid i, s).$$

Site invasibility (per site) quantifies openness to newcomers:

$$V_s = |I|^{-1} \sum_i I\{\lambda_{is} > 0\}$$

or

$$\tilde{V}_s = |I|^{-1} \sum_i P(\lambda > 0 \mid i, s).$$

Trait invasiveness scores which invader traits explain variation in invasiveness, using standardized slopes for continuous traits and ANOVA R^2 for categorical traits.

Usage

```
summarise_invasiveness_invasibility(
  lambda_is = NULL,
  p_is = NULL,
  use_probabilistic = FALSE,
  prob_threshold = 0.5,
  site_df = NULL,
  traits_inv = NULL,
  comm_res = NULL,
  return_long = TRUE,
  make_plots = TRUE,
  label = NULL
)
```

Arguments

`lambda_is`	Matrix of invasion fitness with dimensions sites by invaders. Row and column names are required.
`p_is`	Optional matrix of establishment probabilities with the same dimensions as `lambda_is`.
`use_probabilistic`	Logical. If `TRUE`, summaries use `p_is`. If `FALSE`, summaries use the hard rule $I\{\lambda > 0\}$.
`prob_threshold`	Numeric between 0 and 1. If probabilistic summaries are used and a binary view is requested, values with `p_is >= prob_threshold` are treated as 1.
`site_df`	Optional data frame with columns `site`, `x`, and `y` for spatial mapping.

traits_inv	Optional data frame of invader traits; row names must match invader IDs. Numeric and factor traits are supported.
comm_res	Optional site-by-resident matrix used to compute resident richness per site.
return_long	Logical. If TRUE, returns a tidy long table of the site-by-invader surface.
make_plots	Logical. If TRUE, returns ggplot objects.
label	Optional character label added to plot titles and metadata.

Details

Summarize invasion fitness into species, trait, and site metrics (with plots)

Value

A list with the following components:

- species: data frame of invader-level invasiveness metrics
- site: data frame of site-level invasibility metrics
- trait_effects: data frame of trait effect sizes
- establish_long: tidy long-format table of the working surface
- plots: list of ggplot objects (or NULL)
- meta: list describing the summary mode and threshold

Examples

```
set.seed(42)
S = 8; I = 5
sites = paste0("s", 1:S)
inv   = paste0("i", 1:I)

# Fake fitness and probabilities
lambda_is = matrix(rnorm(S*I, sd=1), S, I, dimnames=list(sites, inv))
p_is      = pnorm(lambda_is)  # crude probit just for the example

# Minimal site coordinates for plotting (optional)
site_df = data.frame(site = sites,
                     x = rep(1:4, each=2)[1:S],
                     y = rep(1:2, times=4)[1:S])

# Minimal trait table for invaders (one numeric, one factor)
traits_inv = data.frame(trait_size = runif(I, 0, 1),
                        trait_type = factor(sample(c("A","B"), I, TRUE)),
                        row.names = inv)

# 1) Probabilistic summaries (use p_is)
outP = summarise_invasiveness_invasibility(
  lambda_is = lambda_is,
  p_is      = p_is,
  use_probabilistic = TRUE,
```

```
  site_df   = site_df,
  traits_inv = traits_inv,
  make_plots = FALSE
)
names(outP)
head(outP$species)

# 2) Hard rule summaries (use I{lambda>0})
outH = summarise_invasiveness_invasibility(
  lambda_is = lambda_is,
  use_probabilistic = FALSE,
  site_df = site_df,
  traits_inv = traits_inv,
  make_plots = FALSE
)
head(outH$site)
```

`summarise_results` *Summarise invasiveness and invasibility (tables, maps, rankings)*

Description

Thin wrapper around summarise_invasiveness_invasibility that extracts inputs from a new_invasimapr_fit container, forwards optional arguments, and optionally overlays a boundary layer on the site map. This function is intended as a reporting step after computing invasion fitness and/or establishment probabilities.

Usage

```
summarise_results(
  fit,
  boundary_sf = NULL,
  boundary_params = list(inherit.aes = FALSE, fill = NA, color = "black", size = 0.3),
  traits_inv = NULL,
  ...
)
```

Arguments

`fit`	An object produced by the invasimapr workflow, containing `fitness` and/or `prob` components created by upstream steps.
`boundary_sf`	Optional object of class `sf` providing a boundary layer to overlay on the site map.
`boundary_params`	Named list of aesthetics passed to `ggplot2::geom_sf()`. Defaults to `list(inherit.aes = FALSE, fill = NA, color = "black", size = 0.3)`.

`traits_inv`	Optional override of the invader trait table used for invader-ranked summaries. If `NULL`, traits are taken from the container.
...	Additional arguments forwarded to summarise_invasiveness_invasibility.

Details

Inputs used from the container

- `fit$fitness$lambda_is`: site-by-invader invasion fitness matrix (optional).
- `fit$prob$p_is`: site-by-invader establishment probability matrix (optional).
- `fit$inputs$site_df`: site metadata with coordinates `x` and `y` (optional).
- `fit$inputs$comm_res`: resident community matrix for context summaries (optional).
- invader trait tables stored in the container, used for invader rankings.

At least one of `lambda_is` or `p_is` must be present; otherwise an error is thrown.

Behaviour

1. Extract invasion fitness and/or establishment probability matrices from the container.
2. Select site metadata and an effective invader trait table when available.
3. Call summarise_invasiveness_invasibility to construct tidy summary tables and plots (site maps, rankings, heatmaps).
4. If a boundary layer is supplied and plots are available, overlay it on the site map using `geom_sf()`.

Output layout The returned container gains a `summary` element mirroring the structure returned by summarise_invasiveness_invasibility, typically containing summary tables, plots, and the arguments used.

Value

The input `fit` object with an added or updated `fit$summary` list. Invisibly returns `fit` to support piping.

See Also

compute_invasion_fitness, compute_establishment_probability, summarise_invasiveness_invasibility, assemble_matrices

Examples

```
## Not run:
fit <- prepare_inputs(
  sites = site_df,
  residents = resident_df,
  invaders = invader_df,
  traits = trait_df
)

fit <- learn_sensitivities(fit)
```

```
fit <- predict_invaders(fit, traits_inv = invader_traits)

fit <- summarise_results(fit)
fit$summary$plots$site_map

## End(Not run)
```

Index