Package: EnvRtype (via r-universe)

October 23, 2026

Type Package

Title Envirotyping for Quantitative Genetics and Plant Breeding

Version 0.1.3

Description An enviromics toolkit for genotype-by-environment analysis and envirotype-informed genomic prediction. Collects worldwide daily and hourly weather from 'NASA POWER', soil profiles from 'SoilGrids', elevation and bioclimatic layers from 'WorldClim', agro-ecological zones from 'FAO GAEZ', and downscaled 'CMIP6' climate scenarios, with checkpointed retrieval that can resume after an interrupted run. Derives agro-meteorological parameters (solar radiation, evapotranspiration, thermal time, atmospheric demand), runs a FAO-56 single crop coefficient soil water balance, and maps daily weather onto crop developmental stages. Builds environmental covariable matrices, envirotype typologies and frequency matrices, mega-environment clusters, historical risk profiles and target population of environments weights. Reduces interval-resolved envirotypes by principal component analysis, delineates soil zones with Gaussian mixture models, and constructs linear, Gaussian and arc-cosine environmental relatedness kernels. Fits Bayesian reaction-norm kernel models and predicts trained genotypes at untested locations and planting dates, with coverage diagnostics, spatial mapping and planting-window optimisation. Simulation functions generate multi-environment trials and enviromes with known architecture for power analysis. Methods are described in Costa-Neto, Galli, Carvalho, Crossa and Fritsche-Neto (2021) <doi:10.1093/g3journal/jkab040>, and build on the Bayesian genotype-by-environment kernel sampler of Granato, Cuevas, Luna-Vazquez, Crossa, Montesinos-Lopez, Burgueno and Fritsche-Neto (2018) <doi:10.1534/g3.118.200435>.

License GPL-3

URL https://github.com/gcostaneto/EnvRtype, https://gcostaneto.github.io/EnvRtype/

BugReports https://github.com/gcostaneto/EnvRtype/issues

Encoding UTF-8

LazyData true

Config/roxygen2/version 8.1.0

Imports ggplot2, grDevices, graphics, reshape2, stats, tools, utils

Suggests cluster, curl, geobr, ggrepel, httr, jsonlite, knitr, mclust, nasapower, parallel, randomForest, rmarkdown, sf, terra, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/pak/sysreqs libicu-dev

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

Date/Publication 2026-10-22 18:20:18 UTC

RemoteUrl https://github.com/gcostaneto/envrtype

RemoteRef HEAD

RemoteSha add5c59616e13876b9250723ffa435a322ac2fac

Contents

aez_default_url 4
aez_legend 5
as.data.frame.sim_met 6
as.matrix.sim_envirome 6
as.matrix.sim_W 7
best_planting_date 7
clay_5_15 9
cluster_environments 9
coverage_summary 12
decompose_kernels 13
env_cluster 14
env_copula 15
env_cor 18
env_cor_heatmap 19
env_expand 20
env_indices 21
env_kernel 24
env_loading_curve 26
env_pc_associate 28
env_pca 31
env_pca_biplot 33
env_pca_scree 34
env_phenology 35
env_risk_profile 38
env_target_importance 41
env_typing 43

G2F_2014_17 . . . 47
get_AEZ . . . 48
get_bioclim . . . 51
get_climate_scenario . . . 52
get_elevation . . . 55
get_kernel . . . 56
get_soil . . . 59
get_soil_resumable . . . 61
get_spatial . . . 62
get_weather . . . 64
get_weather_hourly . . . 66
get_weather_resumable . . . 68
grid_scan . . . 70
kernel_cv . . . 71
kernel_model . . . 72
kernel_model_clustered . . . 77
kernel_model_mc . . . 78
maizeG . . . 79
maizeWTH . . . 80
maizeYield . . . 80
map_scan . . . 81
param_atmospheric . . . 85
param_radiation . . . 86
param_temperature . . . 88
phenology_templates . . . 89
planting_window_table . . . 90
plot.sim_met . . . 91
plot.sim_W . . . 91
plot_planting_window . . . 92
print.kernel_set . . . 93
print.scan_untested_envs . . . 93
print.sim_envirome . . . 94
print.sim_markers . . . 95
print.sim_met . . . 95
print.sim_W . . . 96
print.soil_gmm . . . 96
print.T_matrix . . . 97
processWTH . . . 98
project_risk . . . 99
read_progress_log . . . 100
restart_from_log . . . 101
scan_spatial_table . . . 103
scan_untested_envs . . . 103
sim_envirome . . . 107
sim_markers . . . 109
sim_met . . . 111
sim_met_C . . . 117
sim_W . . . 118

sim_W_grid 121
soil_classification 123
summary_water_balance 128
summaryWTH 130
T_matrix 132
tpe_weights 134
truncate_gxe_kernel 136
undecompose_kernels 136
varcomp_summary 137
W_matrix 138
water_balance 141

Index **147**

`aez_default_url` *Default GAEZ Download URL*

Description

Returns the URL template `get_AEZ` uses when no local file is given and the bundled raster is unavailable.

Usage

```
aez_default_url(layer = "aez57")
```

Arguments

`layer`	character. Which GAEZ layer. Currently `"aez57"`.

Details

This URL is unverified. It was constructed from the documented GAEZ v4 portal structure but could not be tested against a live FAO server, and FAO has reorganised these paths between releases. It is exposed as a function, rather than hidden inside `get_AEZ()`, so it can be inspected and overridden without editing the package:

```
get_AEZ(env, lat, lon, url = "https://.../correct/path.tif")
```

For reproducible analyses, rely on the bundled raster or download the layer once and pass `file=`.

Value

A single character string.

See Also

`get_AEZ`

Examples

```
aez_default_url()
```

`aez_legend` *FAO GAEZ Agro-Ecological Zone Legend*

Description

Returns the class table used to decode GAEZ agro-ecological zone raster values into human-readable zone names.

Usage

```
aez_legend(classes = 57)
```

Arguments

`classes`	integer. Which legend to return. Currently only `57` (the GAEZ v4 57-class AEZ layer) is built in.

Details

Verify before relying on these labels. FAO has revised AEZ class definitions between GAEZ releases. The table here follows the GAEZ v4 documentation, but if your raster came from a different release the codes may not line up. `get_AEZ` will warn when a raster contains codes that are absent from the legend – that warning is the signal to check.

Value

A data.frame with `code`, `label`, `thermal`, `moisture` and `arable` (logical; `FALSE` for water, urban, barren and the constraint classes).

See Also

`get_AEZ`

Examples

```
leg <- aez_legend()
head(leg)
table(leg$thermal)
```

as.data.frame.sim_met *Coerce a Simulated MET to a data.frame*

Description

Returns the long phenotype table, so a `sim_met` object can be handed straight to modelling functions or inspected as a plain frame.

Usage

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

Arguments

`x`	a `sim_met` object.
`row.names, optional`	passed to the default method for consistency; unused.
`...`	ignored.

Value

A `data.frame` with one row per plot and columns `env`, `gid`, `rep` and `value`.

See Also

`sim_met`

as.matrix.sim_envirome *Coerce a Simulated Envirome Diversity Matrix to a Plain Matrix*

Description

Coerce a Simulated Envirome Diversity Matrix to a Plain Matrix

Usage

```
## S3 method for class 'sim_envirome'
as.matrix(x, ...)
```

Arguments

`x`	a `sim_envirome` object.
`...`	ignored.

Value

A numeric matrix with no `sim_envirome` class or attributes.

See Also

`sim_envirome`

`as.matrix.sim_W` *Coerce a Simulated Envirome to a Plain Matrix*

Description

Strips the `sim_W` class and its diagnostic attributes, returning the bare $q \times n_{var}$ covariable matrix for use with `env_kernel`, `W_matrix` consumers, or base matrix ops.

Usage

```
## S3 method for class 'sim_W'
as.matrix(x, ...)
```

Arguments

`x`	a `sim_W` object.
`...`	ignored.

Value

A numeric matrix (environments in rows, covariables in columns) with no `sim_W` class or attributes.

See Also

`sim_W`

`best_planting_date` *Best planting date per genotype*

Description

Best planting date per genotype

Usage

```
best_planting_date(
  scan,
  dates = NULL,
  date_col = "date",
  genotypes = NULL,
  group = NULL,
  exclude_flagged = TRUE,
  envelope_level = c("gross", "mild")
)
```

Arguments

`scan`, `dates`, `date_col`, `genotypes`, `group`
: as in [planting_window_table()].

`exclude_flagged`
: drop dates outside the envelope before ranking. Default TRUE: an optimum found in a region the training data does not cover is not an optimum you can act on.

`envelope_level`
: ‘"gross"‘ (default) or ‘"mild"‘.

Value

data.frame: ‘gid‘, ‘best_date‘, ‘yHat‘, ‘lower‘, ‘upper‘, ‘position‘, ‘n_dates_considered‘.

Examples

```
## A scan is normally produced by scan_untested_envs(); here we build a
## minimal one by hand -- 3 genotypes across 6 candidate sowing windows --
## so the example is self-contained.
gids <- paste0("G", 1:3)
envs <- paste0("sow", 1:6)
mu <- matrix(c(6, 5.5, 5), 3, 6) +
      matrix(rep(c(0, .4, .7, .6, .2, -.3), each = 3), 3, 6)  # peaks mid-window
dimnames(mu) <- list(gids, envs)
scan <- list(
  yHat_matrix = mu, yHat_lower = mu - 0.5, yHat_upper = mu + 0.5,
  interpolation = data.frame(
    env               = envs,
    position          = c("interpolation", "interpolation", "edge",
                          "interpolation", "mild_extrapolation",
                          "gross_extrapolation"),
    mahalanobis_ratio = c(0.6, 0.8, 1.4, 0.9, 2.1, 3.6),
    prop_outside      = c(0, 0, 0.1, 0, 0.3, 0.7),
    weight_negativity = 0, stringsAsFactors = FALSE))
class(scan) <- "scan_untested_envs"

## Map each scanned environment to a calendar sowing date
dates <- data.frame(env = envs,
                    date = as.Date("2024-04-01") + c(0, 14, 28, 42, 56, 70))
```

```
## Best *supported* sowing date per genotype (gross extrapolations dropped)
best_planting_date(scan, dates)

## Rank every date, including the out-of-envelope ones
best_planting_date(scan, dates, exclude_flagged = FALSE)

## Stricter: also drop mild extrapolations before ranking
best_planting_date(scan, dates, envelope_level = "mild")
```

---

clay_5_15 *Clay content (g/kg) from 5 to 15cm depth for Nairobi, Kenya*

Description

Soil profile for clay content (g/kg) around Nairobi, downloaded from SoilGrids (https://soilgrids.org/). Shipped as a GeoTIFF under `inst/extdata` and read with **terra**.

Format

A single-layer GeoTIFF raster read with `terra::rast()`.

Examples

```
f <- system.file("extdata", "clay_5_15.tif", package = "EnvRtype")
clay <- terra::rast(f)
terra::plot(clay)
```

---

cluster_environments *Group environments into mega-environments (clusters)*

Description

Delineates mega-environments by clustering the **genetic correlation** among environments, i.e. how similarly genotypes rank from one environment to another. Environments in which genotypes respond alike are placed in the same cluster, which is the operational definition of a mega-environment in a target population of environments (TPE).

The input is either a fitted `kernel_model` (carrying the environment genetic-correlation matrix in `$genetic_cor_env`, produced with `keep_effects = TRUE`) or any square environment correlation matrix you already have (for example from `env_cor` or an external GxE analysis).

Usage

```
cluster_environments(
  object,
  k = NULL,
  method = c("hclust", "pam", "kmeans"),
  distance = c("one_minus", "sqrt"),
  hclust_linkage = "ward.D2",
  k_range = 2:6,
  seed = NULL
)
```

Arguments

object	a kernel_model fit carrying $genetic_cor_env (fit with keep_effects = TRUE), or a square environment-by-environment correlation matrix.
k	integer or NULL. Number of clusters. When NULL (default) it is chosen by maximum average silhouette width over k_range.
method	character. Clustering algorithm: "hclust" (default), "pam" or "kmeans".
distance	character. Distance built from the genetic correlation: "one_minus" ($1 - r_g$, default) or "sqrt" ($\sqrt{1 - r_g}$).
hclust_linkage	character. Linkage passed to hclust when method = "hclust". Default "ward.D2".
k_range	integer vector. Candidate numbers of clusters tested when k is NULL. Default 2:6.
seed	integer or NULL. Optional seed for reproducibility of the randomised starts in "pam" and "kmeans".

Details

The genetic correlation r_g is turned into a distance and the chosen algorithm is run on that distance:

- distance = "one_minus" uses $d = 1 - r_g$ (default).
- distance = "sqrt" uses $d = \sqrt{1 - r_g}$, which spreads out highly correlated environments.

Correlations are clamped to $[-1, 1]$ and the distance matrix is symmetrised before clustering. Three algorithms are available through method: agglomerative hierarchical clustering ("hclust", controlled by hclust_linkage), partitioning around medoids ("pam"), and "kmeans" run on a classical MDS embedding of the distance matrix.

When k is NULL the number of clusters is selected automatically by maximising the **average silhouette width** over the candidate values in k_range; supplying k skips this search. At least three environments are required.

Value

A list with:

clusters named integer vector giving the cluster of each environment.

k the number of clusters used (selected or supplied).

method the clustering algorithm used.

distance the symmetric distance matrix that was clustered.

silhouette per-environment silhouette widths plus their `average` (higher is a tighter, better-separated solution).

medoids representative environment of each cluster (`"pam"`/`"kmeans"`; `NULL` for `"hclust"`).

hclust the `hclust` tree for `method = "hclust"`, otherwise `NULL`.

table a data.frame (`env`, `cluster`, `sil_width`) ordered by cluster and decreasing silhouette width.

See Also

`kernel_model` (fit with `keep_effects = TRUE` to obtain `$genetic_cor_env`), `kernel_model_clustered` to fit a model informed by the clusters, `env_cluster` for clustering from an environmental covariable matrix, and `env_cor`.

Examples

```
## ------------------------------------------------------------------
## 1. Cluster a genetic-correlation matrix directly (no model needed)
## ------------------------------------------------------------------
## Six environments forming two obvious groups
R <- matrix(0.2, 6, 6)
R[1:3, 1:3] <- 0.9
R[4:6, 4:6] <- 0.85
diag(R) <- 1
dimnames(R) <- list(paste0("E", 1:6), paste0("E", 1:6))

## Automatic number of clusters via silhouette width
cl <- cluster_environments(R)
cl$k
cl$table
cl$silhouette["average"]

## ------------------------------------------------------------------
## 2. Force a specific number of clusters
## ------------------------------------------------------------------
cluster_environments(R, k = 2)$clusters

## ------------------------------------------------------------------
## 3. Try different algorithms and distances
## ------------------------------------------------------------------
cluster_environments(R, k = 2, method = "pam", seed = 1)$medoids
cluster_environments(R, method = "kmeans", k_range = 2:4, seed = 1)$clusters
cluster_environments(R, distance = "sqrt",
                     hclust_linkage = "complete")$clusters


## ------------------------------------------------------------------
## 4. Full workflow from phenotypes, kernels and a fitted model
## ------------------------------------------------------------------
data("maizeYield"); data("maizeG"); data("maizeWTH")
```

```
ECs <- W_matrix(env.data = maizeWTH[maizeWTH$daysFromStart < 100, ],
                var.id = c("FRUE", "PETP", "SRAD", "T2M_MAX"))
K <- get_kernel(K_G = list(G = maizeG),
                K_E = list(W = env_kernel(env.data = ECs)[[2]]),
                data = maizeYield, model = "RNMM")

## keep_effects = TRUE is required for $genetic_cor_env
fit <- kernel_model(y = "value", env = "env", gid = "gid",
                    data = maizeYield, random = K,
                    keep_effects = TRUE)

megaenv <- cluster_environments(fit)
megaenv$table

## 5. Feed the clusters into a cluster-aware model
fit_cl <- kernel_model_clustered(y = "value", data = maizeYield,
                                 random = K, env = "env", gid = "gid",
                                 clusters = megaenv, use = "block")
```

`coverage_summary` *Summarise coverage of a scanned region*

Description

Tabulates how much of a scanned area falls in each envelope class, and which covariates most often drive sites outside. The second table answers "what should I go and measure".

Usage

```
coverage_summary(scan)
```

Arguments

`scan`	a ‘scan_untested_envs‘ / [grid_scan()] result.

Value

list of ‘by_position‘ and ‘by_covariate‘ data.frames.

decompose_kernels *Attach spectral decompositions to a list of dense kernels*

Description

Turns each dense kernel (`list(Kernel = <n x n>, Type = "D"|"BD")`) into a cached spectral decomposition stored on `$decomp`. Call once and reuse across any number of `kernel_model` fits: kernels depend only on genotypes and environments – never on the phenotype vector – so masking `y` for a cross-validation fold does NOT invalidate the decomposition.

Usage

```
decompose_kernels(
  K,
  ne = NULL,
  tol = 1e-10,
  scale_kernels = TRUE,
  keep_var = 1,
  verbose = TRUE
)
```

Arguments

`K`	list of kernels.
`ne`	per-environment observation counts; defaults to `attr(K, "ne")`.
`tol`	eigenvalue cutoff; must match `kernel_model(tol=)`.
`scale_kernels`	must match `kernel_model(scale_kernels=)`.
`keep_var`	numeric in (0, 1]. Retain only the leading eigenvalues carrying this proportion of each kernel's trace. 1 (default) keeps everything. A named vector or list keyed by kernel name / regex applies per kernel, e.g. `c(KGE_ = 0.95)`.
`verbose`	report rank, timing and route per kernel.

Details

Factored kernels from `get_kernel` arrive pre-decomposed and do not need this function.

Value

`K` with `$decomp` on each element and a `"kd_meta"` attribute.

See Also

`get_kernel`, `kernel_model`, `undecompose_kernels`

`env_cluster` *Envirotype clustering into mega-environments*

Description

Clusters environments into homogeneous groups ("mega-environments") from an environmental covariable matrix (`W`) using K-means. The number of groups can be set by the user or selected automatically by maximising the average silhouette width. An environmental relationship (kinship) matrix is also returned for convenience.

Usage

```
env_cluster(
  W,
  k = NULL,
  k.max = 10,
  scale = TRUE,
  nstart = 25,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`W`	matrix or data.frame. Environmental covariables (environments in rows, variables in columns), typically a `W_matrix` output. An envirotype frequency matrix from `T_matrix` works identically.
`k`	integer or NULL. Number of mega-environments. If `NULL` (default) it is chosen by silhouette.
`k.max`	integer. Maximum number of clusters tested when `k` is `NULL`. Default 10.
`scale`	boolean. If `TRUE` (default) `W` is centred and scaled before clustering.
`nstart`	integer. Number of random starts for K-means. Default 25.
`seed`	integer or NULL. Optional seed for reproducibility.
`verbose`	boolean. If `TRUE` (default) prints progress messages.

Details

Mega-environments are repeatable groups of locations that rank genotypes similarly. Here they are approximated by grouping environments that share a similar enviromic profile, which is useful to stratify a Target Population of Environments (TPE) before multi-environment prediction. The silhouette width $s(i)$ contrasts within- and between-cluster distances; the `k` maximising its average is retained.

Value

A list with `clusters` (named integer vector), `k`, `silhouette` (data.frame of average silhouette per tested `k`, or `NULL` when `k` was supplied), `kmeans` (the fitted object) and `env.kinship` (an environment x environment correlation matrix).

Author(s)

Germano Costa Neto

See Also

`W_matrix`, `env_kernel`, `env_target_importance`, `T_matrix`

Examples

```
data("maizeWTH")

W <- W_matrix(env.data = maizeWTH[maizeWTH$daysFromStart < 100, ],
              var.id = c("T2M", "T2M_MAX", "PRECTOT"), statistic = "mean")

## Automatic number of mega-environments (chosen by silhouette)
mega <- env_cluster(W, seed = 1)
mega$k
mega$clusters
mega$silhouette

## Force a fixed number of groups
mega2 <- env_cluster(W, k = 2, seed = 1)
mega2$clusters

## Environmental relationship (kinship) matrix
round(mega$env.kinship, 2)

## Use the grouping downstream (e.g. as a fixed effect or to stratify CV)
table(mega$clusters)
```

---

env_copula — *Copula-based Dependence Indices for an Environmental Covariable Matrix*

Description

Performs a copula analysis of an environment x covariable matrix (`W`) and returns a matrix of *copula indices* that can replace the raw covariables. Sklar's theorem lets the joint distribution of the environmental variables be split into (i) the marginal behaviour of each variable and (ii) the dependence structure linking them (the copula). By moving to the copula scale, each environment is described by where it sits in the *multivariate* environmental distribution rather than by raw physical units.

Usage

```
env_copula(
  W,
  index = c("pobs", "joint", "survival", "kendall", "all"),
  groups = NULL,
  ties.method = "average",
  normalize = TRUE,
  verbose = TRUE
)
```

Arguments

`W`	matrix or data.frame. Environmental covariables with environments in rows and variables in columns (typically a `W_matrix` output).
`index`	character. Which copula index to return: `"pobs"` Pseudo-observations $u_{ij} = r_{ij}/(q+1)$, the marginal probability-integral transform. Same dimension as `W`; one column per variable. `"joint"` Multivariate empirical copula $C_q(u_i)$, the joint non-exceedance probability of environment i: the probability that all variables are simultaneously at or below their observed levels. One column per block. `"survival"` Survival (joint exceedance) copula: the probability that all variables are simultaneously at or above their observed levels. Highlights compound-stress environments. One column per block. `"kendall"` Kendall function $K_C(z) = P(C(U) \leq z)$ evaluated at each environment's joint probability. A scalar multivariate return level ordering environments from jointly-benign to jointly-extreme. One column per block. `"all"` Column-binds `pobs`, `joint`, `survival`, `kendall`.
`groups`	list or NULL. Optional named list splitting the columns of `W` into variable blocks, e.g. `list(heat = c("T2M_mean","T2M_MAX_mean"), water = "PRECTOT_mean")`. Elements may be column names or column indices. A separate multivariate copula is fitted per block, so joint/survival/Kendall describe dependence *within* a block. If `NULL` (default) all columns form a single block named `"all"`.
`ties.method`	character. How ranks are computed when there are ties, passed to `rank`. Default `"average"`.
`normalize`	boolean. If `TRUE` (default) the Kendall index is rescaled to $[0, 1]$ by its empirical range when non-degenerate; `FALSE` keeps raw K_C.
`verbose`	boolean. If `TRUE` (default) prints progress messages.

Details

Why rank-based. All indices are computed from ranks, so they are invariant to any monotone transformation of the marginals: storing precipitation in mm or log(mm) gives the same result, and no distributional assumption is imposed. This is the main practical gain over a z-scored **W**, where one extreme site can dominate the Euclidean geometry.

Pseudo-observations. For q environments and variable j,

$$u_{ij} = r_{ij}/(q + 1)$$

with r_{ij} the rank of environment i. Dividing by $q+1$ keeps values strictly inside $(0,1)$, avoiding boundary problems in downstream density evaluation.

Empirical copula. The joint index is the empirical copula evaluated at each environment's own pseudo-observation, a nonparametric estimator of C that requires no copula family to be chosen.

Kendall function. K_C is the CDF of $Z = C(U)$, estimated by the Genest-Rivest empirical construction. It collapses a d-dimensional dependence structure into one interpretable number per environment and underpins multivariate return periods in hydrology.

Sample-size caveat. These are rank statistics *across environments*, so their resolution is bounded by q. With five environments each pseudo-observation can only take values in 1/6,...,5/6 and the joint copula is very coarse; the indices become informative with dozens of environments. A warning is issued when $q < 10$.

Scope. The dependence modelled is that *among environmental variables across the environment panel*. Geographic coordinates are not used and no spatial autocorrelation (variogram / distance decay) is fitted. Use `groups` to control which variables share a copula.

Value

A numeric matrix with q environments in rows (rownames preserved from `W`) and the requested copula indices in columns. Attributes `"copula.index"` and `"copula.groups"` record the index produced and the block structure used.

Author(s)

Germano Costa Neto

References

Sklar, A. (1959). Fonctions de repartition a n dimensions et leurs marges. *Publ. Inst. Statist. Univ. Paris* 8, 229-231.

Genest, C., Rivest, L.-P. (1993). Statistical inference procedures for bivariate Archimedean copulas. *JASA* 88, 1034-1043.

Salvadori, G., De Michele, C., Kottegoda, N.T., Rosso, R. (2007). *Extremes in Nature: An Approach Using Copulas*. Springer.

See Also

`W_matrix`, `env_kernel`, `env_cluster`

Examples

```
data("maizeWTH")
env.data <- maizeWTH[maizeWTH$daysFromStart < 100, ]

W <- W_matrix(env.data = env.data, var.id = c("T2M", "T2M_MAX", "PRECTOT"),
              statistic = "mean")

## 1. Marginal copula scale (pseudo-observations)
round(env_copula(W, index = "pobs"), 3)
```

```
## 2. Joint non-exceedance probability per environment
env_copula(W, index = "joint")

## 3. Compound-stress (joint exceedance) index
env_copula(W, index = "survival")

## 4. Kendall multivariate return level
env_copula(W, index = "kendall")

## 5. Blocks: heat and water handled as separate copulas
env_copula(W, index = "all",
           groups = list(heat  = c("T2M_mean", "T2M_MAX_mean"),
                         water = "PRECTOT_mean"))

## 6. Straight from W_matrix() via the 'copula' argument
Wc <- W_matrix(env.data = env.data, var.id = c("T2M", "T2M_MAX", "PRECTOT"),
               statistic = "mean", copula = "kendall",
               center = FALSE, scale = FALSE)

## 7. Copula-scale kernel for reaction-norm models
Kc <- env_kernel(env.data = env_copula(W, index = "pobs"), gaussian = TRUE)
round(Kc$envCov, 2)
```

---

env_cor *Extract the Correlation Among Environments*

Description

Accessory generic that pulls the environment correlation out of a sim_met object or a bare matrix, so a hand-off to sim_W never depends on internal structure.

Usage

```
env_cor(x, type = c("target", "realised"), ...)

## S3 method for class 'sim_met'
env_cor(x, type = c("target", "realised"), ...)

## S3 method for class 'matrix'
env_cor(x, type = c("target", "realised"), ...)

## S3 method for class 'sim_W'
env_cor(x, type = c("target", "realised"), ...)

## Default S3 method:
env_cor(x, type = c("target", "realised"), ...)
```

```
## S3 method for class 'sim_envirome'
env_cor(x, type = c("target", "realised"), ...)
```

Arguments

x	an object (sim_met, matrix, ...).
type	"target" (the specified correlation) or "realised" (what the simulation actually produced, on the K-whitened scale).
...	unused.

Value

A $q \times q$ correlation matrix.

See Also

sim_met, sim_W

Examples

```
met <- sim_met(maizeG, n_env = 6, seed = 1)
env_cor(met, "target")
env_cor(met, "realised")
```

---

env_cor_heatmap *Correlation Heatmap of Environmental Indices*

Description

Pearson correlation among all environmental indices, as in Fig. 1b of Della Coletta et al. (2023), which motivated the dimensionality reduction: the 867 indices were highly correlated.

Usage

```
env_cor_heatmap(
  W,
  order = c("factor", "interval", "none"),
  max.index = 400L,
  title = "Correlation among environmental indices",
  verbose = TRUE
)
```

Arguments

`W`	matrix. Environment x index matrix from `env_indices`.
`order`	character. Column ordering: `"factor"` (default, groups all intervals of a factor together, giving the block structure of Fig. 1b), `"interval"` (chronological across factors) or `"none"`.
`max.index`	integer. Guard against enormous plots; if the matrix has more columns than this, a regularly spaced subset is shown. Default 400.
`title`	character. Plot title.
`verbose`	boolean. If `TRUE` (default) prints a progress banner.

Details

The mean absolute off-diagonal correlation is printed as a one-number summary of how redundant the index set is – the quantity that justifies running a PCA at all.

Value

Invisibly, the correlation matrix that was plotted.

See Also

`env_indices`, `env_pca`

Examples

```
W <- env_indices(maizeWTH, end.day = 130)
R <- env_cor_heatmap(W)
```

---

env_expand — *Merge an environmental covariable matrix onto a phenotypic table*

Description

Expands an environment-level **W** matrix to the n observations of a phenotypic data.frame, so that each row of the output corresponds to one genotype-in-environment record.

Usage

```
env_expand(env.data, df.pheno, env.id = "env", skip = 3, verbose = TRUE)
```

Arguments

`env.data`	matrix. Environment x covariable matrix with environments as rownames.
`df.pheno`	data.frame. Phenotypic records containing the environment column.
`env.id`	character. Name of the environment column in `df.pheno`.
`skip`	integer. Number of leading columns of the merged table to drop (the phenotypic id columns). Default 3.
`verbose`	boolean. If `TRUE` (default) prints a progress banner.

Value

A numeric matrix of n rows (observations) by k covariables.

See Also

`W_matrix`, `env_kernel`

Examples

```
data("maizeYield"); data("maizeWTH")
W <- W_matrix(env.data = maizeWTH[maizeWTH$daysFromStart < 100, ],
              var.id = c("T2M", "PRECTOT"), statistic = "mean")
Wn <- env_expand(env.data = W, df.pheno = maizeYield, env.id = "env")
dim(Wn)
```

`env_indices` *Build Interval-Resolved Environmental Indices*

Description

Converts daily environmental data into the environment x index matrix used by Della Coletta et al. (2023): each weather factor is summarised within fixed-width time intervals from planting, producing one column per *factor x interval* combination.

Usage

```
env_indices(
  env.data,
  env.id = "env",
  day.id = "daysFromStart",
  var.id = NULL,
  interval = 3L,
  end.day = 151L,
  statistic = c("mean", "sum", "min", "max", "median"),
  statistic.by = NULL,
```

```
  id.names = NULL,
  sep = "_",
  require.complete = TRUE,
  verbose = TRUE
)
```

Arguments

`env.data`	data.frame of daily weather, e.g. a `get_weather` or `processWTH` output.
`env.id`	character. Column identifying the environment. Default `"env"`.
`day.id`	character. Column giving days from planting. Default `"daysFromStart"`.
`var.id`	character vector. Which weather factors to use. `NULL` (default) uses every numeric column that is not an id column.
`interval`	integer. Width of each time window in days. Default 3, as in the paper.
`end.day`	integer. Last day retained. Default 151, as in the paper. Days beyond this are discarded so that every environment contributes the same intervals.
`statistic`	character. Within-interval summary: `"mean"` (default), `"sum"`, `"min"`, `"max"` or `"median"`. Precipitation is arguably better summed than averaged – see `statistic.by`.
`statistic.by`	named character vector. Per-factor override of `statistic`, e.g. `c(PRECTOT = "sum")`. Names are factor names, values are statistics.
`id.names`	character vector. Additional columns to treat as identifiers (never summarised). Default `NULL`.
`sep`	character. Separator between factor and interval in the output column names. Default `"_"`.
`require.complete`	boolean. If `TRUE` (default) an error is raised when any environment x interval x factor cell is missing, because an incomplete matrix silently biases the PCA. Set `FALSE` to allow `NA`s through.
`verbose`	boolean. Print a summary. Default `TRUE`.

Details

Why a fixed end day. Environments differ in season length. Without a common cut-off, environments contribute different numbers of intervals and the matrix is ragged. The paper fixed the last interval at 151 days after planting; the same device is used here via `end.day`.

Interval labelling. With `interval = 3` and `end.day = 151`, days 1-3 form interval 1, days 4-6 interval 2, and so on; the final partial window (day 151) forms interval 51. This reproduces the 51 intervals of the paper, and with 17 factors gives the 867 indices reported.

Mean vs sum. The paper does not state the within-interval statistic. A mean is used by default because it is defined for every factor, but for accumulating quantities such as precipitation a sum is more interpretable; `statistic.by` allows a per-factor choice.

Value

A numeric matrix with environments in rows and $n_{factor} \times n_{interval}$ indices in columns. Attributes:

`"factors"` the factor names used

`"intervals"` the interval labels

`"interval.mid"` named numeric vector of interval mid-point days

`"interval.range"` data.frame of interval start/end days

`"n.missing"` number of missing cells

Author(s)

Implementation following Della Coletta et al. (2023)

References

Della Coletta R., Liese S.E., Fernandes S.B., Mikel M.A., Bohn M.O., Lipka A.E., Hirsch C.N. (2023). Linking genetic and environmental factors through marker effect networks to understand trait plasticity. *GENETICS* 224(4), iyad103.

See Also

`env_pca`, `env_cor_heatmap`

Examples

```
data("maizeWTH")

## 3-day intervals across the shared season. The bundled sample does not
## reach the paper's 151 days in every environment, so end.day is trimmed to
## a fully observed window (the default 151 would leave gaps and error out).
W <- env_indices(maizeWTH, env.id = "env", day.id = "daysFromStart",
                 end.day = 130)
dim(W)
attr(W, "intervals")[1:5]

## Sum precipitation within each window, average everything else
W2 <- env_indices(maizeWTH, statistic = "mean",
                  statistic.by = c(PRECTOT = "sum"), end.day = 130)

## Weekly windows instead
W3 <- env_indices(maizeWTH, interval = 7, end.day = 140)
```

env_kernel *Easily Building of Environmental Relatedness Kernels*

Description

Returns environmental kinships for reaction-norm models. The output is a list containing `varCov`, the relatedness among environmental covariables, and `envCov`, the relatedness among environments. Linear (GBLUP-like), nonlinear Gaussian and arc-cosine deep kernels are supported.

Usage

```
env_kernel(
  env.data,
  Y = NULL,
  is.scaled = TRUE,
  sd.tol = 10,
  digits = 5,
  tol = 0.001,
  merge = FALSE,
  Z_E = NULL,
  stages = NULL,
  env.id = "env",
  gaussian = FALSE,
  h.gaussian = NULL,
  deep.kernel = FALSE,
  deep.layers = 1,
  QC = FALSE,
  verbose = TRUE
)
```

Arguments

`env.data`	matrix. Environmental variables (or markers) per environment (or per genotype-environment combination), typically a `W_matrix` output. An envirotype frequency matrix from `T_matrix` is accepted identically (rows = environments); pass `is.scaled = TRUE` since `T_matrix` rows are already normalised compositions.
`Y`	data.frame. Phenotypic data set containing environment id, genotype id and trait value. Only used when `merge = TRUE`.
`is.scaled`	boolean. If the environmental data are already mean-centred and scaled (default `TRUE`), assuming $x \sim N(0, 1)$.
`sd.tol`	numeric. Maximum standard deviation tolerated in quality control. Columns above this value are eliminated. Default 10.
`digits`	numeric. Number of digits used for rounding. Default 5.
`tol`	numeric. Numerical tolerance. Default 1E-3.

`merge`	boolean. If `TRUE`, the environmental covariables are merged with `Y` to build an $n \times n$ kernel.
`Z_E`	matrix. Model matrix for environments, used when `merge = TRUE`. If `NULL` it is built from `Y`.
`stages`	vector (character). Names of each stage or time interval. If not `NULL`, one kernel is produced per development stage.
`env.id`	character. Identification of the environment column. Default `'env'`.
`gaussian`	boolean. If `TRUE`, uses the Gaussian kernel parametrisation $envCov = exp(-h \cdot d/q)$.
`h.gaussian`	numeric. Bandwidth h used when `gaussian = TRUE`.
`deep.kernel`	boolean. If `TRUE`, an arc-cosine deep kernel is built from the environmental data instead of the linear/Gaussian kernel. Default `FALSE`.
`deep.layers`	integer. Number of hidden layers when `deep.kernel = TRUE` (>= 1).
`QC`	boolean. If `TRUE`, applies quality control based on `sd.tol` regardless of `is.scaled`. Default `FALSE`.
`verbose`	boolean. If `TRUE` (default) prints quality-control messages.

Details

Three kernel methods are available.

Linear (GB). $K = WW'/\mathrm{trace}(WW')/n$, the environmental analogue of the GBLUP kernel of VanRaden (2008).

Gaussian (GK). $K = \exp(-h\, d/q)$, with d the squared Euclidean distance and q its median, so the bandwidth is scale-free.

Arc-cosine deep (DK). Emulates a deep neural network with `deep.layers` hidden layers (Cuevas et al. 2019); no bandwidth parameter is required.

Note that quality control is now controlled by the explicit `QC` argument. In earlier versions it ran only when `is.scaled = FALSE`, so with the default `is.scaled = TRUE` no filtering happened despite `sd.tol` being documented as active.

Value

A list with two elements: `varCov`, the covariable x covariable relatedness, and `envCov`, the environment x environment relatedness. When `stages` is supplied, each element is itself a named list of kernels, one per stage.

Author(s)

Germano Costa Neto

References

Cuevas J. et al. (2019). Deep kernel for genomic and near infrared predictions in multi-environment breeding trials. *G3* 9(9), 2913-2924.

Costa-Neto G., Fritsche-Neto R., Crossa J. (2021). Nonlinear kernels, dominance, and envirotyping data increase the accuracy of genome-based prediction in multi-environment trials. *Heredity* 126, 92-106.

See Also

W_matrix, get_kernel, env_cluster, T_matrix

Examples

```
data('maizeYield'); data("maizeWTH")

## Environmental covariable matrix (environments x covariables)
W.cov <- W_matrix(env.data = maizeWTH[maizeWTH$daysFromStart < 100, ],
                  var.id = c("T2M", "T2M_MAX", "PRECTOT"), statistic = "mean")

## 1. Linear (co)variance kernel
K.lin <- env_kernel(env.data = W.cov, gaussian = FALSE)
round(K.lin$envCov, 2)

## 2. Nonlinear Gaussian kernel
K.gau <- env_kernel(env.data = W.cov, gaussian = TRUE)
round(K.gau$envCov, 2)

## 3. Arc-cosine deep kernel with 2 hidden layers
K.deep <- env_kernel(env.data = W.cov, deep.kernel = TRUE, deep.layers = 2)
round(K.deep$envCov, 2)

## 4. One kernel per development stage
stages   <- c('VE', 'V1_V6', 'V6_VT', 'VT_R1')
interval <- c(0, 7, 30, 65, 90)
W.stage  <- W_matrix(env.data = maizeWTH, var.id = c('FRUE', 'PETP'),
                     by.interval = TRUE, time.window = interval,
                     names.window = stages)
K.stage <- env_kernel(env.data = W.stage, stages = stages, gaussian = TRUE)
names(K.stage$envCov)

## 5. Feeding a reaction-norm model
KE <- list(W = env_kernel(env.data = W.cov)$envCov)
```

---

env_loading_curve *Seasonal Loading Curves of an Environmental PC*

Description

Reproduces Fig. 6c of Della Coletta et al. (2023): for a chosen PC, the loading of every index is plotted against its position in the growing season, with one loess curve per weather factor. This answers "which weather factors, at which point in the season, drive this PC".

Usage

```
env_loading_curve(
  x,
  pc = 1,
  factors = NULL,
  span = 0.4,
  points = TRUE,
  facet = FALSE,
  title = NULL,
  verbose = TRUE
)
```

Arguments

x	an env_pca object.
pc	integer. Which PC to display. Default 1.
factors	character vector. Restrict to these weather factors. NULL (default) shows all.
span	numeric. Loess span. Default 0.4; larger is smoother.
points	boolean. Draw the raw per-interval loadings under the curves. Default TRUE.
facet	boolean. One panel per factor instead of overlaying. Default FALSE.
title	character. Plot title. NULL builds one automatically.
verbose	boolean. If TRUE (default) prints a progress banner.

Details

Interpretation. A factor whose curve departs strongly from zero over a stretch of the season contributes heavily to that PC during that window. Sign is arbitrary up to the usual PCA reflection: only relative signs within a PC are meaningful.

Loess caveat. The smoother is cosmetic. With 51 intervals per factor the curve is well determined, but with few intervals it can imply structure that the underlying points do not support – keep points = TRUE to see the evidence.

Value

Invisibly, a data.frame of the plotted loadings (index, factor, day, loading), so the curves can be re-plotted or tabulated. When **ggplot2** is available the ggplot object is attached as the "plot" attribute.

See Also

env_pca, env_pc_associate

Examples

```
pc <- env_pca(env_indices(maizeWTH, end.day = 130))

## All factors on one panel
env_loading_curve(pc, pc = 1)

## Temperature factors only, one panel each
env_loading_curve(pc, pc = 2,
                  factors = c("T2M", "T2M_MAX", "T2M_MIN"), facet = TRUE)

## Recover the plotted values
L <- env_loading_curve(pc, pc = 1)
head(L[order(-abs(L$loading)), ])
```

`env_pc_associate` *Correlate Environmental PCs with a Per-Environment Variable*

Description

Pearson correlation between each environmental PC and an external per-environment quantity – in the paper, module eigenmarkers; equally, a trait mean, heritability or predictive ability per environment. P-values are FDR-adjusted across PCs, as in the paper.

Usage

```
env_pc_associate(
  x,
  y,
  n.pc = NULL,
  method = "pearson",
  p.adjust = "BH",
  verbose = TRUE
)
```

Arguments

`x`	an `env_pca` object.
`y`	numeric vector or matrix. One value (or column of values) per environment. Names, or rownames, are matched against the environments in `x`.
`n.pc`	integer. How many PCs to test. Default `min(9, available)`, 9 being the paper's choice.
`method`	character. Correlation method passed to `cor.test`. Default `"pearson"`.
`p.adjust`	character. Multiple-testing correction. Default `"BH"` (Benjamini-Hochberg), as in the paper.
`verbose`	boolean. Report the strongest association. Default `TRUE`.

Details

Power warning. With q environments each correlation has $q-2$ degrees of freedom. At the paper's $q = 9$ even $r = 0.86$ gives an unadjusted $p \approx 0.003$, and a handful of PCs tested against a handful of modules exhausts the evidence quickly. Treat these associations as hypothesis- generating, not confirmatory, and prefer many environments.

FDR does not cover module selection. The adjustment applied here accounts for testing several PCs. It does *not* account for the eigenmarker being PC1 of a module that was itself chosen by clustering the same effect matrix. That selection inflates correlations and is invisible to any p-value computed from the observed data. See the worked example below for a permutation check, and for a demonstration that the permutation check is itself insufficient.

Value

A data.frame with `column`, `PC`, `r`, `p`, `p.adj` and `n`, sorted by adjusted p-value.

Examples

```
pc <- env_pca(env_indices(maizeWTH, end.day = 130))

## Trait mean per environment
ybar <- tapply(maizeYield$value, maizeYield$env, mean)
env_pc_associate(pc, ybar)



## ---------------------------------------------------------------------
## WORKED EXAMPLE: marker effect networks -> eigenmarkers -> environmental PCs
##
## The full analysis chain of Della Coletta et al. (2023), on simulated data
## where the answer is known in advance so the method can be checked.
##
## Planted truth:
##   module A markers respond to mid-season TEMPERATURE
##   module B markers respond to late-season PRECIPITATION
##   module C markers have real but environment-INDEPENDENT effects
## ----------------------------------------------------------------------

## --- 0. Simulate markers, weather and phenotypes ---------------------
set.seed(2026)
n.gid <- 160
n.marker <- 90
n.env <- 20
envs <- paste0("E", seq_len(n.env))
drivers <- scale(matrix(rnorm(n.env * 2), n.env, 2))
rownames(drivers) <- envs
M <- scale(matrix(rbinom(n.gid * n.marker, 2, 0.5), n.gid, n.marker))
dimnames(M) <- list(paste0("G", seq_len(n.gid)),
                    paste0("m", seq_len(n.marker)))
beta <- matrix(0.2, n.marker, n.env)
beta[1:30, ] <- rep(0.2 + 0.3 * drivers[, 1], each = 30)
beta[31:60, ] <- rep(0.2 + 0.3 * drivers[, 2], each = 30)
```

```
signal <- M %*% beta
Y <- signal + matrix(rnorm(length(signal), sd = 0.3), n.gid, n.env)
dimnames(Y) <- list(rownames(M), envs)

wth <- expand.grid(daysFromStart = 1:151, env = envs)
ie <- match(wth$env, envs)
day <- wth$daysFromStart
wth$T2M <- 22 + 4 * sin(pi * day / 151) +
  3 * drivers[ie, 1] * (day >= 50 & day <= 100) +
  rnorm(nrow(wth), sd = 0.5)
wth$PRECTOT <- exp(1 + 0.6 * drivers[ie, 2] * (day >= 100) +
                     rnorm(nrow(wth), sd = 0.1))

## --- 1. Marker effects per environment (RR-BLUP) ----------------------
## Solve via the n x n system: with p markers >> n genotypes this is far
## cheaper than the p x p form and gives the same ridge solution.
rrblup_effects <- function(y, Z, h2 = 0.5) {
  yc <- as.numeric(y) - mean(y)
  lambda <- ncol(Z) * (1 - h2) / h2
  a <- solve(tcrossprod(Z) + lambda * diag(nrow(Z)), yc)
  as.numeric(crossprod(Z, a))
}
beta.hat <- sapply(colnames(Y), function(e) rrblup_effects(Y[, e], M, h2 = 0.55))
dimnames(beta.hat) <- list(colnames(M), colnames(Y))

## --- 2. Marker effect network -> modules ------------------------------
## Markers whose effects do not VARY across environments cannot covary with
## anything; screening them out first avoids NA correlations and spurious
## modules. The paper used WGCNA; this is the same idea in base R.
eff.sd <- apply(beta.hat, 1, sd)
Bk     <- beta.hat[eff.sd > quantile(eff.sd, 0.35), , drop = FALSE]
adj    <- abs(cor(t(Bk)))^6                 # soft-thresholding power
mods   <- cutree(hclust(as.dist(1 - adj), method = "average"), h = 0.92)
mods   <- mods[mods %in% as.integer(names(table(mods))[table(mods) >= 15])]

## --- 3. Eigenmarkers --------------------------------------------------
## PC1 of a module's effect profiles, the analogue of a WGCNA module
## eigengene. Sign is arbitrary in PCA, so orient it deterministically --
## otherwise a module can flip between runs and reverse every downstream
## correlation.
eigenmarker <- function(B.mod) {
  X  <- scale(t(B.mod))
  em <- prcomp(X, center = FALSE, scale. = FALSE)$x[, 1]
  if (mean(cor(em, X)) < 0) em <- -em
  em
}
ids <- sort(unique(mods))
EM  <- sapply(ids, function(m) eigenmarker(Bk[names(mods)[mods == m], , drop = FALSE]))
dimnames(EM) <- list(colnames(beta.hat), paste0("ME", ids))

## --- 4. Environmental PCA and association ------------------------------
W  <- env_indices(wth, interval = 3, end.day = 151,
                  statistic.by = c(PRECTOT = "sum"))
```

```
pc <- env_pca(W)
assoc <- env_pc_associate(pc, EM, n.pc = 9, p.adjust = "BH")
head(assoc)

## --- 5. Name the drivers of an associated PC (paper Fig. 6c) ----------
k <- as.integer(sub("PC", "", assoc$PC[1]))
L <- data.frame(factor = pc$meta$factor, day = pc$meta$day,
                loading = pc$loadings[, k])
sort(tapply(abs(L$loading), L$factor, mean), decreasing = TRUE)
env_loading_curve(pc, pc = k)

## --- 6. Permutation null, and why it is NOT enough --------------------
## Permuting environment labels of the eigenmarker breaks the
## environment<->module link while preserving both structures.
perm.max <- replicate(500, {
  EMp <- EM[sample(nrow(EM)), , drop = FALSE]
  rownames(EMp) <- rownames(EM)
  max(abs(env_pc_associate(pc, EMp, n.pc = 9, verbose = FALSE)$r))
})
quantile(perm.max, 0.95)   # compare observed |r| with the permutation null

## A spurious module can pass both FDR adjustment and this permutation test.
## The permutation preserves the module definitions, so it cannot detect a
## module that was spuriously assembled in the first place. Guard against
## this with module stability across resampled genotypes, or by requiring
## an independent set of environments -- not with permutation alone.
```

---

env_pca *PCA of Interval-Resolved Environmental Indices*

Description

Runs prcomp on the environment x index matrix, as in Della Coletta et al. (2023), and returns scores, loadings and variance explained in a form the plotting functions in this module can consume.

Usage

```
env_pca(
  W,
  center = TRUE,
  scale. = TRUE,
  rank. = NULL,
  drop.constant = TRUE,
  verbose = TRUE
)
```

Arguments

W	matrix. Environment x index matrix, typically from env_indices.
center	boolean. Centre the indices. Default TRUE.
scale.	boolean. Scale the indices to unit variance. Default TRUE; see Details for why this differs from a bare prcomp call.
rank.	integer or NULL. Maximum number of PCs to compute. NULL (default) computes all available, which is min(nrow - 1, ncol).
drop.constant	boolean. If TRUE (default) indices with zero variance are removed before the PCA rather than producing NaN under scaling.
verbose	boolean. Print a summary. Default TRUE.

Details

Scaling. The paper reports only that prcomp was used. Because the 17 factors are on incomparable scales – radiation in MJ/m2/day, precipitation in mm, FRUE unitless in [0,1] – an unscaled PCA is dominated by the highest-variance factor and the leading PCs largely describe that one variable. scale. = TRUE is therefore the default here. Set scale. = FALSE for a covariance-matrix PCA.

Rank. With q environments and $k \gg q$ indices, at most $q - 1$ PCs carry variance. With the 9 environments of the paper that is 8, so the reported "ninth PC" explaining $< 0.01\%$ is effectively numerical noise. Expect very few informative PCs unless you have many environments.

Value

A list of class "env_pca":

scores environment x PC matrix (prcomp$x)

loadings index x PC matrix (prcomp$rotation)

variance data.frame with PC, sd, prop, cum

prcomp the underlying prcomp object

dropped indices removed as constant

meta data.frame mapping each index to its factor, interval and day

See Also

env_indices, env_pca_biplot, env_loading_curve

Examples

```
data("maizeWTH")
W  <- env_indices(maizeWTH, end.day = 130)
pc <- env_pca(W)
head(pc$variance)
env_pca_scree(pc)
env_pca_biplot(pc)
```

`env_pca_biplot` *Environments in Principal Component Space*

Description

Scatter of environments on two PCs, reproducing Fig. 1c of Della Coletta et al. (2023), where each point is one growth environment labelled by its code.

Usage

```
env_pca_biplot(
  x,
  pc = c(1, 2),
  label = TRUE,
  group = NULL,
  title = "Environments in PC space",
  verbose = TRUE
)
```

Arguments

`x`	an `env_pca` object.
`pc`	integer vector of length 2. Which PCs to plot. Default `c(1, 2)`.
`label`	boolean. Draw environment names. Default `TRUE`.
`group`	factor or named vector. Optional grouping used to colour points, e.g. a mega-environment assignment. Names are matched to environment names.
`title`	character. Plot title.
`verbose`	boolean. If `TRUE` (default) prints a progress banner.

Value

A ggplot object when **ggplot2** is available (invisibly), otherwise `NULL` after drawing a base plot.

See Also

`env_pca`, `env_pca_scree`

Examples

```
pc <- env_pca(env_indices(maizeWTH, end.day = 130))
env_pca_biplot(pc)
env_pca_biplot(pc, pc = c(1, 3))
```

env_pca_scree — *Scree Plot of Environmental PCA*

Description

Plots the proportion of variance explained per PC, the visualisation the paper used to decide how many PCs to retain.

Usage

```
env_pca_scree(
  x,
  n.pc = NULL,
  cumulative = TRUE,
  title = "Variance explained by environmental PCs",
  verbose = TRUE
)
```

Arguments

x	an env_pca object.
n.pc	integer. How many PCs to show. Default min(15, available).
cumulative	boolean. Overlay the cumulative proportion. Default TRUE.
title	character. Plot title.
verbose	boolean. If TRUE (default) prints a progress banner.

Value

A ggplot object when **ggplot2** is available (invisibly), otherwise NULL after drawing a base plot.

See Also

env_pca

Examples

```
pc <- env_pca(env_indices(maizeWTH, end.day = 130))
env_pca_scree(pc)
```

env_phenology *Thermal-Time Phenology: Map Daily Weather onto Developmental Stages*

Description

Accumulates daily thermal time from planting and cuts each environment's season at crop-specific developmental thresholds, so that every downstream interval-based routine compares LIKE DEVELOPMENTAL STATES across sites rather than like calendar dates.

The returned `stage` column is a drop-in replacement for the `interval` produced by `time.window` in `W_matrix()`, `summaryWTH()` and `env_typing()`.

Usage

```
env_phenology(
  env.data,
  env.id = "env",
  day.id = "daysFromStart",
  Tmax = "T2M_MAX",
  Tmin = "T2M_MIN",
  crop = NULL,
  stages = NULL,
  Tbase = NULL,
  Tupper = NULL,
  method = c("gdd", "baskerville", "trapezoid"),
  Tmaxdev = 45,
  photoperiod = FALSE,
  lat = "LAT",
  daylength.id = NULL,
  p.opt = NULL,
  p.sens = 0.02,
  day.type = NULL,
  planting.id = NULL,
  merge = TRUE,
  verbose = TRUE
)
```

Arguments

`env.data`	data.frame. Daily weather, typically a `get_weather()` or `processWTH()` output.
`env.id`	character. Environment-id column. Default `"env"`.
`day.id`	character. Day-counter column. Default `"daysFromStart"`.
`Tmax`, `Tmin`	character. Daily maximum/minimum temperature columns. Defaults `"T2M_MAX"` / `"T2M_MIN"`.

crop	character. Name of a built-in template, e.g. "maize", "wheat", "soybean", "rice", "cotton", "canola". Supplies default stages, Tbase, Tupper, p.opt and day.type. Any explicitly supplied argument wins. Call show_phenology() for the full list of the 20 templates and their provenance.
stages	named numeric. Cumulative thermal time (C d) at the END of each stage, strictly increasing, e.g. c(VE = 125, V6 = 475, VT = 1135, R6 = 2700). Overrides crop.
Tbase, Tupper	numeric. Cardinal temperatures for thermal time. Override the crop template when supplied.
method	character. Thermal-time method: "gdd" (default, McMaster & Wilhelm with upper cutoff), "baskerville" (Baskerville-Emin sine integration), or "trapezoid" (linear supra-optimal decline to Tmaxdev).
Tmaxdev	numeric. Absolute developmental maximum, used by method = "trapezoid". Default 45.
photoperiod	boolean. If TRUE, multiply daily thermal time by a photoperiod factor, producing photothermal time (PTT).
lat	character or numeric. Latitude column name or a numeric vector, used to derive daylength when photoperiod = TRUE and no daylength column exists. Default "LAT".
daylength.id	character. Existing daylength column (h), e.g. "N" from param_radiation(). Used in preference to deriving it.
p.opt, p.sens	numeric. Optimum daylength (h) and sensitivity (fractional slowdown per hour of deviation). When p.opt is NULL (default) the crop template's value is used – 12.5 h for short-day maize, 16 h for long-day wheat, and so on.
day.type	character. "short", "long" or "neutral". NULL (default) takes the crop template's value. A "neutral" crop silently keeps a photoperiod factor of 1 even when photoperiod = TRUE.
planting.id	character. Optional column giving planting date, used to reset accumulation. If NULL, accumulation starts at the first record of each environment.
merge	boolean. If TRUE (default) return env.data with the new columns appended; if FALSE return only the new columns.
verbose	boolean. Print a progress banner.

Details

Which method. "gdd" is the convention and what param_temperature() already computes; use it for comparability. "baskerville" is materially better where daily range straddles Tbase – high-elevation or early-spring sites – because the simple mean then understates accumulation. "trapezoid" is the only option that penalises supra-optimal heat, relevant for heat-stress work.

Photothermal time. Maize, sorghum and rice are quantitative short-day plants: long days delay flowering. With photoperiod = TRUE the daily increment is scaled by a linear daylength factor, which keeps tropical and temperate plantings on a common developmental scale. The default p.sens = 0.02 is deliberately mild; calibrate it before relying on the absolute stage dates.

Caveat. These thresholds are cultivar-generic. Maturity group alone moves maize R6 by 300-500 C d. Treat the stage boundaries as a consistent relative scale, not as a validated phenology prediction, unless you have calibrated stages against observed dates.

Value

A data.frame with these columns added:

dTT daily thermal time (C d), or photothermal time if `photoperiod = TRUE`

cumTT cumulative thermal time from planting

stage ordered factor, the developmental stage of that day

dstage relative position within the stage, in [0, 1)

stage_day day counter within the stage (1-based)

The attribute `"phenology"` carries the stage table, the method and the cardinal temperatures. Environments whose season ends before the last threshold get `NA` for the stages never reached, and a warning names them – silently recycling the final stage would fabricate development that the weather does not support.

Author(s)

Germano Costa Neto

References

McMaster, G.S. & Wilhelm, W.W. (1997) Growing degree-days: one equation, two interpretations. *Agric. For. Meteorol.* 87(4), 291-300.

Baskerville, G.L. & Emin, P. (1969) Rapid estimation of heat accumulation from maximum and minimum temperatures. *Ecology* 50(3), 514-517.

Abendroth, L.J. et al. (2011) *Corn Growth and Development*. PMR 1009, Iowa State University Extension.

See Also

`param_temperature`, `W_matrix`, `env_risk_profile`, `summaryWTH`

Examples

```
data("maizeWTH")

## 1. Default maize staging on cumulative GDD
ph <- env_phenology(maizeWTH, crop = "maize")
table(ph$env, ph$stage)

## 2. Photothermal time, daylength taken from param_radiation()'s N column
ph2 <- env_phenology(processWTH(maizeWTH), crop = "maize",
                     photoperiod = TRUE, daylength.id = "N")

## 3. Custom stages and the Baskerville-Emin method
ph3 <- env_phenology(maizeWTH,
                     stages = c(veg = 500, flow = 1200, fill = 2200),
                     Tbase = 8, Tupper = 32, method = "baskerville")

## 4. Hand the stage factor to the existing covariable machinery.
```

```
##    This is the point of the function: W columns now align on
##    development instead of on calendar date.
W <- W_matrix(env.data = ph, var.id = c("T2M", "PRECTOT"),
              by.interval = TRUE, time.window = NULL)
```

`env_risk_profile` *Historical Envirotype Risk Profile Across Years*

Description

Quantifies how OFTEN each stress envirotype occurs at a site, by developmental stage, across many historical seasons. Where `env_typing()` describes the environments you actually observed, `env_risk_profile()` estimates the underlying climatology those environments were drawn from – the frequency, not the realisation.

This is the quantity a breeding programme needs to weight its testing network: a mega-environment that appears in 1 season out of 20 should not carry the same weight as one that appears in 12.

Usage

```
env_risk_profile(
  env.data,
  site.id = "env",
  date.id = "YYYYMMDD",
  var.id = NULL,
  years = NULL,
  planting = NULL,
  season.length = 150,
  cardinals = NULL,
  by.stage = TRUE,
  crop = "maize",
  stages = NULL,
  Tbase = NULL,
  Tupper = NULL,
  method = "gdd",
  min.days = 30L,
  return.seasons = FALSE,
  verbose = TRUE
)
```

Arguments

`env.data`	data.frame. Multi-year daily weather for one or more sites, typically `get_weather()` run over a long window, then `processWTH()`. Must contain a site id, a date, and the variables in `var.id`.

site.id	character. Site-id column. Default "env". Note this is a SITE, not a site-year; the site-year is formed internally.
date.id	character. Date column (Date or YYYYMMDD). Default "YYYYMMDD".
var.id	vector (character). Variables to profile. Defaults to those present among the names of cardinals.
years	vector (numeric). Seasons to include. Defaults to every year present with a complete season.
planting	character, Date vector or "DOY". Planting date per season. A single "MM-DD" string applies the same sowing date every year; a named vector keyed by site applies a per-site date; a Date vector is used as given.
season.length	numeric. Days after planting retained per season when phenology is not used. Default 150.
cardinals	named list. Per-variable list(breaks =, labels =) defining stress classes. Defaults in .RISK_CARDINALS cover T2M_MAX, T2M_MIN, T2M, PETP, PRECTOT, VPD, FRUE and Ks.
by.stage	boolean. If TRUE (default) compute risk within thermal-time stages by calling env_phenology() per season.
crop, stages, Tbase, Tupper, method	Passed to env_phenology() when by.stage = TRUE.
min.days	integer. Minimum days a site-year must contribute to be kept. Default 30.
return.seasons	boolean. If TRUE, also return the per-site-year table behind the frequencies.
verbose	boolean. Print progress.

Details

Two frequencies are reported and they answer different questions. freq is the proportion of SEASONS in which the class occurred at least once – the probability of encountering that stress. mean_days is the expected NUMBER OF DAYS per season – the intensity. A site can have freq = 1.0 for mild heat while averaging only 3 days of it; that is a very different target from one averaging 40 days.

Return period is simply 1 / freq and inherits all the usual caveats: it is an average recurrence interval estimated from a finite record, not a schedule. With 20 years of data a return period above ~10 seasons is barely distinguishable from noise. The function will not print a return period longer than the record.

Stationarity. Frequencies computed over 1990-2024 describe that period. If the site is warming, they are already an underestimate of future heat risk. Use project_risk() with a scenario to see the shift.

Value

An object of class "env_risk": a list with

frequency data.frame of site x stage x variable x class with n_years, n_occur, freq (probability a season shows that class at all), and mean_days (expected days per season)

severity data.frame of site x stage x variable with the mean, sd and 10th/90th percentile of the variable across seasons

return_period data.frame with `1 / freq`, the expected number of seasons between occurrences of each stress class

seasons per-site-year detail if `return.seasons = TRUE`

call, years, cardinals provenance

Author(s)

Germano Costa Neto

References

Chapman, S.C. et al. (2000) Using crop simulation to generate genotype by environment interaction effects for sorghum in water-limited environments. *Aust. J. Agric. Res.* 51(2), 209-221.

Heinemann, A.B. et al. (2015) Drought impact on rainfed common bean production areas in Brazil. *Agric. For. Meteorol.* 225, 1-12.

See Also

`env_phenology`, `env_typing`, `tpe_weights`, `project_risk`

Examples

```
## 30 seasons of weather for three sites
sites <- data.frame(env = c("PIRA", "SETE", "NMAC"),
                    lat = c(-22.7, -19.4, -21.2),
                    lon = c(-47.6, -44.2, -45.2))

wth <- get_weather(env.id = sites$env, lat = sites$lat, lon = sites$lon,
                   start.day = "1994-01-01", end.day = "2024-12-31")
wth <- processWTH(wth)

## Risk of heat and drought at flowering, sown 15 October each year
rp <- env_risk_profile(wth, site.id = "env", planting = "10-15",
                       var.id = c("T2M_MAX", "PETP"),
                       crop = "maize", by.stage = TRUE)
rp

## How often is flowering heat-stressed at each site?
subset(rp$frequency, stage == "R1" & class == "heat_severe")

## Weights for a target population of environments
w <- tpe_weights(rp, stage = "R1", var = "PETP")
```

env_target_importance *Environmental drivers of phenotypic/target clustering*

Description

Groups environments by their target metrics (`Y`: phenotypic means, heritabilities, selection accuracies, etc.) using K-means, then trains a Random Forest to predict those clusters from the environmental covariable matrix (`W`). The Random Forest variable importance ranks which environmental variables best explain the target-based grouping of environments. The number of clusters can be supplied by the user or optimised automatically with the average silhouette width.

Usage

```
env_target_importance(
  Y,
  W,
  k = NULL,
  k.max = 10,
  scale.Y = TRUE,
  ntree = 1000,
  importance.type = c("gini", "permutation"),
  nstart = 25,
  seed = NULL,
  save = FALSE,
  dir.path = NULL,
  verbose = TRUE
)
```

Arguments

`Y`	numeric vector, matrix or data.frame. Target metrics with environments in rows (e.g. adjusted means, H^2, predictive ability). A named vector is also accepted.
`W`	matrix or data.frame. Environmental covariables with environments in rows and variables in columns (e.g. a `W_matrix` output).
`k`	integer or NULL. Number of clusters. If `NULL` (default) it is chosen by maximising the average silhouette width over `2:k.max`.
`k.max`	integer. Maximum number of clusters tested when `k` is `NULL`. Default 10.
`scale.Y`	boolean. If `TRUE` (default) `Y` is centred and scaled before clustering.
`ntree`	integer. Number of trees in the Random Forest. Default 1000.
`importance.type`	character. `"gini"` (MeanDecreaseGini, default) or `"permutation"` (MeanDecreaseAccuracy).
`nstart`	integer. Number of random starts for K-means. Default 25.
`seed`	integer or NULL. Optional seed for reproducibility.

save	boolean. If TRUE, writes the importance table and cluster assignment as .csv.
dir.path	character. Output directory when save = TRUE. Defaults to the working directory.
verbose	boolean. If TRUE (default) prints progress messages.

Details

This is a supervised counterpart of env_cluster: instead of grouping environments by their enviromic profile, environments are grouped by *what the breeder cares about* (yield level, heritability, accuracy) and the enviromic data are then asked to explain that grouping. Variables at the top of importance are candidate drivers of the observed environmental stratification and good candidates for env_threshold_gxe.

Permutation importance ("permutation") is generally preferred when covariables differ in scale or in number of categories, since MeanDecreaseGini is biased towards high-cardinality predictors.

Value

A list with:

- clusters: named integer vector of the cluster of each environment.
- k: the number of clusters used.
- silhouette: data.frame of average silhouette width per tested k (NULL if k was given).
- kmeans: the kmeans object.
- randomForest: the fitted randomForest object.
- importance: data.frame of environmental variables ranked by importance (descending).

Author(s)

Germano Costa Neto

References

Costa-Neto, G., da Matta, D., Fernandes, I. K., & Heinemann, A. B. (2023). Environmental clusters defining breeding zones for tropical irrigated rice in Brazil. *Agronomy Journal*, 115(5). doi:10.1002/agj2.21481

See Also

env_cluster, W_matrix

Examples

```
if (requireNamespace("randomForest", quietly = TRUE)) {
  data("maizeYield"); data("maizeWTH")

  ## Environment-level target: mean grain yield per environment
  Y <- tapply(maizeYield$value, maizeYield$env, mean)
```

```
  ## Environmental covariables (environments x covariables)
  W <- W_matrix(env.data = maizeWTH[maizeWTH$daysFromStart < 100, ],
                var.id = c("T2M", "T2M_MAX", "PRECTOT"), statistic = "mean")

  ## Which environmental variables best explain the target-based grouping?
  out <- env_target_importance(Y = Y, W = W, seed = 1)
  out$k
  out$clusters
  head(out$importance)

  ## Permutation importance and a fixed number of clusters
  out2 <- env_target_importance(Y = Y, W = W, k = 2,
                                importance.type = "permutation", seed = 1)
  head(out2$importance)

  ## Multi-trait target (e.g. mean and heritability per environment)
  Ymat <- cbind(mean = Y, sd = tapply(maizeYield$value, maizeYield$env, stats::sd))
  out3 <- env_target_importance(Y = Ymat, W = W, seed = 1)
  out3$importance
}
```

`env_typing` *Environmental Typologies based on Cardinal, Quantilic Limits or Data-Driven Mining*

Description

Returns environmental typologies (*envirotypes*) that can be used as envirotype markers. Typologies are given by cardinals (discrete intervals for each variable), by empirical quantiles, or – when `envirotype_mining = TRUE` – learned from the data: for each variable a K-means clustering is run over a grid of candidate k values and the optimal number of envirotypes is selected by the Calinski-Harabasz (variance ratio) criterion.

Usage

```
env_typing(
  env.data,
  var.id,
  env.id,
  cardinals = NULL,
  days.id = NULL,
  time.window = NULL,
  names.window = NULL,
  quantiles = NULL,
  id.names = NULL,
  by.interval = FALSE,
  scale = FALSE,
```

```
  format = NULL,
  ratio = FALSE,
  joint = FALSE,
  joint.by.interval = TRUE,
  combine.features = FALSE,
  combine.max.order = 2L,
  envirotype_mining = FALSE,
  k.range = 2:10,
  nstart = 25,
  iter.max = 100,
  seed = 1234,
  plot_envirotypes = FALSE,
  min.window = 2L,
  verbose = TRUE
)
```

Arguments

`env.data`	data.frame of environmental variables from `get_weather`.
`var.id`	character. Which variables will be used in the analysis.
`env.id`	character. Column used as id for environments.
`cardinals`	list (numeric). Cardinal thresholds per variable. If `NULL`, see `quantiles`. Ignored when `envirotype_mining = TRUE`.
`days.id`	character. Name of the column indicating the days from start.
`time.window`	vector (numeric). If `by.interval = TRUE`, the temporal breaks.
`names.window`	vector (character). If `by.interval = TRUE`, the interval names. Note that `cut()` is applied with an open-ended final break, so `time.window = c(0,40,80,120)` defines FOUR intervals; supply either 4 names or 3, in which case the trailing one is auto-named.
`quantiles`	vector (numeric). Probability quantiles used when `cardinals` is `NULL`. Default `c(0.01, .25, .50, .75, .99)`. Ignored when mining.
`id.names`	vector (character). Columns used as id for the environmental variables.
`by.interval`	boolean. Compute temporal intervals inside each environment. Default `FALSE`.
`scale`	boolean. If `TRUE`, variables assume $x \sim N(0,1)$. Default `FALSE`.
`format`	character. Output shape, `'long'` (default) or `'wide'`.
`ratio`	boolean. If `TRUE`, `Freq` is rescaled to a relative frequency in $[0,1]$ within each environment x interval x variable group. Default `FALSE`.
`joint`	boolean. Only used when `envirotype_mining = TRUE`. If `FALSE` (default) one univariate K-means is run per variable. If `TRUE`, the variables in `var.id` are COMBINED: the vector of all variables observed at the same time point is clustered jointly, so each envirotype is a recurring multi-variable state of the environment (analogous to a haplotype across loci).
`joint.by.interval`	boolean. Only used when `joint = TRUE`. If `TRUE` (default) a separate K-means is fitted inside each time interval; if `FALSE` a single K-means is fitted over all intervals pooled.

`combine.features`
: boolean. Only valid when `envirotype_mining = TRUE` and `joint = FALSE`. If `TRUE`, envirotypes are mined for every unique COMBINATION of the variables in `var.id`: each variable on its own, then every pair, every triple, and so on. A combination of two or more variables is clustered jointly (as in `joint = TRUE`); a single-variable combination reduces to the univariate case. Each combination receives a compact generic id (`FC001`, `FC002`, ...) and the returned `feature_combinations` table maps every id back to its variables and time intervals. Default `FALSE`.

`combine.max.order`
: integer or `NULL`. Largest combination size to enumerate when `combine.features = TRUE`. Because the number of combinations grows as a power set ($2^n - 1$), this defaults to `2` (single variables and pairs only), which stays tractable for dozens of variables. Increase it to include triples and higher (e.g. `3`), or set `NULL` to enumerate all `length(var.id)` orders. A guard stops the run if the requested number of combinations is impractically large.

`envirotype_mining`
: boolean. If `TRUE`, envirotype classes are mined from the data by K-means with `k` chosen by the Calinski-Harabasz index. Default `FALSE`.

`k.range`
: integer vector. Candidate numbers of envirotypes tested when mining. Default `2:10`.

`nstart`
: numeric. Random starts passed to `stats::kmeans`. Default 25.

`iter.max`
: numeric. Maximum iterations passed to `stats::kmeans`. Default 100.

`seed`
: numeric. Random seed for reproducible K-means. Default 1234; `NULL` skips seeding.

`plot_envirotypes`
: boolean. If `TRUE`, draws a barplot of the frequency (or ratio) of each envirotype per environment, faceted by variable (univariate) or by time interval (joint). Uses **ggplot2** when available, base graphics otherwise. Default `FALSE`.

`min.window`
: integer. Minimum number of observations required in the open-ended trailing interval. Default 2; use 0 or `NULL` to always keep it.

`verbose`
: boolean. If `TRUE` (default) prints progress messages.

Details

The Calinski-Harabasz index for a partition into k groups of n observations is

$$CH(k) = \frac{BGSS/(k-1)}{WGSS/(n-k)}$$

where BGSS is the between-group and WGSS the within-group sum of squares. The k maximising CH is retained. Clusters are relabelled in increasing order of their centres so that `Envirotype_<var>_00001` is always the lowest range of the variable. For joint (multivariate) mining there is no natural ordering, so centroids are ordered along their first principal component and variables are standardised within the window before clustering so that no feature dominates the Euclidean distance.

Value

If `envirotype_mining = FALSE` (default) and `plot_envirotypes = FALSE`: a data.frame (or matrix, if `format = 'wide'`) of envirotype frequencies.

Otherwise a `list` with:

`typologies` data.frame (long) or matrix (wide) of envirotype frequencies.

`envirotype_description` data.frame describing the numeric range captured by each envirotype: variable, label, cluster index, `n`, `min`, `max`, `mean`, `sd`, `median` and the `lower`/`upper` cut boundaries.

`mining_summary` data.frame with the optimal `k`, the CH index at the optimum and the full CH profile over `k.range`.

`feature_combinations` only when `combine.features = TRUE`: a data.frame mapping each generic combination id to its number of features (`n_features`), the `features` themselves and the time `intervals`.

`plot` the plot object, or `NULL`.

Author(s)

Germano Costa Neto

See Also

`W_matrix`, `env_kernel`, `T_matrix`

Examples

```
data("maizeWTH")
env.data <- maizeWTH[maizeWTH$daysFromStart < 100, ]

## 1. Generic time intervals
env_typing(env.data = env.data, env.id = 'env', var.id = 'T2M', by.interval = TRUE)

## 2. Specific time intervals with names
env_typing(env.data = env.data, env.id = 'env', var.id = 'T2M',
           by.interval = TRUE,
           time.window  = c(0, 15, 35, 65, 90),
           names.window = c('1-initial growing', '2-leaf expansion I',
                            '3-leaf expansion II', '4-flowering', '5-grain filling'))

## 3. Cardinal (ecophysiological) thresholds
env_typing(env.data = env.data, env.id = 'env',
           var.id    = c('T2M', 'PRECTOT'),
           cardinals = list(T2M = c(0, 9, 22, 32, 45), PRECTOT = c(0, 5, 10, 25, 100)))

## 4. Data-driven univariate mining (K-means + Calinski-Harabasz)
out <- env_typing(env.data = env.data, env.id = 'env',
                  var.id = c('T2M', 'PRECTOT'),
                  envirotype_mining = TRUE, k.range = 2:8)
out$typologies
```

```
out$envirotype_description
out$mining_summary

## 5. Joint (multi-variable) envirotypes per development stage
outj <- env_typing(env.data = env.data, env.id = 'env',
                   var.id = c('T2M', 'PRECTOT'),
                   by.interval = TRUE, time.window = c(0, 30, 60, 90),
                   envirotype_mining = TRUE, joint = TRUE)
head(outj$envirotype_description)

## 6. Relative frequencies, wide format (ready for env_kernel)
ET <- env_typing(env.data = env.data, env.id = 'env', var.id = 'T2M',
                 format = 'wide', ratio = TRUE)

## 7. Envirotypes for feature combinations (singles + pairs by default)
outc <- env_typing(env.data = env.data, env.id = 'env',
                   var.id = c('T2M', 'PRECTOT', 'VPD'),
                   envirotype_mining = TRUE, combine.features = TRUE)
outc$feature_combinations        # generic id -> which variables
head(outc$envirotype_description)

## include triples too (combine.max.order = 3), or NULL for all orders
outc3 <- env_typing(env.data = env.data, env.id = 'env',
                    var.id = c('T2M', 'PRECTOT', 'VPD'),
                    envirotype_mining = TRUE, combine.features = TRUE,
                    combine.max.order = 3)
```

G2F_2014_17 *Genome to Fields (G2F) GxE panel (2014-2017)*

Description

Genomes to fields (G2F) is a multi-institutional, public collaborative to develop information and tools that support the translation of maize (Zea mays L.) genomic information into relevant phenotypes for the benefit of growers, consumers, and society. Building on existing maize genome sequence resources, the project focuses on developing approaches to improve phenomic predictability and facilitate the development and deployment of tools and resources that help address fundamental problems of sustainable agricultural productivity. Specific projects within G2F involve collaboration from research fields such as genetics, genomics, plant physiology, agronomy, climatology and crop modeling, computational sciences, statistics, and engineering. See https://bmcresnotes.biomedcentral.com/articles/10.11 020-4922-8.

Here we used the geographic coordinates from 93 environments (diverse locations across 4 years)

Usage

```
data(G2F_2014_17)
```

Format

A data.frame containing environmental identifications (env), geographic coordinates (lat, lon) and the time window (start, end) given in YYYY-MM-DD.

data.frame file

Examples

```
require(EnvRtype)
data(G2F_2014_17)
head(G2F_2014_17)

# let's use the first 10 locations only
data  = G2F_2014_17
env   = data$env[1:10]
lon   = data$lon[1:10]
lat   = data$lat[1:10]
start = data$start[1:10]
end   = data$end[1:10]


# get_weather() queries the NASA POWER web service
weather.data =
  EnvRtype::get_weather(env.id = env,
                        lat = lat,
                        lon = lon,
                        start.day = start,
                        end.day = end)

head(weather.data)
```

---

get_AEZ *Retrieve FAO GAEZ Agro-Ecological Zone Classification for Sites*

Description

Extracts the FAO/IIASA Global Agro-Ecological Zone (GAEZ v4) class at a set of coordinates. By default it reads the GAEZ 57-class raster **bundled with the package**, so it works offline with no arguments beyond the coordinates. Companion to `get_weather` and `get_soil`.

Usage

```
get_AEZ(
  env.id,
  lat,
  lon,
  file = NULL,
  url = aez_default_url(),
```

```
  dir.path = NULL,
  legend = NULL,
  buffer = 0,
  overwrite = FALSE,
  timeout = 1800L,
  verbose = TRUE
)
```

Arguments

`env.id`	character vector. Environment/site identifiers.
`lat`	numeric vector. Latitude in decimal degrees (WGS84), same length as `env.id`.
`lon`	numeric vector. Longitude in decimal degrees (WGS84), same length as `env.id`.
`file`	character. Path to a GAEZ GeoTIFF already on disk. If `NULL` (default), `get_AEZ` uses the raster **shipped with the package** (`system.file("extdata", "aez_v9v2red_5m_CRUTS32_Hist = "EnvRtype")`). Only if that bundled file is unavailable does it fall back to downloading from `url`. Supply `file` to use a different GAEZ release or scenario.
`url`	character. Download URL for a GAEZ raster, used only when `file` is `NULL` *and* the bundled raster is missing. Defaults to `aez_default_url()`, a best-effort template that has *not* been verified against a live FAO server – see Details.
`dir.path`	character. Where to cache a downloaded raster. Default `tempdir()`. Use a persistent directory to avoid re-downloading.
`legend`	data.frame or NULL. Lookup table with at least `code` and `label` columns. `NULL` (default) uses `aez_legend()`. Pass `NA` to skip labelling and return raw codes only.
`buffer`	numeric. If greater than 0, the radius in metres of a circular neighbourhood summarised around each point (majority class and its share), rather than a single-cell lookup. Default 0.
`overwrite`	boolean. Re-download even if the cached file exists. Default `FALSE`.
`timeout`	integer. Seconds allowed for the download. Default 1800; GAEZ rasters are large and R's 60-second default will abort them.
`verbose`	boolean. Progress messages. Default `TRUE`.

Details

Where the raster comes from. GAEZ v4 has no documented per-point API: it publishes whole-globe GeoTIFFs through a JavaScript portal. `get_AEZ` is therefore a raster-extract tool and resolves its raster in three steps – an explicit `file`, else the layer bundled under `inst/extdata`, else a download from `url`. The bundled layer is the GAEZ v4 57-class AEZ historical baseline (`CRUTS32_Hist_8110`, ~5 arc-min), which makes the default call fully offline and reproducible.

The default URL is unverified. It is only used when both `file` and the bundled raster are unavailable. FAO has reorganised GAEZ download paths between releases and the portal is JavaScript-rendered, so no stable path could be confirmed. If a download is triggered and fails:

1. open https://gaez.fao.org/pages/data-access-download,

2. download the AEZ classification layer manually,
3. pass the file with `file = "path/to/layer.tif"`.

Coordinates must be WGS84 lon/lat. They are reprojected to the raster's CRS automatically, but only if the raster declares one. A raster with an undefined CRS triggers an error rather than a silent mis-extraction.

Interpreting `buffer`. A single-cell lookup at ~9 km resolution can be misleading near a zone boundary. With `buffer > 0` the returned `majority_share` shows how homogeneous the neighbourhood is; a value near 1 means the site sits well inside one zone, while 0.4 means the classification is effectively arbitrary at that location.

Value

A data.frame with one row per environment:

`env` environment id

`lat`, `lon` coordinates as supplied

`aez_code` raster cell value

`aez_label`, `thermal`, `moisture`, `arable` from the legend, when one is used

`majority_share` when `buffer > 0`, the proportion of cells in the neighbourhood holding the majority class – a purity measure

The raster path used is attached as the `"source"` attribute.

References

FAO & IIASA (2021). *Global Agro-Ecological Zones (GAEZ v4)*. Rome, FAO. https://gaez.fao.org

See Also

`aez_legend`, `get_weather`, `get_soil`

Examples

```
sites <- data.frame(
  env = c("Ames_IA", "Lincoln_NE", "Piracicaba_BR"),
  lat = c(42.03, 40.81, -22.71),
  lon = c(-93.62, -96.68, -47.63))

## Default: uses the raster bundled with the package (offline)
aez <- get_AEZ(sites$env, sites$lat, sites$lon)
aez

## Check zone purity in a 25 km neighbourhood
aez25 <- get_AEZ(sites$env, sites$lat, sites$lon, buffer = 25000)
aez25[, c("env", "aez_label", "majority_share")]

## Raw codes, no labelling
get_AEZ(sites$env, sites$lat, sites$lon, legend = NA)
```

```
## Not run:
## A different GAEZ layer you downloaded yourself
get_AEZ(sites$env, sites$lat, sites$lon, file = "GAEZ_AEZ_scenario.tif")

## End(Not run)
```

`get_bioclim` *WorldClim Bioclimatic Variables at Trial Sites*

Description

Returns the 19 standard bioclimatic indices (BIO1-BIO19) for each site, from the 1970-2000 WorldClim 2.1 normals.

Usage

```
get_bioclim(
  env.id = NULL,
  lat = NULL,
  lon = NULL,
  vars = 1:19,
  resolution = c("2.5m", "5m", "10m", "30s"),
  dir.path = NULL,
  verbose = TRUE
)
```

Arguments

`env.id`	character. Environment identifiers.
`lat`, `lon`	numeric. Site coordinates (decimal degrees).
`vars`	integer or character. Which indices, e.g. `c(1, 12)` or `c("BIO1", "BIO12")`. Default all 19.
`resolution`	character. `"10m"`, `"5m"`, `"2.5m"` or `"30s"`. Default `"2.5m"`.
`dir.path`	character. Cache directory.
`verbose`	boolean.

Value

data.frame with `env`, `LAT`, `LON` and one column per requested index. Units follow WorldClim: temperature indices in C, precipitation in mm, BIO3/BIO4/BIO15 dimensionless or percent.

What these are, and are not

Bioclim variables are LONG-TERM CLIMATE NORMALS, not the weather of your trial year. They describe the climate a site is drawn from, so they are appropriate for characterising a TPE, clustering locations, or as static site covariates. They are NOT a substitute for `get_weather()` + `env_phenology()` when you want to explain what happened in a particular season: a site's BIO1 is identical in a drought year and a wet year. Mixing normals and realised weather in one kernel without saying which is which is a common and serious error.

References

Fick, S.E. & Hijmans, R.J. (2017) WorldClim 2: new 1-km spatial resolution climate surfaces for global land areas. *International Journal of Climatology* 37(12), 4302-4315. doi:10.1002/joc.5086

O'Donnell, M.S. & Ignizio, D.A. (2012) Bioclimatic predictors for supporting ecological applications in the conterminous United States. *U.S. Geological Survey Data Series* 691.

Examples

```
sites <- data.frame(env = c("PIRA", "SETE"),
                    lat = c(-22.7, -19.4), lon = c(-47.6, -44.2))
bc  <- get_bioclim(env.id = sites$env, lat = sites$lat, lon = sites$lon)
bc2 <- get_bioclim(sites$env, sites$lat, sites$lon, vars = c(1, 12))
```

`get_climate_scenario` *Future-Climate Envirotyping from CMIP6 Downscaled Scenarios*

Description

Retrieves downscaled CMIP6 monthly climate (WorldClim 2.1) for a set of sites and converts it into a DAILY weather series compatible with `processWTH()`, `env_phenology()` and `water_balance()`, by applying the monthly change signal to an observed baseline series (delta-change downscaling).

Usage

```
get_climate_scenario(
  env.id = NULL,
  lat = NULL,
  lon = NULL,
  baseline = NULL,
  model = "MPI-ESM1-2-HR",
  models = c("IPSL-CM6A-LR", "MPI-ESM1-2-HR", "MRI-ESM2-0", "UKESM1-0-LL"),
  scenario = "ssp245",
  period = "2041-2060",
  resolution = c("2.5m", "5m", "10m", "30s"),
  method = c("delta", "raw"),
  dir.path = NULL,
```

```
  baseline.id = "env",
  baseline.date = "YYYYMMDD",
  precip.scale = c("multiplicative", "additive"),
  verbose = TRUE
)
```

Arguments

`env.id`	vector (character). Site identifiers.
`lat`, `lon`	vector (numeric). Site coordinates.
`baseline`	data.frame. Observed daily weather for the same sites, from `get_weather()`. Supplies the day-to-day variability that monthly projections cannot. REQUIRED for `method = "delta"`.
`model`	character. CMIP6 GCM name, e.g. `"MPI-ESM1-2-HR"`. One of `.CS_GCMS`. Use `"ensemble"` to average several.
`models`	vector (character). Used when `model = "ensemble"`. Default is a four-model spread across climate sensitivities.
`scenario`	character. `"ssp126"`, `"ssp245"`, `"ssp370"` or `"ssp585"`.
`period`	character. `"2021-2040"`, `"2041-2060"`, `"2061-2080"` or `"2081-2100"`.
`resolution`	character. WorldClim tile resolution: `"2.5m"` (default), `"5m"`, `"10m"` or `"30s"`. Finer is a much larger download. Resolution does NOT affect which model x scenario combinations exist: archive gaps are per GCM x SSP and are identical at every resolution and period. See `.CS_GAPS`.
`method`	character. `"delta"` (default) perturbs the baseline; `"raw"` returns the monthly projected values without downscaling.
`dir.path`	character. Cache directory for downloaded rasters. Defaults to `tools::R_user_dir("et", "cache")`.
`baseline.id`, `baseline.date`	character. Columns in `baseline`.
`precip.scale`	character. `"multiplicative"` (default) or `"additive"` application of the precipitation delta.
`verbose`	boolean. Print progress.

Details

What delta-change does and does not do. The monthly mean change from the GCM is added to (temperature) or multiplied onto (precipitation) the observed daily series. The result therefore keeps the OBSERVED day-to-day sequence, wet-day frequency and spell structure, and shifts only the monthly means. This is the standard, defensible way to drive a crop model from monthly projections (Hawkins et al. 2013), but note the consequence: **changes in variability, in dry-spell length, and in extreme-event frequency are NOT represented**. A scenario that in reality brings the same total rain in fewer, heavier events will look identical to one that brings it evenly. Heat-stress day counts under delta-change are driven purely by the mean shift, so they are a lower bound.

If extremes matter to your question, you need daily GCM output and a quantile-mapping bias correction, not this function.

Model choice is not innocuous. Across CMIP6, equilibrium climate sensitivity ranges roughly 1.8-5.6 K, and several high-sensitivity models are now regarded as implausibly hot. A single-model result is one draw from a wide distribution. Prefer `model = "ensemble"` and report the spread; a projection whose sign flips across models is not a finding.

Network access. WorldClim is an external host. In restricted environments the download will fail; pre-populate `dir.path` with the GeoTIFFs and the function will use the cache.

Value

For `method = "delta"`, a data.frame in `get_weather()` layout (`env`, `YYYYMMDD`, `daysFromStart`, `T2M`, `T2M_MAX`, `T2M_MIN`, `PRECTOT`, ...) with the scenario signal applied, plus `scenario`, `model` and `period` columns. For `method = "raw"`, a monthly data.frame of projected `tmin`, `tmax` and `prec` per site. The attribute `"deltas"` holds the monthly change factors that were applied, so the perturbation is auditable.

Author(s)

Germano Costa Neto

References

Fick, S.E. & Hijmans, R.J. (2017) WorldClim 2. *Int. J. Climatol.* 37(12), 4302-4315.

Hawkins, E. et al. (2013) Calibration and bias correction of climate projections for crop modelling. *Agric. For. Meteorol.* 170, 19-31.

O'Neill, B.C. et al. (2016) The Scenario Model Intercomparison Project (ScenarioMIP) for CMIP6. *Geosci. Model Dev.* 9, 3461-3482.

See Also

`get_weather`, `project_risk`, `env_phenology`

Examples

```
sites <- data.frame(env = c("PIRA", "SETE"),
                    lat = c(-22.7, -19.4), lon = c(-47.6, -44.2))

base <- get_weather(env.id = sites$env, lat = sites$lat, lon = sites$lon,
                    start.day = "2015-10-01", end.day = "2016-03-31")

## Mid-century, intermediate scenario, 4-model ensemble.
## The default 'models' set covers all four SSPs.
fut <- get_climate_scenario(env.id = sites$env, lat = sites$lat,
                            lon = sites$lon, baseline = base,
                            model = "ensemble", scenario = "ssp245",
                            period = "2041-2060")

## GFDL-ESM4 is fine for ssp126 or ssp370, just not ssp245/ssp585:
fut126 <- get_climate_scenario(env.id = sites$env, lat = sites$lat,
                               lon = sites$lon, baseline = base,
                               model = "GFDL-ESM4", scenario = "ssp126",
```

```
                                period = "2041-2060")

## Drop straight into the existing pipeline
fut <- processWTH(fut)
ph  <- env_phenology(fut, crop = "maize")

## Compare thermal time now vs mid-century
tapply(ph$dTT, ph$env, sum)
```

`get_elevation` *Elevation at Trial Sites from WorldClim 2.1*

Description

Returns elevation (m) for each site. Elevation is a strong covariate for temperature and radiation, so it is often worth carrying alongside the weather-derived envirotypes – but note it is STATIC: it adds between-site structure, never within-season dynamics.

Usage

```
get_elevation(
  env.id = NULL,
  lat = NULL,
  lon = NULL,
  resolution = c("2.5m", "5m", "10m", "30s"),
  dir.path = NULL,
  verbose = TRUE
)
```

Arguments

`env.id`	character. Environment identifiers.
`lat`, `lon`	numeric. Site coordinates (decimal degrees).
`resolution`	character. `"10m"`, `"5m"`, `"2.5m"` or `"30s"`. Default `"2.5m"`. 30s is ~9.7 GB – use only if you genuinely need ~1 km cells.
`dir.path`	character. Cache directory. Default is the same cache used by `get_climate_scenario()`.
`verbose`	boolean.

Details

Sites in the sea or outside the land mask return `NA` with a warning naming them, rather than silently propagating NA into downstream kernels.

Value

data.frame with `env`, `LAT`, `LON`, `ALT` (m).

References

Fick, S.E. & Hijmans, R.J. (2017) WorldClim 2: new 1-km spatial resolution climate surfaces for global land areas. *International Journal of Climatology* 37(12), 4302-4315. doi:10.1002/joc.5086

Examples

```
sites <- data.frame(env = c("PIRA", "SETE"),
                    lat = c(-22.7, -19.4), lon = c(-47.6, -44.2))
alt <- get_elevation(env.id = sites$env, lat = sites$lat, lon = sites$lon)
```

---

get_kernel — *Envirotype-informed Kernels for Multi-Environment Genomic Prediction*

Description

Builds the genomic, envirotypic and genotype-by-environment kernels required by `kernel_model`. Given a genomic relationship matrix (`p x p` genotypes) and optionally an environmental relationship matrix (`q x q` environments), it returns one kernel per model term, already spectrally decomposed and ready to sample from.

Two computational engines are available and produce numerically identical results:

`engine = "factored"` **(default)** Never allocates the `n x n` kernel. Each kernel is represented by its eigenvectors `U` (`n x r`) and eigenvalues `s`, obtained from the small `p x p` / `q x q` inputs. Memory is O(n*r) instead of O(n^2), which is what makes large multi-environment trials feasible.

`engine = "dense"` Builds the explicit `n x n` matrix and attaches it as `$Kernel`. Slower and quadratic in memory, but needed if you want to inspect or post-process the kernel itself.

Set `verbose = TRUE` to trace each step with timings; `verbose = FALSE` (default) is entirely silent.

Usage

```
get_kernel(
  K_G = NULL,
  K_E = NULL,
  data = NULL,
  model = "RNMM",
  intercept.random = FALSE,
  engine = c("factored", "dense"),
  env = "env",
  gid = "gid",
  y = "value",
  tol = 1e-10,
  keep_var = 1,
  scale_kernels = TRUE,
  verbose = FALSE
)
```

Arguments

- `K_G` — list of genomic relationship matrices (`p x p`), named. If `NULL`, an identity genotype kernel is used.
- `K_E` — list of environmental relationship matrices (`q x q`), named. If `NULL`, benchmark genomic-only models are built and `model` is restricted to `MM` / `MDs`.
- `data` — data.frame with the environment, genotype and phenotype columns.
- `model` — character. One of `"MM"`, `"MDs"`, `"EMM"`, `"EMDs"`, `"RNMM"`, `"RNMDs"`. See *Model structures*.
- `intercept.random` — logical. Append an identity genotype kernel `KG_Gi`, separating marker-predictable genetic variance from genotype-specific deviation the markers do not capture. Requires replication (a genotype seen in more than one environment) to be identifiable; with one observation per genotype it is confounded with the residual and a warning is issued. Default `FALSE`.
- `engine` — character. `"factored"` (default) or `"dense"`.
- `env`, `gid`, `y` — character. Column names in `data`.
- `tol` — numeric. Eigenvalue cutoff, relative to the largest eigenvalue. Must match `kernel_model(tol =)`. Default `1e-10`.
- `keep_var` — numeric in (0, 1]. Retain only the leading eigenvalues carrying this proportion of each kernel's trace. Sampler cost scales with total rank, so e.g. `0.99` is often a large speedup for negligible change in predictions. Default `1` (keep everything).
- `scale_kernels` — logical. Normalise each kernel to `mean(diag(K)) = 1` before decomposing. Must match `kernel_model(scale_kernels =)`. Default `TRUE`.
- `verbose` — logical. Narrate each step with timings. Default `FALSE`.

Value

A named list of kernels, class `"kernel_set"`. Each element is a list with:

`Type` `"D"` (dense) or `"BD"` (block-diagonal).

`decomp` list of blocks, each `list(s, U, nr, deltav, type, pos, scaled, tol)`.

`diag_mean` `mean(diag(K))`, computed in O(n*r).

`Kernel` the `n x n` matrix – only when `engine = "dense"`.

Attributes: `"ne"` (observations per environment), `"kd_meta"` (decomposition settings, so `kernel_model()` can validate the cache), and `"obs"` (index vectors and level names).

Model structures

model	terms	kernels
MM	y = mu + G	KG_G
MDs	y = mu + G + GE	KG_G, KGE_GE
EMM	y = mu + E + G	KE_W, KG_G
EMDs	y = mu + E + G + GE	KE_W, KG_G, KGE_GE

RNMM	y = mu + E + G + GxW	KE_W, KG_G, KGE_GW
RNMDs	y = mu + E + G + GE + GxW	KE_W, KG_G, KGE_GE, KGE_GW

GE is a block-diagonal (within-environment) interaction; GxW is the reaction-norm interaction built from the environmental kernel.

Why the factored engine is exact

Every kernel here is a congruence transform of a small matrix. With ig mapping observations to genotypes and ie to environments,

$$K_G = Z_g K_g Z_g' = K_g[ig, ig], \quad K_E = K_e[ie, ie]$$

so the eigendecomposition of the n x n kernel follows from the small one in O(p^3) rather than O(n^3). Writing $K = AA'$ with $A = V[ig,]\sqrt{D}$, the nonzero eigenpairs come from the small Gram matrix $A'A$, giving an orthonormal U and exact eigenvalues.

The interaction kernel is an elementwise product of two expansions, which is the expansion of a Kronecker product: with $j = (ie - 1)n_g + ig$, $K_{GE} = (K_e \otimes K_g)[j, j]$. On a complete balanced grid its eigenpairs are closed-form (eigenvalues $d_e \otimes d_g$, eigenvectors a Khatri-Rao product) so no large eigendecomposition is ever required.

Original Version

Costa-Neto et al (2021)

EnvRtype v.0.1.3, Sep 2026

Reusing kernels

Kernels depend only on genotypes and environments, never on the phenotype. Masking y for a cross-validation fold does **not** invalidate them, so build once and reuse across folds, chains and models.

Author(s)

Germano Costa Neto. Refactored version.

References

Costa-Neto G, Galli G, Carvalho HF, Crossa J, Fritsche-Neto R (2021). EnvRtype: a software to interplay enviromics and quantitative genomics in agriculture. *G3*, 11(4), jkab040.

See Also

kernel_model, decompose_kernels

Examples

```
data("maizeYield"); data("maizeG"); data("maizeWTH")

ECs <- W_matrix(env.data = maizeWTH[maizeWTH$daysFromStart < 100, ],
                var.id = c("FRUE", "PETP", "SRAD", "T2M_MAX"),
                statistic = "mean")
KE <- list(W = env_kernel(env.data = ECs)[[2]])
KG <- list(G = maizeG)

## factored engine (default): no n x n matrix is ever allocated
K <- get_kernel(K_G = KG, K_E = KE, data = maizeYield, model = "RNMM",
                env = "env", gid = "gid", y = "value", verbose = TRUE)

fit <- kernel_model(y = "value", data = maizeYield, random = K,
                    env = "env", gid = "gid",
                    iterations = 5000, burnin = 1000, verbose = TRUE)

## trim the eigenvalue tail: usually a large speedup, tiny accuracy cost
K99 <- get_kernel(K_G = KG, K_E = KE, data = maizeYield, model = "RNMM",
                  keep_var = 0.99)

## separate marker-predictable from residual genetic variance
Ki <- get_kernel(K_G = KG, K_E = KE, data = maizeYield, model = "RNMM",
                 intercept.random = TRUE)
```

---

get_soil — *Collect Soil Features from SoilGrids (250 m)*

Description

Queries the ISRIC SoilGrids REST API for soil properties at given coordinates.

Usage

```
get_soil(
  env.id = NULL,
  lat = NULL,
  lon = NULL,
  variables.names = "clay",
  depths = c("0-5cm", "5-15cm", "15-30cm", "30-60cm", "60-100cm", "100-200cm"),
  stat = c("mean", "q_5", "q_50", "q_95", "uncertainty"),
  wide = TRUE,
  sleep = 12,
  tries = 3L,
  timeout = 60,
  verbose = TRUE
)
```

Arguments

`env.id`	vector (character). Environment identifiers.
`lat`, `lon`	vector (numeric). Coordinates in decimal degrees (WGS84).
`variables.names`	vector (character). SoilGrids properties. See Details.
`depths`	vector (character). Depth intervals to request.
`stat`	character. Which summary to return: `"mean"` (default), `"q_5"`, `"q_50"`, `"q_95"` or `"uncertainty"`.
`wide`	boolean. Return one row per environment with one column per property-depth (default), else long format.
`sleep`	numeric. Minimum seconds between successive API requests. Minimum 12 (SoilGrids fair-use limit of 5 requests/min). Network time counts toward this interval, and duplicate coordinates are fetched only once.
`tries`	integer. Retry attempts per failed request.
`timeout`	numeric. Maximum seconds to wait for a single SoilGrids response before the request is aborted and retried. Prevents one stalled connection from blocking the whole sequential download. Default 60.
`verbose`	boolean. Print progress messages.

Details

Accepted properties: `bdod`, `cec`, `cfvo`, `clay`, `nitrogen`, `ocd`, `ocs`, `phh2o`, `sand`, `silt`, `soc`, `wv0010`, `wv0033`, `wv1500`.

SoilGrids returns values in mapped units (e.g. clay in g/kg x 10). No unit conversion is applied; the `unit` column records what was served.

Value

A data.frame of soil features.

Author(s)

Brandon Monier, modified by Germano Costa Neto

References

Poggio, L. et al. (2021). SoilGrids 2.0: producing soil information for the globe with quantified spatial uncertainty. *SOIL* 7, 217-240.

See Also

`get_weather`, `get_spatial`

Examples

```
## Not run:
## Requires the live SoilGrids service.
get_soil(env.id = "NM", lat = -13.05, lon = -56.05,
         variables.names = c("clay", "nitrogen"))

## End(Not run)
```

`get_soil_resumable` *Crash-Resilient Wrapper for get_soil()*

Description

Downloads SoilGrids properties one environment at a time with the same checkpointing scheme as `get_weather_resumable`: each result is written to disk immediately and recorded in an append-only log, so a crashed run resumes with `restart_from_log`.

SoilGrids is rate-limited and slower than NASA POWER, which makes long soil jobs the most frequent victims of a mid-run failure.

Usage

```
get_soil_resumable(
  env.id = NULL,
  lat = NULL,
  lon = NULL,
  variables.names = "clay",
  depths = c("0-5cm", "5-15cm", "15-30cm", "30-60cm", "60-100cm", "100-200cm"),
  stat = c("mean", "q_5", "q_50", "q_95", "uncertainty"),
  dir.path = NULL,
  run.id = "soil_run",
  sleep = 20,
  tries = 3L,
  resume = TRUE,
  verbose = TRUE
)
```

Arguments

`env.id`, `lat`, `lon`, `variables.names`, `depths`, `stat`
: Passed to `get_soil`.

`dir.path`
: character. Parent directory. Default `getwd()`.

`run.id`
: character. Run directory name. Default `"soil_run"`.

`sleep`
: numeric. Seconds between environments. Default 20, matching `get_soil`; lowering it much invites throttling by ISRIC.

`tries`
: integer. Retry attempts per environment. Default 3.

resume	logical. Skip environments already logged ok. Default TRUE.
verbose	logical. Print progress. Default TRUE.

Details

Results are stored **long** on disk, one row per property-depth, and reshaped to wide only at assembly. Storing long is what makes resumption safe: parts fetched at different times still stack correctly, whereas pre-widened parts could have mismatched columns.

See get_weather_resumable for the full checkpoint design.

Value

A data.frame of soil properties (wide by default), with attributes run.dir, n_ok, n_failed and failed.

See Also

restart_from_log, read_progress_log, get_soil

Examples

```
## Not run:
## Long-running download job requiring access to SoilGrids.
grid <- data.frame(env = paste0("P", 1:200),
                   lat = runif(200, -13, -12),
                   lon = runif(200, -56, -55))

soil <- get_soil_resumable(
  env.id = grid$env, lat = grid$lat, lon = grid$lon,
  variables.names = c("clay", "sand", "silt", "soc", "phh2o"),
  run.id = "zoning_grid")

## After a crash, in a new session:
soil <- restart_from_log(run.id = "zoning_grid")

## Points in the ocean never resolve -- keep the successes
soil <- restart_from_log(run.id = "zoning_grid", retry.failed = FALSE)

## End(Not run)
```

---

get_spatial *Extract Point Estimates from Raster Files*

Description

Extracts point values from a raster at the coordinates of each environment. This function supersedes extract_GIS(), which relied on the retired **raster** and **sp** packages.

Usage

```
get_spatial(
  digital.raster = NULL,
  which.raster.number = NULL,
  env.dataframe = NULL,
  env.id = NULL,
  lat = NULL,
  lng = NULL,
  name.feature = NULL,
  merge = TRUE,
  crs = 4326,
  verbose = TRUE
)
```

Arguments

`digital.raster`	SpatRaster, or a file path readable by `terra::rast()`.
`which.raster.number`	integer. Layer(s) to keep when the input is multi-layer.
`env.dataframe`	data.frame containing the environment id and coordinates.
`env.id`	character. Name of the environment-id column in `env.dataframe`.
`lat`, `lng`	character. Names of the latitude/longitude columns, or numeric vectors of coordinates used directly.
`name.feature`	character. Name(s) for the extracted layer(s).
`merge`	boolean. Join the extracted values back onto `env.dataframe`.
`crs`	numeric or character. CRS of the input coordinates. Default 4326 (WGS84).
`verbose`	boolean. Print progress messages.

Details

Coordinates are validated before extraction: latitude must lie in [-90, 90] and longitude in [-180, 180]. This catches the common lat/lon transposition, which otherwise yields silent `NA`s or values from the wrong hemisphere.

Value

A data.frame of extracted values, joined to `env.dataframe` when `merge = TRUE`.

Author(s)

Germano Costa Neto

See Also

`get_weather`, `get_soil`

Examples

```
if (requireNamespace("terra", quietly = TRUE) &&
    requireNamespace("sf", quietly = TRUE)) {
## A small synthetic elevation raster covering the example site.
elevation <- terra::rast(nrows = 2, ncols = 2, xmin = -57, xmax = -55,
                         ymin = -14, ymax = -12, crs = "EPSG:4326")
terra::values(elevation) <- c(100, 200, 300, 400)
ref <- data.frame(env = "NM", lat = -13.05, lng = -56.05)
get_spatial(env.dataframe = ref, digital.raster = elevation,
            lat = "lat", lng = "lng", env.id = "env",
            name.feature = "Elevation")
}
```

`get_weather` *Easily Collect Worldwide Daily Weather Data*

Description

Imports daily weather data from the NASA POWER API and augments it with derived agro-meteorological indices (VPD, photoperiod, extraterrestrial radiation, temperature-humidity indices).

Usage

```
get_weather(
  env.id = NULL,
  lat = NULL,
  lon = NULL,
  start.day = NULL,
  end.day = NULL,
  variables.names = NULL,
  dir.path = NULL,
  save = FALSE,
  parallel = FALSE,
  workers = NULL,
  chunk_size = 29,
  sleep = 60,
  tries = 3L,
  verbose = TRUE
)
```

Arguments

`env.id`	vector (character). Identifier of the site/environment. If `NULL`, environments are named `env1 ... envN`.
`lat`	vector (numeric). Latitude in decimal degrees (WGS84).

lon	vector (numeric). Longitude in decimal degrees (WGS84).
start.day, end.day	vector (character or Date). First and last date of collection, e.g. "2015-02-15". Recycled if length 1.
variables.names	vector (character). POWER parameters to request. See Details.
dir.path	character. Output directory when save = TRUE. Defaults to getwd().
save	boolean. Write one .csv per environment.
parallel	boolean. Download chunks in parallel. Default FALSE (serial), which is gentler on the NASA POWER rate limit; set TRUE to speed up large jobs.
workers	integer. Number of parallel processes. Defaults to 90% of available cores, capped at chunk_size.
chunk_size	integer. Points per chunk. Default 29; raising this may exceed the API rate limit.
sleep	numeric. Seconds to pause between chunks. Default 60.
tries	integer. Retry attempts per failed request. Default 3.
verbose	boolean. Print progress messages.

Details

daysFromStart is date-based. It is computed as as.numeric(YYYYMMDD - start.day) + 1, not as the row position. The previous implementation used 1:nrow(), which silently mis-numbered every row after a gap whenever the API returned fewer days than requested – assigning observations to the wrong phenological stage in summaryWTH.

Missing values. POWER fill values (-999 and -99) are converted to NA before any derived variable is computed.

Commonly used parameters include T2M, T2M_MAX, T2M_MIN, T2MDEW, PRECTOTCORR (returned as PRECTOT), RH2M, WS2M, ALLSKY_SFC_SW_DWN, ALLSKY_SFC_SW_DNI, ALLSKY_SFC_PAR_TOT, GWETROOT, GWETTOP, EVPTRNS.

Derived variables, added when their inputs are present:

- VPD: vapour pressure deficit (kPa), from T2MDEW, T2M_MAX, T2M_MIN
- N: photoperiod (h), Forsythe et al. (1995)
- RTA: extraterrestrial radiation (MJ/m2/day), FAO-56
- n: actual sunshine duration (h)
- TH1, TH2: temperature-humidity indices (NRC 1971; Yousef 1985)
- PAR_TEMP: ratio of PAR to mean temperature

Value

A data.frame with one row per environment-day. Always contains env, LON, LAT, YYYYMMDD, DOY and daysFromStart, followed by the requested and derived variables.

Author(s)

Germano Costa Neto and Giovanni Galli, modified by Tiago Olivoto

References

Sparks, A. (2018). nasapower: NASA-POWER Data from R. *JOSS* 3(30), 1035.

Forsythe, W.C., Rykiel, E.J., Stahl, R.S., Wu, H., Schoolfield, R.M. (1995). A model comparison for daylength as a function of latitude and day of the year. *Ecological Modelling* 80, 87-95.

See Also

summaryWTH, processWTH, get_soil, get_weather_hourly

Examples

```
## Single location
get_weather(env.id = "NM", lat = -13.05, lon = -56.05,
            start.day = "2015-02-15", end.day = "2015-06-15",
            variables.names = c("T2M", "PRECTOTCORR"))

## Two locations with different planting dates
get_weather(env.id = c("NM", "SO"),
            lat = c(-13.05, -12.32), lon = c(-56.05, -55.42),
            start.day = c("2015-02-15", "2015-02-13"),
            end.day   = rep("2015-06-15", 2))
```

---

get_weather_hourly *Day/Night Partitioned Hourly Weather from NASA POWER*

Description

Sibling collector to get_weather that downloads the NASA POWER *hourly* endpoint, partitions each 24 h cycle into day / night / pre-dawn, and returns one row per environment-day. It is a PARALLEL producer of a covariate block; it does NOT feed processWTH() and must not be wired into the daily pipeline by analogy.

Usage

```
get_weather_hourly(
  env.id,
  lat,
  lon,
  start.day,
  end.day,
  variables.names = NULL,
  night.def = c("solar", "fixed", "radiation"),
  night.hours = NULL,
  predawn.hours = 2:5,
  time_standard = c("LST", "UTC"),
```

```
  summarise = TRUE,
  thresholds = NULL,
  tries = 3,
  save = FALSE,
  dir.path = NULL,
  verbose = TRUE
)
```

Arguments

`env.id` character. Environment identifiers.

`lat, lon` numeric. Coordinates, recycled if length 1.

`start.day, end.day` character/Date. Collection window, recycled if length 1.

`variables.names` character. POWER parameters. **Maximum 15** on the hourly endpoint; validated before the request is sent.

`night.def` character. How to define night:

- `"solar"` Sunset/sunrise from Forsythe daylength. Night length varies with season and latitude.
- `"fixed"` User window via `night.hours`. Constant and directly comparable across sites and dates.
- `"radiation"` Hours with `ALLSKY_SFC_SW_DWN <= 0`. Empirical; costs one parameter slot.

`night.hours` integer. Hours 0-23 for `night.def = "fixed"`. Wrap-around (e.g. `c(22,23,0,1,2,3,4,5)`) is handled.

`predawn.hours` integer. Window for pre-dawn statistics. Default 2:5.

`time_standard` character. `"LST"` (default) or `"UTC"`. Keep LST: hour 3 is then ~3 a.m. solar time at every site, so pre-dawn windows are comparable across longitudes.

`summarise` boolean. `TRUE` (default) returns one row per environment-day. `FALSE` returns raw hourly rows plus `period` and `night_id` – the 24x volume path.

`thresholds` named numeric. Night-hour counts above each threshold, e.g. `c(T2M = 22)` gives `hours_above_T2M`.

`tries` integer. Retry attempts per site.

`save` boolean. Write one per-site CSV.

`dir.path` character. Output directory when `save = TRUE`.

`verbose` boolean. Print progress messages.

Details

POWER hourly is MERRA-2 reanalysis, not observation. The diurnal cycle at 0.5 x 0.625 degrees is modelled and sub-daily fidelity is weaker than the daily aggregate.

A daily `T2M_MIN` is not a substitute for a night mean: the two differ materially, and on a `Q10 ~ 2` process a few degrees is a large difference in implied rate (Peng et al. 2004).

Value

A data.frame of class `c("weather_hourly", "data.frame")`. With `summarise = TRUE`, one row per environment-day, carrying day/night/ pre-dawn statistics per variable. **Radiation-type parameters (shortwave, longwave, PAR) are summarised for DAYTIME only**: they are ~0 at night by construction, so night statistics would add no information.

References

Peng, S. et al. (2004) Rice yields decline with higher night temperature from global warming. *PNAS* 101(27), 9971-9975.

Forsythe, W.C. et al. (1995) A model comparison for daylength as a function of latitude and day of the year. *Ecological Modelling* 80, 87-95.

Allen, R.G. et al. (1998) Crop evapotranspiration. *FAO Irrigation and Drainage Paper* 56.

See Also

`get_weather`, `processWTH`, `summaryWTH`

Examples

```
## Reuse the coordinates of a shipped daily dataset
sites <- unique(maizeWTH[, c("env", "LON", "LAT")])
h <- get_weather_hourly(env.id = sites$env, lat = sites$LAT, lon = sites$LON,
                        start.day = "2016-01-01", end.day = "2016-01-31",
                        night.def = "solar")

## Night temperature is not the daily minimum
with(h, cor(T2M_night_mean, T2M_day_min, use = "complete.obs"))

## Fixed night window, wrap-around handled correctly
h2 <- get_weather_hourly(sites$env, sites$LAT, sites$LON,
                         "2016-01-01", "2016-01-31",
                         night.def = "fixed", night.hours = c(22:23, 0:5))
```

`get_weather_resumable` *Crash-Resilient Wrapper for get_weather()*

Description

Downloads daily weather one environment at a time, writing each result to disk and appending a line to a progress log as soon as it succeeds. If the session dies part-way through, `restart_from_log` resumes from the last completed environment instead of restarting the whole job.

Use this instead of `get_weather` whenever losing the run would hurt – in practice, more than a few dozen environments.

Usage

```
get_weather_resumable(
  env.id = NULL,
  lat = NULL,
  lon = NULL,
  start.day = NULL,
  end.day = NULL,
  variables.names = NULL,
  dir.path = NULL,
  run.id = "weather_run",
  sleep = 1,
  tries = 3L,
  resume = TRUE,
  verbose = TRUE
)
```

Arguments

`env.id`, `lat`, `lon`, `start.day`, `end.day`, `variables.names`	Passed to `get_weather`. `lat` and `lon` are required.
`dir.path`	character. Parent directory for run directories. Default `getwd()`.
`run.id`	character. Name of this run's directory. Default `"weather_run"`. Use a distinct name per job.
`sleep`	numeric. Seconds between environments. Default 1.
`tries`	integer. Retry attempts per environment before it is logged as failed. Default 3.
`resume`	logical. If `TRUE` (default), environments already marked `ok` in an existing log are skipped.
`verbose`	logical. Print progress. Default `TRUE`.

Details

Why per-environment files. Results are written as one `.rds` per environment under `parts/`, never accumulated only in memory. A process killed at any point loses at most the environment in flight.

Why an append-only log. Each line is appended and flushed individually. Rewriting one state file every iteration would risk the whole file to a kill mid-write; appending risks only the last line, and the reader tolerates a truncated final record.

Why a fingerprint. The manifest stores a hash of coordinates, dates and variables. Resuming with different arguments is refused, because silently concatenating two different queries produces a table that looks fine and means nothing.

Failures do not abort the run. A failed environment is logged and the loop continues. Inspect with `read_progress_log`, then retry with `restart_from_log`.

Serial by design. Unlike `get_weather`, this wrapper does not parallelise. Interleaved writes from several workers make progress ordering ambiguous, and NASA POWER rate-limits aggressive clients anyway. The cost is wall-clock time; the benefit is a log you can trust. For speed on a job you can afford to lose, use `get_weather(parallel = TRUE)`.

Value

A `data.frame` of daily weather for every environment that succeeded, with attributes `run.dir`, `n_ok`, `n_failed` and `failed`.

See Also

`restart_from_log`, `read_progress_log`, `get_weather`

Examples

```
## Not run:
## Long-running download job requiring access to NASA POWER.
sites <- data.frame(env = paste0("E", 1:200),
                    lat = runif(200, -30, -5),
                    lon = runif(200, -60, -40))

wth <- get_weather_resumable(
  env.id = sites$env, lat = sites$lat, lon = sites$lon,
  start.day = "2023-10-01", end.day = "2024-02-01",
  run.id = "trial2024")

## --- session dies at environment 137 ---
## In a NEW session, no arguments needed:
wth <- restart_from_log(run.id = "trial2024")

read_progress_log(run.id = "trial2024")
attr(wth, "failed")

## Straight into the characterization pipeline
wth <- processWTH(wth)
W   <- W_matrix(wth, env.id = "env", var.id = c("T2M", "PRECTOT"))

## End(Not run)
```

`grid_scan` *Scan many sites in memory-bounded chunks*

Description

[scan_untested_envs()] allocates an ‘ndraws x g x m‘ array, which is infeasible for raster-sized ‘m‘. This wrapper splits ‘W_new‘ into chunks, scans each, and stitches the per-site results. Quantities that are defined ACROSS new sites – ‘vcov‘ and ‘cor‘ – cannot be recovered from chunks and are returned as ‘NULL‘ with a warning if requested.

Usage

```
grid_scan(
  object,
  W_train,
  W_new,
  K_G = NULL,
  chunk_size = 500,
  max_draws = 200,
  verbose = TRUE,
  ...
)
```

Arguments

`object`, `W_train`, `K_G`	as in [scan_untested_envs()].
`W_new`	covariates for every grid cell / site (m x p).
`chunk_size`	sites per chunk.
`max_draws`	MCMC draws retained per chunk. Lower than the scan default because mapping needs the posterior mean far more than a tight interval.
`verbose`	report chunk progress.
`...`	passed to [scan_untested_envs()].

Value

list with ‘yHat_matrix‘, ‘yHat_lower‘, ‘yHat_upper‘, ‘yHat_prob_win‘, ‘genetic_var‘, ‘interpolation‘, ‘exceedance_detail‘, ‘meta‘. Class ‘scan_untested_envs‘ so downstream functions accept it, but with ‘vcov‘ and ‘cor‘ set to ‘NULL‘.

---

kernel_cv *k-fold cross-validation predictive ability (out-of-sample)*

Description

In-sample yHat overstates accuracy. This masks a fold of phenotypes as NA, refits, and correlates held-out predictions against observed values.

Usage

```
kernel_cv(
  y,
  data,
  random,
  env,
  gid,
```

```
  folds = 5,
  scheme = c("cv1", "cv0"),
  seed = NULL,
  tol = 1e-10,
  scale_kernels = TRUE,
  ...
)
```

Arguments

y	character. Name of the response column in data.
data	data.frame with the response, environment and genotype columns.
random	named list of kernels passed to kernel_model.
env	character. Name of the environment column in data.
gid	character. Name of the genotype column in data.
folds	integer. Number of cross-validation folds. Default 5.
scheme	"cv1" (random cells) or "cv0" (leave-one-environment-out).
seed	integer or NULL. Optional seed for reproducible fold assignment.
tol	numeric. Eigenvalue tolerance passed to the kernel decomposition.
scale_kernels	logical. If TRUE (default) kernels are scaled before fitting.
...	further arguments passed to kernel_model.

Details

The kernels are decomposed ONCE and reused across all folds: they depend only on genotypes and environments, never on the phenotype vector, so masking y does not invalidate them. This is the single largest saving in the package.

Value

A list summarising out-of-sample predictive ability, with components scheme (the validation scheme used), folds (number of folds), accuracy (overall correlation between held-out predictions and observations), rmse (overall root-mean-square error), per_fold (a data.frame of per-fold accuracy and rmse), fold_id (integer fold assignment for each row of data), predicted (held-out predictions) and observed (the observed response).

---

kernel_model *Kernel Models for Predicting Phenotypes across Multi-Environment Conditions*

Description

Fits Bayesian linear mixed models for multi-environment trials (MET) using the kernels produced by `get_kernel()`. The model is

$$y = 1\mu + X\beta + \sum_j u_j + e$$

with $u_j \sim N(0, K_j\sigma_j^2)$ and $e \sim N(0, I\sigma_e^2)$. Sampling runs in the spectral basis of each kernel, so every kernel is used through its eigendecomposition rather than as an n x n matrix.

The function returns genotype predictions, per-kernel variance components, broad- and narrow-sense heritabilities, and – with `keep_effects = TRUE` – posterior effect chains and breeder-facing report tables (per-environment rankings, stability, selection candidates).

Set `verbose = TRUE` to trace the six execution steps with per-step and total timings; `verbose = FALSE` (default) is entirely silent.

Usage

```
kernel_model(
  y,
  data = NULL,
  random = NULL,
  fixed = NULL,
  env,
  gid,
  verbose = FALSE,
  iterations = 1000,
  burnin = 200,
  thining = 10,
  tol = 1e-10,
  R2 = 0.5,
  digits = 4,
  seed = NULL,
  probs = c(0.025, 0.975),
  variance = c("realized", "scale"),
  scale_kernels = TRUE,
  keep_effects = FALSE,
  compute_CI = TRUE,
  R2_split = c("per_kernel", "shared"),
  prior_scale = c("eigen", "diag"),
  varcomp_method = c("postmean", "gibbs")
)
```

Arguments

`y`	character. Name of the phenotype column in `data`.
`data`	data.frame with the environment, genotype and phenotype columns. Rows must be grouped by environment if any kernel is block-diagonal.

random	list of kernels from get_kernel. Each element is a list carrying $decomp (the spectral decomposition) and $Type ("D" or "BD"); $Kernel is optional and is not required by the sampler. Factored kernels (no $Kernel) and dense kernels may be mixed in one list. Names should follow the KG_ / KE_ / KGE_ convention, which drives the variance-component roles.
fixed	matrix (n x p) of fixed effects, or NULL.
env, gid	character. Environment and genotype column names in data.
verbose	logical or 0/1. TRUE traces the six execution steps with per-step and total timings; FALSE (default) is entirely silent – no cat(), no message(). Results are identical either way.
iterations	integer. Total Gibbs iterations. Default 1000.
burnin	integer. Iterations discarded before collecting draws. Default 200.
thining	integer. Keep every thining-th post-burn-in draw. Default 10.
tol	numeric. Eigenvalue cutoff. Must match get_kernel(tol =) or the cached decomposition is rejected and recomputed. Default 1e-10.
R2	numeric in (0, 1). Prior proportion of phenotypic variance explained by the random terms. Default 0.5.
digits	integer. Rounding for the variance-component tables.
seed	integer or NULL. Sets the RNG state for reproducibility.
probs	length-2 numeric. Quantiles for posterior prediction intervals. Default c(0.025, 0.975).
variance	"realized" (default) or "scale". See *Details*.
scale_kernels	logical. Normalise each kernel to mean(diag(K)) = 1. Must match get_kernel(scale_kernels =) for the cache to be reused. Default TRUE.
keep_effects	logical. Retain per-draw effect chains. Required for varcomp_method = "postmean" and for the breeder-facing report tables ($predictions, $genotype_effects, $env_summary). Costs O(k * draws * n) memory. Default FALSE.
compute_CI	logical. Compute posterior prediction intervals for yHat. Requires storing effect chains. Default TRUE.
R2_split	"per_kernel" (default) or "shared". Controls how the prior variance R2 is distributed across kernels. See *Prior parameterisation*.
prior_scale	"eigen" (default) or "diag". Whether the prior scale is divided by the mean eigenvalue of each kernel, making kernels of differing rank comparable. See *Prior parameterisation*.
varcomp_method	"postmean" (default) or "gibbs". The estimator used for $varcomp. "postmean" needs keep_effects = TRUE; when unavailable it falls back to "gibbs" (reported in the trace when verbose = TRUE).

Details

Fits

$$y = 1\mu + X\beta + \sum_j u_j + e$$

with $u_j \sim N(0, K_j\sigma^2_{u_j})$ and $e \sim N(0, I\sigma^2_e)$, by Gibbs sampling in the spectral basis of each kernel. Writing $K_j = U_j S_j U_j'$, the effects are sampled as $u_j = U_j b_j$ with $b_j \sim N(0, S_j\sigma^2_{u_j})$, so the cost per iteration scales with the retained **rank** of each kernel, not with n^2. Trimming the eigenvalue tail via `get_kernel(keep_var =)` is therefore the most direct way to speed up sampling.

`variance = "scale"` reports the raw kernel-scale parameter `sigb[j]` (original BGGE behaviour, inflated for low-rank incidence kernels such as `KE_`). `variance = "realized"` reports `var(u_j)` averaged over draws – what most users mean by a variance component. The raw scale is always returned as `$sigb`.

Value

An object of class `"kernel_model"`, a list with:

`yHat` numeric vector of fitted/predicted values, length n.

`yHat.CI` matrix of posterior prediction intervals (`NULL` unless `compute_CI = TRUE`).

`varE` posterior mean residual variance.

`random` per-kernel posterior effects and variances.

`fit` the full sampler object: chains, kernel names, MCMC settings and – with `keep_effects = TRUE` – `Uchain`. (Named `$BGGE` in versions before v1.2.3.)

`meta` the aligned env / gid / y data frame.

`VarComp` variance-component table with confidence intervals.

`varcomp` validated components and heritabilities, class `"km_varcomp"`; see `varcomp_summary`.

`varcomp_gibbs` the same computed by the Gibbs-chain estimator, for comparison.

`runtime` sampler wall time in seconds.

With `keep_effects = TRUE` it additionally carries `predictions`, `genetic_cor_env`, `genotype_effects`, `env_summary` and `variance_proportions`.

Original Version

Costa-Neto et al (2021)

EnvRtype v.0.1.3, Sep 2026

Performance changes vs. the original

- **Cached eigendecomposition.** If `random` carries `$decomp` (attached by `get_kernel(decompose = TRUE)`) and the cache matches `tol` / `scale_kernels` / `n`, `eigen()` is skipped entirely. Cross-validation folds and multiple chains reuse one decomposition.
- **No stored transposes.** The original kept both `U` and `t(U)` per kernel (doubling eigenvector memory) and computed `crossprod(tU, b)`. Now only `U` is stored and the back-transform is `U %*% b`.
- **Optional chain storage.** Full per-draw effect matrices (`k` x `nCum` x `n` doubles) are allocated only when needed (`keep_effects = TRUE` or `yHat.CI`). Otherwise running sums reduce memory from O(k*nCum*n) to O(k*n).
- **Vectorised post-processing.** Row-wise `apply()` over draw matrices replaced by `rowSums`/`rowMeans` identities; `sweep()` replaced by recycled arithmetic.

- **Scalar chains in a matrix** rather than a nested list, removing per-iteration copy-on-modify of the chain container.

Prior parameterisation

Two defects in the original parameterisation are addressed by arguments:

`R2_split`. The original gave *every* kernel the full `R2` share, so with `nk` kernels the priors collectively assert `nk * R2` of the phenotypic variance – with `RNMDs` (4 kernels, `R2 = 0.5`) that is twice `var(y)`. `"per_kernel"` (default) uses `R2 / nk` so total prior mass stays at `R2` regardless of model size, keeping model comparisons on equal footing. `"shared"` restores the original behaviour.

`prior_scale`. Kernels normalised to `mean(diag(K)) = 1` are equal in scale but not in effective dimension: a rank-5 and a rank-750 kernel with the same mean diagonal carry very different total variance. `"eigen"` (default) divides the prior scale by `mean(s)` (= trace / rank), making priors comparable across kernels of differing rank.

Author(s)

Germano Costa Neto. Gibbs sampler adapted from Granato et al. (2018), BGGE package. Refactored version.

References

Costa-Neto G, Galli G, Carvalho HF, Crossa J, Fritsche-Neto R (2021). EnvRtype: a software to interplay enviromics and quantitative genomics in agriculture. *G3*, 11(4), jkab040.

Granato I, Cuevas J, Luna-Vazquez F, Crossa J, Montesinos-Lopez O, Burgueno J, Fritsche-Neto R (2018). BGGE: a new package for genomic-enabled prediction incorporating genotype x environment interaction models. *G3*, 8(9), 3039-3047.

See Also

`get_kernel`, `varcomp_summary`, `kernel_model_mc`, `kernel_cv`

Examples

```
data("maizeYield"); data("maizeG"); data("maizeWTH")

ECs <- W_matrix(env.data = maizeWTH[maizeWTH$daysFromStart < 100, ],
                var.id = c("FRUE", "PETP", "SRAD", "T2M_MAX"),
                statistic = "mean")
K <- get_kernel(K_G = list(G = maizeG),
                K_E = list(W = env_kernel(env.data = ECs)[[2]]),
                data = maizeYield, model = "RNMM",
                env = "env", gid = "gid", y = "value")

## silent fit
fit <- kernel_model(y = "value", data = maizeYield, random = K,
                    env = "env", gid = "gid",
                    iterations = 5000, burnin = 1000)
fit
fit$varcomp
```

```
## traced fit, with effect chains for the report tables
fit2 <- kernel_model(y = "value", data = maizeYield, random = K,
                     env = "env", gid = "gid",
                     iterations = 5000, burnin = 1000,
                     keep_effects = TRUE, verbose = TRUE)
head(fit2$genotype_effects)

## leave-one-environment-out: kernels do not depend on y, so build once
for (e in unique(maizeYield$env)) {
  tr <- maizeYield; tr$value[tr$env == e] <- NA
  f <- kernel_model(y = "value", data = tr, random = K,
                    env = "env", gid = "gid", iterations = 2000, burnin = 500)
}
```

---

kernel_model_clustered

Fit kernel_model() informed by an environment-cluster table

Description

"fixed" adds mega-environment means as fixed effects; "block" zeroes between-cluster entries of the GxE kernel so genotypes may rank differently across mega-environments; "both" does each.

Usage

```
kernel_model_clustered(
  y,
  data,
  random,
  env,
  gid,
  clusters,
  use = c("block", "fixed", "both"),
  gxe_pattern = "^KGE_|GE",
  fixed = NULL,
  ...
)
```

Arguments

y	character. Name of the response column in data.
data	data.frame with the response, environment and genotype columns.
random	named list of kernels passed to kernel_model.
env	character. Name of the environment column in data.

gid	character. Name of the genotype column in data.
clusters	an environment-cluster mapping: a cluster_environments result, a data.frame with columns env and cluster, or a named vector.
use	character. How clusters enter the model: "block" (default), "fixed" or "both".
gxe_pattern	character. Regular expression selecting the GxE kernels to block-diagonalise. Default "^KGE_|GE".
fixed	optional fixed-effects specification passed to kernel_model.
...	further arguments passed to kernel_model.

Details

Block-diagonalisation MODIFIES the GxE kernels, so any cached decomposition for those kernels is dropped and recomputed. Both dense and factored kernels are supported. Selected factored kernels are reconstructed as dense observation-by-observation matrices before blocking; unselected kernels are left unchanged.

Value

An object of class "kernel_model" (see kernel_model) with an added cluster_info element: a list recording the environment-to-cluster mapping, the number of clusters k, the mechanism used ("block", "fixed" or "both"), the gxe_pattern, the number of added fixed columns n_fixed_cols and the names of the blocked_kernels.

---

kernel_model_mc — *Multi-chain fit with convergence diagnostics (Gelman-Rubin R-hat, ESS)*

Description

Runs independent chains with different seeds and reports R-hat and a crude ESS for the residual and each kernel-scale parameter. R-hat above 1.01 or a tiny ESS means the chain has not mixed: increase ‘iterations‘.

Usage

```
kernel_model_mc(..., n_chains = 3, seed = 1, tol = 1e-10, scale_kernels = TRUE)
```

Arguments

...	arguments passed to kernel_model (e.g. y, data, random, env, gid).
n_chains	integer. Number of independent chains. Default 3.
seed	integer. Base seed; chain c uses seed + c. Default 1.
tol	numeric. Eigenvalue tolerance passed to the kernel decomposition.
scale_kernels	logical. If TRUE (default) kernels are scaled before fitting.

Details

The decomposition is shared across chains.

Value

An object of class "kernel_model" (the fit of the first chain; see kernel_model) with two added elements: diagnostics, a data.frame giving the Gelman-Rubin Rhat and effective sample size ESS for the residual variance and each kernel-scale parameter, and all_chains, the list of individual per-chain fits.

---

maizeG *Maize Genomic Relationship Matrix (GRM)*

Description

This data set was included in Souza et al. (2017) and Cuevas et al (2019) is from the Helix Seeds Company (HEL). Genotyping Array of 616 K SNPs (Single Nucleotide Polymorphism) (Unterseer et al. 2014). Standard quality controls (QC) were applied to the data, removing markers with a Call Rate $ 0.95. The remaining missing data in the lines were imputed with the Synbreed package (Wimmer et al. 2015) using the algorithms from the Beagle 4.0 software (Browning and Browning 2009). Markers with Minor Allele Frequency (MAF) of # 0.05 were removed. After applying QC, 52,811 SNPs were available to make the predictions. The phenotypic and genomic data of inbred lines are credited to Helix Seeds Ltda. Company.

Usage

```
data(maizeG)
```

Format

An object of class matrix (inherits from array) with 150 rows and 150 columns.

Examples

```
data(maizeG)
```

maizeWTH *Maize Weather Data*

Description

Weather data collected from NASA POWER for each one of the five environments in HELIX dataset.

Usage

```
data(maizeWTH)
```

Format

A data.frame containing weather variables for each environment in HEL dataset

Examples

```
data(maizeWTH); head(maizeWTH)
```

maizeYield *Maize Phenotypic Data Set*

Description

This data set was included in Souza et al. (2017) and Cuevas et al (2019) is from the Helix Seeds Company (HEL). It consists of grain yied from 452 maize hybrids obtained by crossing 111 pure lines (inbreds); the hybrids were evaluated in 2015 at five Brazilian sites (E1-E5). The experimental design used in each site was a randomized block with two replicates per hybrid. However, to facilitate the demonstration of functions, only 150 hybrids per environment are being considered, thus counting 750 genotype x environment observations. Grain yield data are mean-centered and scaled.

Usage

```
data(maizeYield)
```

Format

A data frame with 750 rows and 3 variables:

env environmental id (factor)

gid genotypic id (factor)

value grain yield value of genotype plus genotype by location interaction effects in kg ha-1 (numeric) ...

Examples

```
data(maizeYield); head(maizeYield)
```

`map_scan` *Map scanned predictions, or training-network coverage*

Description

Map scanned predictions, or training-network coverage

Usage

```
map_scan(
  scan,
  coords,
  lon_col = "lon",
  lat_col = "lat",
  date_col = NULL,
  type = c("performance", "coverage", "which_won_where"),
  genotypes = NULL,
  summarise = c("none", "mean"),
  facet_by = c("genotype", "date"),
  shp = NULL,
  clip = FALSE,
  mark_envelope = TRUE,
  envelope_level = c("gross", "mild"),
  win_threshold = 0.25,
  point_size = 2,
  palette = "viridis",
  output = c("plot", "table")
)
```

Arguments

`scan`	a ‘scan_untested_envs‘ / [grid_scan()] result.
`coords`, `lon_col`, `lat_col`, `date_col`	as in [scan_spatial_table()].
`type`	‘"performance"‘ (predicted value), ‘"coverage"‘ (where the training network supports prediction), or ‘"which_won_where"‘ (the top genotype per site, and how many genotypes tie it).
`genotypes`	genotypes to map. Required for ‘type = "performance"‘ unless ‘summarise = "mean"‘. For ‘type = "which_won_where"‘ it restricts the pool of genotypes that compete for the win (default: all).
`summarise`	for ‘type = "performance"‘: ‘"none"‘ (panel per genotype) or ‘"mean"‘ (single panel, mean over selected genotypes).

`facet_by` ‘"genotype"‘ (A) or ‘"date"‘ (C). ‘"date"‘ needs ‘date_col‘.

`shp` optional ‘sf‘ polygon layer (e.g. a state or country border from ‘geobr::read_state()‘) drawn underneath the points; also used to clip when ‘clip = TRUE‘.

`clip` if ‘TRUE‘ and ‘shp‘ is supplied, keep only the scanned sites that fall inside ‘shp‘ (points are matched to ‘shp‘ in its CRS).

`mark_envelope` hatch/grey cells outside the envelope.

`envelope_level` ‘"gross"‘ (default) or ‘"mild"‘.

`win_threshold` for ‘type = "which_won_where"‘: the minimum posterior probability of being the best genotype for another genotype to count as a co-winner at a site (default ‘0.25‘).

`point_size`, `palette` cosmetic.

`output` ‘"plot"‘ (default) returns a ggplot; ‘"table"‘ returns the data.frame that would have been plotted (one row per site, or per site-by-genotype for ‘type = "performance"‘), so you can build your own map. Clipping and ‘summarise‘ are applied first, and a logical ‘flagged‘ column marks sites outside the envelope. ‘"table"‘ needs no ggplot2.

Details

Three kinds of map are available through ‘type‘:

- ‘"performance"‘: the predicted value (‘yHat‘) of one or more genotypes across the scanned sites, one panel per genotype (or a single panel of the mean with ‘summarise = "mean"‘). Sites outside the training covariate envelope are drawn as grey crosses rather than on the value scale.
- ‘"coverage"‘: where the training network actually supports prediction, colouring each site by its geometric position relative to the training covariate cloud (interpolation, edge, mild- or gross-extrapolation).
- ‘"which_won_where"‘: a classic *which-won-where* view for selection. It answers two questions with two maps (and two tables). *Who won where* is the single top genotype per site, by posterior mean (‘yHat‘; ties broken by the first genotype). *Winners* is how many genotypes are statistically tied for the win, taken from the scan’s posterior draws: a genotype is a co-winner when its probability of being the best at that site (‘prob_win‘) is at least ‘win_threshold‘. The count (‘n_winners‘) and the tied genotypes (‘winners‘, most probable first) are reported per site, so you can see where the decision is clear (one winner) versus where several genotypes are indistinguishable. This needs a scan from the current [scan_untested_envs()], which stores the per-site win probabilities.

Value

a ggplot object, or a data.frame when ‘output = "table"‘. For ‘type = "which_won_where"‘ a two-element list is returned instead: ‘who_won_where‘ (the single top genotype per site, by posterior mean) and ‘winners‘ (how many genotypes have posterior ‘prob_win >= win_threshold‘), each a ggplot, or a data.frame when ‘output = "table"‘.

Examples

```
## A scan is normally produced by scan_untested_envs(); here we build a
## minimal one by hand -- 2 genotypes predicted at 9 sites on a 3x3 grid.
sites <- paste0("S", 1:9)
grid  <- expand.grid(lon = 1:3, lat = 1:3)
yh <- rbind(G1 = 6 + grid$lon * 0.3 - grid$lat * 0.1,
            G2 = 5 + grid$lat * 0.4)
colnames(yh) <- sites
scan <- list(
  yHat_matrix = yh, yHat_lower = yh - 0.4, yHat_upper = yh + 0.4,
  ## posterior P(best) per genotype x site (illustrative softmax of yHat)
  yHat_prob_win = prop.table(exp(yh), margin = 2),
  interpolation = data.frame(
    env               = sites,
    position          = rep(c("interpolation", "edge",
                              "gross_extrapolation"), each = 3),
    mahalanobis_ratio = seq(0.5, 3.5, length.out = 9),
    prop_outside      = seq(0, 0.8, length.out = 9),
    weight_negativity = 0, stringsAsFactors = FALSE))
class(scan) <- "scan_untested_envs"

## Site coordinates, keyed by the scanned environment name
coords <- data.frame(env = sites, lon = grid$lon, lat = grid$lat)


if (requireNamespace("ggplot2", quietly = TRUE)) {
  ## Where does the training network support prediction?
  map_scan(scan, coords, type = "coverage")

  ## Predicted performance for a single genotype
  map_scan(scan, coords, type = "performance", genotypes = "G1")

  ## Mean over several genotypes in one panel
  map_scan(scan, coords, type = "performance",
           genotypes = c("G1", "G2"), summarise = "mean")

  ## Which genotype wins at each site, and how many genotypes tie it
  ww <- map_scan(scan, coords, type = "which_won_where")
  ww$who_won_where
  ww$winners
}


## Export the plotted data instead of a figure, to map it your own way
head(map_scan(scan, coords, type = "coverage", output = "table"))

## Both winner tables (no ggplot2 needed), one row per site
ww_tab <- map_scan(scan, coords, type = "which_won_where", output = "table")
head(ww_tab$who_won_where)
head(ww_tab$winners)
```

```
## ---- Realistic scan: 50 new locations across Mato Grosso, Brazil --------
## The bundled maize datasets are multi-environment trials from Mato Grosso:
## maizeWTH (daily weather), maizeYield (grain yield) and maizeG (genomic
## relationships). We train on them, then predict the trained genotypes at
## 50 candidate sites spread across the state. Needs BGGE (fitting) and a
## network connection (get_weather() queries NASA POWER).
data("maizeWTH"); data("maizeYield"); data("maizeG")

## 1. Training environmental covariates (first 100 days after sowing)
vars <- c("FRUE", "PETP", "GDD", "T2M_MAX")
Wtr  <- W_matrix(env.data = maizeWTH[maizeWTH$daysFromStart < 100, ],
                 var.id = vars, statistic = "mean")

## 2. Kernels + a fit that KEEPS the effect chains (required by any scan)
K   <- get_kernel(K_G = list(G = maizeG),
                  K_E = list(W = env_kernel(env.data = Wtr)[[2]]),
                  data = maizeYield, model = "RNMM",
                  env = "env", gid = "gid", y = "value")
fit <- kernel_model(y = "value", data = maizeYield, random = K,
                    env = "env", gid = "gid",
                    iterations = 5000, burnin = 1000, keep_effects = TRUE)

## 3. Fifty candidate locations across Mato Grosso (WGS84 decimal degrees).
##    The state spans roughly 7.3-18.0 S and 50.2-61.6 W.
set.seed(2024)
coords <- data.frame(
  env = sprintf("MT%02d", 1:50),
  lon = runif(50, -61.6, -50.2),
  lat = runif(50, -18.0,  -7.3))

## 4. Daily weather for each site over one summer season, processed into the
##    SAME covariates as the training matrix.
wth_new <- get_weather(env.id = coords$env, lat = coords$lat, lon = coords$lon,
                       start.day = "2023-11-01", end.day = "2024-02-15",
                       variables.names = c("T2M", "T2M_MAX", "T2M_MIN",
                                           "T2MDEW", "PRECTOT", "RH2M",
                                           "ALLSKY_SFC_SW_DWN"))
wth_new <- processWTH(wth_new)          # derives FRUE, PETP, GDD, ...
Wnew <- W_matrix(env.data = wth_new[wth_new$daysFromStart < 100, ],
                 var.id = vars, statistic = "mean")

## 5. Predict the trained genotypes at all 50 sites
sc <- scan_untested_envs(fit, W_train = Wtr, W_new = Wnew, K_G = maizeG)

## 6. Coverage first: where does the trial network actually support prediction?
map_scan(sc, coords, type = "coverage")

## Predicted performance of one genotype across Mato Grosso
g1 <- rownames(maizeG)[1]
map_scan(sc, coords, type = "performance", genotypes = g1)

## Mean predicted performance over the first ten genotypes
map_scan(sc, coords, type = "performance",
```

```
         genotypes = rownames(maizeG)[1:10], summarise = "mean")

## 7. Overlay the Mato Grosso state border for context. geobr serves the
##    official IBGE boundaries for Brazil as sf objects; clip = TRUE also
##    drops any scanned site that falls outside the polygon.
if (requireNamespace("geobr", quietly = TRUE) &&
    requireNamespace("sf", quietly = TRUE)) {
mt <- geobr::read_state(code_state = "MT", year = 2020)
map_scan(sc, coords, type = "coverage", shp = mt)
map_scan(sc, coords, type = "performance", genotypes = g1,
         shp = mt, clip = TRUE)
}
```

`param_atmospheric` *Estimate Atmospheric Parameters Related to Evapotranspiration*

Description

Computes vapour pressure deficit, the slope of the saturation vapour pressure curve, Priestley-Taylor potential evapotranspiration and the precipitation balance, following FAO-56.

Usage

```
param_atmospheric(
  env.data,
  PREC = NULL,
  Tdew = NULL,
  Tmin = NULL,
  Tmax = NULL,
  RH = NULL,
  Rad = NULL,
  G = 0,
  Alt = 600,
  alpha = 1.26,
  merge = FALSE,
  verbose = TRUE
)
```

Arguments

`env.data`	data.frame. A `get_weather` output or equivalent.
`PREC`, `Tdew`, `Tmin`, `Tmax`, `RH`, `Rad`	character. Column names for precipitation, dew point, minimum and maximum temperature, relative humidity and solar radiation. Defaults follow NASA POWER naming.
`G`	numeric. Soil heat flux (MJ/m2/day). Default 0.

`Alt`	numeric scalar, or character naming an elevation column in `env.data`. Default 600 m.
`alpha`	numeric. Priestley-Taylor coefficient. Default 1.26.
`merge`	boolean. Bind results onto `env.data`.
`verbose`	boolean. Print a summary of what was computed.

Details

Elevation now works. `Alt` accepts either a numeric elevation or the name of a column. Previously the default `Alt = 600` was never `NULL`, so the elevation column was never read and every site was treated as 600 m – a 26% error in atmospheric pressure at 2500 m, propagating into the psychrometric constant and ETP.

`VPD` is the difference between mean saturation vapour pressure (from Tmin and Tmax) and actual vapour pressure (from dew point). Values are floored at 0; negatives can only arise from a dew point above air temperature, which is unphysical and indicates a data problem. A warning is raised when this occurs.

Value

A data.frame with `VPD` (kPa), `SPV` (kPa/C), `ETP` (mm/day) and `PETP` (mm/day).

Author(s)

Germano Costa Neto

See Also

`param_radiation`, `param_temperature`, `processWTH`

Examples

```
env.data <- get_weather(lat = -13.05, lon = -56.05)
param_atmospheric(env.data, Alt = 850)
```

`param_radiation` *Estimate Basic Solar-Radiation Parameters*

Description

Computes extraterrestrial radiation, daylength and actual sunshine duration from day of year and latitude, following FAO-56.

Usage

```
param_radiation(
  env.data,
  day.id = "DOY",
  latitude = "LAT",
  solarRadiation = NULL,
  merge = TRUE,
  verbose = TRUE
)
```

Arguments

`env.data`	data.frame. A `get_weather` output or equivalent.
`day.id`	character. Column holding day of year. Default `"DOY"`.
`latitude`	character. Column holding latitude. Default `"LAT"`.
`solarRadiation`	character. Column holding incoming shortwave radiation. Default `"ALLSKY_SFC_SW_DWN"`.
`merge`	boolean. Bind results onto `env.data`. Default `TRUE`.
`verbose`	boolean. Print a summary of what was computed.

Details

Daylength uses Forsythe et al. (1995), consistent with `get_weather`. The two functions previously used different formulas that disagreed by up to 0.5 h at high latitude.

The sunset hour angle is clamped so polar day/night yields 0 rather than `NaN`. Missing radiation is median-imputed within the supplied data, and `n` is bounded to `[0, N]`.

Value

A data.frame with `n` (actual sunshine hours), `N` (daylength, h) and `RTA` (extraterrestrial radiation, MJ/m2/day).

Author(s)

Germano Costa Neto

See Also

`param_atmospheric`, `param_temperature`, `processWTH`

Examples

```
env.data <- get_weather(lat = -13.05, lon = -56.05)
param_radiation(env.data)
```

param_temperature *Estimate Thermal Parameters*

Description

Computes growing degree days, the temperature-limited radiation use efficiency factor, and daily temperature range.

Usage

```
param_temperature(
  env.data,
  Tmax = NULL,
  Tmin = NULL,
  Tbase1 = 9,
  Tbase2 = 45,
  Topt1 = 26,
  Topt2 = 32,
  method = c("standard", "legacy"),
  merge = FALSE,
  verbose = TRUE
)
```

Arguments

`env.data`	data.frame. A `get_weather` output or equivalent.
`Tmax`, `Tmin`	character. Column names for maximum and minimum air temperature (Celsius).
`Tbase1`	numeric. Base temperature below which development stops.
`Tbase2`	numeric. Upper temperature at which development stops.
`Topt1`, `Topt2`	numeric. Lower and upper bounds of the optimal range.
`method`	character. GDD formulation: `"standard"` (default) or `"legacy"`. See Details.
`merge`	boolean. Bind results onto `env.data`.
`verbose`	boolean. Print a summary of what was computed.

Details

GDD methods.

`"standard"` $GDD = \max(\frac{\min(T_{max}, T_{base2}) + T_{min}}{2} - T_{base1},\ 0)$. Tmax is capped at the upper threshold; the result is floored at zero. This is the conventional agronomic definition.

`"legacy"` Clamps *both* Tmax and Tmin into $[T_{base1}, T_{base2}]$ before averaging. This is the behaviour of earlier EnvRtype versions and corresponds to the "Method 2" variant. It credits heat units on days colder than T_{base1} – e.g. with Tmax 12, Tmin 5, Tbase1 9 it returns 1.5 where the standard method returns 0.

The default changed from legacy to standard; results will differ on cold days. Pass `method = "legacy"` to reproduce earlier output.

`FRUE` rises linearly from 0 at `Tbase1` to 1 at `Topt1`, holds at 1 across the optimum, and falls to 0 at `Tbase2`.

Value

A data.frame with `GDD` (C/day), `FRUE` (0-1) and `T2M_RANGE` (C).

Author(s)

Germano Costa Neto

See Also

`param_radiation`, `param_atmospheric`, `processWTH`

Examples

```
env.data <- get_weather(lat = -13.05, lon = -56.05)
param_temperature(env.data)
param_temperature(env.data, method = "legacy")  # pre-refactor behaviour
```

---

phenology_templates *List Built-in Phenology Templates*

Description

`phenology_templates()` returns the built-in crop templates as a data.frame; `show_phenology()` prints them, optionally for one crop with its stage table, provenance and caveats.

Every template is a published tabulation for a REFERENCE cultivar. They give a consistent relative scale across sites – which is what fixes the calendar-day misalignment in `W_matrix()` – but they are not calibrated predictions for your germplasm. Maturity group alone shifts maize R6 by 300-500 C d. Validate against observed flowering dates before treating stage labels as phenology.

Usage

```
phenology_templates()

show_phenology(crop = NULL)
```

Arguments

`crop`	character. Optional single crop; if omitted, all are listed.

Value

`phenology_templates()` a data.frame, one row per crop. `show_phenology()` returns its input invisibly and is called for the side effect of printing.

Examples

```
phenology_templates()
show_phenology("safflower")
```

---

planting_window_table *Tabulate scan predictions against planting date*

Description

Reshapes a ‘scan_untested_envs()‘ result into one row per genotype x planting date, carrying the credible bounds and the envelope position. Useful on its own when you want the numbers rather than a plot.

Usage

```
planting_window_table(
  scan,
  dates,
  date_col = "date",
  genotypes = NULL,
  group = NULL
)
```

Arguments

`scan`	a ‘scan_untested_envs‘ object whose new "environments" are candidate planting dates.
`dates`	either (a) a data.frame with a column ‘env‘ matching the scan’s new-environment names and a column giving the date, or (b) a vector of dates in the same order as the scan’s new environments.
`date_col`	name of the date column when ‘dates‘ is a data.frame.
`genotypes`	optional character vector to subset genotypes.
`group`	optional named vector or data.frame (‘gid‘, ‘group‘) assigning genotypes to groups, for group-mean curves.

Value

data.frame with ‘gid‘, ‘env‘, ‘date‘, ‘yHat‘, ‘lower‘, ‘upper‘, ‘ci_width‘, ‘position‘, ‘mahalanobis_ratio‘, ‘prop_outside‘, ‘weight_negativity‘, and ‘group‘ when supplied.

plot.sim_met *Diagnostic Plot for a Simulated MET*

Description

Scatter of the target against the realised (K-whitened) off-diagonal environment correlations, with the 1:1 line.

Usage

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

Arguments

x	a sim_met object.
...	passed to plot.

Value

x, invisibly.

See Also

sim_met

plot.sim_W *Diagnostic Plot for a Simulated Envirome*

Description

Two panels: the realised kernel off-diagonals against the target environment correlation, and the scree of per-PC variance shares.

Usage

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

Arguments

x	a sim_W object.
...	passed to plot.

Value

x, invisibly.

See Also

sim_W

---

plot_planting_window *Plot genotype performance across candidate planting dates*

Description

Curves of predicted performance against sowing date, with credible ribbons and out-of-envelope dates marked. Dates classified ‘gross_extrapolation‘ are desaturated and shaded, and counted in the subtitle, because position is reported but must not be read as a reliability guarantee.

Usage

```
plot_planting_window(
  scan,
  dates = NULL,
  date_col = "date",
  genotypes = NULL,
  group = NULL,
  summarise = c("genotype", "group", "overall"),
  ribbon = TRUE,
  ribbon_max = 6,
  mark_envelope = TRUE,
  envelope_level = c("gross", "mild"),
  engine = c("ggplot2", "base"),
  ylab = "Predicted performance",
  title = NULL
)
```

Arguments

scan	a ‘scan_untested_envs‘ object, or a table from [planting_window_table()].
dates, date_col, genotypes, group	passed to [planting_window_table()] when ‘scan‘ is a scan object.
summarise	one of ‘"genotype"‘ (one curve per genotype), ‘"group"‘ (mean curve per group, requires ‘group‘), or ‘"overall"‘ (single mean curve across all selected genotypes).
ribbon	draw credible ribbons. Switched off automatically when more than ‘ribbon_max‘ curves are drawn, since overlapping ribbons are unreadable.
ribbon_max	curve count above which ribbons are suppressed.

`mark_envelope`	shade and desaturate dates outside the envelope.
`envelope_level`	which positions to mark: ‘"gross"‘ (default) or ‘"mild"‘ (also marks ‘mild_extrapolation‘).
`engine`	‘"ggplot2"‘ (default) or ‘"base"‘.
`ylab`, `title`	axis label and title.

Value

a ggplot object (invisibly for ‘engine = "base"‘).

`print.kernel_set` *Print a kernel set*

Description

Print a kernel set

Usage

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

Arguments

`x`	object of class "`kernel_set`" from `get_kernel`.
`...`	ignored.

Value

Returns `x` invisibly. Called for its side effect of printing a one-line header and a per-kernel table (kernel name, retained rank, mean diagonal and rank as a percentage of the number of observations) to the console.

`print.scan_untested_envs` *Print a Scan of Untested Environments*

Description

Compact console summary of a `scan_untested_envs` result: the per-site prediction and its extrapolation diagnostics, a tally of position labels, the covariates that fell outside the training range, and a reminder of how the geometric measures relate to calibrated accuracy.

Usage

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

Arguments

x	an object of class "scan_untested_envs".
...	ignored; present for S3 compatibility.

Value

x, invisibly.

See Also

scan_untested_envs

---

print.sim_envirome *Print a Simulated Envirome Diversity Matrix*

Description

Print a Simulated Envirome Diversity Matrix

Usage

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

Arguments

x	a sim_envirome object.
...	ignored.

Value

x, invisibly.

See Also

sim_envirome

print.sim_markers *Print a Simulated Marker Matrix*

Description

Print a Simulated Marker Matrix

Usage

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

Arguments

x	a sim_markers object.
...	ignored.

Value

x, invisibly.

See Also

sim_markers

print.sim_met *Print a Simulated Multi-Environment Trial*

Description

Print a Simulated Multi-Environment Trial

Usage

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

Arguments

x	a sim_met object.
...	ignored.

Value

x, invisibly.

See Also

sim_met

---

print.sim_W — *Print a Simulated Envirome*

Description

Print a Simulated Envirome

Usage

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

Arguments

x	a sim_W object.
...	ignored.

Value

x, invisibly.

See Also

sim_W

---

print.soil_gmm — *Print a soil_gmm object*

Description

Compact summary of a fitted soil mixture model: the selected model and cluster count, the BIC gap to the runner-up, cluster sizes, the number of uncertain sites, and the per-cluster profile means in original units.

The BIC gap is reported because a point estimate of the zone count is misleading on its own. Differences are graded weak (< 2), positive (2-6), strong (6-10) and very strong (> 10). A weak gap means several partitions fit about equally well.

Usage

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

Arguments

x	an object of class soil_gmm.
...	ignored.

Value

x, invisibly.

Examples

```
## Not run:
## Requires a soil_grid data set, as in the soil_classification() example.
fit <- soil_classification(soil_grid, risk.vars = "soc")
print(fit)

## End(Not run)
```

---

print.T_matrix *Print an Envirotype (T) Matrix*

Description

Print an Envirotype (T) Matrix

Usage

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

Arguments

x	a T_matrix object.
...	ignored.

Value

x, invisibly.

See Also

T_matrix

processWTH *Enrich a get_weather() Output with Derived Parameters*

Description

Convenience wrapper chaining param_radiation, param_atmospheric and param_temperature.

Usage

```
processWTH(
  env.data,
  PREC = NULL,
  Tdew = NULL,
  Tmin = NULL,
  Tmax = NULL,
  RH = NULL,
  Rad = NULL,
  G = 0,
  Alt = 600,
  Tbase1 = 9,
  Tbase2 = 45,
  Topt1 = 26,
  Topt2 = 32,
  method = c("standard", "legacy"),
  alpha = 1.26,
  verbose = TRUE
)
```

Arguments

env.data	data.frame. A get_weather output.
PREC, Tdew, Tmin, Tmax, RH, Rad	character. Column names passed through to the underlying functions.
G	numeric. Soil heat flux. Default 0.
Alt	numeric or character. Elevation in metres, or the name of an elevation column.
Tbase1, Tbase2, Topt1, Topt2	numeric. Cardinal temperatures.
method	character. GDD method, see param_temperature.
alpha	numeric. Priestley-Taylor coefficient.
verbose	boolean. Print progress messages.

Details

If env.data already carries n, N or RTA from get_weather, those columns are dropped before recomputation to avoid duplicated names. Note that get_weather() derives n from ALLSKY_SFC_SW_DNI whereas param_radiation() uses ALLSKY_SFC_SW_DWN, so the recomputed values may differ.

Value

The input data.frame with radiation, atmospheric and thermal parameters appended.

Author(s)

Germano Costa Neto

See Also

`get_weather`, `summaryWTH`

Examples

```
env.data <- get_weather(lat = -13.05, lon = -56.05)
processWTH(env.data)
```

---

project_risk — *Compare Envirotype Risk Between Baseline and a Climate Scenario*

Description

Runs `env_risk_profile()` on an observed series and on a delta-shifted scenario series, and returns the change in stress frequency per site, stage and class.

Usage

```
project_risk(baseline, scenario, ..., verbose = TRUE)
```

Arguments

`baseline`	data.frame. Observed multi-year daily weather.
`scenario`	data.frame. Output of `get_climate_scenario()` built from the same baseline, or any comparably structured series.
`...`	Arguments passed to `env_risk_profile()` for BOTH runs, so the two profiles are strictly comparable.
`verbose`	boolean. Print progress.

Details

Because `get_climate_scenario(method = "delta")` preserves the observed day-to-day sequence, differences here reflect the MEAN shift only. Rising frequencies of heat classes are real but conservative; unchanged precipitation-class frequencies do not mean rainfall risk is unchanged, only that the mean-preserving perturbation left wet-day counts alone.

Value

A list of class "env_risk_delta" with baseline, scenario (both "env_risk" objects) and change, a data.frame with freq_base, freq_scen, freq_diff and days_diff.

See Also

env_risk_profile, get_climate_scenario

Examples

```
## Not run:
## Requires site coordinates and an observed weather series (wth).
fut <- get_climate_scenario(env.id = sites$env, lat = sites$lat,
                            lon = sites$lon, baseline = wth,
                            scenario = "ssp585", period = "2061-2080")

chg <- project_risk(baseline = processWTH(wth),
                    scenario = processWTH(fut),
                    planting = "10-15", crop = "maize",
                    var.id = c("T2M_MAX", "PETP"))
subset(chg$change, class == "heat_severe")

## End(Not run)
```

---

read_progress_log *Inspect the Progress Log of a Checkpointed Run*

Description

Returns the per-environment retrieval history written by get_weather_resumable or get_soil_resumable: what succeeded, what failed, when, and with which error message.

Usage

```
read_progress_log(run.id, dir.path = NULL, all = FALSE, verbose = TRUE)
```

Arguments

run.id	character. Run directory name.
dir.path	character. Parent directory. Default getwd().
all	logical. If TRUE, return every record including superseded retries. If FALSE (default), the latest record per environment.
verbose	boolean. If TRUE (default) prints a progress banner.

Value

A data.frame with env, key, status ("ok" or "failed"), time and message.

See Also

restart_from_log

Examples

```
## Not run:
## Requires a saved run created by get_soil_resumable().
lg <- read_progress_log(run.id = "zoning_grid")
table(lg$status)

## Why did they fail?
subset(lg, status == "failed")[, c("env", "message")]

## Full history including retries
read_progress_log(run.id = "zoning_grid", all = TRUE)

## End(Not run)
```

---

restart_from_log *Resume an Interrupted get_weather() or get_soil() Run*

Description

Reads the manifest and progress log written by `get_weather_resumable` or `get_soil_resumable`, skips every environment already retrieved, fetches those still missing or failed, and returns the assembled result.

Safe to call repeatedly. If everything already succeeded it fetches nothing and simply assembles what is on disk.

Usage

```
restart_from_log(
  run.id,
  dir.path = NULL,
  retry.failed = TRUE,
  sleep = NULL,
  tries = NULL,
  verbose = TRUE
)
```

Arguments

`run.id`	character. The run directory name used originally.
`dir.path`	character. Parent directory. Default `getwd()`.
`retry.failed`	logical. Retry environments previously logged as failed. Default `TRUE`. Set `FALSE` to assemble only the successes – useful when failures are permanent, such as coordinates in the ocean.

sleep	numeric or NULL. Override the stored delay. NULL (default) reuses the manifest value.
tries	integer or NULL. Override the stored retry count.
verbose	logical. Print progress. Default TRUE.

Details

All retrieval parameters come from manifest.json, so a resumed run necessarily uses the same coordinates, dates and variables as the original. You cannot accidentally resume with different settings – that is what the stored fingerprint prevents.

Permanently failing environments. Some failures never resolve: a point in the ocean has no SoilGrids profile. Repeated calls will retry them every time. Once you have confirmed a failure is permanent, pass retry.failed = FALSE.

Value

A data.frame as returned by the original function, with attributes run.dir, n_ok, n_failed and failed.

See Also

get_weather_resumable, get_soil_resumable, read_progress_log

Examples

```
## Not run:
## A long soil job dies at environment 137 of 200.
## In a brand-new session -- no need to remember the arguments:
soil <- restart_from_log(run.id = "zoning_grid")

## Inspect what failed and why
lg <- read_progress_log(run.id = "zoning_grid")
subset(lg, status == "failed")[, c("env", "message")]

## Give up on permanent failures
soil <- restart_from_log(run.id = "zoning_grid", retry.failed = FALSE)

## Weather resumes identically
wth <- restart_from_log(run.id = "trial2024")

## End(Not run)
```

scan_spatial_table *Join scan predictions to coordinates*

Description

Join scan predictions to coordinates

Usage

```
scan_spatial_table(
  scan,
  coords,
  lon_col = "lon",
  lat_col = "lat",
  date_col = NULL,
  genotypes = NULL
)
```

Arguments

scan	a ‘scan_untested_envs‘ (or [grid_scan()]) result.
coords	data.frame with ‘env‘ matching the scan’s new environments, plus longitude and latitude columns.
lon_col, lat_col	coordinate column names.
date_col	optional column giving a planting date per site, for (C).
genotypes	optional subset.

Value

long data.frame: ‘gid‘, ‘env‘, ‘lon‘, ‘lat‘, [‘date‘], ‘yHat‘, ‘lower‘, ‘upper‘, ‘position‘, ‘mahalanobis_ratio‘, ‘prop_outside‘, ‘weight_negativity‘, ‘prob_win‘ (posterior probability the genotype is the best at that site; ‘NA‘ for scans that do not carry it).

scan_untested_envs *Predict Trained Genotypes at Untested Environments Without Refitting*

Description

Transfers a fitted `kernel_model` to environments that were never planted – new locations, new planting dates, or both – using only their environmental covariates. No re-fitting and no phenotypes at the new sites are required.

The environmental main effect estimated at the training sites is *kriged* onto each new site through a linear environmental kernel, and the whole posterior is carried draw-by-draw so that credible intervals, win probabilities and an extrapolation diagnostic are all internally consistent.

The transfer. Covariates are standardised on the training moments,

$$\tilde{W} = (W - \bar{W}_{tr}) \;/\; s_{tr},$$

and a linear (product) environmental kernel is formed from the p covariates,

$$K_{tt} = \tilde{W}_{tr}\tilde{W}_{tr}^{\top}/p, \quad K_{nt} = \tilde{W}_{n}\tilde{W}_{tr}^{\top}/p, \quad K_{nn} = \tilde{W}_{n}\tilde{W}_{n}^{\top}/p.$$

The new-site environmental effects are a kriging (Gaussian-process conditional-mean) map of the training effects,

$$H = K_{nt}\,(K_{tt} + \lambda I)^{-1}, \qquad \hat{e}_{n}^{(d)} = H\, e_{tr}^{(d)},$$

applied to every retained MCMC draw d. When a GxE kernel is present its interaction effects are kriged the same way. The ridge λ stabilises the inverse; H is *not* constrained to be positive, and large negative row weights are the signal that a site is being reached by extrapolation (see `weight_negativity` below).

The prediction. For draw d, genotype g and new site j,

$$\hat{y}_{gj}^{(d)} = \mu + u_{g}^{(d)} + \hat{e}_{j}^{(d)} + \widehat{(ge)}_{gj}^{(d)},$$

with μ the training grand mean. Point predictions are posterior means over draws and the credible interval uses the empirical quantiles at $\alpha = (1-\texttt{level})/2$ and $1-\alpha$. The probability that a genotype is the best at a site is the fraction of draws in which it attains the row maximum, so it respects the genotype correlation *within* a draw. Per-site genomic heritability is reported as $h^2 = \sigma_g^2/(\sigma_g^2 + \sigma_e^2)$.

Usage

```
scan_untested_envs(
  object,
  W_train,
  W_new,
  K_G = NULL,
  standardize = TRUE,
  lambda = 1e-06,
  level = 0.95,
  max_draws = 500,
  envelope = c("keep", "flag", "mask", "drop"),
  envelope_level = c("gross", "mild")
)
```

Arguments

`object`	a fitted model from `kernel_model`, fitted with `keep_effects = TRUE`. The per-draw effect chains (not posterior means) are required; without them the intervals would be fabricated.
`W_train`	training environmental covariate matrix, environments $\times$ covariates ($e \times p$). If it has no `rownames` they are taken to be the sorted training environments.
`W_new`	new-environment covariate matrix, sites $\times$ covariates ($m \times p$). Must have the *same columns* as `W_train`. Unnamed rows are labelled `NewEnv1..NewEnvm`.
`K_G`	optional genomic relationship matrix. Used only to inform the GxE transfer; when omitted, GxE is kriged from environmental similarity alone (a warning is issued).
`standardize`	logical. Standardise covariates on the training mean and SD before building the kernels. Default `TRUE` and strongly advised when covariates are on different scales.
`lambda`	numeric ridge added to K_{tt} before inversion, as a fraction of its mean diagonal. Default `1e-6`. Raise it if the kernel is near-singular (highly collinear covariates).
`level`	credible-interval mass. Default `0.95`.
`max_draws`	integer. Cap on the number of MCMC draws used; draws are thinned by even spacing when the chain is longer. Default `500`.
`envelope`	what to do about sites that fall outside the training covariate envelope: `"keep"` (default, return everything unchanged), `"flag"` (return everything, warn), `"mask"` (set flagged predictions to `NA` but keep the matrix structure so indexing by environment name still resolves), or `"drop"` (remove flagged rows/columns and record them in `$excluded`).
`envelope_level`	which sites count as flagged: `"gross"` (default) or `"mild"`. Only `"gross"` separates cleanly in calibration; `"mild"` acts on a weak signal and is not recommended as a gate.

Details

Extrapolation diagnostics. The scan reports *where* each new site sits relative to the training covariate cloud. Several complementary measures are returned because no single one is sufficient:

- *Cosine similarity* to the nearest training site, $s_{ij} = K_{nt,ij}/\sqrt{K_{nn,ii}\,K_{tt,jj}}$, summarised as `max_similarity` and `mean_similarity`.
- A *Mahalanobis-type distance* in the leading principal-component subspace (enough PCs to reach 95% variance), $d_i = \sqrt{\sum_k z_{ik}^2}$, divided by the 95th percentile of the training distances to give `mahalanobis_ratio`.
- A *marginal range test* per covariate (`prop_outside`, `max_exceedance_sd`). This is marginal: a site inside every covariate's range individually can still occupy a *combination* that never occurred, which only the distance and weight measures catch.
- *Kriging-weight geometry*: `weight_negativity` $= \sum_j \min(H_{ij}, 0)$ and the effective number of training sites leaned on, $n_{\mathrm{eff},i} = (\sum_j |H_{ij}|)^2 / \sum_j H_{ij}^2$.
- A combined `divergence_index` $= \min\big(1, \max(0,\ 0.5(1-s_{\max})+0.5\,t)\big)$ with $t = \mathrm{clamp}((\texttt{mahalanobis_ratio}-0.5)/2,\ 0,\ 1)$.

Sites are labelled "interpolation", "edge", "mild_extrapolation" or "gross_extrapolation" from these.

Position is geometry, not a correctness guarantee. Calibration against 60 held-out environments found the envelope measures correlate only $r \approx -0.37$ to -0.66 with realised accuracy. Treat the position label as a description of where a site lies, never as a verdict that the prediction is right or wrong. The model-internal ci_width_ratio was the strongest single accuracy correlate ($r = -0.66$); to know your true accuracy, run leave-environments-out cross-validation on your own data.

Value

An object of class "scan_untested_envs": a list with

yHat_matrix, yHat_lower, yHat_upper genotype $\times$ new-site matrices of the posterior mean and credible bounds.

yHat the same in long form, with ci_width.

yHat_prob_win probability each genotype is best at each site.

genetic_var per-site genetic variance, its interval and genomic_h2.

vcov, cor between-new-site genetic covariance and correlation matrices.

interpolation, divergence, diagnostics the extrapolation measures described above, one row per new site.

exceedance_detail one row per (site $\times$ offending covariate) that fell outside the training range.

excluded sites masked or dropped by envelope, else NULL.

meta dimensions, settings, similarity and kriging weights.

References

Costa-Neto, G., Fritsche-Neto, R. & Crossa, J. (2021) Nonlinear kernels, dominance, and envirotyping data increase the accuracy of genome-based prediction in multi-environment trials. *Heredity* 126, 92-106.

Jarquin, D. et al. (2014) A reaction norm model for genomic selection using high-dimensional genomic and environmental data. *Theoretical and Applied Genetics* 127, 595-607.

Cressie, N. (1993) *Statistics for Spatial Data*. Wiley. (kriging as a Gaussian-process conditional mean.)

See Also

kernel_model, get_kernel, grid_scan, map_scan, plot_planting_window, best_planting_date

Examples

```
data("maizeWTH"); data("maizeYield"); data("maizeG")

## Training covariates (first 100 days after sowing)
vars <- c("FRUE", "PETP", "GDD", "T2M_MAX")
Wtr  <- W_matrix(env.data = maizeWTH[maizeWTH$daysFromStart < 100, ],
                 var.id = vars, statistic = "mean")
```

```
## Fit KEEPING the effect chains -- required by the scan
K   <- get_kernel(K_G = list(G = maizeG),
                  K_E = list(W = env_kernel(env.data = Wtr)[[2]]),
                  data = maizeYield, model = "RNMM",
                  env = "env", gid = "gid", y = "value")
fit <- kernel_model(y = "value", data = maizeYield, random = K,
                    env = "env", gid = "gid", keep_effects = TRUE)

## Predict the trained genotypes at new sites (reuse the training W here)
sc <- scan_untested_envs(fit, W_train = Wtr, W_new = Wtr, K_G = maizeG)
sc                                  # print method: per-site diagnostics
head(sc$yHat)                       # predictions + credible intervals
sc$interpolation[, c("env", "position", "mahalanobis_ratio")]
```

---

sim_envirome *Simulate an Envirome with a Controllable Environmental Diversity*

Description

Generates an environmental covariable matrix (environments in rows, covariables in columns) whose diversity among environments is set by explicit knobs: the structure (unstructured, mega-environment clusters, or a continuous gradient), the redundancy among covariables and their marginal shape. The induced environment correlation and environmental kernel are attached, so the result feeds sim_met (as W or C) and get_kernel directly. It is the environmental analogue of sim_markers.

Usage

```
sim_envirome(
  n_env,
  n_var = 20L,
  structure = c("random", "clusters", "gradient"),
  n_cluster = 3L,
  between.cor = 0.2,
  spread = 1,
  redundancy = 0,
  n_blocks = 1L,
  marginal = c("gaussian", "right-skewed", "heavy-tailed", "bounded"),
  env_names = NULL,
  var_names = NULL,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`n_env`	integer. Number of environments (rows, >= 2).
`n_var`	integer. Number of covariables (columns, >= 2).
`structure`	character. `"random"` (default), `"clusters"` or `"gradient"`.
`n_cluster`	integer. Number of mega-environments when `structure = "clusters"`.
`between.cor`	numeric in [0, 1). Similarity between clusters (0 = well separated).
`spread`	numeric. Within-cluster / along-gradient dispersion; larger values make environments within a group more diverse.
`redundancy`	numeric in [0, 1). Share of covariables collapsed onto drivers (collinearity), lowering effective rank.
`n_blocks`	integer. Number of covariable drivers when `redundancy > 0`.
`marginal`	character. Marginal of the covariables: `"gaussian"` (default), `"right-skewed"`, `"heavy-tailed"` or `"bounded"`.
`env_names, var_names`	character. Optional names for rows / columns.
`seed`	integer. RNG seed. The caller's RNG stream is restored on exit.
`verbose`	logical. Print a diversity summary. Default `TRUE`.

Details

Each environment gets a latent profile that drives its covariables; the `structure` controls how those profiles are arranged. `"random"` places environments independently (maximal diversity), `"clusters"` groups them into `n_cluster` mega-environments (high within-cluster similarity, between-cluster similarity set by `between.cor`), and `"gradient"` orders environments along a latent continuum (neighbours similar, extremes different), as for a latitude or season gradient. `redundancy` collapses a share of covariables onto `n_blocks` drivers, lowering the effective rank, and `marginal` reshapes the covariables away from Gaussian.

Value

A matrix of class `"sim_envirome"` (environments x covariables). Attributes: `"cluster"` (mega-environment of each environment, when applicable), `"C_env"` (the induced environment correlation), `"K_E"` (the environmental kernel) and `"diversity"` (`mean_similarity` off the environment correlation and `eff_dim`, the effective dimension of the environmental kernel).

See Also

`sim_W`, `sim_markers`, `sim_met`, `env_kernel`

Examples

```
## Not run:
## High diversity: unstructured environments
E_hi <- sim_envirome(20, n_var = 30, structure = "random", seed = 1)

## Mega-environments: three tight clusters
```

```
E_cl <- sim_envirome(20, n_var = 30, structure = "clusters",
                     n_cluster = 3, between.cor = 0.1, seed = 1)
attr(E_cl, "diversity")

## A latitudinal / seasonal gradient
E_gr <- sim_envirome(20, n_var = 30, structure = "gradient", seed = 1)

## Feed into sim_met as the envirome (reaction norm) or as a target C
met <- sim_met(markers = sim_markers(100, 500, seed = 1),
               W = as.matrix(E_cl), seed = 1)
KE  <- list(W = attr(E_cl, "K_E"))

## Contrast G x E under low vs high environmental diversity (genetics fixed)
X <- sim_markers(150, 2000, seed = 1)
g_hi <- cor(sim_met(markers = X, C = attr(E_hi, "C_env"), seed = 1)$truth$g)
g_cl <- cor(sim_met(markers = X, C = attr(E_cl, "C_env"), seed = 1)$truth$g)
c(diverse = mean(g_hi[lower.tri(g_hi)]),   # low genetic cor -> more G x E
  clustered = mean(g_cl[lower.tri(g_cl)]))

## End(Not run)
```

---

sim_markers *Simulate a Marker Matrix with a Controllable Genetic Diversity*

Description

Generates a bi-allelic SNP dosage matrix (lines in rows, markers in columns) whose genetic diversity is set by explicit knobs: the minor-allele-frequency spectrum, population structure (F_{ST} across subpopulations) and linkage disequilibrium (redundant marker blocks). The VanRaden genomic relationship matrix is attached, so the result feeds `sim_met` (as `markers` or via the kinship) and `get_kernel` directly.

Usage

```
sim_markers(
  n_lines,
  n_marker = 1000L,
  maf = c("uniform", "rare", "common"),
  n_subpop = 1L,
  Fst = 0,
  ld = 0,
  ld_blocks = 1L,
  line_names = NULL,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`n_lines`	integer. Number of lines (rows, >= 2).
`n_marker`	integer. Number of markers (columns).
`maf`	character. Minor-allele-frequency spectrum: `"uniform"` (default), `"rare"` (U-shaped, many low-frequency variants) or `"common"` (bell-shaped, mostly intermediate).
`n_subpop`	integer. Number of subpopulations. Lines are assigned in round-robin order.
`Fst`	numeric in [0, 1). Between-subpopulation differentiation. `0` gives a single panmictic population.
`ld`	numeric in [0, 1). Fraction of each block's markers forced to equal the block lead marker, creating linkage disequilibrium / redundancy.
`ld_blocks`	integer. Number of linkage blocks when `ld` > `0`.
`line_names`	character. Optional line names (default `G1..Gn`).
`seed`	integer. RNG seed. The caller's RNG stream is restored on exit.
`verbose`	logical. Print a diversity summary. Default `TRUE`.

Details

Allele frequencies are drawn per marker and genotypes as $x_{ij} \sim \mathrm{Binomial}(2, p_j)$. Population structure follows the Balding–Nichols model: within subpopulation s the frequency drifts from the ancestral p_j as $p_{sj} \sim \mathrm{Beta}\big(p_j \frac{1-F_{ST}}{F_{ST}}, (1-p_j)\frac{1-F_{ST}}{F_{ST}}\big)$, so larger F_{ST} means more between-group differentiation and lower within-group diversity. Linkage disequilibrium is emulated by copying each block's lead marker into a fraction `ld` of the other markers in the block, lowering the effective rank of the kinship.

Value

An integer matrix of class `"sim_markers"` (lines x markers). Attributes: `"freq"` (per-marker allele frequency), `"subpop"` (subpopulation of each line), `"G"` (the VanRaden kinship, positive definite) and `"diversity"` (`He` expected heterozygosity, `mean_relatedness` and `eff_dim`, the effective dimension of the kinship).

See Also

`sim_met`, `sim_envirome`, `get_kernel`

Examples

```
## Not run:
## High diversity: many common markers, one panmictic population
X_hi <- sim_markers(150, 2000, maf = "common", seed = 1)

## Low diversity: fewer, rarer markers
X_lo <- sim_markers(150, 300, maf = "rare", seed = 1)

## Structured: four subpopulations with strong drift
X_st <- sim_markers(150, 2000, n_subpop = 4, Fst = 0.15, seed = 1)
```

```
attr(X_st, "diversity")

## Feed straight into sim_met (markers mechanism) or via the kinship
met <- sim_met(markers = X_st, n_env = 10, n.qtl = 50, seed = 1)
KG  <- list(G = attr(X_st, "G"))

## Compare phenotype variance under low vs high genetic diversity
E <- sim_envirome(10, 30, structure = "clusters", seed = 1)
v_lo <- var(sim_met(markers = X_lo, C = attr(E, "C_env"), seed = 1)$data$value)
v_hi <- var(sim_met(markers = X_hi, C = attr(E, "C_env"), seed = 1)$data$value)
c(low = v_lo, high = v_hi)

## End(Not run)
```

---

sim_met *Simulate a Multi-Environment Trial with a Known Genetic Architecture*

Description

Core simulator. Generates genotype-by-environment phenotypes whose truth is known exactly: a genomic kinship K among lines, a genetic (type-B) correlation C among environments, and a per-environment heritability. It is the reference-answer generator against which `kernel_model` and `scan_untested_envs` can be checked, and it pairs with `sim_W` to study how well an envirome recovers C.

Usage

```
sim_met(
  K = NULL,
  n_env = NULL,
  min.h2 = 0.2,
  max.h2 = 0.8,
  min.cor = 0,
  max.cor = 0.8,
  C = NULL,
  h2 = NULL,
  n_rep = 1,
  sg2 = 1,
  exact = TRUE,
  env_effects = NULL,
  subset = NULL,
  subset.by = c("cell", "gid_env"),
  K.weights = NULL,
  W = NULL,
  reaction.norm = !is.null(W),
  rn.main = 0.5,
```

```
  gxe.specific = 0,
  markers = NULL,
  n.qtl = NULL,
  rep.var = 0,
  spatial = FALSE,
  field.dim = NULL,
  spatial.rho = 0.6,
  spatial.var = 1,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

K	n x n genomic kinship AMONG LINES (e.g. maizeG), or a named *list* of such matrices for multiple variance components (combined with K.weights). Made positive definite internally, so a singular marker-based kinship is fine. May be NULL when markers is given.
n_env	integer. Number of environments; inferred from C if given.
min.h2, max.h2	numeric. Plot-basis heritability range, used when h2 is not supplied. Must satisfy $0 < min.h2 \leq max.h2 < 1$.
min.cor, max.cor	numeric. Genetic correlation range among environments, used when C is not supplied. Mutually exclusive with C.
C	q x q correlation AMONG ENVIRONMENTS (see sim_met_C). Mutually exclusive with min.cor/max.cor.
h2	numeric vector of per-environment heritabilities (length 1 or n_env). Mutually exclusive with min.h2/max.h2.
n_rep	integer. Replicates per genotype x environment cell.
sg2	numeric. Genetic variance per environment (length 1 or n_env).
exact	logical. Orthonormalise the draw so realised moments match the targets. Default TRUE.
env_effects	numeric. Optional fixed environment means μ_j.
subset	numeric in (0, 1]. Fraction of cells retained, to induce unbalancedness.
subset.by	"cell" or "gid_env". Whether missingness is drawn per observation or per genotype-by-environment cell.
K.weights	numeric. Non-negative weights (rescaled to sum 1) for a list of kinships in K; ignored for a single matrix. Default equal weights.
W	q x k envirome matrix. When supplied, genetic values are generated as reaction norms and the environment correlation is induced from W.
reaction.norm	logical. Use the reaction-norm mechanism. Defaults to TRUE when W is supplied.
rn.main	numeric in [0, 1]. Share of genetic variance assigned to the main (across-environment) effect under the reaction-norm mechanism; the rest is reaction-norm interaction. Default 0.5.

gxe.specific	numeric in [0, 1). Share of genetic variance made line-independent and environment-specific, breaking separability. Default 0.
markers	n x m marker matrix. When supplied, a marker/QTL mechanism replaces the infinitesimal draw and K is derived from the markers if not given.
n.qtl	integer. Number of markers acting as QTL (sampled) when markers is used; NULL uses all markers.
rep.var	numeric. Variance of a random replicate/block effect added to phenotypes. Default 0.
spatial	logical. Add an AR1×AR1 spatial field to each environment. Default FALSE.
field.dim	integer length-2 c(nrow, ncol). Field layout per environment; defaults to a near-square grid sized to the plots.
spatial.rho	numeric. Lag-1 spatial autocorrelation. Default 0.6.
spatial.var	numeric. Variance of the spatial field. Default 1.
seed	integer. RNG seed. The caller's RNG stream is restored on exit.
verbose	logical. Print a summary. Default TRUE.

Details

Model. For n lines and q environments, genetic values are drawn from a separable (Kronecker) Gaussian

$$\mathrm{vec}(G) \sim \mathcal{N}(0,\ K \otimes \Sigma_g)\,, \qquad \Sigma_g = D^{1/2}\, C\, D^{1/2},\ D = \mathrm{diag}(\sigma_g^2),$$

realised as $G = L_K\, U\, L_C^\top$ with $L_K L_K^\top = K$, $L_C L_C^\top = \Sigma_g$. When exact = TRUE the driver U is orthonormalised so that $U^\top U = nI$, making the realised moments match their targets instead of merely scattering around them.

Phenotypes. For replicate r,

$$y_{ijr} = \mu_j + g_{ij} + \varepsilon_{ijr}, \qquad \varepsilon_{ijr} \sim \mathcal{N}(0, \sigma_{e,j}^2), \quad \sigma_{e,j}^2 = \sigma_{g,j}^2\, \frac{1 - h_j^2}{h_j^2}.$$

Heritability, two bases. The plot-basis value $h^2 = V_g/(V_g + V_e)$ is what min.h2/max.h2 target; the line-mean basis $h^2/(h^2 + (1 - h^2)/n_{rep})$ is also reported.

Realised environment correlation is measured on the K-whitened scale. Because lines are related through K, cor(G) across lines is not C; forcing it to be is wrong. The realised correlation is defined as $\mathrm{cor}(L_K^{-1} G)$ and stored for validation.

The two matrices are different objects. K is $n \times n$ among lines; the environment correlation is $q \times q$. A genomic kinship passed where an environment correlation is expected is the most common error and is rejected by an elementwise unit-diagonal check.

Alternative genetic mechanisms. Besides the default separable draw, three opt-in mechanisms generate the genetic values:

- *Reaction norm* (supply W): genetic values are built causally from an envirome, $g_{ij} = m_i + \sum_k w_{jk} b_{ik}$ with a main effect m and line sensitivities $b_{\cdot k}$ both K-structured. The environment correlation is then *induced* by W (and reported), not imposed. rn.main splits variance between the main effect and the reaction-norm interaction.

- *Marker-based* (supply `markers`): QTL effects are sampled per environment with the target correlation `C`, giving large-effect architectures instead of the infinitesimal draw.
- *Multiple kinships* (supply a *list* for `K` with `K.weights`): independent additive, dominance or epistatic components are summed.

`gxe.specific` adds a line-independent, environment-specific deviation that makes the covariance *non-separable*. `rep.var` and `spatial` (an AR1×AR1 field) add trial realism to the phenotypes.

Value

An object of class `"sim_met"`: a list with

`data` long `data.frame` with `env`, `gid`, `rep`, `value`.

`C_env` the target environment correlation, with the realised correlation attached as attribute `"realised"`.

`truth` list of everything known: genetic values `g`, target/realised correlations, target/realised heritabilities on both bases, variances, means.

`K` the positive-definite kinship actually used.

References

Costa-Neto, G., et al. (2023). Envirome-wide associations enhance multi-environment prediction. *G3* 13(2), jkac313.

See Also

`sim_W`, `sim_met_C`, `env_cor`, `sim_markers`, `sim_envirome`, `kernel_model`, `scan_untested_envs`

Examples

```
data(maizeG)

## 1. Baseline: random C and random h2 within a range
met <- sim_met(maizeG, n_env = 10, min.cor = 0.2, max.cor = 0.8,
               min.h2 = 0.3, max.h2 = 0.7, seed = 1)
head(met$data)
env_cor(met, "realised")

## 2. Supply an explicit environment correlation (e.g. a year-like series)
C   <- sim_met_C(q = 8, structure = "ar1", rho = 0.6)
met <- sim_met(maizeG, C = C, h2 = 0.5, seed = 1)

## 3. Fixed per-environment heritabilities and environment means
met <- sim_met(maizeG, n_env = 4, h2 = c(0.3, 0.5, 0.6, 0.8),
               env_effects = c(2, 4, 6, 8), seed = 1)

## 4. Replicates and an unbalanced design (retain 70% of g x e cells)
met <- sim_met(maizeG, n_env = 6, n_rep = 3,
               subset = 0.7, subset.by = "gid_env", seed = 1)

## 5. Several genetic variance components (additive + dominance + epistasis)
met <- sim_met(K = list(add = maizeG, dom = maizeG, epi = maizeG),
```

```
               K.weights = c(0.6, 0.3, 0.1), n_env = 8, seed = 1)

## 6. Causal reaction-norm: C is INDUCED by the envirome, not imposed
Wenv <- scale(matrix(rnorm(10 * 6), 10, 6))
rn   <- sim_met(maizeG, W = Wenv, rn.main = 0.4, seed = 1)
env_cor(rn, "target")

## 7. Marker / QTL architecture instead of the infinitesimal draw
X   <- matrix(rbinom(200 * 500, 2, 0.3), 200, 500)
qtl <- sim_met(markers = X, n_env = 6, n.qtl = 30, seed = 1)

## 8. Non-separable GxE: 25% of genetic variance is environment-specific
met <- sim_met(maizeG, n_env = 6, gxe.specific = 0.25, seed = 1)

## 9. Field realism: replicate/block noise plus an AR1 x AR1 spatial trend
met <- sim_met(maizeG, n_env = 4, n_rep = 2, rep.var = 0.5,
               spatial = TRUE, field.dim = c(20, 15),
               spatial.rho = 0.7, spatial.var = 1.5, seed = 1)

## 10. Pair with a simulated envirome and diagnose the recovery
W <- sim_W(met$C_env, noise = 0.3, n_var = 50, seed = 1)
plot(met)

## 11. End-to-end: simulate kinship + MET + envirome, then fit a model
##     Simulate a genomic kinship KG from markers (VanRaden)
X <- matrix(rbinom(150 * 1000, 2, 0.3), 150, 1000)
rownames(X) <- paste0("G", seq_len(150))        # gid names carry through
Z <- scale(X); Z[is.na(Z)] <- 0
Kg <- tcrossprod(Z) / ncol(Z) + diag(1e-6, 150) # positive-definite kinship
dimnames(Kg) <- list(rownames(X), rownames(X))
KG <- list(G = Kg)

##     Simulate the MET truth on that kinship (one replicate)
met <- sim_met(Kg, n_env = 10, n_rep = 1,
               min.cor = 0.3, max.cor = 0.8,
               min.h2 = 0.3, max.h2 = 0.7, seed = 1)
df  <- as.data.frame(met$data)                  # base data.frame for get_kernel()

##     Simulate a redundant, noisier envirome on the same environments
W2  <- sim_W(met$C_env, noise = 0.6, n_var = 200,
             collinearity = 0.8, n_blocks = 5, seed = 1)
KE2 <- list(W = env_kernel(env.data = as.matrix(W2))[[2]])

##     Build reaction-norm kernels and fit the Bayesian model
K2  <- get_kernel(K_G = KG, K_E = KE2, data = df, model = "RNMM",
                  env = "env", gid = "gid", y = "value")
fit <- kernel_model(y = "value", data = df, random = K2,
                    env = "env", gid = "gid",
                    iterations = 5000, burnin = 1000)

## 12. Phenotypes for contrasting GENETIC diversity (envirome fixed)
##     Low diversity: few rare markers, strong structure; high: many common.
E    <- sim_envirome(12, 40, structure = "clusters", n_cluster = 3, seed = 1)
```

```
X_lo <- sim_markers(200, 500,  maf = "rare",   n_subpop = 5, Fst = 0.2, seed = 1)
X_hi <- sim_markers(200, 4000, maf = "common", n_subpop = 1, Fst = 0.0, seed = 1)
met_lo <- sim_met(markers = X_lo, C = attr(E, "C_env"), n.qtl = 80, seed = 1)
met_hi <- sim_met(markers = X_hi, C = attr(E, "C_env"), n.qtl = 80, seed = 1)
c(low = var(met_lo$data$value), high = var(met_hi$data$value))

## 13. Phenotypes for contrasting ENVIROME diversity (genetics fixed)
##     Diverse environments induce lower cross-environment correlation (more GxE).
X    <- sim_markers(200, 3000, n_subpop = 3, Fst = 0.08, seed = 2)
E_lo <- sim_envirome(12, 40, structure = "gradient", redundancy = 0.7,
                     n_blocks = 2, seed = 2)
E_hi <- sim_envirome(12, 40, structure = "random", redundancy = 0, seed = 2)
met_eLo <- sim_met(markers = X, C = attr(E_lo, "C_env"), n.qtl = 80, seed = 2)
met_eHi <- sim_met(markers = X, C = attr(E_hi, "C_env"), n.qtl = 80, seed = 2)
cor(met_eLo$truth$g)[1, 2]   # high  cross-environment genetic correlation
cor(met_eHi$truth$g)[1, 2]   # low   correlation -> stronger GxE

## 14. Full 2 x 2 factorial of genetic x envirome diversity
scenario <- function(gen, env, seed = 1) {
  X <- if (gen == "low")
         sim_markers(200, 500, maf = "rare", n_subpop = 5, Fst = 0.2,
                     seed = seed, verbose = FALSE)
       else
         sim_markers(200, 4000, maf = "common", n_subpop = 1, Fst = 0,
                     seed = seed, verbose = FALSE)
  E <- if (env == "low")
         sim_envirome(12, 40, structure = "gradient", redundancy = 0.7,
                      n_blocks = 2, seed = seed, verbose = FALSE)
       else
         sim_envirome(12, 40, structure = "random", redundancy = 0,
                      seed = seed, verbose = FALSE)
  met <- sim_met(markers = X, C = attr(E, "C_env"),
                 n.qtl = 80, min.h2 = 0.3, max.h2 = 0.6, seed = seed)
  Cg  <- cor(met$truth$g)
  data.frame(gen = gen, env = env,
             gxe_cor   = mean(Cg[lower.tri(Cg)]),
             pheno_var = var(met$data$value))
}
do.call(rbind, list(scenario("low",  "low"),  scenario("low",  "high"),
                    scenario("high", "low"),  scenario("high", "high")))

## 15. Sweep population structure (Fst) and record phenotype variance
E <- sim_envirome(10, 40, structure = "clusters", n_cluster = 3, seed = 3)
do.call(rbind, lapply(c(0, 0.05, 0.1, 0.2), function(f) {
  X   <- sim_markers(200, 3000, n_subpop = 4, Fst = f, seed = 3, verbose = FALSE)
  met <- sim_met(markers = X, C = attr(E, "C_env"), n.qtl = 80, seed = 3)
  data.frame(Fst = f, eff_dim = attr(X, "diversity")["eff_dim"],
             pheno_var = var(met$data$value))
}))

## 16. Reaction-norm phenotypes driven directly by a diverse envirome
X    <- sim_markers(150, 2500, n_subpop = 2, Fst = 0.06, seed = 6)
E2   <- sim_envirome(10, 25, structure = "gradient",
```

```
                    marginal = "right-skewed", seed = 6)
met2 <- sim_met(markers = X, W = as.matrix(E2),
                reaction.norm = TRUE, rn.main = 0.6, seed = 6)
met2$truth$mechanism
```

---

sim_met_C *Simulate a q x q Genetic Correlation Among Environments*

Description

Accessory to sim_met and sim_W. Builds a valid $q \times q$ correlation matrix among environments, so the same C can seed both simulators explicitly.

Usage

```
sim_met_C(
  q,
  min.cor = 0,
  max.cor = 0.8,
  structure = c("random", "vine", "equi", "ar1", "block", "factor"),
  rho = NULL,
  n_blocks = 2L,
  between.cor = 0,
  n_factor = 2L,
  env_names = NULL,
  max.tries = 200L,
  seed = NULL
)
```

Arguments

q	integer. Number of environments (>= 3).
min.cor, max.cor	numeric. Range for off-diagonal correlations. For the structured generators these bound the random inputs (vine partial correlations, factor loadings) rather than the realised off-diagonals exactly.
structure	character. How the matrix is built: "random" (default, rejection-sampled uniform off-diagonals, as before), "vine" (C-vine partial correlations, always PSD without rejection), "equi" (equicorrelated), "ar1" (first-order autoregressive, for ordered environments such as a year series), "block" (block-diagonal mega-environments) or "factor" (a random factor model $\Lambda\Lambda^\top + \Psi$).
rho	numeric or NULL. The correlation used by "equi" (off-diagonal), "ar1" (lag-1) and "block" (within-block). If NULL, a sensible value is derived from min.cor/max.cor.
n_blocks	integer. Number of mega-environments when structure = "block".

`between.cor`	numeric. Between-block correlation when `structure = "block"`.
`n_factor`	integer. Number of latent factors when `structure = "factor"`.
`env_names`	character. Optional environment names.
`max.tries`	integer. Rejection-sampling attempts before repair.
`seed`	integer. RNG seed.

Details

Sampling off-diagonals uniformly from `[min.cor, max.cor]` does not generally give a positive semi-definite matrix. The equicorrelated bound is $\rho \geq -1/(q-1)$, so for $q = 6$ nothing below -0.2 is attainable. A naive eigenvalue repair silently returns something else, so this function errors up front when the request is infeasible and rejection-samples otherwise, only falling back to eigenvalue clamping (with a warning) if sampling fails.

Value

A $q \times q$ correlation matrix of class `"C_env"` with a logical attribute `"repaired"` and an attribute `"requested"` recording the target range.

See Also

`sim_met`, `sim_W`, `env_cor`

Examples

```
C <- sim_met_C(q = 8, min.cor = 0.2, max.cor = 0.8, seed = 1)
range(C[lower.tri(C)])

## Structured mega-environments and an autoregressive year series
Cb <- sim_met_C(q = 9, structure = "block", n_blocks = 3, rho = 0.7,
                between.cor = 0.1)
Car <- sim_met_C(q = 8, structure = "ar1", rho = 0.6)
```

`sim_W` *Simulate an Envirome with a Known Relationship to an Environment Correlation*

Description

Core simulator. Given a target correlation C_{env} among environments, it builds a $q \times n_{var}$ covariable matrix W whose environmental kernel recovers a *known* share of C_{env}. It answers: if the true similarity among my environments is C_{env}, what would an envirome of this size and quality look like, and how much of that similarity would it recover? Pairs with `sim_met`.

Usage

```
sim_W(
  C_env,
  noise = 0,
  n_var = 50,
  collinearity = 0,
  n_blocks = 1,
  calibrate = TRUE,
  cal.nrep = 15L,
  marginal = c("gaussian", "right-skewed", "heavy-tailed", "bounded"),
  kernel = c("linear", "rbf"),
  bandwidth = NULL,
  na.frac = 0,
  family.sizes = NULL,
  var_names = NULL,
  env_names = NULL,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`C_env`	q x q correlation AMONG ENVIRONMENTS, from any source (e.g. `sim_met$C_env`, `env_cor`, `cov2cor(env_kernel(W)$envCov)`, or hand-specified). NOT a kinship.
`noise`	numeric in [0, 1]. With `calibrate = TRUE`, the expected off-diagonal correlation between the simulated kernel and `C_env` is `1 - noise`.
`n_var`	integer. Number of covariables (columns of W).
`collinearity`	numeric in [0, 1). Share of columns collapsed onto drivers (also the within-family strength when `family.sizes` is used).
`n_blocks`	integer. Collinear columns grouped into this many families.
`calibrate`	logical. Solve for the internal variance share a numerically so `noise` is on the outcome scale. Default `TRUE`.
`cal.nrep`	integer. Monte-Carlo replicates used during calibration.
`marginal`	character. Marginal distribution of the covariables: `"gaussian"` (default), `"right-skewed"`, `"heavy-tailed"` or `"bounded"`. Applied rank-preserving, so recovery is barely changed.
`kernel`	character. Environmental kernel: `"linear"` (default, $WW^\top$) or `"rbf"` (Gaussian).
`bandwidth`	numeric or `NULL`. RBF bandwidth; `NULL` uses the median heuristic.
`na.frac`	numeric in [0, 1). Fraction of entries set to `NA` in the returned W (truth is computed on the complete matrix first).
`family.sizes`	integer vector (optionally named). Column counts per covariable family; must sum to `n_var`. Overrides `n_blocks` redundancy and names the columns by family.

var_names, env_names
: character. Optional names for columns / rows.

seed
: integer. RNG seed. The caller's RNG stream is restored on exit.

verbose
: logical. Print a summary. Default TRUE.

Details

Construction. With $C_{env}^{1/2}$ the symmetric PSD square root of the target, an orthonormal driver O ($q \times n_{var}$) and iid noise E,

$$W = \sqrt{1-a}\, C_{env}^{1/2} O \;+\; \sqrt{a}\, E,$$

and the environmental kernel follows the env_kernel convention

$$K_W = \frac{WW^\top}{\mathrm{tr}(WW^\top)/q}.$$

Recovery is measured by the off-diagonal Pearson correlation $r = \mathrm{cor}\big(\mathrm{offdiag}(K_W),\, \mathrm{offdiag}(C_{env})\big)$.

Calibration. When calibrate = TRUE, noise is on the *outcome* scale: the internal variance share a is solved by uniroot so that $E[r] = 1 - \mathrm{noise}$. Because r decreases in a, noise = 0 gives $a = 0$ (full recovery) and noise = 1 gives $a = 1$ (pure noise).

Collinearity. With collinearity > 0, a share of columns is collapsed onto n_blocks driver families, mimicking the redundancy of real enviromes and lowering the effective rank.

C_env **is an environment correlation, not a kinship.** It must be $q \times q$ with unit diagonal. Passing a genomic kinship is rejected.

Realism knobs. marginal maps each covariable to a non-Gaussian shape (rank-preserving), kernel = "rbf" builds a Gaussian instead of a linear kernel, family.sizes groups columns into named covariable families (as in real enviromes), and na.frac injects missing values into the returned W *after* the truth is computed, so downstream cleaning (e.g. W_matrix imputation) can be tested against a known answer.

Value

A $q \times n_{var}$ matrix of class "sim_W". Attributes: "C_env" (the target), "K_W" (the realised kernel) and "explained" (a list with r, r2, per-PC variance shares, effective rank eff_rank, the internal share a_internal, the marginal/kernel/na.frac used, and the requested/target values).

References

Costa-Neto, G., et al. (2023). Envirome-wide associations enhance multi-environment prediction. *G3* 13(2), jkac313.

See Also

sim_met, sim_met_C, sim_W_grid, env_cor, env_kernel

Examples

```
C <- sim_met_C(q = 10, min.cor = 0.2, max.cor = 0.8, seed = 1)

## 1. An envirome that recovers ~70% of C (noise is on the outcome scale)
W <- sim_W(C, noise = 0.3, n_var = 50, seed = 1)
attr(W, "explained")$r

## 2. Perfect vs pure-noise enviromes
W_good  <- sim_W(C, noise = 0.0, n_var = 50)
W_noise <- sim_W(C, noise = 1.0, n_var = 50)

## 3. Redundant envirome: many collinear columns, low effective rank
W <- sim_W(C, noise = 0.3, n_var = 200, collinearity = 0.8, n_blocks = 5)

## 4. Non-Gaussian covariables (rank-preserving marginal transform)
W <- sim_W(C, noise = 0.3, n_var = 60, marginal = "right-skewed")
W <- sim_W(C, noise = 0.3, n_var = 60, marginal = "heavy-tailed")
W <- sim_W(C, noise = 0.3, n_var = 60, marginal = "bounded")

## 5. Gaussian RBF kernel instead of the linear WW'
W <- sim_W(C, noise = 0.3, n_var = 60, kernel = "rbf")
W <- sim_W(C, noise = 0.3, n_var = 60, kernel = "rbf", bandwidth = 2)

## 6. Inject missing values AFTER the truth is computed
W <- sim_W(C, noise = 0.3, n_var = 60, na.frac = 0.1)
sum(is.na(W))

## 7. Named covariable families (sizes must sum to n_var)
W <- sim_W(C, n_var = 12, family.sizes = c(temp = 4, rain = 5, rad = 3))
colnames(W)

## 8. Custom names and a faster/looser calibration
W <- sim_W(C, noise = 0.4, n_var = 30, cal.nrep = 5,
           env_names = paste0("Loc", 1:10),
           var_names = paste0("bio", 1:30), seed = 1)

## 9. Raw (uncalibrated) internal variance share
W <- sim_W(C, noise = 0.5, n_var = 50, calibrate = FALSE)
plot(W)
```

`sim_W_grid` *Sweep Envirome Size and Quality (Power Analysis)*

Description

Accessory to `sim_W`. Repeatedly simulates enviromes across a grid of `noise`, `n_var` and `collinearity`, returning the mean recovery and effective rank at each design point.

Usage

```
sim_W_grid(
  C_env,
  noise = seq(0, 1, 0.25),
  n_var = c(10, 50, 200),
  collinearity = c(0, 0.5, 0.9),
  n_rep = 10,
  calibrate = TRUE,
  cal.nrep = 15L,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

C_env	q x q correlation AMONG ENVIRONMENTS (see sim_W).
noise, n_var, collinearity	numeric vectors defining the grid.
n_rep	integer. Replicate simulations per design point.
calibrate	logical. Passed to sim_W.
cal.nrep	integer. Calibration replicates per sim_W call.
seed	integer. RNG seed; each design point gets a reproducible sub-seed.
verbose	logical. Print per-design-point progress. Default TRUE.

Value

A data.frame with one row per design point: noise, n_var, collinearity, mean recovery r and its SD r_sd, r2, and mean eff_rank.

See Also

sim_W, sim_met_C

Examples

```
C <- sim_met_C(q = 10, min.cor = 0.2, max.cor = 0.8, seed = 1)
grid <- sim_W_grid(C, noise = c(0, 0.3, 0.6), n_var = c(20, 100),
                   collinearity = c(0, 0.8), n_rep = 5)
```

soil_classification *Delineate Soil Zones with a Gaussian Mixture Model*

Description

Clusters soil profiles returned by `get_soil` into a small number of zones using a Gaussian mixture model, selecting both the number of clusters and the covariance structure by BIC. Returns hard cluster labels, the full matrix of posterior membership probabilities, and a per-site uncertainty measure.

Unlike k-means, a mixture model gives every site a probability of belonging to every zone. A site on a boundary is genuinely ambiguous, and that ambiguity is reported rather than hidden behind a hard label.

Usage

```
soil_classification(
  soil.data,
  env.id = "env",
  variables = NULL,
  depths = NULL,
  G = 1:9,
  model.names = NULL,
  scale.data = TRUE,
  clr.texture = TRUE,
  convert.units = TRUE,
  use.pca = FALSE,
  pca.var = 0.95,
  risk.vars = NULL,
  risk.direction = c("high", "low"),
  uncertainty.threshold = 0.7,
  seed = 1,
  verbose = TRUE
)
```

Arguments

`soil.data`	data.frame. Wide-format output of `get_soil(wide = TRUE)`, one row per site, with columns named `property|depth` (for example `"clay|0_5cm"`).
`env.id`	character. Name of the site-id column. Default `"env"`.
`variables`	character. Soil properties to cluster on, for example `c("clay", "sand", "soc")`. `NULL` (default) uses every property present. Filtering is strongly recommended – see **Choosing variables** below.
`depths`	character. Depth layers to use, for example `c("0_5cm", "30_60cm")`. `NULL` (default) uses all layers present. Adjacent SoilGrids layers are highly correlated, so using all six rarely helps.

`G` integer vector. Candidate numbers of clusters. Default `1:9`. `G = 1` is included deliberately: if BIC selects it, the data provide no evidence for more than one zone, which is a legitimate and useful finding.

`model.names` character. `mclust` covariance families to try, for example `"EII"` (spherical, equal volume) or `"VVV"` (ellipsoidal, all free). `NULL` (default) lets `mclust` try every family it can fit. Restrict this when sites are few relative to variables.

`scale.data` logical. Standardise variables to zero mean and unit variance before fitting. Default `TRUE`, and strongly recommended: soil properties differ by orders of magnitude, and an unscaled fit is dominated by whichever variable happens to have the largest units.

`clr.texture` logical. Apply an isometric log-ratio (ilr) transform to texture fractions (clay/sand/silt) at each depth. Default `TRUE`.

`convert.units` logical. Apply SoilGrids conversion factors, turning mapped integers into conventional units. Requires `.SG_FACTORS` from `water_balance.R`. Default `TRUE`.

`use.pca` logical. Reduce the feature matrix to principal components before fitting. Default `FALSE`. Useful when variables are many and collinear.

`pca.var` numeric in (0, 1]. Proportion of variance the retained components must explain when `use.pca = TRUE`. Default `0.95`.

`risk.vars` character. Variables used to order the cluster labels, for example `"soc"`. `NULL` (default) orders by the mean of all scaled features. See **Label ordering** below.

`risk.direction` `"high"` or `"low"`. Whether high values of `risk.vars` should sort last (`"high"`, the default, so cluster 1 is the lowest) or first.

`uncertainty.threshold` numeric in (0, 1). A site whose maximum posterior probability falls below this is flagged `uncertain`. Default `0.7`.

`seed` integer or `NULL`. Seed for reproducible EM starts. Default `1`. Cluster *labels* are seed-invariant because of the ordering step, though the underlying fit may vary.

`verbose` logical. Print progress. Default `TRUE`.

Details

What the model does. A Gaussian mixture assumes the sites are drawn from G multivariate normal components with unknown means, covariances and mixing weights, fitted by EM. `mclust` additionally searches over constrained covariance families (spherical, diagonal, ellipsoidal; equal or varying across components), and BIC selects both G and the family in one criterion. See Fraley & Raftery (2002) and Scrucca et al. (2016).

Compositional data. Texture fractions sum to a constant. Feeding them to a Gaussian untransformed is a category error: the closure constraint forces spurious negative correlations. The default `clr.texture = TRUE` applies an *isometric* log-ratio transform per depth, mapping D parts to $D-1$ orthonormal coordinates. A plain clr transform is deliberately not used for fitting: its coordinates sum to zero, leaving the covariance singular and causing BIC to select the wrong number of clusters. Turn this off only if your variables are not compositional.

Label ordering. Mixture component labels are arbitrary and permute between runs. Clusters are therefore renumbered by `risk.vars` so that cluster 1 is always the low end and cluster G the high

end. Without this, "cluster 3" would mean something different on every run and results could not be compared or cached.

Choosing variables. A mixture estimates a mean vector and a covariance per component, so the parameter count grows quickly with the number of features. Passing every property at every depth gives 20+ highly correlated columns and will overwhelm a modest number of sites. The function warns when `n < 10 * p`. Prefer a handful of properties at two contrasting depths, or set `use.pca = TRUE`.

Sample size. This is the single most common way to get a meaningless result. Clustering a handful of sites produces a model in which every site owns its own component: the output reports `max_posterior = 1` and `uncertainty = 0` for every row, which looks like total confidence and means nothing. Zone delineation samples a grid of points across the area of interest – typically hundreds – not one point per farm. Check n against p before believing any output.

Reading the BIC gap. `print()` reports the BIC difference between the selected model and the runner-up, labelled weak (< 2), positive (2-6), strong (6-10) or very strong (> 10) following the usual convention for BIC differences. A weak gap means several partitions fit about equally well and the zone count should not be presented as settled.

Relationship to the source paper. The paper behind this function (Zhang et al., doi:10.34133/ehs.0402) could not be read: the publisher blocks automated access. This implements the standard GMM zoning workflow its title describes – log-ratio transform, standardise, BIC-selected mixture, posterior-based uncertainty – but the paper's exact preprocessing, covariance families and thresholds are unverified. Treat this as a methodologically conventional starting point, not a reproduction.

What this cannot do. SoilGrids contains no cadmium, and no heavy metal of any kind. The source paper's subject is cadmium risk; this function can only cluster the properties SoilGrids actually provides (texture, organic carbon, pH, CEC, water retention). Any "risk" interpretation rests entirely on whichever proxies you pass to `risk.vars`, and the paper's machine learning step – predicting the contaminant before zoning – has no counterpart here. Do not present output from this function as a contaminant risk map.

Value

An object of class `soil_gmm`, a list with:

`classification` data.frame, one row per input site in input order: `env`, `cluster`, `max_posterior`, `uncertainty` (= 1 - max_posterior), and logical `uncertain`.

`posterior` matrix of posterior membership probabilities, sites x clusters, each row summing to 1.

`profiles` data.frame of per-cluster means in original units, plus `n_sites`.

`model` list with `G`, `modelName`, `bic`, `loglik`.

`BIC` the full BIC table over G and covariance family.

`features` the feature matrix actually clustered.

`pca` the `prcomp` object, or `NULL`.

`settings` the call arguments, for reproducibility.

Output granularity

The function classifies *rows*, one output row per input row, in input order. If each row is a grid point, you get a cluster per grid point; a farm spanning several rows may well split across zones, and that split is usually the informative result. Join back to your own table on `env`.

References

Zhang, F., Jia, Z., Wu, S., Chen, C., Chen, X., Zheng, C., Xu, M. (2025) Machine Learning and Gaussian Mixture Model for Delineating Soil Cadmium Risk Zones. *Ecosystem Health and Sustainability*. doi:10.34133/ehs.0402

Fraley, C. & Raftery, A.E. (2002) Model-based clustering, discriminant analysis and density estimation. *Journal of the American Statistical Association* 97(458), 611-631. doi:10.1198/016214502760047131

Scrucca, L., Fop, M., Murphy, T.B. & Raftery, A.E. (2016) mclust 5: clustering, classification and density estimation using Gaussian finite mixture models. *The R Journal* 8(1), 205-233. doi:10.32614/RJ2016021

Aitchison, J. (1982) The statistical analysis of compositional data. *Journal of the Royal Statistical Society B* 44(2), 139-177.

Egozcue, J.J., Pawlowsky-Glahn, V., Mateu-Figueras, G. & Barcelo-Vidal, C. (2003) Isometric logratio transformations for compositional data analysis. *Mathematical Geology* 35(3), 279-300. doi:10.1023/A:1023818214614

Schwarz, G. (1978) Estimating the dimension of a model. *Annals of Statistics* 6(2), 461-464. doi:10.1214/aos/1176344136

Poggio, L. et al. (2021) SoilGrids 2.0. *SOIL* 7, 217-240. doi:10.5194/soil72172021

See Also

`get_soil` for the input, `water_balance` for per-site indices that need no cross-site sample, `Mclust` for the underlying model.

Examples

```
## Not run:
## This workflow downloads SoilGrids data for a 200-site grid.
library(mclust)

## ---------------------------------------------------------------
## 1. Minimal use
## ---------------------------------------------------------------
soil <- get_soil(env.id = c("NM", "SO", "IP"),
                 lat = c(-13.05, -12.32, -21.98),
                 lon = c(-56.08, -55.71, -47.88),
                 variables.names = c("clay", "sand", "silt", "soc", "phh2o"))

fit <- soil_classification(soil)
fit                                # summary, BIC gap, cluster profiles
head(fit$classification)           # per-site cluster + uncertainty


## ---------------------------------------------------------------
## 2. Realistic zoning: sample a GRID, not one point per farm
## ---------------------------------------------------------------
## A mixture cannot be identified from a handful of rows. Sample the
## area of interest, here 40 points around each of 5 locations.
sites <- data.frame(
```

```
  env = c("SOR", "LON", "SLG", "PAL", "BAR"),
  lat = c(-12.5453, -23.3045, -28.4083, -10.1689, -12.1530),
  lon = c(-55.7113, -51.1696, -54.9608, -48.3317, -44.9900))

g <- expand.grid(k = 1:40, i = seq_len(nrow(sites)))
pts <- data.frame(
  env  = paste0(sites$env[g$i], "_", sprintf("%03d", g$k)),
  farm = sites$env[g$i],
  lat  = sites$lat[g$i] + runif(nrow(g), -0.05, 0.05),
  lon  = sites$lon[g$i] + runif(nrow(g), -0.05, 0.05))

soil_grid <- get_soil(env.id = pts$env, lat = pts$lat, lon = pts$lon,
                      variables.names = c("clay", "sand", "silt",
                                          "soc", "phh2o"))

fit <- soil_classification(soil_grid,
                           variables = c("clay", "sand", "silt",
                                         "soc", "phh2o"),
                           depths    = c("0_5cm", "30_60cm"),
                           risk.vars = "soc")
fit

## Do farms split across zones? Usually the interesting question.
table(farm = pts$farm, cluster = fit$classification$cluster)


## ----------------------------------------------------------------
## 3. Posterior probabilities and ambiguous sites
## ----------------------------------------------------------------
round(head(fit$posterior), 3)        # membership in EVERY zone

## Sites the model is least sure about -- these sit on a boundary.
amb <- fit$classification[fit$classification$uncertain, ]
amb[order(amb$max_posterior), ]

## Tighten or relax the flag without refitting the model:
fit2 <- soil_classification(soil_grid, uncertainty.threshold = 0.9)
sum(fit2$classification$uncertain)


## ----------------------------------------------------------------
## 4. Ordering zones so cluster 1 always means the same thing
## ----------------------------------------------------------------
lo  <- soil_classification(soil_grid, risk.vars = "soc",
                           risk.direction = "high")   # 1 = lowest SOC
hi  <- soil_classification(soil_grid, risk.vars = "soc",
                           risk.direction = "low")    # 1 = highest SOC
lo$profiles[, c("cluster", "n_sites")]


## ----------------------------------------------------------------
## 5. Many correlated variables: reduce first
## ----------------------------------------------------------------
```

```
fit_pca <- soil_classification(soil_grid, use.pca = TRUE, pca.var = 0.95)
fit_pca$pca                          # how many components were kept

## Or restrict the covariance family when sites are few:
fit_par <- soil_classification(soil_grid, model.names = c("EII", "VII",
                                                          "EEI", "VVI"))


## ----------------------------------------------------------------
## 6. Joining results back to your own table
## ----------------------------------------------------------------
out <- merge(pts, fit$classification, by = "env")
head(out[, c("env", "farm", "lat", "lon", "cluster",
             "max_posterior", "uncertain")])

## Dominant zone per farm, and whether the farm is homogeneous
with(out, table(farm, cluster))


## ----------------------------------------------------------------
## 7. Is the cluster count actually supported?
## ----------------------------------------------------------------
fit$BIC                              # full table over G and family
fit$model$G                          # selected number of zones
## print(fit) reports the gap to the runner-up; a weak gap (< 2) means
## the zone count is not settled, whatever the point estimate says.

## End(Not run)
```

---

summary_water_balance *Summarise Water-Balance Output by Environment or Growth Stage*

Description

Condenses the daily output of `water_balance` into one row per environment, or per environment and growth stage, producing the season-level indices most often used as environmental covariates in genotype-by-environment models.

Usage

```
summary_water_balance(
  wb,
  env.id = "env",
  days.id = "daysFromStart",
  by.interval = FALSE,
  time.window = NULL,
  names.window = NULL,
  verbose = TRUE
)
```

Arguments

`wb`	data.frame. Output of `water_balance`.
`env.id`	character. Environment-id column. Default `"env"`.
`days.id`	character. Day counter column. Default `"daysFromStart"`.
`by.interval`	logical. Summarise within growth stages as well as by environment. Default `FALSE`.
`time.window`	numeric. Stage breakpoints in days after planting. When `NULL`, 10-day windows are used.
`names.window`	character. Stage names, one per interval. Ignored if the length does not match the number of intervals.
`verbose`	boolean. If `TRUE` (default) prints a progress banner.

Details

`ETa_ETc_ratio` is the relative transpiration deficit, the quantity entering most water-limited yield models – notably the FAO Ky framework of Doorenbos & Kassam (1979), where relative yield loss is proportional to $1 - ET_a/ET_c$.

`max_consecutive_stress` distinguishes one damaging spell from the same number of scattered days, which a mean cannot. Ten consecutive stressed days at flowering is a different event from ten isolated days across the season, and yield responds very differently to the two.

Stage timing matters more than season totals. Water deficit during flowering and grain fill costs disproportionately more yield than the same deficit during vegetative growth (Steduto et al. 2012). Prefer `by.interval = TRUE` with breakpoints matched to your crop's phenology over a single season-long mean.

Value

A data.frame with, per group: `n_days`, `mean_Ks`, `min_Ks`, `total_deficit`, `n_stress_days`, `max_consecutive_stress`, `total_ETc`, `total_ETa`, `ETa_ETc_ratio`, `total_rain`, `total_irrigation`, `total_drainage`, `mean_ASW`, `mean_depletion`.

References

Doorenbos, J. & Kassam, A.H. (1979) Yield response to water. *FAO Irrigation and Drainage Paper 33*. FAO, Rome.

Steduto, P., Hsiao, T.C., Fereres, E. & Raes, D. (2012) Crop yield response to water. *FAO Irrigation and Drainage Paper 66*. FAO, Rome.

Costa-Neto, G., Fritsche-Neto, R. & Crossa, J. (2021) Nonlinear kernels, dominance, and envirotyping data increase the accuracy of genome-based prediction in multi-environment trials. *Heredity* 126, 92-106. doi:10.1038/s41437020003531

See Also

`water_balance`, `summaryWTH`

Examples

```
## Synthetic daily weather and a uniform soil profile (mapped units).
wth <- expand.grid(daysFromStart = 1:120, env = c("E1", "E2"))
wth$PRECTOT <- ifelse(wth$daysFromStart %% 10 == 0, 25, 0)
wth$ETP <- 5
soil <- data.frame(env = c("E1", "E2"))
for (layer in c("0_5cm", "5_15cm", "15_30cm", "30_60cm", "60_100cm")) {
  soil[[paste0("wv0033|", layer)]] <- 300
  soil[[paste0("wv1500|", layer)]] <- 150
}
wb <- water_balance(wth, soil)

## Season-level, one row per environment
summary_water_balance(wb)

## By maize growth stage
summary_water_balance(wb, by.interval = TRUE,
                      time.window  = c(0, 14, 35, 60, 90, 120),
                      names.window = c("P-E", "E-V1", "V1-V4",
                                       "V4-VT", "VT-GF", "GF-PM"))

## Default 10-day windows when no breakpoints are given
summary_water_balance(wb, by.interval = TRUE)

## Rank environments by flowering-period stress
s <- summary_water_balance(wb, by.interval = TRUE,
                           time.window  = c(0, 60, 90, 120),
                           names.window = c("veg", "flower", "fill"))
s[s$interval == "flower", ][order(s$mean_Ks[s$interval == "flower"]), ]
```

---

summaryWTH *Basic Summary Statistics for Environmental Data*

Description

Summarises a `get_weather` output by environment and, optionally, by user-defined time intervals such as phenological stages.

Usage

```
summaryWTH(
  env.data,
  id.names = NULL,
  env.id = NULL,
  days.id = NULL,
  var.id = NULL,
  statistic = NULL,
```

```
  probs = NULL,
  by.interval = FALSE,
  time.window = NULL,
  names.window = NULL,
  verbose = TRUE
)
```

Arguments

`env.data`	data.frame. A `get_weather` output.
`id.names`	vector (character). Columns treated as identifiers.
`env.id`	character. Environment-id column. Default `"env"`.
`days.id`	character. Column holding days from start. Default `"daysFromStart"`.
`var.id`	vector (character). Variables to summarise. Defaults to every non-identifier column.
`statistic`	character. One of `"all"`, `"sum"`, `"mean"`, `"quantile"`.
`probs`	vector (numeric). Quantile probabilities. Default `c(0.25, 0.50, 0.75)`.
`by.interval`	boolean. Summarise within time intervals.
`time.window`	vector (numeric). Interval breakpoints in days.
`names.window`	vector (character). Interval names.
`verbose`	boolean. If `TRUE` (default) prints a progress banner.

Details

Quantiles are computed with `stats::aggregate` rather than the previous nested `foreach` loops, which registered no parallel backend and therefore ran sequentially while emitting a warning on every call.

Value

A data.frame in long format with one row per environment (and interval) and variable.

Author(s)

Germano Costa Neto

See Also

`get_weather`, `processWTH`

Examples

```
env.data <- get_weather(lat = -13.05, lon = -56.05)
summaryWTH(env.data, env.id = "env", statistic = "mean")

summaryWTH(env.data, env.id = "env", by.interval = TRUE,
           time.window  = c(0, 14, 35, 60, 90, 120),
```

```
names.window = c("P-E", "E-V1", "V1-V4", "V4-VT", "VT-GF", "GF-PM"))
```

T_matrix *Envirotype Frequency Matrix, Ready for the Kernel Pipeline*

Description

The envirotype counterpart of W_matrix. Runs env_typing and returns, unconditionally, an environment x envirotype numeric matrix suitable for env_kernel and env_cluster – regardless of which of the four env_typing() branches ran. Where W_matrix summarises continuous covariables, T_matrix summarises the *frequencies* of discrete environmental types (envirotypes), the "enviromic assembly" building block of Costa-Neto, Crossa & Fritsche-Neto (2021).

Usage

```
T_matrix(
  env.data,
  var.id = NULL,
  env.id = NULL,
  cardinals = NULL,
  days.id = NULL,
  time.window = NULL,
  names.window = NULL,
  quantiles = NULL,
  id.names = NULL,
  by.interval = FALSE,
  joint = FALSE,
  joint.by.interval = TRUE,
  envirotype_mining = FALSE,
  k.range = 2:10,
  nstart = 25,
  iter.max = 100,
  seed = 1234,
  combine.features = FALSE,
  combine.max.order = 2L,
  ratio = TRUE,
  center = FALSE,
  scale = FALSE,
  sd.tol = 10,
  tol = 0.001,
  QC = FALSE,
  impute = c("none", "mean", "drop"),
  drop.constant = TRUE,
  verbose = TRUE
)
```

Arguments

- `env.data` — data.frame of processed weather data, or an object already returned by `env_typing` (any of its four shapes), in which case it is reshaped rather than recomputed.
- `var.id`, `env.id`, `cardinals`, `days.id`, `time.window`, `names.window`, `quantiles` — Passed to `env_typing`.
- `id.names`, `by.interval`, `joint`, `joint.by.interval` — Passed to `env_typing`.
- `envirotype_mining`, `k.range`, `nstart`, `iter.max`, `seed` — Passed to `env_typing`.
- `combine.features`, `combine.max.order` — Passed to `env_typing`.
- `ratio` — logical. Convert counts to within-environment relative frequencies. Default `TRUE` – see Details.
- `center`, `scale` — logical. Passed to the shared scaler. **Both default to** `FALSE`, unlike `W_matrix` – see Details.
- `sd.tol`, `tol`, `QC`, `impute` — Passed to the shared scaler.
- `drop.constant` — logical. Drop envirotype columns that are constant across environments (they contribute nothing to a kernel). Default `TRUE`.
- `verbose` — logical.

Details

Why `ratio = TRUE` **by default.** Raw envirotype counts are days, so an environment with a longer season has larger counts in every column. A kernel built on raw counts is then partly a kernel on season length. Row-normalising makes each environment a composition – the share of its season spent in each envirotype – which is what "envirotype frequency" is meant to convey.

Why `center = scale = FALSE` **by default.** This deliberately differs from `W_matrix`, where `center = scale = TRUE`. After `ratio = TRUE` the columns are already on a common [0, 1] scale and each row sums to 1. Scaling a composition to unit variance per column inflates rare envirotypes – a type present in 2% of days in one environment and 0% elsewhere becomes a high-leverage column. If you want the `W_matrix` convention, set them explicitly; the choice is yours but it should be a choice.

Compositional caveat. With `ratio = TRUE` the rows sum to 1, so the columns are linearly dependent and the matrix is rank-deficient by exactly one. This is harmless for `env_kernel` (a Gaussian/linear kernel is still positive semi-definite) but will produce one zero eigenvalue. Do not "fix" it by dropping a column – that makes the result depend on which column you dropped.

Value

A numeric matrix of class `c("T_matrix", "matrix", "array")`, with `rownames` = environments and `colnames` = envirotypes. Attributes: `"typologies"` (the long frame), `"envirotype_description"`, `"mining_summary"`, `"plot"`, `"ratio"`, `"dropped"`, and the `.env_w_scale()` attributes when scaling is requested.

References

Costa-Neto, G., Crossa, J., & Fritsche-Neto, R. (2021). Enviromic assembly increases accuracy and reduces costs of the genomic prediction for yield plasticity in maize. *Frontiers in Plant Science* 12, 717552.

See Also

W_matrix for the covariable counterpart, env_typing for the underlying typology, env_kernel and env_cluster for the consumers.

Examples

```
data(maizeWTH)
Tm <- T_matrix(maizeWTH, var.id = c("T2M", "PRECTOT"), env.id = "env")
dim(Tm)
K <- env_kernel(env.data = Tm, is.scaled = TRUE)
```

---

tpe_weights *Target Population of Environments (TPE) Weights*

Description

Converts an env_risk_profile() into per-environment-type weights summing to one, so that selection indices, weighted GxE models and testing network decisions can be made proportional to how often each environment type actually occurs – rather than how often it happened to be trialled.

Usage

```
tpe_weights(
  risk,
  stage = NULL,
  var = NULL,
  by = c("class", "site"),
  measure = c("freq", "days"),
  drop.benign = FALSE,
  verbose = TRUE
)
```

Arguments

risk	An "env_risk" object from env_risk_profile().
stage	character. Restrict to one stage (e.g. "R1"). If NULL, weights are computed across all stages jointly.
var	character. Restrict to one variable. If NULL, classes from all variables are pooled, which is only sensible if they are disjoint.

by	character. "class" (default) weights environment TYPES; "site" weights SITES by how representative they are of the TPE.
measure	character. "freq" (probability of occurrence, default) or "days" (expected days, which weights by intensity).
drop.benign	boolean. Exclude classes named safe, optimal, none, non_limiting. Default FALSE.
verbose	boolean. Print the weight table.

Details

What "site weight" means. With by = "site" the weight is the share of the TPE's total stress-occurrence mass that a site contributes. A high weight means the site frequently expresses the stresses of interest, NOT that it is a good discriminator of genotypes – those are different criteria, and a site can be highly representative yet have poor heritability. Combine these weights with per-site repeatability before cutting a testing location.

Weights are as good as the record. They inherit every sampling limitation of the underlying profile, and they assume the historical climatology still holds.

Value

A data.frame with the grouping column, the raw measure, and weight summing to 1. Attribute "measure" records which measure was used.

See Also

env_risk_profile

Examples

```
## Not run:
## Requires an observed multi-year weather series (wth).
rp <- env_risk_profile(wth, planting = "10-15", crop = "maize")

## How much of the TPE is each drought class at flowering?
tpe_weights(rp, stage = "R1", var = "PETP")

## Which sites carry the most of the TPE's stress mass?
tpe_weights(rp, var = "PETP", by = "site", drop.benign = TRUE)

## End(Not run)
```

truncate_gxe_kernel *Truncate the GxE kernel to its structural rank.*

Description

The GxE kernel has a flat eigenvalue tail that behaves like an identity matrix and competes with the residual. Dropping it reduces sigma2_GE inflation (mean abs bias 0.333 -> 0.130) at essentially no cost in cross-validated prediction (CV2 0.678 full vs 0.669 truncated).

Usage

```
truncate_gxe_kernel(random, keep_prop = 0.75, pattern = "GE")
```

Arguments

random	a named list of kernels (the random argument of kernel_model).
keep_prop	numeric in (0, 1]. Proportion of the GxE eigenvalue mass to retain. Default 0.75.
pattern	character. Regular expression selecting the GxE kernels to truncate. Default "GE".

Details

CAUTION: truncation distorts the G:GxE ratio – rg moved to 0.72 vs a true 0.595 in testing. Use only when sigma2_GE specifically matters.

Any cached decomposition is dropped, since the kernel changes.

Value

The input random list of kernels, with the selected GxE kernels replaced by their reduced-rank (truncated) version and any cached spectral decomposition removed so the kernels are re-decomposed on the next fit. When no kernel matches pattern, or keep_prop >= 1, the list is returned unchanged.

undecompose_kernels *Drop cached decompositions*

Description

Use after modifying a kernel in place, to force recomputation.

Usage

```
undecompose_kernels(K)
```

Arguments

K	a kernel list from get_kernel.

Value

K with decompositions and metadata removed.

---

varcomp_summary *Validated variance components, heritabilities and genetic correlation.*

Description

var_total = var(y) (PHENOTYPIC; var(yHat) carries no residual and yields a NEGATIVE residual by subtraction). var_resid = var_total - (varG + varE + varGE) h2_entry = varG / (varG + varE/q + varGE/(p*q) + var_resid) varG is retained in the denominator; omitting it produced h2 > 1 even with true inputs (observed 1.025 and 0.983).

Usage

```
varcomp_summary(
  fit,
  method = c("postmean", "gibbs"),
  y = NULL,
  p = NULL,
  q = NULL,
  digits = 4
)
```

Arguments

fit	a fitted kernel_model object (or a compatible list carrying variance components).
method	character. "postmean" (default) uses posterior-mean effects; "gibbs" uses the stored Gibbs variance components.
y	numeric. Observed phenotypes used as the total-variance reference. Taken from fit$meta$y when NULL.
p	integer. Number of genotypes. Inferred from fit$meta$gid when NULL.
q	integer. Number of environments. Inferred from fit$meta$env when NULL.
digits	integer. Number of decimal places in the summary table. Default 4.

Value

An object of class "km_varcomp", a list with elements table (a data.frame of variance components with their absolute values and proportions of the phenotypic variance), the individual variances varG, varE, varGE, var_resid and var_total, the heritabilities h2_entry_mean and h2_plot, the genetic correlation rg, the counts p and q, the method used, and the logical flags negative_residual and few_environments. A dedicated print method displays the table and derived quantities.

W_matrix *Build the Matrix of Environmental Covariables for Reaction Norm*

Description

Estimates a weather covariable realized matrix **W** with dimensions $q \times k$, for q environments and k covariables. This matrix is the basic input of the reaction-norm and enviromic-assisted models in **EnvRtype** (env_kernel, env_cluster, env_target_importance, get_kernel).

Usage

```
W_matrix(
  env.data,
  is.processed = FALSE,
  id.names = NULL,
  env.id = NULL,
  var.id = NULL,
  probs = NULL,
  by.interval = NULL,
  time.window = NULL,
  names.window = NULL,
  center = TRUE,
  scale = TRUE,
  sd.tol = 10,
  statistic = NULL,
  tol = 0.001,
  QC = FALSE,
  impute = c("none", "mean", "median", "knn", "drop"),
  knn = 5L,
  max.cor = NULL,
  group = FALSE,
  cor.report = NULL,
  verbose = TRUE,
  copula = NULL,
  copula.args = list()
)
```

Arguments

`env.data`	data.frame of environmental variables from `get_weather`, or an already summarised object when `is.processed = TRUE`.
`is.processed`	boolean. Indicates whether the data.frame was previously processed with `summaryWTH` and already contains means, medians, etc.
`id.names`	character. Columns used as id for the environmental variables.
`env.id`	character. Column used as id for environments.
`var.id`	vector (character). Which variables will be used in the analysis.
`probs`	vector (numeric). Probability quantiles in $[0, 1]$, used when `statistic = 'quantile'`. If `NULL`, `c(0.25, 0.50, 0.75)`.
`by.interval`	boolean. Indicates if temporal intervals must be computed inside each environment. Default `FALSE`.
`time.window`	vector (numeric). If `by.interval = TRUE`, the temporal breaks.
`names.window`	vector (character). If `by.interval = TRUE`, the interval names.
`center`	boolean. Indicates whether the matrix should be centred. Default `TRUE`.
`scale`	boolean. If `TRUE`, variables assume $x \sim N(0, 1)$. Default `TRUE`.
`sd.tol`	numeric. Maximum standard deviation tolerated in quality control. Default 10.
`statistic`	vector (character). Statistic to be computed, `c('all','sum','mean','quantile')`. Default `'mean'`.
`tol`	numeric. Numerical tolerance; variables whose standard deviation is at or below `tol` are treated as near-constant and flagged for removal. Default 1E-3.
`QC`	boolean. Indicates whether Quality Control is applied. QC removes variables with `sd(x) > sd.tol` and near-constant variables.
`impute`	character. How to handle missing values before scaling: `"none"` (default, keep NAs), `"mean"` (column mean), `"median"` (column median), `"knn"` (k-nearest-environment imputation, see `knn`) or `"drop"` (remove columns with any NA).
`knn`	integer. Number of nearest environments used when `impute = "knn"`. Each missing cell is filled with the mean of that covariable over the `knn` most similar environments (Euclidean distance on the standardised, commonly observed covariables). Default 5.
`max.cor`	numeric in $(0, 1]$ or `NULL`. If not `NULL`, a greedy collinearity filter (same rule as `caret::findCorrelation`) drops covariables so that no pair of retained covariables has absolute Pearson correlation above `max.cor`. When two covariables are too correlated the one with the larger mean absolute correlation to the rest is removed. Unlike the `sd`-based rules this is an explicit opt-in and the collinear covariables are always dropped (independently of `QC`). Default `NULL` (no collinearity filtering).
`group`	boolean. Only used when `max.cor` is not `NULL`. If `TRUE`, collinear covariables are *grouped* instead of dropped: covariables are clustered by average-linkage hierarchical clustering on $1 - \lvert r \rvert$, the tree is cut at height $1-$`max.cor`, and each correlated block is replaced by its mean (a single composite covariable). The block-to-member mapping is returned in the `"groups"` attribute. Default `FALSE` (collinear covariables are dropped).

cor.report	NULL, TRUE, or a character path. If not NULL, a per-covariable collinearity diagnosis (its largest absolute correlation, the partner responsible, the final status, and the block it belongs to) is written as a CSV. Pass TRUE to write "W_matrix_collinearity.csv" in the current working directory, a directory to write that file there, or a full file path. Default NULL (no file written). The same table is always attached as the "collinearity" attribute.
verbose	boolean. If TRUE (default) prints quality-control messages.
copula	character or NULL. If not NULL, the covariable matrix is replaced by copula-based indices computed by env_copula. One of "none" (default, same as NULL), "pobs", "joint", "survival", "kendall" or "all". See env_copula for the meaning of each index.
copula.args	list. Extra arguments passed to env_copula, e.g. list(groups = list(heat = c("T2M_mean","T2M_MAX_mean")), ties.method = "average").

Details

Quality control follows Morais Junior et al. (2018): covariables whose standard deviation across environments exceeds sd.tol are discarded, as are near-constant covariables ($sd \leq tol$) which carry no information about environmental differences.

Three further, independent cleaning steps are available. Missing values can be imputed column-wise (impute = "mean"/"median") or from the most similar environments (impute = "knn") before any statistic is computed, which avoids silently shrinking the covariable set the way impute = "drop" does. Redundant covariables can be pruned with max.cor: enviromic matrices frequently contain near-duplicated columns (e.g. several temperature summaries), which inflate Euclidean geometry and relatedness kernels; the greedy filter keeps one representative per correlated block.

Unlike earlier versions, the numerical tolerance is *not* added to the data before scaling (which silently shifted unscaled outputs); it is used only to detect near-constant columns.

Value

An environmental covariable realized matrix with dimensions $q \times k$. The centring and scaling values, plus the list of removed markers, are attached as attributes ("scaled:center", "scaled:scale", "removed") so that new environments can be projected onto the same space. When quality control or collinearity filtering drop covariables, the reason for each removal is reported in the "removed.reason" attribute (a named character vector with values "too.variable", "near.constant" or "collinear"). The full collinearity diagnosis is attached as the "collinearity" data.frame attribute, and when group = TRUE the block-to-member mapping is attached as "groups".

Author(s)

Germano Costa Neto

References

Costa-Neto G., Galli G., Carvalho H.F., Crossa J., Fritsche-Neto R. (2021). EnvRtype: a software to interplay enviromics and quantitative genomics in agriculture. *G3* 11(4), jkab040.

See Also

summaryWTH, env_kernel, env_typing, T_matrix

Examples

```
data("maizeWTH")
env.data <- maizeWTH[maizeWTH$daysFromStart < 100, ]

## Mean-centred and scaled matrix (default statistic = 'mean')
W <- W_matrix(env.data = env.data)
dim(W)

## Adding time windows (one block of covariables per development stage)
W <- W_matrix(env.data = env.data, by.interval = TRUE,
              time.window = c(0, 14, 35, 60, 90, 120))

## Selecting the statistic to be used
W <- W_matrix(env.data = env.data, by.interval = TRUE, statistic = 'quantile',
              time.window = c(0, 14, 35, 60, 90, 120))

## With Quality Control based on the maximum sd tolerated
W <- W_matrix(env.data = env.data, QC = TRUE, sd.tol = 3)
attr(W, "removed")

## Dropping redundant (collinear) covariables and imputing missing cells
W <- W_matrix(env.data = env.data, impute = "knn", max.cor = 0.95)
attr(W, "removed.reason")

## Grouping collinear covariables into composite blocks + CSV diagnosis
W <- W_matrix(env.data = env.data, max.cor = 0.9, group = TRUE,
              cor.report = tempdir())
attr(W, "groups")

## Creating W for specific variables
W <- W_matrix(env.data = env.data, var.id = c('T2M_MAX', 'T2M_MIN', 'T2M'))

## Combining with summaryWTH by using is.processed = TRUE
data <- summaryWTH(env.data, env.id = 'env', statistic = 'quantile')
W <- W_matrix(env.data = data, is.processed = TRUE)
```

---

water_balance *Daily Soil Water Balance and Crop Water-Stress Indices*

Description

Runs a FAO-56 single crop coefficient soil water balance by combining daily weather from get_weather with soil hydraulic properties from get_soil, returning daily root-zone depletion, a water-stress coefficient, actual evapotranspiration, and a categorical stress level.

The stress coefficient Ks is the main output: bounded in [0, 1], dimensionless, and directly proportional to the reduction in transpiration.

Usage

```
water_balance(
  env.data,
  soil.data,
  env.id = "env",
  days.id = "daysFromStart",
  PREC = "PRECTOT",
  ETo = NULL,
  irrigation = 0,
  root.depth = 1000,
  root.init = 300,
  dap.root.max = 60,
  kc.stages = c(1, 20, 45, 90, 120),
  kc.values = c(0.4, 0.75, 1.15, 1.15, 0.6),
  p = 0.55,
  p.adjust = TRUE,
  initial.depletion = NULL,
  fc.var = "wv0033",
  pwp.var = "wv1500",
  convert.units = TRUE,
  verbose = TRUE
)
```

Arguments

env.data	data.frame. Daily weather, typically from get_weather or processWTH. Must contain the environment id, a day counter, precipitation, and a reference evapotranspiration column.
soil.data	data.frame. Soil properties from get_soil(wide = TRUE). Must include wv0033 (field capacity) and wv1500 (wilting point) for the layers spanning the root zone.
env.id	character. Environment-id column present in both inputs. Default "env".
days.id	character. Column giving days from planting. Default "daysFromStart".
PREC	character. Precipitation column (mm/day). Default "PRECTOT".
ETo	character. Reference evapotranspiration column (mm/day). If NULL (default), the function looks for "ETP" (added by param_atmospheric), then "EVPTRNS".
irrigation	numeric or character. Irrigation in mm/day: a single value applied daily, a vector as long as env.data, or the name of a column. Default 0.
root.depth	numeric. Maximum rooting depth in mm. Default 1000.
root.init	numeric. Rooting depth at emergence in mm. Default 300.
dap.root.max	numeric. Days after planting at which maximum rooting depth is reached. Default 60.

`kc.stages`, `kc.values`
: numeric. Days after planting and the corresponding crop coefficients, linearly interpolated between. Defaults describe a generic 120-day maize crop (FAO-56 Table 12). **Calibrate these.**

`p`
: numeric. Soil-water depletion fraction for no stress (FAO-56 Table 22). Default 0.55 for maize.

`p.adjust`
: logical. Adjust `p` daily for evaporative demand using FAO-56 Eq. 84. Default `TRUE`.

`initial.depletion`
: numeric. Root-zone depletion at planting in mm, or `NULL` (default) to start at field capacity.

`fc.var`, `pwp.var`
: character. SoilGrids property names for field capacity and wilting point. Defaults `"wv0033"` and `"wv1500"`.

`convert.units`
: logical. Apply ISRIC conversion factors to the soil data. Set `FALSE` only if already converted. Default `TRUE`.

`verbose`
: logical. Print progress messages. Default `TRUE`.

Details

Method. FAO-56 single crop coefficient approach (Allen et al. 1998, Ch. 8). Root-zone depletion is stepped forward daily:

$$D_{r,i} = D_{r,i-1} - P_i - I_i + ET_{a,i} + DP_i$$

bounded to $[0, TAW]$. Water in excess of field capacity becomes deep percolation; depletion cannot exceed total available water.

The stress coefficient follows FAO-56 Eq. 84:

$$K_s = 1 \quad \text{if } D_r \leq RAW$$

$$K_s = \frac{TAW - D_r}{TAW - RAW} \quad \text{if } D_r > RAW$$

so transpiration proceeds at the potential rate until readily available water is exhausted, then declines linearly to zero at wilting point.

Causal ordering. `Ks` on day i is computed from *yesterday's* depletion against *today's* capacity. Using the same day's closing depletion would let today's rainfall relieve stress that already reduced today's transpiration – a subtle look-ahead that systematically understates stress.

Units. SoilGrids serves integers in mapped units: `wv0033 = 314` means 0.314 cm3/cm3. Conversion is automatic; see `convert.units`.

Total available water is integrated across the SoilGrids layers intersecting the root zone, each weighted by the depth it contributes, not taken from a single layer. Because rooting depth grows over the season, TAW is recomputed daily (cached by unique depth for speed).

What this does not model. No dual crop coefficient, so soil evaporation is not separated from transpiration. No capillary rise, no lateral flow, and **no surface runoff** – all rainfall is assumed to infiltrate, which overestimates stored water and understates stress on steep, crusted or low-infiltration soils. Subtract runoff from the precipitation column beforehand if that matters. Irrigation is applied as given; the balance does not schedule it.

Interpreting the output. Regress `Ks` against yield: it is bounded, dimensionless, and scales transpiration directly. `depletion_frac` is easier to explain to agronomists. `stress_level` is a convenience for grouping and tabulation and should not be used as a modelling covariate – it discards magnitude.

Value

A data.frame with one row per environment-day, containing the input columns plus:

`TAW` total available water in the root zone (mm)

`RAW` readily available water, $p \times TAW$ (mm)

`Dr` root-zone depletion at end of day (mm)

`ASW` available soil water remaining, $TAW - D_r$ (mm)

`depletion_frac` D_r/TAW; 0 at field capacity, 1 at wilting point

`Ks` water-stress coefficient; 1 unstressed, 0 fully stressed

`ETc` crop evapotranspiration, standard conditions (mm/day)

`ETa` actual evapotranspiration, $K_s \times ET_c$

`deficit` $ET_c - ET_a$ (mm/day)

`drainage` deep percolation below the root zone (mm/day)

`irrigation` irrigation applied (mm/day)

`Kc` interpolated crop coefficient

`root_depth` rooting depth that day (mm)

`stress_level` ordered factor: none, mild, moderate, severe

References

Allen, R.G., Pereira, L.S., Raes, D. & Smith, M. (1998) Crop evapotranspiration: guidelines for computing crop water requirements. *FAO Irrigation and Drainage Paper 56*. FAO, Rome.

Pereira, L.S. et al. (2021) Standard single and basal crop coefficients for field crops. *Agricultural Water Management* 243, 106466. doi:10.1016/j.agwat.2020.106466

Doorenbos, J. & Kassam, A.H. (1979) Yield response to water. *FAO Irrigation and Drainage Paper 33*. FAO, Rome.

Steduto, P., Hsiao, T.C., Fereres, E. & Raes, D. (2012) Crop yield response to water. *FAO Irrigation and Drainage Paper 66*. FAO, Rome.

Poggio, L. et al. (2021) SoilGrids 2.0. *SOIL* 7, 217-240. doi:10.5194/soil72172021

Turek, M.E. et al. (2023) Global mapping of volumetric water retention at 100, 330 and 15000 cm suction using the WoSIS database. *International Soil and Water Conservation Research* 11(2), 225-239. doi:10.1016/j.iswcr.2022.08.001

Borg, H. & Grimes, D.W. (1986) Depth development of roots with time. *Transactions of the ASAE* 29(1), 194-197. doi:10.13031/2013.30125

See Also

`summary_water_balance` to condense the daily output, `get_weather` and `get_soil` for the inputs, `processWTH` to derive ETo, `soil_classification` for cross-site soil zoning.

Examples

```
## Not run:
## Requires live NASA POWER and SoilGrids services.
## ---------------------------------------------------------------
## 1. Minimal use
## ---------------------------------------------------------------
wth  <- get_weather(env.id = "NM", lat = -13.05, lon = -56.05,
                    start.day = "2015-02-15", end.day = "2015-06-15")
wth  <- processWTH(wth)                # adds ETP
soil <- get_soil(env.id = "NM", lat = -13.05, lon = -56.05,
                 variables.names = c("wv0033", "wv1500", "clay"))

wb <- water_balance(wth, soil)
head(wb[, c("env", "daysFromStart", "TAW", "Dr", "Ks",
            "deficit", "stress_level")])


## ---------------------------------------------------------------
## 2. Several environments at once
## ---------------------------------------------------------------
sites <- data.frame(
  env = c("SOR", "LON", "BAR"),
  lat = c(-12.5453, -23.3045, -12.1530),
  lon = c(-55.7113, -51.1696, -44.9900))

wth  <- processWTH(get_weather(env.id = sites$env, lat = sites$lat,
                               lon = sites$lon,
                               start.day = "2023-10-15",
                               end.day   = "2024-02-12"))
soil <- get_soil(env.id = sites$env, lat = sites$lat, lon = sites$lon,
                 variables.names = c("wv0033", "wv1500"))

wb <- water_balance(wth, soil, root.depth = 1000)
summary_water_balance(wb)


## ---------------------------------------------------------------
## 3. Irrigation: constant, per-day, or from a column
## ---------------------------------------------------------------
water_balance(wth, soil, irrigation = 5)              # 5 mm every day
water_balance(wth, soil, irrigation = rep(0, nrow(wth)))

wth$irr <- ifelse(wth$daysFromStart %in% 40:70, 8, 0)  # top-up at flowering
wb_irr  <- water_balance(wth, soil, irrigation = "irr")

## How much stress did irrigation remove?
c(rainfed   = sum(water_balance(wth, soil, verbose = FALSE)$deficit),
  irrigated = sum(wb_irr$deficit))


## ---------------------------------------------------------------
## 4. A different crop: soybean
```

```
## ---------------------------------------------------------------
## Kc and p come from FAO-56 Tables 12 and 22. Always check them
## against your own crop, cycle length and region.
wb_soy <- water_balance(
  wth, soil,
  kc.stages = c(1, 25, 50, 95, 120),
  kc.values = c(0.40, 0.80, 1.15, 1.15, 0.50),
  p         = 0.50,          # soybean, FAO-56 Table 22
  root.depth = 800)


## ---------------------------------------------------------------
## 5. Restricted root zone (compaction, shallow profile)
## ---------------------------------------------------------------
shallow <- water_balance(wth, soil, root.depth = 400, verbose = FALSE)
deep    <- water_balance(wth, soil, root.depth = 1200, verbose = FALSE)
c(shallow_TAW = max(shallow$TAW), deep_TAW = max(deep$TAW))
## A shallower profile stores less water and stresses sooner.


## ---------------------------------------------------------------
## 6. Dry planting: start part-depleted rather than at field capacity
## ---------------------------------------------------------------
wb_dry <- water_balance(wth, soil, initial.depletion = 60)


## ---------------------------------------------------------------
## 7. Season and growth-stage summaries
## ---------------------------------------------------------------
summary_water_balance(wb)

summary_water_balance(wb, by.interval = TRUE,
                      time.window  = c(0, 14, 35, 60, 90, 120),
                      names.window = c("P-E", "E-V1", "V1-V4",
                                       "V4-VT", "VT-GF", "GF-PM"))
## Stress at VT-GF (flowering to grain fill) costs far more yield
## than the same stress during vegetative growth.


## ---------------------------------------------------------------
## 8. Using the indices as envirotype covariates
## ---------------------------------------------------------------
W <- summary_water_balance(wb, by.interval = TRUE,
                           time.window = c(0, 30, 60, 90, 120))
wide <- reshape(W[, c("env", "interval", "mean_Ks")],
                idvar = "env", timevar = "interval", direction = "wide")
head(wide)   # one row per environment, ready to join to trial data

## End(Not run)
```

Index

∗ **datasets**
G2F_2014_17, 47
maizeG, 79
maizeWTH, 80
maizeYield, 80

aez_default_url, 4
aez_legend, 5, *49, 50*
as.data.frame.sim_met, 6
as.matrix.sim_envirome, 6
as.matrix.sim_W, 7

best_planting_date, 7, *106*

clay_5_15, 9
cluster_environments, 9
cor.test, *28*
coverage_summary, 12

decompose_kernels, 13, *58*

env_cluster, *11*, 14, *132*, *134*
env_copula, 15
env_cor, *9*, *11*, 18, *114*, *118–120*
env_cor_heatmap, 19, *23*
env_expand, 20
env_indices, *20*, 21, *32*
env_kernel, *7*, 24, *108*, *120*, *132–134*
env_loading_curve, 26, *32*
env_pc_associate, *27*, 28
env_pca, *20*, *23*, *27, 28*, 31, *33, 34*
env_pca_biplot, *32*, 33
env_pca_scree, *33*, 34
env_phenology, 35, *40*, *54*
env_risk_profile, *37*, 38, *100*, *135*
env_target_importance, 41
env_typing, *40*, 43, *132–134*

G2F_2014_17, 47
get_AEZ, *4, 5*, 48
get_bioclim, 51
get_climate_scenario, 52, *100*
get_elevation, 55
get_kernel, *13*, 56, *74*, *76*, *93*, *106, 107*, *109, 110*, *137*
get_soil, *48*, *50*, 59
get_soil_resumable, 61
get_spatial, 62
get_weather, *22*, *48*, *50*, *54*, 64, *66*, *68*
get_weather_hourly, 66, *66*
get_weather_resumable, 68
grid_scan, 70, *106*

hclust, *10, 11*

kernel_cv, 71
kernel_model, *9–11*, *13*, *56*, *58*, 72, *78, 79*, *104–106*, *111*, *114*
kernel_model_clustered, *11*, 77
kernel_model_mc, 78

maizeG, 79
maizeWTH, 80
maizeYield, 80
map_scan, 81, *106*

param_atmospheric, 85
param_radiation, 86
param_temperature, *37*, 88
phenology_templates, 89
planting_window_table, 90
plot, *91*
plot.sim_met, 91
plot.sim_W, 91
plot_planting_window, 92, *106*
prcomp, *31*
print.kernel_set, 93
print.scan_untested_envs, 93
print.sim_envirome, 94
print.sim_markers, 95
print.sim_met, 95

print.sim_W, 96
print.soil_gmm, 96
print.T_matrix, 97
processWTH, *22*, *68*, 98
project_risk, *40*, *54*, 99

read_progress_log, 100
restart_from_log, 101

scan_spatial_table, 103
scan_untested_envs, *93, 94*, 103, *111*, *114*
show_phenology *(*phenology_templates*)*, 89
sim_envirome, *7*, *94*, 107, *110*, *114*
sim_markers, *95*, *107, 108*, 109, *114*
sim_met, *6*, *18, 19*, *91*, *96*, *107–110*, 111, *117–120*
sim_met_C, *112*, *114*, 117, *120*, *122*
sim_W, *7*, *18, 19*, *92*, *96*, *108*, *111*, *114*, *117*, 118, *118*, *121, 122*
sim_W_grid, *120*, 121
soil_classification, 123
summary_water_balance, 128
summaryWTH, *37*, *68*, 130

T_matrix, *14, 15*, *24*, *26*, *46*, *97*, 132, *141*
tpe_weights, *40*, 134
truncate_gxe_kernel, 136

undecompose_kernels, *13*, 136
uniroot, *120*

varcomp_summary, 137

W_matrix, *7*, *37*, *120*, *132–134*, 138
water_balance, 141