

Package: mgcca (via r-universe)

October 3, 2026

Type Package

Title Generalized Canonical Correlation Analysis with Missing Individuals

Version 0.99.6

Date 2026-10-02

Description Implements Generalized Canonical Correlation Analysis (GCCA) with missing individuals (whole missing rows; van de Velden and Bijmolt, 2006) for multi-block omics integration. The package operates directly on HDF5 matrices via the BigDataStatMeth C++ API to scale computations; R functions are thin wrappers around the C++ core.

Depends R (>= 4.5.0)

Imports BigDataStatMeth (>= 2.0.6), MASS, rlang (>= 1.1.0), withr, Rcpp (>= 1.0.6), ggplot2 (>= 3.4.0), ggthemes, impute, SummarizedExperiment, graphics, grDevices, grid, methods, stats, utils

LinkingTo Rcpp, RcppEigen, Rhdf5lib, BigDataStatMeth, BH

Suggests BiocStyle, knitr, rmarkdown, testthat (>= 3.0.0), MultiAssayExperiment, Biobase, wordcloud, GenomicRanges, IRanges, BiocGenerics, made4, RSpectra, GenomeInfoDb, curatedTCGADData, TCGAutils, TxDb.Hsapiens.UCSC.hg19.knownGene, org.Hs.eg.db

SystemRequirements GNU make, C++17

License MIT + file LICENSE

URL <https://isglobal-brge.github.io/mgcca/>,
<https://github.com/isglobal-brge/mgcca>,
<http://brge.isglobal.org>

BugReports <https://github.com/isglobal-brge/mgcca/issues>

biocViews StatisticalMethod, DimensionReduction, PrincipalComponent, GeneExpression, Transcriptomics, Epigenetics, Metabolomics

VignetteBuilder knitr
Config/testthat/edition 3
Encoding UTF-8
RoxygenNote 7.3.3
NeedsCompilation yes
Config/pak/sysreqs make libicu-dev zlib1g-dev
Repository <https://biocstaging.r-universe.dev>
Date/Publication 2026-10-03 11:29:27 UTC
RemoteUrl <https://github.com/BiocStaging/mgcca>
RemoteRef HEAD
RemoteSha 4476903d1eae19f5619cfb33732df3cd4035213d

Contents

cardiovascular	3
corTestP	4
genCoords	5
getSignif	6
getTables	8
matrixChr2Num	9
mgcca	10
mgcca_associate	13
mgcca_audit	14
mgcca_audit_load	18
mgcca_block_profile	19
mgcca_import_hdf5	20
mgcca_load	22
mgcca_permtest	23
mgcca_results	24
mgcca_select_lambda	26
mgcca_sensitivity	30
mgcca_stability	34
mgccaImpute	37
plot.mgcca	38
plot.mgcca_audit	39
plot.mgcca_block_profile	42
plot.mgcca_sensitivity	43
plotAVE	45
plotBiplot	46
plotIndividuals	47
plotInds	48
plotLoadings	50
plotScores	51
plotScree	52

plotVariables	53
plotVars	54
predict.mgcca	56
print.ListMAE	57
print.mgcca	58
print.mgcca_block_profile	59
print.mgcca_lambda	60
summary.mgcca	61
topVars	62
Index	64

cardiovascular	<i>Simulated cardiovascular multi-table example data</i>
----------------	--

Description

A small, self-contained example for [mgcca](#): three simulated tables measured on partly overlapping sets of individuals, illustrating the missing-individuals setting [mgcca](#) is designed for. Loading the data set creates three data frames, X1, X2 and X3, in the calling environment.

The data are simulated. They are drawn from aggregate summary statistics of a real cardiovascular methylation study (see *Source*) and mimic its correlation and missingness structure, but no individual in the data set corresponds to a real participant. The tables are meant for illustration and testing only; they support no substantive conclusion about methylation, cardiovascular risk or blood composition.

Usage

`data(cardiovascular)`

Format

- Three numeric data frames with individuals in rows and variables in columns:
- X1 `data.frame` with 618 rows and 69 columns (simulated methylation-derived scores).
 - X2 `data.frame` with 633 rows and 6 columns (simulated clinical measurements).
 - X3 `data.frame` with 645 rows and 4 columns (simulated cell-type proportions).

Details

Each table has individuals in rows – identified by synthetic identifiers `id0001–id0646` shared across tables – and variables in columns; every column is numeric and there are no missing values within a table. The three tables do not share exactly the same individuals: the union is 646 individuals and only 606 are common to all three, which is the incomplete-blocks situation [mgcca](#) handles:

- X1 618 x 69 data frame of simulated standardized methylation-derived scores (columns `cg1–cg69`), each centred and scaled.

X2 633 x 6 data frame of simulated clinical measurements: TC (total cholesterol), LDL, HDL, BMI, waist (waist circumference) and Gluc (glucose).

X3 645 x 4 data frame of simulated blood cell-type proportion estimates: CD4T, NK, Mono and Gran. These are four components of a six-type composition; the remaining two were set aside, so the four columns do not sum to one by construction.

[mgcca](#) expects numeric matrices, so coerce the tables with `as.matrix()` before fitting (see the examples).

Source

Simulated data, generated by the script `system.file("scripts", "make-cardiovascular.R", package = "mgcca")` from aggregate summary statistics – a joint correlation matrix, per-variable marginal distributions and the sizes of the missing-value pattern – of a real, de-identified cardiovascular methylation study that cannot be redistributed. Those aggregates are stored alongside the script in `cardiovascular-parameters.rds`; no individual-level information from the source study is retained in the package.

Examples

```
data(cardiovascular)
X <- list(methylation = as.matrix(X1),
         clinical     = as.matrix(X2),
         other        = as.matrix(X3))
vapply(X, dim, integer(2))
```

corTestP

Two-sided p-value of a correlation coefficient

Description

Compute the two-sided p-value of a Pearson correlation coefficient from the coefficient and the sample size, without needing the underlying data. Used internally by [mgcca_results](#) to turn the between-table correlations into p-values, and exported because it is useful on its own.

Usage

```
corTestP(r, n)
```

Arguments

r	Correlation coefficient.
n	Sample size (≥ 3).

Details

This is a plain function, not an S3 method: `stats::cor.test` is not a generic, and the name carries no dot so that it cannot be read as one.

Value

A numeric vector of the same length as `r` with the two-sided p-values from a t distribution on $n - 2$ degrees of freedom.

Examples

```
corTestP(0.35, 100)
corTestP(c(0.1, 0.5, 0.9), 50)
```

genCoords	<i>Subset a MultiAssayExperiment object by specific genomic coordinates</i>
-----------	---

Description

Keeps, in every assay of a `MultiAssayExperiment`, only the features that fall inside a genomic region of interest. Assays stored as a `RangedSummarizedExperiment` are filtered on their `rowRanges` (features contained *within* the region); the assay indicated by `meth.index` – typically methylation, held in a plain `SummarizedExperiment` without ranges – is filtered on a numeric position column of its `rowData`. The resulting feature names are then used to subset the whole object row-wise.

Usage

```
genCoords(
  multiassayexperiment,
  gen.coords,
  meth.index,
  col.name,
  seqname.col = NULL
)
```

Arguments

<code>multiassayexperiment</code>	a <code>MultiAssayExperiment</code> object whose assays carry annotation (<code>rowRanges</code> , or <code>rowData</code> for the assay pointed at by <code>meth.index</code>).
<code>gen.coords</code>	a <code>GRanges</code> object holding <i>one</i> region. Several ranges are an error: pass them one at a time.
<code>meth.index</code>	is the index where the methylation data (or a <code>SummarizedExperiment</code> non <code>Ranged</code>) is located inside the mae.
<code>col.name</code>	name of the <code>rowData</code> column of the assay pointed at by <code>meth.index</code> that holds the feature position, in the same coordinate system as <code>gen.coords</code> .
<code>seqname.col</code>	name of the <code>rowData</code> column of that same assay that holds the chromosome, so that features at the same position on another chromosome are dropped. The default <code>NULL</code> looks for a column called (case-insensitively) <code>seqnames</code> , <code>seqname</code> , <code>chr</code> , <code>chrom</code> or <code>chromosome</code> ; when none is found the position filter is applied on its own and a warning is issued.

Value

A MultiAssayExperiment with the same assays as multiassayexperiment, each restricted to the features lying inside gen.coords.

Examples

```
if (requireNamespace("MultiAssayExperiment", quietly = TRUE) &&
    requireNamespace("SummarizedExperiment", quietly = TRUE) &&
    requireNamespace("GenomicRanges", quietly = TRUE)) {
  library(MultiAssayExperiment)
  library(SummarizedExperiment)
  library(GenomicRanges)

  data(cardiovascular)
  sel <- rownames(X2)[1:20]

  ## a non-ranged assay (methylation): positions live in rowData, and the
  ## chromosome is read from the 'chr' column without having to name it
  m1 <- t(as.matrix(X1[rownames(X1) %in% sel, 1:8, drop = FALSE]))
  se1 <- SummarizedExperiment(
    assays = list(beta = m1),
    rowData = DataFrame(position = seq(1000, 8000,
                                     length.out = nrow(m1)),
                        chr = rep(c("chr1", "chr2"),
                                length.out = nrow(m1)),
                        row.names = rownames(m1)))

  ## a ranged assay: positions live in rowRanges
  m2 <- t(as.matrix(X3[rownames(X3) %in% sel, , drop = FALSE]))
  gr <- GRanges(seqnames = "chr1",
                ranges = IRanges(start = c(1200, 2200, 3200, 9000),
                                width = 50))

  names(gr) <- rownames(m2)
  rse2 <- SummarizedExperiment(assays = list(expr = m2), rowRanges = gr)

  mae <- MultiAssayExperiment(
    ExperimentList(list(methylation = se1, cells = rse2)))

  region <- GRanges(seqnames = "chr1", ranges = IRanges(start = 1000,
                                                        end = 5000))
  sub <- genCoords(mae, region, meth.index = 1, col.name = "position")
  dim(sub[[1]]) # methylation, region and chromosome
  dim(sub[[2]]) # cells restricted to the region
}
```

Description

Collects, from the per-table p-values stored in `x$pval.cor`, the variables whose correlation with *at least one* shared canonical component is significant at level `pval.cut`. The fit must therefore carry p-values (see [mgcca](#)).

Usage

```
getSignif(x, df = NA, pval.cut = 0.05, ...)
```

Arguments

<code>x</code>	an object of class 'mgcca'
<code>df</code>	tables to inspect: a vector of positions in <code>x\$pval.cor</code> or of table names. The default NA means all tables.
<code>pval.cut</code>	significance level. A variable is reported when any of its component p-values is strictly below this value. Default 0.05.
<code>...</code>	further arguments, currently ignored.

Value

A data.frame with one row per selected variable and two character columns: `variable` (the variable name) and `table` (the table it comes from). A variable significant in more than one table appears once per table.

See Also

[topVars](#), [plotVariables](#)

Examples

```
data(cardiovascular)
u <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))
sel <- u[1:150]
mk <- function(d, cols = NULL) {
  m <- as.matrix(d[rownames(d) %in% sel, , drop = FALSE])
  if (!is.null(cols)) m <- m[, cols, drop = FALSE]
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = mk(X1, 1:20), clinical = mk(X2), other = mk(X3))
fit <- mgcca(X, nfac = 2, method = "penalized", lambda = rep(0.1, 3))

sig <- getSignif(fit, pval.cut = 0.01)
head(sig)
table(sig$table)

## only the clinical table
getSignif(fit, df = "clinical", pval.cut = 0.01)

## or several tables at once, by position
```

```
table(getSignif(fit, df = c(2, 3), pval.cut = 0.01)$table)
```

getTables

Splits a MultiAssayExperiment in a list of tables

Description

Turns the assays of a MultiAssayExperiment into the list of tables [mgcca](#) expects. Assays are stored with features in rows and individuals in columns, so each one is transposed on the way out: the tables returned have individuals in rows and variables in columns.

Usage

```
getTables(multiassayexperiment)
```

Arguments

```
multiassayexperiment
      MultiAssayExperiment
```

Value

A list of matrices, one per assay, named after the assays and each transposed to individuals-by-variables. The list carries class "ListMAE", which [matrixChr2Num](#) requires.

See Also

[matrixChr2Num](#), [mgcca](#)

Examples

```
if (requireNamespace("MultiAssayExperiment", quietly = TRUE)) {
  data(cardiovascular)
  sel <- rownames(X2)[1:20]
  a1 <- t(as.matrix(X1[rownames(X1) %in% sel, 1:8, drop = FALSE]))
  a2 <- t(as.matrix(X2[rownames(X2) %in% sel, , drop = FALSE]))
  mae <- MultiAssayExperiment::MultiAssayExperiment(
    MultiAssayExperiment::ExperimentList(
      list(methylation = a1, clinical = a2)))

  tabs <- getTables(mae)
  names(tabs)
  vapply(tabs, dim, integer(2))  # individuals x variables
}
```

matrixChr2Num	<i>Convert a character table in a list of tables to numeric</i>
---------------	---

Description

Some assays (e.g. methylation from TCGA) are stored as character matrices; mgcca needs numeric tables. This converts the table at position index in a list of tables to numeric, preserving dimnames.

Usage

```
matrixChr2Num(listMAE, index)
```

Arguments

listMAE	a list of tables (e.g. from getTables).
index	integer position of the table to convert.

Value

The input list of tables (still of class "ListMAE") with the table at position index replaced by a numeric matrix of the same dimensions, row names and column names. All other tables are returned unchanged.

See Also

[getTables](#)

Examples

```
if (requireNamespace("MultiAssayExperiment", quietly = TRUE)) {
  data(cardiovascular)
  sel <- rownames(X2)[1:20]
  ## two assays, features in rows, individuals in columns
  a1 <- t(as.matrix(X1[rownames(X1) %in% sel, 1:8, drop = FALSE]))
  a2 <- t(as.matrix(X2[rownames(X2) %in% sel, , drop = FALSE]))
  ## a character assay, as methylation downloaded from TCGA often is
  storage.mode(a1) <- "character"
  mae <- MultiAssayExperiment::MultiAssayExperiment(
    MultiAssayExperiment::ExperimentList(
      list(methylation = a1, clinical = a2)))

  tabs <- getTables(mae)
  class(tabs[[1]][1, 1]) # "character"
  tabs <- matrixChr2Num(tabs, 1)
  class(tabs[[1]][1, 1]) # "numeric"
}
```

mgcca

*Generalized Canonical Correlation Analysis with missing individuals***Description**

High-level, one-call entry point for the C++/HDF5 MGCCA pipeline: import any supported input into HDF5, then run the single-call orchestrator [mgcca_rcpp](#). Results are written to the HDF5 file under FINAL_RESULTS (Y, corsY, pval, AVE, and scores if requested); with `collect = TRUE` they are also returned in memory as an object of class "mgcca" ready for [plotIndividuals](#) / [plotVariables](#).

Usage

```
mgcca(
  x,
  filename = NULL,
  group = "MGCCA_IN",
  datasets = NULL,
  nfac = 2,
  scale = TRUE,
  method = "penalized",
  lambda = NULL,
  scores = FALSE,
  route = c("auto", "cov", "dual"),
  threads = NULL,
  overwrite = TRUE,
  collect = TRUE,
  outputs = NULL
)
```

Arguments

x	Input data: an HDF5 file path (data already under group), a <code>MultiAssayExperiment</code> , an <code>ExpressionSet</code> / <code>SummarizedExperiment</code> , or a named list of matrices (individuals x variables, rownames = individual IDs).
filename	Target HDF5 file. When given (the default behaviour up to and including 1.3.0), the whole pipeline runs out-of-core on that file. When <code>NULL</code> , mgcca fits <i>in memory</i> instead: the tables are held in RAM and the estimator runs in pure R, with no HDF5 file written. The in-memory path is meant for tables that fit in RAM; it accepts a list of matrices, a <code>MultiAssayExperiment</code> , a <code>SummarizedExperiment</code> or an <code>ExpressionSet</code> (all materialised in RAM), but not an HDF5 path as x (that requires a filename). Ignored when x is itself an HDF5 path. The presence of filename <i>is</i> the choice of backend – there is no automatic fallback by size.
group	Input HDF5 group. Default "MGCCA_IN".
datasets	Optional dataset names (subset/order).

<code>nfac</code>	Number of shared components. Default 2.
<code>scale</code>	If TRUE (default), column-center+scale each table before analysis, with <code>base-R scale()</code> semantics (sd with the $n-1$ divisor), over the individuals present in each table. On the file-backed path (<code>filename</code> given) the scaling is performed <i>inside</i> the HDF5 file (out-of-core) after a raw, untransformed import, so it applies uniformly to every input type – including inputs that are already an HDF5 file – and never loads a full table into RAM. On the in-memory path (<code>filename = NULL</code>) it is done in RAM on the materialised tables.
<code>method</code>	Inversion method: "solve" (SPD Cholesky), "penalized" (requires <code>lambda</code>), or "geninv"/"ginv" (Moore-Penrose pseudoinverse).
<code>lambda</code>	Numeric vector (length = number of tables) for <code>method = "penalized"</code> .
<code>scores</code>	If TRUE, also compute per-table weights and scores. A table's scores are standardised over the individuals <i>present in that table</i> , and individuals absent from it get NA (NaN in the HDF5 file) rather than a score of zero. Only the shared components Y are defined for every individual of the union.
<code>route</code>	Per-table algebra route: "auto" (default) picks per table by $\min(n, p)$ (covariance $X'X$) when $p < n$, Gram XX' when $p \geq n$; "cov" or "dual" force one route for all tables. The Gram/dual route is the one that scales to $p \gg n$ (e.g. full methylation) without ever forming a $p \times p$ matrix.
<code>threads</code>	Optional thread count.
<code>overwrite</code>	If TRUE (default), overwrite the file/datasets on import.
<code>collect</code>	If TRUE (default), read the results back from HDF5 and return an in-memory object of class "mgcca" (via <code>mgcca_results</code>) ready for <code>plotIndividuals</code> / <code>plotVariables</code> / <code>getSignif</code> ; the HDF5 descriptor is kept on its "desc" attribute. With <code>collect = FALSE</code> the lightweight descriptor list is returned instead (results still live in the HDF5 file). On the in-memory path (<code>filename = NULL</code>) <code>collect = FALSE</code> has no meaning – there is no file to leave the results in – and is refused with a clear error.
<code>outputs</code>	Which components to collect when <code>collect = TRUE</code> . NULL (default) collects all of them – exactly the behaviour of every previous version. A character subset of "Y", "corsY", "scores", "pval", "weights", "scaling", "AVE", "eigen", "overlap" skips the HDF5 reads behind the rest; what you do ask for is bit-for-bit identical to a full collection, because the choice only changes what is read back, never what was computed. Ignored when <code>collect = FALSE</code> . See <code>mgcca_results</code> .

Value

With `collect = TRUE` (default), an object of class "mgcca" (see `mgcca_results`); its descriptor is on `attr(, "desc")`. With `collect = FALSE` (file-backed only), the descriptor list (`filename`, `datasets`, `nfac`, `m`, `eig_values`, `route`, ...); results still live in the file under `FINAL_RESULTS`. An in-memory fit (`filename = NULL`) returns the same-shaped "mgcca" object, with a descriptor that reports `backend = "memory"` and `filename = NA`; such a fit is not written to disk and is therefore not reloadable with `mgcca_load`, and the reliability analyses (`mgcca_sensitivity`, `mgcca_stability`) require a file-backed fit.

Tables need not share the same individuals: they are aligned on the union of the row names. Per-table quantities are computed on the individuals a table actually holds – the p-values in `pval` carry that table's $n_{\text{present}} - 2$ degrees of freedom, and its scores are standardised over the same individuals.

Numerical identifiability

The fit carries the external spectral gap $\mu_L - \mu_{L+1}$ of the mgcca operator (`$eigen`, and `attr(, "desc")$eigen`), which is what separates "the solver converged" from "the leading subspace is identified". With `method = "solve"` the two can come apart: in the regime where a table has about as many variables as it has observed rows, the top eigenvalues coincide, the gap falls to machine zero and the directions the solver returns are arbitrary – while the solver reports success. When that happens mgcca emits a warning of class `"mgcca_degenerate_subspace"` recommending `method = "penalized"`, which restores the separation. The warning never changes or aborts the fit.

See Also

[mgcca_results](#), [plotIndividuals](#), [mgcca_import_hdf5](#), [mgcca_rcpp](#)

Examples

```
data(cardiovascular)
ids <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))[1:150]
num <- function(d, cols = seq_len(ncol(d))) {
  m <- as.matrix(d[rownames(d) %in% ids, cols, drop = FALSE])
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = num(X1, 1:20), clinical = num(X2), other = num(X3))

## In-memory fit: no 'filename', so nothing is written to disk.
fit <- mgcca(X, nfac = 2, method = "penalized", lambda = rep(0.1, 3),
  scores = TRUE)

fit
head(fit$Y)
fit$AVE$AVE_outer_model

## The same call with a 'filename' runs out-of-core and leaves a
## self-describing HDF5 results file behind (see mgcca_load()).

h5 <- tempfile(fileext = ".h5")
fit2 <- mgcca(X, filename = h5, nfac = 2, method = "penalized",
  lambda = rep(0.1, 3))
attr(fit2, "desc")$eig_values
unlink(h5)
```

mgcca_associate	<i>Associate the shared components with external phenotypes</i>
-----------------	---

Description

Regresses each shared component on one or more external variables (a phenotype, a clinical group, a batch, ...) and reports how much of the component each explains (R^2) with a p-value. This answers "do the integrated axes capture this variable?" – useful to interpret an mgcca result against clinical or technical annotation.

Usage

```
mgcca_associate(x, phenotypes, comps = NULL)
```

Arguments

x	an mgcca object (from mgcca / mgcca_results).
phenotypes	a single vector (factor or numeric), a named list of such vectors, or a data.frame. Named vectors / data-frame row names are matched to the individuals of x; otherwise they are assumed to be in the same order.
comps	components to test. Default: all.

Value

A data.frame with one row per (phenotype, component): phenotype, component, R2, p, n (non-missing individuals) and p.adj (Benjamini-Hochberg across all rows).

See Also

[plotIndividuals](#), [mgcca_permtest](#)

Examples

```
data(cardiovascular)
ids <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))[1:150]
num <- function(d, cols = seq_len(ncol(d))) {
  m <- as.matrix(d[rownames(d) %in% ids, cols, drop = FALSE])
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = num(X1, 1:20), clinical = num(X2), other = num(X3))
fit <- mgcca(X, nfac = 2, method = "penalized", lambda = rep(0.1, 3))

## An external annotation, matched to the individuals by row name. Here it
## is simulated, so the components are not expected to capture it.
set.seed(1)
pheno <- data.frame(
  group = factor(sample(c("case", "control"), nrow(fit$Y), TRUE)),
```

```

age = rnorm(nrow(fit$Y), 55, 8),
row.names = rownames(fit$Y))

a <- mgcca_associate(fit, pheno)
a
a[a$p.adj < 0.05, ]

```

mgcca_audit

Pre-fit availability and design audit

Description

Audits the availability design of a multi-block study *before* anything is fitted: how many individuals each block and each declared stratum actually holds, which pairs of blocks share enough individuals to be supported directly, which pairs are reachable only through other blocks, how fragile those routes are, and how concentrated any declared availability weights are. It reads presence and absence, the declared strata and the declared weights, and nothing else: it never calls [mgcca](#), never reads a data value, and cannot change any quantity a fit produces.

Usage

```

mgcca_audit(
  data,
  L,
  strata = NULL,
  weights = NULL,
  design_lines = list(ess_min = 30, availability_min = 0.05, max_weight = 20),
  save_hdf5 = FALSE
)

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

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

```

Arguments

- | | |
|------|--|
| data | The design to audit. Either an $n \times J$ logical (or 0/1) availability matrix or data frame (columns named for the blocks, TRUE where the individual is present), or the very object mgcca would fit: a named list of matrices, a <code>MultiAssayExperiment</code> , a <code>SummarizedExperiment</code> , an <code>ExpressionSet</code> , a path to an HDF5 file holding the blocks, or the descriptor list (filename, group, datasets) returned by mgcca_import_hdf5 . A bare path carries no declared order or subset of the blocks, so the file's own listing decides both – as it does for mgcca ; pass the descriptor to fix them. |
| L | Declared rank of the working model, an integer ≥ 1 . It is recorded and compared against the support counts descriptively; nothing is fitted at rank L here. |

strata	Optional declared strata, a factor or vector of length n , in the row order of the audit (<code>\$individuals</code>). Availability, representation and the edge rule are then reported within every stratum. Default NULL: one stratum, the whole table.
weights	Optional user-declared availability weights, a non-negative numeric vector of length n , in the same row order. They are described (ESS, maximum, concentration) and used for the additional ESS-based support column. They are never applied to data, and never replace the raw counts.
design_lines	Declared lines, with margin: <code>ess_min</code> (minimum effective count, default 30, applied to stratum ESS and to pair support), <code>availability_min</code> (minimum realised availability, default 0.05) and <code>max_weight</code> (largest declared weight tolerated in a cell, default 20). A partial list is merged with the defaults.
save_hdf5	If TRUE and data is HDF5-backed, write the complete audit into an <code>availability_audit</code> group of that same file, so the certificate travels with the data, and read it back whole with <code>mgcca_audit_load</code> . The encoding is structured and inspectable rather than a blob: numeric tables as datasets, block names, individual IDs and stratum labels as their dimnames, status codes as integers with the frozen vocabulary stored beside them as a legend, transfer routes as sequences of integer block indices. Rewriting is idempotent. The returned object is the same either way, and is <code>saveRDS</code> -able.
x	An object of class "mgcca_audit", as returned by <code>mgcca_audit()</code> .
object	An object of class "mgcca_audit", as returned by <code>mgcca_audit()</code> .
...	Ignored.

Details

It audits the object the fit will consume. Besides a ready-made availability table, data accepts the same inputs `mgcca` takes in either mode – an in-memory list of blocks (or a `MultiAssayExperiment`, `SummarizedExperiment` or `ExpressionSet`), or an HDF5 file holding them – and derives the availability table itself through the package’s own readers, with the same union-of-IDs rule the estimator uses. Only the individual IDs are read; no block data is materialised, on either route. The union order is the order the rows of the audit are in, and `$individuals` reports it, so declared `strata` and `weights` can be lined up with it.

Support graph. Blocks j and k are joined by an edge when the realised joint availability $\hat{\pi}_{jk,c}$ reaches `design_lines$availability_min` in *every* declared stratum c . With no strata declared there is one stratum, the whole table. A pair that clears the line in the sample as a whole but has an empty joint cell in one declared stratum gets no edge, and the per-stratum table shows which stratum emptied it.

Pair codes (frozen vocabulary, and a partition of the pairs):

`DIRECT_SUPPORT` the pair has its own edge: individuals observed in both blocks carry it.

`MODEL_TRANSFER_ELIGIBLE` no edge, but a route of edges runs from j to k . The pair is then identified conditional on the declared rank-L model and its required structural conditions, which are not empirically checked by this audit. `$provenance` names the route, the support of every edge on it, and its bottleneck.

`UNSUPPORTED` no edge and no route, but the pair does have jointly observed individuals – too few, or absent from a declared stratum.

STRUCTURAL_ZERO no edge, no route, and not one individual anywhere is observed in both blocks. The joint cell is empty by design.

Advisory codes are additive and carry no verdict: LOW_SUPPORT (the support the pair leans on sits below the declared `ess_min` line), SINGLE_BRIDGE ($\kappa_{jk} = 1$: one edge carries the whole connection) and SEVERE_REPRESENTATION_WARNING (a stratum-block cell crosses one of the declared representation lines). The value that raised a code is always reported beside it.

Design-support fragility. κ_{jk} is the number of edge-disjoint routes between two blocks, equivalently the smallest number of edges whose loss would separate them. It measures the *topological redundancy of the design's support* – how many independent ways the design has of connecting two blocks – and it is silent about how well anything is estimated. A bridge is an edge with $\kappa = 1$: lose those individuals and the two sides stop being connected at all.

Support counts are descriptive. The default edge-support metric is the raw joint count n_{jk} . When weights are declared, an ESS-based support is reported *in addition*, in its own column; it never replaces the raw count. Comparing either against L is descriptive support and nothing more: $n_{jk} \geq L$ does not prove that the corresponding cross-block quantity has numerical rank L .

Positivity. A finite entry in the availability table *evidences* positivity for that cell – somebody was observed there. It does not prove it: an empty cell may be empty by chance or impossible by design, and the table cannot tell the two apart.

Design lines are declared, with margin. `ess_min`, `availability_min` and `max_weight` are conventional lines set by the analyst, not thresholds at which identification fails and not quantities estimated from the data. Everything the audit reports is continuous: the object carries the value (an ESS of 23.7, say) together with any code it raised, and no design makes the audit refuse to return one.

Structural checks are not run. `$structural_checks` is always "not_evaluated". Empirical rank and conditioning checks on the blocks and on the transfer routes are a different, later question; the field is present so that a consumer can never confuse "checked and fine" with "not checked".

Among the pre-fit diagnostics we surveyed for multi-block methods with missing individuals, we found none reporting the transfer route and its bottleneck alongside the representation of the declared strata.

`print()` gives the audit as a certificate: the source, the sizes, the declared strata, weights and design lines, the support graph with its bridges and articulation blocks, the tally of pair codes, and every pair or stratum-block cell that raised an advisory, each shown beside the value that raised it. It closes by restating that `structural_checks` is "not_evaluated": nothing printed is a verdict on identification.

`summary()` prints all of that and then the full tables: the availability by block and by declared stratum, every pair with its supports, route and bottleneck, and the representation table. A finite entry there evidences positivity for that cell without proving it, and the printed note says so.

Value

An object of class "mgcca_audit", a list with

`$availability` by_block and by_stratum (n_{jc} and $\hat{\pi}_{jc}$), pairs (n_{jk}) and pairs_by_stratum ($n_{jk,c}$; NULL when no strata are declared).

\$graph adjacency, edges, component, n_components, connected, complete, kappa, bridges and articulation – design-support fragility and topological redundancy, not identification robustness.

\$provenance one row per pair with its code, supports, κ , best route, bottleneck and fragility; \$paths holds the route of every connected pair edge by edge; \$rank_assumption states what transfer is conditional on.

\$representation per declared stratum and block: observed proportion, Kish ESS (only when weights are declared), maximum weight, weight concentration and status.

\$status machine-readable codes, \$pairs and \$strata.

\$structural_checks always "not_evaluated".

plus \$table (the availability table it worked from), \$individuals, \$blocks, \$L, \$design_lines, \$source and \$saved.

print() returns its argument invisibly and summary() returns object invisibly; both are called for what they write to the console.

See Also

[mgcca](#) for the estimator this audit deliberately does not touch, and [mgcca_audit_load](#) for reading a stored audit back out of the file it was written into.

Examples

```
# A design in which one block is only reachable through another: 40
# individuals carry methylation and expression, 60 carry expression and
# proteins, and nobody carries methylation and proteins together.
av <- cbind(methylation = rep(c(TRUE, FALSE), c(40, 60)),
            expression  = rep(TRUE, 100),
            proteins     = rep(c(FALSE, TRUE), c(40, 60)))
a <- mgcca_audit(av, L = 2)
a

# methylation--proteins has no shared individual, but a route through
# expression, and the audit reports the route and its bottleneck.
a$provenance$pairs[, c("block1", "block2", "code", "path", "bottleneck")]

# Declared strata and declared weights: the audit adds the per-stratum
# representation of each block and the concentration of the weights.
st <- rep(c("cohort A", "cohort B"), each = 50)
w <- rep(c(1, 3), each = 50)
summary(mgcca_audit(av, L = 2, strata = st, weights = w))

# The same audit taken from the blocks themselves: the availability table is
# derived from the individual IDs, with mgcca's own union rule.
ids <- sprintf("ind%03d", 1:100)
blocks <- list(
  methylation = matrix(0, 40, 3, dimnames = list(ids[1:40], paste0("m", 1:3))),
  expression  = matrix(0, 100, 2, dimnames = list(ids, paste0("e", 1:2))),
  proteins    = matrix(0, 60, 2, dimnames = list(ids[41:100], paste0("p", 1:2))))
mgcca_audit(blocks, L = 2)$graph$component
```

mgcca_audit_load	<i>Read a stored availability audit back from HDF5</i>
------------------	--

Description

Restores, whole, the audit that [mgcca_audit](#) wrote into a file with `save_hdf5 = TRUE`. The result is the same "mgcca_audit" object, with the same tables, codes and routes.

Usage

```
mgcca_audit_load(file, group = .mgcca_av_group, check_inputs = TRUE)
```

Arguments

<code>file</code>	Path to the HDF5 file holding the audit.
<code>group</code>	Group the audit was written to. Default "availability_audit", which is where <code>mgcca_audit()</code> puts it.
<code>check_inputs</code>	If TRUE (default), also re-derive the audit from the inputs stored beside it and compare, so a file written by a different version of the audit is reported rather than trusted in silence. The object returned is the stored one either way.

Details

The group holds every table the audit produced, as double datasets with their names carried in the `dimnames` and the frozen vocabulary stored beside the integer codes as a legend, plus the audit's five inputs – the availability table, the declared rank `L`, the declared strata, the declared weights and the design lines.

The object is reassembled **component by component from what is stored**: nothing is deserialised and nothing is recomputed, so what comes back is what went in, and a fault in the encoding shows up as a difference rather than being papered over by a recomputation. Because the inputs are stored too, the audit can also be re-derived from them and compared against what was stored; `check_inputs` does exactly that and reports a disagreement – which would mean the file was written by a different version of the audit – instead of leaving it silent.

Nothing here reads a block of data, and no estimator is called. This is a provenance record: the certificate travelling with the file it describes.

Value

An object of class "mgcca_audit"; see [mgcca_audit](#).

See Also

[mgcca_audit](#)

Examples

```
# a small HDF5-backed design: three blocks over a union of 60 individuals
ids <- sprintf("ind%03d", 1:60)
blocks <- list(
  methylation = matrix(rnorm(30 * 4), 30, 4,
    dimnames = list(ids[1:30], paste0("m", 1:4))),
  expression = matrix(rnorm(60 * 3), 60, 3,
    dimnames = list(ids, paste0("e", 1:3))),
  proteins = matrix(rnorm(30 * 2), 30, 2,
    dimnames = list(ids[31:60], paste0("p", 1:2)))
h5 <- tempfile(fileext = ".h5")
desc <- mgcca_import_hdf5(blocks, filename = h5, overwriteFile = TRUE)

a <- mgcca_audit(desc, L = 2, save_hdf5 = TRUE)
b <- mgcca_audit_load(h5)
identical(b$provenance$pairs, a$provenance$pairs)
unlink(h5)
```

mgcca_block_profile	<i>Per-block contribution profile of a sensitivity result</i>
---------------------	---

Description

Reformats the block-level decomposition that `mgcca_sensitivity` already computed: how much of a query's first-order sensitivity each block carries, and how much of that is carried by the grouping.

Usage

```
mgcca_block_profile(x, query = NULL)
```

Arguments

<code>x</code>	An object of class "mgcca_sensitivity".
<code>query</code>	Optional query name, or names, to restrict to. Default: all.

Details

This is an extractor, not a second analysis. It recomputes nothing. The block-level quantities come back from the same kernel pass that produced the totals, so there is exactly one route to them; a second independent implementation would eventually disagree with the first, and the disagreement would be silent.

Shares are relative within a query. A block carrying most of a negligible sensitivity still carries most of it, so the absolute columns are returned alongside and should be read with the shares.

Value

An object of class "mgcca_block_profile": a data frame with one row per query and block, plus the originating provenance.

See Also[mgcca_sensitivity](#)**Examples**

```
## mgcca_sensitivity() needs the HDF5-backed source blocks of a fit, so the
## example works from a minimal stand-in carrying the one element this
## extractor reads, `by_block` -- exactly the shape mgcca_sensitivity()
## returns for an ungrouped analysis of three blocks and two queries.
sens <- structure(
  list(by_block = data.frame(
    query = rep(c("age", "bmi"), each = 3),
    block = rep(c("methylation", "clinical", "other"), 2),
    S_total = c(0.62, 0.25, 0.13, 0.31, 0.44, 0.25),
    share_total = c(0.62, 0.25, 0.13, 0.31, 0.44, 0.25),
    stringsAsFactors = FALSE),
    groups = NULL,
    settings = list(grouped = FALSE),
    provenance = list(file = NA_character_, datasets = NULL),
    call = NULL),
  class = "mgcca_sensitivity")

bp <- mgcca_block_profile(sens)
bp$profile

## restrict to one query
mgcca_block_profile(sens, query = "age")$profile
```

mgcca_import_hdf5

*Import data of any supported type into an HDF5 file for mgcca***Description**

Versatile entry point that normalizes the input tables to the mgcca HDF5 convention and writes them under a single group. Accepts data already stored in HDF5, a MultiAssayExperiment, an ExpressionSet/SummarizedExperiment, or a plain list of R matrices. Each table is stored as **individuals x variables** (R-view) with rownames = individual IDs (required to align tables and detect missing individuals) and colnames = variable names.

Usage

```
mgcca_import_hdf5(
  x,
  filename,
  group = "MGCCA_IN",
  datasets = NULL,
  overwriteDataset = FALSE,
  overwriteFile = FALSE
)
```

Arguments

<code>x</code>	Input data: a length-1 character path to an existing HDF5 file, a <code>MultiAssayExperiment</code> , an <code>ExpressionSet</code> / <code>SummarizedExperiment</code> , or a named list of matrices.
<code>filename</code>	Target HDF5 file. Ignored (and may be missing) when <code>x</code> is itself an HDF5 path.
<code>group</code>	HDF5 group to write the tables under. Default "MGCCA_IN".
<code>datasets</code>	Optional character vector of dataset names. When <code>x</code> is an existing HDF5 file, restricts/orders the tables to use; otherwise names the written datasets (defaults to the names of <code>x</code>).
<code>overwriteDataset</code>	Logical, overwrite datasets if present. Default FALSE.
<code>overwriteFile</code>	Logical, overwrite/create the file. Default FALSE.

Details

Dimnames are written in the `BigDataStatMeth` layout (`<group>/.<dataset>_dimnames/1=colnames, /2=rownames`) by `bdCreate_hdf5_matrix`, which preserves the `dimnames` attribute of the object automatically. For assay-based inputs (features x samples), assays are transposed to individuals x variables. A plain list is assumed to already be individuals x variables (as produced by `getTables`).

Value

A descriptor list with elements `filename`, `group` and `datasets` (the dataset names written/used), for downstream `mgcca`.

See Also

[getTables](#)

Examples

```
data(cardiovascular)
ids <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))[1:100]
num <- function(d, cols = seq_len(ncol(d))) {
  m <- as.matrix(d[rownames(d) %in% ids, cols, drop = FALSE])
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = num(X1, 1:10), clinical = num(X2), other = num(X3))

h5 <- tempfile(fileext = ".h5")
desc <- mgcca_import_hdf5(X, filename = h5, overwriteFile = TRUE)
desc$group
desc$datasets

## the descriptor is what the rest of the HDF5 pipeline consumes
file.exists(desc$filename)
unlink(h5)
```

mgcca_load

*Reload an mgcca result straight from its HDF5 file***Description**

Rebuilds a full "mgcca" object from a results file written by [mgcca](#), using *only* the file: it reads the provenance manifest stored as native HDF5 attributes (method, lambda, route, route_dual, nfac, datasets, eigenvalues, ...) and combines it with [mgcca_results](#) (which reads the numeric blocks). No original R session, descriptor list or .rds is needed – the file is self-describing.

If the file predates the manifest (no attributes), mgcca_load falls back to auto-discovering the table names from the corsY subgroups and returns what it can, with the provenance fields set to NA.

Usage

```
mgcca_load(file, group = "FINAL_RESULTS")
```

Arguments

file	Path to the HDF5 results file.
group	HDF5 group holding the results. Default "FINAL_RESULTS".

Value

An object of class "mgcca" (see [mgcca_results](#)), with the reconstructed provenance descriptor on attr(., "desc").

See Also

[mgcca](#), [mgcca_results](#)

Examples

```
data(cardiovascular)
ids <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))[1:100]
num <- function(d, cols = seq_len(ncol(d))) {
  m <- as.matrix(d[rownames(d) %in% ids, cols, drop = FALSE])
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = num(X1, 1:10), clinical = num(X2), other = num(X3))

## A file-backed fit. collect = FALSE leaves the results in the file only.
h5 <- tempfile(fileext = ".h5")
desc <- mgcca(X, filename = h5, nfac = 2, method = "penalized",
             lambda = rep(0.1, 3), collect = FALSE)
rm(desc)

## Rebuilt from the file alone -- provenance included.
```

```
fit <- mgcca_load(h5)
fit
attr(fit, "desc")$method
attr(fit, "desc")$lambda
unlink(h5)
```

mgcca_permtest

Permutation test for the significance of the shared components

Description

Builds a null distribution for the mgcca eigenvalues by breaking the correspondence between tables (independently permuting each table's individuals) and re-fitting `nperm` times. A component whose observed eigenvalue exceeds the permuted ones carries genuine cross-table signal.

Most informative when $p < n$ (eigenvalues below 1); for $p \gg n$ with penalisation the eigenvalues sit near 1 by construction and the test is uninformative. Re-fits the model `nperm` times, so use it on modest data (or a feature subset).

Usage

```
mgcca_permtest(
  x,
  filename = tempfile(fileext = ".h5"),
  nperm = 99,
  nfac = 2,
  method = "penalized",
  lambda = NULL,
  route = "auto",
  scale = TRUE,
  ...
)
```

Arguments

<code>x</code>	input tables, exactly as passed to mgcca (a named list of matrices, a <code>MultiAssayExperiment</code> , ...).
<code>filename</code>	target HDF5 file for the observed fit. Default a temp file.
<code>nperm</code>	number of permutations. Default 99.
<code>nfac, method, lambda, route, scale</code>	passed to mgcca .
<code>...</code>	further arguments to mgcca .

Value

A list with eigenvalues (observed, descending), `p.value` (per component), the null matrix (`nperm` x `nfac`) and `nperm`.

See Also[mgcca_associate](#)**Examples**

```
## A toy run, small enough to execute here: two three-column blocks over 40
## individuals and nperm = 2. With two permutations the p-values cannot fall
## below 1/3 and carry no evidence whatever; this call is here so the code
## itself is exercised, not to test anything.
u <- seq_len(40) / 40
ids <- sprintf("id%02d", seq_len(40))
one <- cbind(a = sin(2 * pi * u), b = cos(2 * pi * u), c = u)
two <- cbind(d = sin(2 * pi * u + 0.3), e = u^2, f = rev(u))
rownames(one) <- rownames(two) <- ids

h5toy <- tempfile(fileext = ".h5")
pt0 <- mgcca_permtest(list(one = one, two = two), filename = h5toy,
                      nperm = 2, nfac = 2, method = "penalized",
                      lambda = rep(0.1, 2))

pt0$eigenvalues
unlink(h5toy)

## The illustration below is a realistic run: each permutation is a full
## file-backed re-fit, so even a token number of them takes tens of seconds,
## and it is kept out of the check budget.

data(cardiovascular)
ids <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))[1:60]
num <- function(d, cols = seq_len(ncol(d))) {
  m <- as.matrix(d[rownames(d) %in% ids, cols, drop = FALSE])
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = num(X1, 1:4), clinical = num(X2, 1:4),
          other = num(X3, 1:4))

h5 <- tempfile(fileext = ".h5")
pt <- mgcca_permtest(X, filename = h5, nperm = 9, nfac = 2,
                    method = "penalized", lambda = rep(0.1, 3))

pt$eigenvalues
pt$p.value      # nperm = 9 bounds these below at 0.1
unlink(h5)
```

Description

The C++/HDF5 core ([mgcca_rcpp](#) / [mgcca](#)) writes its results to the HDF5 file under FINAL_RESULTS rather than returning them in memory. This reader materialises those results into a compact list of class "mgcca" – the same shape the legacy in-memory `mgcca()` returned – so the post-analysis helpers [plotInds](#), [plotVars](#), [getSignif](#) and [topVars](#) work unchanged on the port's output.

The shared components `Y` and the per-table `corsY` / `pval` are small (individuals or variables $\times$ `nfac`), so collecting them into RAM does not defeat the out-of-core design; the large tables and intermediates stay on disk.

Usage

```
mgcca_results(
  x,
  datasets = NULL,
  final_group = NULL,
  scores = NULL,
  pval = TRUE,
  tmp_group = "MGCCA_TMP",
  outputs = NULL
)
```

Arguments

<code>x</code>	Either the descriptor list returned by <code>mgcca()</code> / <code>mgcca_rcpp()</code> (with elements <code>filename</code> , <code>datasets</code> , <code>final_group</code>), or a length-1 character path to the HDF5 file.
<code>datasets</code>	Optional table names to read. Defaults to <code>x\$datasets</code> when <code>x</code> is a descriptor, otherwise required.
<code>final_group</code>	HDF5 group holding the results. Default <code>x\$final_group</code> or "FINAL_RESULTS".
<code>scores</code> , <code>pval</code>	Whether to read the per-table scores / p-values (they exist only if <code>mgcca()</code> was called with <code>scores = TRUE</code> / the defaults). Default: infer from the descriptor, otherwise TRUE.
<code>tmp_group</code>	HDF5 group holding the intermediates, where the per-table presence masks <code>K</code> live. Only used for the overlap report; if the group is absent the report is NULL and nothing else changes. Default "MGCCA_TMP".
<code>outputs</code>	Which components to collect. NULL (default) collects everything, which is exactly what this function has always done. Otherwise a character vector naming a subset of "Y", "corsY", "scores", "pval", "weights", "scaling", "AVE", "eigen", "overlap" ("pval.cor" and "eig" are accepted as synonyms of the last two). The datasets behind the components you do not ask for are never read: on a large fit collected over a slow filesystem that is where the time goes. The choice is purely subtractive – every component you do ask for is bit-for-bit the one a full collection would have produced, because selecting outputs changes what is read and nothing about what was computed. Components not asked for come back NULL, and the object keeps its shape, so <code>\$corsY</code> on a partial object is NULL rather than an error. What was requested is recorded on <code>attr(, "outputs")</code> .

Value

A list of class "mgcca" with elements `Y` (matrix, individuals $\times$ components), `corsY` and `pval.cor` (named lists of variable $\times$ component matrices), `scores` (named list or NULL) and `AVE` (`AVE_X`, `AVE_outer_model`, `AVE_inner_model`).

Two read-only diagnostics are attached as well, and neither takes part in any computation: `eigen` (the external spectral gap $\mu_L - \mu_{L+1}$ of the mgcca operator, absolute and relative, plus the eigen-residual) and `overlap` (the number of individuals shared by each pair of tables, n_{jk} , its normalised version $\alpha_{jk} = n_{jk}/n$ over the union, and the dispersion of α across pairs). `summary` prints both.

Each scores matrix spans the union of individuals, but an individual that is not part of a given table has NA in that table's scores (the HDF5 layer stores it as NaN). Only `Y` is defined for everyone.

See Also

[mgcca](#), [plotInds](#), [plotVars](#)

Examples

```
data(cardiovascular)
ids <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))[1:100]
num <- function(d, cols = seq_len(ncol(d))) {
  m <- as.matrix(d[rownames(d) %in% ids, cols, drop = FALSE])
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = num(X1, 1:10), clinical = num(X2), other = num(X3))

## collect = FALSE returns the descriptor; the results stay in the file.
h5 <- tempfile(fileext = ".h5")
desc <- mgcca(X, filename = h5, nfac = 2, method = "penalized",
  lambda = rep(0.1, 3), collect = FALSE)

res <- mgcca_results(desc)
dim(res$Y)
names(res$corsY)
res$overlap$pairs

## Subtractive collection: only the shared components are read back, and
## they are bit-for-bit the ones a full collection would have returned.
resY <- mgcca_results(desc, outputs = "Y")
identical(resY$Y, res$Y)
names(Filter(Negate(is.null), resY))
unlink(h5)
```

Description

Selects the penalty λ for `mgcca(method = "penalized")` by K -fold cross-validation over individuals, scoring each candidate by how well the per-block scores of *held-out* individuals agree across blocks. It returns the chosen penalty, the agreement diagnostic and the whole candidate grid, so the choice lives in the user's record rather than inside a fit.

`mgcca()` does not call this function. `method = "penalized"` still requires an explicit `lambda`; nothing about the default behaviour of `mgcca` changes because this function exists.

Usage

```
mgcca_select_lambda(
  x,
  nfac = 2,
  gamma = NULL,
  lambda = NULL,
  scale_matched = TRUE,
  K = 5,
  seed = NULL,
  rule = c("argmax", "1se"),
  scale = TRUE,
  min_pair = NULL,
  rank_tol = 1e-08,
  sd_tol = 1e-08
)
```

Arguments

<code>x</code>	A named list of matrices (individuals $\times$ variables) with rownames giving the individual IDs. Tables need not share individuals; they are aligned on the union, as in <code>mgcca</code> .
<code>nfac</code>	Number of shared components the penalty is being chosen for. Default 2. The selected <code>lambda</code> is conditional on it.
<code>gamma</code>	Candidate grid for the <i>dimensionless</i> ridge (<code>scale_matched = TRUE</code>), where $\lambda_j = \gamma s_j$ and $s_j = \text{tr}(G_j)/\text{rank}(G_j)$ is the mean positive Gram eigenvalue of block j . Default <code>10^seq(-1, 1, by = 0.25)</code> .
<code>lambda</code>	Candidate grid of absolute penalties (<code>scale_matched = FALSE</code>), applied identically to every block. Default <code>c(0.05, 0.2, 0.5, 1, 2, 5, 10, 30, 100, 300, 1000)</code> – the grid of the reference calibration.
<code>scale_matched</code>	If <code>TRUE</code> (default) the grid is over <code>gamma</code> and each block's penalty is put on the scale of that block's own Gram spectrum, recomputed inside every training fold. This is what keeps the amount of regularisation comparable across blocks of very different dimension; a fixed absolute <code>lambda</code> does not.
<code>K</code>	Number of folds. Default 5.
<code>seed</code>	Optional integer. When supplied, random steps run under this seed in a local RNG scope that restores the caller's random-number state afterwards; when <code>NULL</code> (the default) the current RNG stream is used as-is. The random step here

	is the fold assignment, which the whole selection is a deterministic function of, so pass a seed when you want the selection to be reproducible.
rule	"argmax" (default) takes the best mean CV agreement, as in the reference calibration. "1se" takes the most regularised candidate whose mean is within one standard error of the best.
scale	If TRUE (default) each block is centred and scaled inside every fold using the training rows only.
min_pair	Minimum number of held-out individuals a pair of blocks must share for that pair to be scored in a fold. Default $\text{nfac} + 3$.
rank_tol	Relative tolerance for the numerical rank of G_j when <code>scale_matched = TRUE</code> . Default $1\text{e-}8$.
sd_tol	Columns whose training-fold standard deviation is below this are dropped inside the fold. Default $1\text{e-}8$.

Value

An object of class "mgcca_lambda": a list with

lambda	the selected penalty, one per table, named.
gamma	the selected dimensionless ridge (NA when <code>scale_matched = FALSE</code>).
s	the per-block scales s_j on the full data (NULL when <code>scale_matched = FALSE</code>).
agreement	the mean held-out cross-block RV at the selection – the diagnostic that justifies it.
grid	the candidate trace: the tuning value, <code>cv_mean</code> , <code>cv_se</code> , <code>n_cells</code> (how many (fold, pair) cells were scorable), <code>in_1se</code> and selected.
cv_by_fold	the raw fold-by-candidate agreement matrix.
lambda_by_candidate	the penalty each candidate corresponds to.
status	"ok", or "boundary_optimum" when the best candidate is at an edge of the grid, which is also warned about. When no candidate can be scored at all the function stops rather than returning a choice nothing supports.
criterion, rule, nfac, K, seed, fold, call	the record of how the choice was made.

Why not choose lambda by stability

The intuitive rule – keep the lambda whose fit moves least under resampling – is not merely imperfect, it is actively misleading, and this function exists to replace it. Predicted instability is *minimised* in the over-regularised regime, where each block has fallen onto its own high-variance but non-shared axis: the fit no longer moves because there is no longer anything shared in it that could move. In the specific-dominant calibration regime of the reference below, the stability-optimal lambda recovers the true shared subspace **0.666 worse** than the best available lambda, while the criterion implemented here loses **0.002**. Instability is a legitimate diagnostic *conditional* on a chosen (λ, L) ; it is not a selection rule.

Cross-block agreement rejects that regime by construction. When the blocks have latched onto their own private axes they disagree about where a held-out individual sits, so the score collapses exactly where the stability criterion is happiest.

The criterion

For each fold: fit the penalized missing-row estimator on the training individuals; form each block's weights $B_j = X_j'(X_j X_j' + \lambda_j I)^{-1} Y$ from the training fit; project the held-out individuals of block j through B_j using the *training* centring and scaling; and average the RV coefficient between the held-out scores of every pair of blocks over the individuals both blocks hold. RV is invariant to rotation, sign and scale, which is all a subspace is defined up to. The score never sees any outcome or truth.

Reporting it

Report the selected lambda together with agreement and the grid. A wide near-optimal region (many candidates inside one standard error) means the penalty is not sharply identified and the region, not the single number, is the honest summary; the function says so.

References

The criterion, the guardrail and the 0.002 / 0.666 comparison come from the thesis reliability calibration (reliability/05_lambda_selection.md; estimator in reliability/calibration/estimator.R).
van de Velden, M. and Bijmolt, T. H. A. (2006) Generalized canonical correlation analysis of matrices with missing rows. *Psychometrika* 71, 323-331.

See Also

[mgcca](#)

Examples

```
## A small three-block slice of the shipped cohort. The tables overlap only
## partly, which is the situation the criterion is written for.
data(cardiovascular, package = "mgcca")
ids <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))[1:150]
mk <- function(d, j = NULL) {
  m <- as.matrix(d[rownames(d) %in% ids, , drop = FALSE])
  if (!is.null(j)) m <- m[, j, drop = FALSE]
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = mk(X1, 1:20), clinical = mk(X2), cells = mk(X3))

## Three candidates for the dimensionless ridge, three folds. A real
## analysis uses the wider default grid; this one is sized for the example.
## The folds are drawn at random, so a seed is passed to make the selection
## reproducible; without one the draw comes from the session's own stream.
sel <- mgcca_select_lambda(X, nfac = 2, gamma = c(0.01, 0.1, 1), K = 3,
                          seed = 1)

sel

## The whole candidate trace, not only the winner: report the near-optimal
## region whenever several candidates sit within one standard error.
sel$grid
```

```
## The selected penalty is what mgcca(method = "penalized") then takes.
fit <- mgcca(X, nfac = 2, method = "penalized", lambda = sel$lambda)
```

mgcca_sensitivity	<i>Sensitivity of the shared subspace to a participant-level query</i>
-------------------	--

Description

Measures how much of a participant-level quantity – a phenotype, an exposure, a score – aligns with the shared subspace estimated by [mgcca](#), and how much of that alignment’s sensitivity is carried by a grouping variable such as batch, centre, plate or cohort.

Usage

```
mgcca_sensitivity(
  x,
  query,
  group = NULL,
  covariates = NULL,
  lambda = NULL,
  gamma = 1,
  store_name = "sensitivity",
  backend = c("auto", "memory", "hdf5"),
  block_size = 0L,
  store = FALSE,
  threads = NULL
)

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

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

Arguments

x	A fitted object of class "mgcca". It must carry a descriptor pointing at an HDF5 file that still contains the input blocks: the source data is required, and a results-only object is refused rather than half-processed. Use mgcca_load to obtain one from a file. For the print method documented below, an object of class "mgcca_sensitivity" instead.
query	Numeric vector or matrix over participants. A vector must be named; a matrix must have row names, and column names that are unique because they become the query identifiers. Participants absent from the query contribute nothing and are counted in n_used.

group	Optional grouping over participants: factor, character or integer. Supplying it adds the between-group decomposition; omitting it returns the total quantities only, with the group-dependent fields absent rather than filled with zeroes. Levels with fewer than two participants are refused, since a between-group quantity is not defined for them.
covariates	Optional named, participant-aligned numeric matrix or data frame. The query is residualised on these before the sensitivity is taken. The realised design, its rank and the retained participants are archived in the result.
lambda	Ridge for the layer's own operator: NULL (default) uses a scale-aware guarded rule, gamma times each block's mean positive eigenvalue. Supply one value, or one per block, to override it. The fit's own lambda is deliberately NOT inherited: it was chosen for a different operator on a different scale.
gamma	Multiplier for the guarded ridge rule. Ignored when lambda is supplied.
store_name	Name of the group under RELIABILITY/ when store = TRUE. An existing name is refused rather than overwritten.
backend	One of "auto" (default), "memory" or "hdf5". "auto" chooses by block size, as mgcca(route = "auto") does. The other two force the path; the choice is recorded in the result so a run is never ambiguous about how it was produced.
block_size	Block size for the out-of-core steps. 0 lets BigDataStatMeth choose. A positive value forces it, which exists so the blocked path can be exercised on small data.
store	If TRUE, persist the result into the fit's HDF5 file under RELIABILITY/. Default FALSE: an analysis function should not write to a user's file unasked.
threads	Optional thread count.
object	An object of class "mgcca_sensitivity", as returned by mgcca_sensitivity().
...	Ignored.

Details

Two quantities are returned for each query. T is the share of the query that lies in the retained subspace, bounded in $[0, 1]$. S_{total} is the first-order sensitivity of that alignment to a structured perturbation of the blocks; when `group` is supplied, S_{between} is the part of that sensitivity attributable to differences between groups, and R_{between} their ratio.

Read the ratio with its components, never alone. A high R_{between} can accompany a negligible absolute sensitivity, and a low one can accompany a large signal; the ratio says how the sensitivity is distributed, not how much there is.

Participants are matched by identifier, never by position. A query without names is an error rather than an assumption, because a silent misalignment produces a plausible number instead of a failure and cannot be detected from the output.

Why the ridge cannot be zero, and cannot be the fit's. A block Gram is $n \times n$ in *participants*, so its rank is at most $\min(n, p_j)$, and centring costs one dimension more. A wide block ($p_j \gg n$, say methylation) therefore returns rank $n - 1$; a narrow one ($p_j < n$, say a clinical table of four columns sitting beside it) returns rank p_j , which can be far below n . Measured: $n = 50$ with $p = 500$ gives rank 49, and $n = 620$ with $p = 4$ gives rank 4. So $(G_j + \lambda I)^{-1}$ does not exist at $\lambda = 0$ in *either* case – the two simply fail by very different margins. Inheriting the fit's own lambda is no better: it was chosen for a different operator on a different scale. The default rule is proportional to each block's

own mean positive eigenvalue, which is what lets one number serve a 485,000-column block and a 4-column one in the same fit.

What this is not. These are first-order sensitivity coefficients of a fitted operator. They are not causal, and they are not a psychometric reliability coefficient.

`print()` reports the settings the result was produced under – how many queries and blocks, the grouping, the backend, the rank and where the ridge came from – followed by the overall table, and states plainly when the retained and discarded subspaces are not separated, in which case the first-order expansion these numbers rest on does not hold.

`summary()` prints all of that and then the per-block decomposition and, when the result carries one, the top participants by sensitivity contribution. The share it displays there is taken against the query’s own total, not the `by_individual` table’s `share_total`, which is a share within one block.

Value

An object of class “mgcca_sensitivity”: a list with overall (a data frame, one row per query, carrying `n_used`, `T`, `S_total` and – only when `group` is supplied – `S_between` and `R_between`), `by_block` (one row per query and block), `by_individual`, `by_individual_check`, `validity` (the eigenvalue-gap check on the first-order expansion), `reference` (the layer’s own participant coordinates `Y`, the query directions, and the grouping actually used), `groups` (the group levels, or `NULL`), `settings`, `provenance` and the matched call. When `store = TRUE` a storage element records the HDF5 group the result was written to.

`by_individual` decomposes each block’s sensitivity over participants, in long form with columns `query`, `id`, `block`, `S_total`, `share_total` and – only when `group` is supplied – `S_between` and `share_between`. Writing s_{ij} for a participant’s contribution and S_j for the block quantity already reported in `by_block`, the decomposition is $\sum_i s_{ij} = S_j$.

Rows. `by_individual` contains exactly the ordered participant set used by the grouped sensitivity calculation, including participants numerically absent from a particular block. Key it by the query, `id` and `block` columns; row order is not part of the contract. A grouping that covers fewer participants therefore yields different contributions: that is the estimand being conditioned on the grouped set, not an instability.

Shares. `share_total` is s_{ij}/S_j , so the shares of a given query and block sum to one; `share_between` is defined the same way over the between-group parts. Note that `S_between` is built from group means, so it is constant across the members of a group: it is the group’s between-group contribution carried in each member’s row, and ranking participants by it ranks their groups. `S_total` is the per-participant quantity. When a denominator is not finite and strictly positive the share is `NA`, never a silent zero. The absolute contributions inherit the non-identified scale of the parent sensitivity quantity; the shares remove this common multiplicative scale and are the preferred dimensionless representation, but they remain conditional on the fitted model, the ridge specification, the query, the block and the grouped participant set. No invariant ranking of participants is implied. Like S_j , the contributions are invariant to rescaling the query or a block and are NOT invariant to the layer’s ridge. A participant absent from a block contributes a numerical zero there (of order $1e-34$), not an exact zero, which is why a per-entry relative comparison is the wrong way to check them.

`by_individual_check` reports the decomposition identity actually achieved: `sum_rel_total`, and `sum_rel_between` when grouped, are the largest $|\sum_i s_{ij} - S_j|/\max(1, |S_j|)$ over queries and blocks. The identity is definitional but not bit for bit, because a trace and a sum over rows accumulate in different orders.

What these values are. `by_individual` is an exploratory decomposition of the first-order sensitivity diagnostic. Its block-level parent has empirical calibration evidence with a MIXTO verdict; no individual-level calibration has been established. The reported values are sensitivity contributions, not variance, uncertainty, causal influence, or reliability guarantees.

`print()` returns its argument invisibly, and `summary()` returns object invisibly; both are called for what they write to the console.

See Also

[mgcca_block_profile](#) to reformat the per-block decomposition, [mgcca_stability](#) for the resampling companion, and [mgcca](#) for the fit itself.

Examples

```
## A small three-block slice of the shipped cohort. The reliability layer
## reads the SOURCE blocks back from HDF5, so the fit must be file-backed:
## an in-memory fit (filename = NULL) is refused, not silently approximated.
data(cardiovascular, package = "mgcca")
ids <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))[1:120]
mk <- function(d, j = NULL) {
  m <- as.matrix(d[rownames(d) %in% ids, , drop = FALSE])
  if (!is.null(j)) m <- m[, j, drop = FALSE]
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = mk(X1, 1:10), clinical = mk(X2), cells = mk(X3))

fit <- mgcca(X, filename = tempfile(fileext = ".h5"), nfac = 2,
            method = "penalized", lambda = rep(0.1, 3), scores = TRUE)

## The query is BMI; the grouping splits the cohort on CD4T cell proportion.
## A one-column matrix is used so the query carries its own name into the
## result; a named vector would work too.
bmi <- X$clinical[, "BMI", drop = FALSE]
cd4 <- X$cells[, "CD4T"]
grp <- stats::setNames(
  ifelse(cd4 > stats::median(cd4), "CD4T high", "CD4T low"),
  rownames(X$cells))

sens <- mgcca_sensitivity(fit, query = bmi, group = grp)
sens

## Read the ratio with its components, never alone.
sens$overall
sens$by_block

## summary() adds the per-block decomposition to what print() shows.
summary(sens)
```

mgcca_stability

*Stability of the shared subspace under resampling***Description**

Refits the shared subspace on resamples of the participants and measures how much it moves. "subsample" draws random subsets; "loco" leaves out one level of group at a time – the generic form of leave-one-cohort-out, for any grouping the user has.

Usage

```
mgcca_stability(
  x,
  method = c("subsample", "loco", "both"),
  group = NULL,
  strata = NULL,
  B = 200L,
  fraction = 0.632,
  seed = NULL,
  lambda = NULL,
  gamma = 1,
  store_name = "stability",
  backend = c("auto", "memory", "hdf5"),
  block_size = 0L,
  store = FALSE,
  threads = NULL
)

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

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

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

Arguments

x	A fitted "mgcca" object; see mgcca_sensitivity for what it must carry. For the print and plot methods documented below, an object of class "mgcca_stability" instead.
method	"subsample" (default), "loco", or "both". "loco" requires group; "subsample" does not, so the function is usable with no grouping at all.
group	Grouping over participants, required for "loco".

strata	Optional grouping to stratify the subsampling by. Deliberately separate from group: leaving out study centres while stratifying subsamples by sex is a reasonable thing to want, and one argument serving both roles would make it impossible.
B	Number of subsamples.
fraction	Fraction of participants retained in each subsample.
seed	Optional integer. When supplied, random steps run under this seed in a local RNG scope that restores the caller's random-number state afterwards; when NULL (the default) the current RNG stream is used as-is. The random step here is the resampling draw. The realised resampling plan is archived in the result either way, so a run can be reproduced from what it returns even when no seed was given.
lambda, gamma	Ridge for the layer's operator; see mgcca_sensitivity .
store_name	Name of the group under RELIABILITY/ when store = TRUE. Default "stability". An existing name is refused rather than overwritten, so a second run with different settings cannot leave a manifest that no longer describes the data beside it.
backend, block_size, threads	As in mgcca_sensitivity .
store	If TRUE, persist under RELIABILITY/. Default FALSE: an analysis function should not write to a user's file unasked.
object	An object of class "mgcca_stability", as returned by <code>mgcca_stability()</code> .
...	Ignored.

Details

Two quantities are returned per resample. `overlap` is the agreement between the resampled subspace and the reference one, compared *on the participants both contain*. `rank_margin` is the smallest singular value of the reference frame restricted to those shared participants.

The rank margin is not a measure of component separation. It says whether the comparison is well posed on the people both fits have: if the reference frame becomes near-degenerate once restricted, the overlap is comparing directions that are barely present in the shared set, and a high overlap would mean little. An orthonormal frame has every singular value equal to one, so a value well below one is the warning, not the reassurance.

Why the ridge cannot be zero, and cannot be the fit's. A block Gram is $n \times n$ in *participants*, so its rank is at most $\min(n, p_j)$, and centring costs one dimension more. A wide block ($p_j \gg n$, say methylation) therefore returns rank $n - 1$; a narrow one ($p_j < n$, say a clinical table of four columns sitting beside it) returns rank p_j , which can be far below n . Measured: $n = 50$ with $p = 500$ gives rank 49, and $n = 620$ with $p = 4$ gives rank 4. So $(G_j + \lambda I)^{-1}$ does not exist at $\lambda = 0$ in *either* case – the two simply fail by very different margins. Inheriting the fit's own `lambda` is no better: it was chosen for a different operator on a different scale. The default rule is proportional to each block's own mean positive eigenvalue, which is what lets one number serve a 485,000-column block and a 4-column one in the same fit.

What moves and what does not. The blocks' standardisation is that of the full fit and is held fixed; only the participant set changes. So this measures stability of the estimated subspace to the sample, not to the whole preprocessing pipeline.

`print()` reports the resampling settings, then `summary_metrics`, then the number of resamples whose refit failed, and closes with the reminder that the rank margin says whether the comparison is well posed rather than whether the components are separated.

`summary()` prints all of that and then the ten worst resamples, ordered by overlap, so the tail of the distribution is what is shown rather than its head.

`plot()` draws one point per successful resample, overlap against rank margin and coloured by method. Reading the two axes together is the point: a high overlap at a low rank margin compares directions barely present in the shared participants. Resamples whose refit failed carry no overlap and are absent from the panel; they stay in `$resamples` with their reason.

Value

An object of class "mgcca_stability": a list with

`resamples` one row per resample, with method, id, left_out, the retained n, the overlap with the reference subspace, the rank_margin, and ok / reason for resamples whose refit failed – reported, never dropped.

`summary_metrics` one row per method: number of resamples, number failed, minimum and median overlap, and the minimum rank margin.

`reference` the reference spectrum μ , its eigenvalue gap, and L .

`validity` the gap check on the reference fit.

`plan` the realised resampling plan, so a run reproduces from what it returns rather than from a seed plus matching RNG settings.

`settings, groups, provenance, call` the record of how it was produced. When `store = TRUE` a storage element records the HDF5 group written to.

`print()` returns its argument invisibly and `summary()` returns object invisibly, both being called for what they write to the console; `plot()` returns a ggplot object, which draws when printed.

See Also

[mgcca_sensitivity](#) for the query-based companion, and [mgcca_select_lambda](#) – instability is a diagnostic at a chosen penalty, never the way to choose one.

Examples

```
## A small three-block slice of the shipped cohort. As for
## mgcca_sensitivity(), the fit must be file-backed: the resampling refits
## the operator from the SOURCE blocks in HDF5.
data(cardiovascular, package = "mgcca")
ids <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))[1:120]
mk <- function(d, j = NULL) {
  m <- as.matrix(d[rownames(d) %in% ids, , drop = FALSE])
  if (!is.null(j)) m <- m[, j, drop = FALSE]
  storage.mode(m) <- "double"
  m
}
```

```

X <- list(methylation = mk(X1, 1:10), clinical = mk(X2), cells = mk(X3))

fit <- mgcca(X, filename = tempfile(fileext = ".h5"), nfac = 2,
            method = "penalized", lambda = rep(0.1, 3), scores = TRUE)

## B is tiny here so the example runs quickly; a real run uses the default.
## "both" adds leave-one-level-out on top of the subsampling, so the two
## methods appear side by side in the summary.
cd4 <- X$cells[, "CD4T"]
grp <- stats::setNames(
  ifelse(cd4 > stats::median(cd4), "CD4T high", "CD4T low"),
  rownames(X$cells))

stab <- mgcca_stability(fit, method = "both", group = grp, B = 3, seed = 1)
stab

## Read the overlap next to the rank margin: a high overlap on a low margin
## compares directions barely present in the shared participants.
stab$summary_metrics
stab$resamples

## summary() adds the worst resamples; plot() puts the two quantities on
## the same panel.
summary(stab)
plot(stab)

```

mgccaImpute

Imputes each assay matrix data from a MultiAssayExperiment object

Description

Fills in the missing values of every assay of a MultiAssayExperiment by k-nearest-neighbour imputation ([impute.knn](#)), replacing each assay in place. This is imputation *within* a table (scattered missing entries); it is not the whole-row missingness across tables that [mgcca](#) handles natively.

Usage

```

mgccaImpute(
  multiassayexperiment,
  method = "knn",
  remove.col = FALSE,
  impute.zero = FALSE,
  k = 10,
  rowmax = 0.5,
  colmax = 0.8,
  ...
)

```

Arguments

<code>multiassayexperiment</code>	a <code>MultiAssayExperiment</code> object whose assays are numeric matrices.
<code>method</code>	imputation method, given as a character string. Only "knn" is implemented, which is also the default; any other value is an error.
<code>remove.col</code>	logical; when TRUE, columns with a proportion of missing values of <code>colmax</code> or more are dropped from an assay before it is imputed. Default FALSE.
<code>impute.zero</code>	logical; when TRUE, exact zeros are treated as missing values (recoded to NA) before imputation. Default FALSE.
<code>k</code>	number of neighbours used to impute a missing entry, passed to <code>impute.knn</code> as its <code>k</code> . Default 10.
<code>rowmax</code>	largest proportion of missing values tolerated in a row before <code>impute.knn</code> falls back to the overall column mean for that row, passed on as its <code>rowmax</code> . Default 0.5.
<code>colmax</code>	largest proportion of missing values tolerated in a column; <code>impute.knn</code> stops when a column exceeds it. Passed on as its <code>colmax</code> , and also used by <code>remove.col</code> . Default 0.8.
<code>...</code>	further arguments passed on to <code>impute.knn</code> .

Value

The input `MultiAssayExperiment` with each assay replaced by its imputed version.

Examples

```
if (requireNamespace("MultiAssayExperiment", quietly = TRUE)) {
  data(cardiovascular)
  sel <- rownames(X2)[1:30]
  a1 <- t(as.matrix(X1[rownames(X1) %in% sel, 1:8, drop = FALSE]))
  a1[1, 1:3] <- NA # a few missing entries to fill in
  a2 <- t(as.matrix(X2[rownames(X2) %in% sel, , drop = FALSE]))
  mae <- MultiAssayExperiment::MultiAssayExperiment(
    MultiAssayExperiment::ExperimentList(
      list(methylation = a1, clinical = a2)))

  anyNA(mae[[1]])
  mae.imp <- mgccaImpute(mae, method = "knn", k = 5)
  anyNA(mae.imp[[1]])
}
```

`plot.mgcca`

Plot an mgcca object

Description

Default plot for an `mgcca` object: the individuals on the two shared components. A thin wrapper around `plotIndividuals`; all its arguments (`group`, `comps`, `ellipse`, ...) are forwarded.

Usage

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

Arguments

```
x          an mgcca object.
...        passed to plotIndividuals.
```

Value

A ggplot object: the individuals of x on two shared components. Drawn when printed.

Examples

```
data(cardiovascular)
ids <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))[1:150]
num <- function(d, cols = seq_len(ncol(d))) {
  m <- as.matrix(d[rownames(d) %in% ids, cols, drop = FALSE])
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = num(X1, 1:20), clinical = num(X2), other = num(X3))

fit <- mgcca(X, nfac = 2, method = "penalized", lambda = rep(0.1, 3),
             scores = TRUE)
p <- plot(fit)
class(p)
```

plot.mgcca_audit	<i>Plot an availability audit</i>
------------------	-----------------------------------

Description

Six views of one audit. "overview" (the default, and what a bare plot() gives you) is a curated four-panel composite: the classification matrix as the anchor, realised availability, representation, and a small support graph. The other five are single panels and come back as ggplot objects you can style, save or assemble yourself.

Usage

```
## S3 method for class 'mgcca_audit'
plot(
  x,
  type = c("overview", "matrix", "graph", "availability", "representation", "patterns"),
  pair = NULL,
  base_size = 13,
  ...
)
```

Arguments

x	An object of class "mgcca_audit".
type	One of "overview" (default), "matrix", "graph", "availability", "representation" or "patterns".
pair	Optional two block names, for type = "graph" only: draw the stored transfer route between them. Only the stored provenance is read; no route is computed here.
base_size	Base font size passed to the theme. Every label, legend and caption in every panel is a multiple of it, so one number resizes the whole view. "overview" draws its four panels at a fixed fraction of it, since they share one device.
...	Ignored.

Details

"overview" Four panels drawn straight to the device through base grid viewports, returning the panel list invisibly. Curated and deliberately stable: it will not silently grow a panel when a new type is added. Give it room – roughly 11 by 9 inches.

"matrix" One triangle of the pair classification, the diagonal carrying each block's marginal n_j . Fill encodes the frozen classification and nothing else; the cell text carries the support quantity that belongs to that class (n_{jk} for a direct pair, B = the route bottleneck for a transfer pair, 0 for a structural zero, blank for unsupported); advisory flags are corner glyphs with their own legend.

"graph" The support graph on a circle, node order fixed by connected component, then degree, then block name. With pair it becomes a provenance overlay: the stored route between those two blocks is emphasised, everything else recedes, and the bottleneck is marked on its weakest edge.

"availability" Realised availability per block and declared stratum, on a fill scale fixed to [0, 1].

"representation" Two aligned panels in the same order: an effective count (Kish ESS where weights are declared, the observed count otherwise, each annotated with its stratum size) and the retention proportion. Never one dual axis – a count and a proportion do not share a scale.

"patterns" Which combinations of blocks the participants actually carry, as an UpSet-like dot matrix with frequency bars, ordered by frequency and then lexicographically. Aggregated at plot time from the availability table the object already holds. Not part of "overview": it is as tall as the design is varied.

What the colours mean, and what they do not. Colour encodes exactly one variable per panel, and in the matrix that variable is the frozen classification – never how much support a pair has, which is why a thinly supported DIRECT_SUPPORT cell is the same blue as a richly supported one, with the count printed in it. There is no traffic-light scale here and no red or green anywhere: these are properties of a design, not verdicts on it. DIRECT_SUPPORT is the visually strongest class because every other one leans on something extra.

Every other panel colours by a variable too, never for decoration: graph nodes by connected component (blocks of one colour can reach each other, blocks of different colours cannot), representation bars by declared stratum (the same colour in both panels, so a stratum can be followed across them), availability by realised proportion on a teal ramp deliberately unlike the classification blue, and the

pattern bars in their own accent. Every palette is colour-blind-safe, and the two traffic-light hues appear in none of them.

Edge width, and only edge width. In the graph, the width of an edge is the raw count of participants observed in both blocks and nothing else. What the graph shows is *design-support redundancy* – how many independent routes the design gives you between two blocks – which is a statement about topology, not about conditioning or how well anything is estimated.

Declared lines. The dashed lines in the representation panel are the declared design lines, drawn neutrally and labelled as such. They are set by the analyst, they carry margin, and they are not thresholds at which anything fails.

Provenance carries its condition. A pair overlay is only available for a MODEL_TRANSFER_ELIGIBLE pair, and it always says so: such a pair is identified conditional on the declared rank-L model and its required structural conditions, which are not empirically checked by this audit. If the stored provenance records that alternative routes exist, the subtitle says so rather than letting one route look privileged.

Value

For every type except "overview", a ggplot object. For "overview", the four panels as a named list, returned invisibly after the composite is drawn.

See Also

[mgcca_audit](#)

Examples

```
# a design where one block is reachable only through a single edge
ids <- sprintf("ind%02d", 1:85)
avail <- cbind(methylation = seq_along(ids) %in% 1:60,
               expression  = seq_along(ids) %in% 1:60,
               proteins     = rep(TRUE, length(ids)),
               metabolites  = seq_along(ids) %in% 61:85)
rownames(avail) <- ids
a <- mgcca_audit(avail, L = 2,
                 strata = rep(c("cohort A", "cohort B"), length.out = 85),
                 weights = ifelse(seq_along(ids) %in% 1:60, 1, 3))

# the curated composite (give it room: ~11 x 9 inches)
panels <- plot(a)
names(panels)

# the precise view, on its own
plot(a, type = "matrix")

# which combinations of blocks people actually carry
plot(a, type = "patterns")

# the route a transfer-eligible pair depends on
plot(a, type = "graph", pair = c("methylation", "metabolites"))
```

plot.mgcca_block_profile

Plot a block contribution profile

Description

Bar chart of share_total, the share of a query's first-order sensitivity each block carries, with one panel per query and the blocks ordered by their share. The shares are relative within a query, so read them next to the absolute S_total column of x\$profile: a block carrying most of a negligible sensitivity still carries most of it.

Usage

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

Arguments

x	An object of class "mgcca_block_profile".
...	ignored.

Value

A ggplot object. Drawn when printed.

See Also

[mgcca_block_profile](#)

Examples

```
## A minimal stand-in for an ungrouped mgcca_sensitivity() result; see
## mgcca_block_profile() for why the example does not fit one here.
sens <- structure(
  list(by_block = data.frame(
    query = rep(c("age", "bmi"), each = 3),
    block = rep(c("methylation", "clinical", "other"), 2),
    S_total = c(0.62, 0.25, 0.13, 0.31, 0.44, 0.25),
    share_total = c(0.62, 0.25, 0.13, 0.31, 0.44, 0.25),
    stringsAsFactors = FALSE),
    groups = NULL,
    settings = list(grouped = FALSE),
    provenance = list(file = NA_character_, datasets = NULL),
    call = NULL),
  class = "mgcca_sensitivity")

p <- plot(mgcca_block_profile(sens))
class(p)
```

plot.mgcca_sensitivity

Plot a sensitivity result

Description

Four views. "map" places the individuals on the layer's reference axes and draws each query as an arrow, so a query that points along the direction separating your groups is visible rather than inferred from R_between. "blocks" (default) shows how much each block contributes to a query's sensitivity and how much of that is carried by the grouping; "overall" compares the queries; "individuals" cuts the same sensitivity the other way and shows which participants carry it.

Usage

```
## S3 method for class 'mgcca_sensitivity'
plot(
  x,
  type = c("blocks", "overall", "map", "individuals"),
  comps = c(1, 2),
  noise = NULL,
  top_n = 15,
  ...
)
```

Arguments

x	An object of class "mgcca_sensitivity".
type	"blocks" (default), "overall", "map" or "individuals".
comps	For type = "map", which two components to draw. With more than two components, call it repeatedly on different pairs: three dimensions need rotating to be read and do not survive a printed page, and beyond three there is nothing to rotate.
noise	For type = "overall", the names of the control queries – columns of random numbers put through the same machinery. Their range is shaded as the level chance alone reaches, which is the only scale T has.
top_n	For type = "individuals", how many participants to draw per query, ranked by their sensitivity contribution summed over blocks. Default 15. Everyone else is aggregated into a single "all others" bar rather than dropped.
...	Ignored.

Details

The absolute sensitivities are drawn, not only the shares, because a share cannot be interpreted without the magnitude it is a share of: a block can carry most of a sensitivity that is itself negligible.

Why the between-group part is overlaid rather than placed beside it. Side by side it is routinely two orders of magnitude shorter than the total and renders as an invisible sliver, so the plot fails to

show the one comparison it exists for. The between-group part is a component of the total, so it is drawn inside the same bar with its share printed above – a sliver, however honest, cannot be read off an axis.

Why the "individuals" view always draws an "all others" bar. A top-N chart without the remainder silently overstates concentration: fifteen tall bars look like the whole story whether they carry 80% of the total or 8%. The remainder is therefore drawn as one further bar, stacked by block like the rest, so the panel accounts for the query's entire sensitivity and the reader can see how much the top N leaves out. It is the same rule that puts the shares next to the absolutes everywhere else in this package.

The per-participant quantities it draws are an exploratory decomposition: the block-level parent has empirical calibration evidence with a MIXTO verdict, no individual-level calibration has been established, and the bars are sensitivity contributions – not variance, uncertainty, causal influence, or reliability guarantees. They are conditioned on the grouped participant set, so a different grouping is a different question.

Value

A ggplot object, which draws when printed.

See Also

[mgcca_sensitivity](#), which produces the object plotted here.

Examples

```
## A small three-block slice of the shipped cohort; the fit must be
## file-backed for the reliability layer to reach the source blocks.
data(cardiovascular, package = "mgcca")
ids <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))[1:120]
mk <- function(d, j = NULL) {
  m <- as.matrix(d[rownames(d) %in% ids, , drop = FALSE])
  if (!is.null(j)) m <- m[, j, drop = FALSE]
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = mk(X1, 1:10), clinical = mk(X2), cells = mk(X3))

fit <- mgcca(X, filename = tempfile(fileext = ".h5"), nfac = 2,
  method = "penalized", lambda = rep(0.1, 3), scores = TRUE)

bmi <- X$clinical[, "BMI", drop = FALSE]
cd4 <- X$cells[, "CD4T"]
grp <- stats::setNames(
  ifelse(cd4 > stats::median(cd4), "CD4T high", "CD4T low"),
  rownames(X$cells))
sens <- mgcca_sensitivity(fit, query = bmi, group = grp)

## Where the sensitivity sits, block by block, with the between-group part
## drawn inside the total rather than beside it.
plot(sens, type = "blocks")
```

```
## The same fit as a map: individuals on the layer's own reference axes,
## the query as an arrow, and the dashed line showing how the groups split.
plot(sens, type = "map")

## And cut the other way: which participants carry it. The "all others" bar
## is always drawn, so the panel accounts for the whole total.
plot(sens, type = "individuals", top_n = 8)
```

plotAVE

Barplot of the Average Variance Explained (AVE) per table

Description

The outer-model AVE (AVE_X) measures how much of each table's variance is captured by each shared component. This draws it as a grouped bar chart (component on the x axis, one bar per table).

Usage

```
plotAVE(x, title = "Average Variance Explained")
```

Arguments

x	an mgcca object.
title	plot title. Default "Average Variance Explained".

Value

A ggplot object.

See Also

[plotScree](#)

Examples

```
data(cardiovascular)
u <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))
sel <- u[1:150]
mk <- function(d, cols = NULL) {
  m <- as.matrix(d[rownames(d) %in% sel, , drop = FALSE])
  if (!is.null(cols)) m <- m[, cols, drop = FALSE]
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = mk(X1, 1:20), clinical = mk(X2), other = mk(X3))
fit <- mgcca(X, nfac = 2, method = "penalized", lambda = rep(0.1, 3))

plotAVE(fit)
```

plotBiplot

Biplot: individuals and variable loadings together

Description

Overlays the individuals (shared components Y) and, for one table, the loadings of its top variables as arrows – a compact view of which variables pull the individuals where. Loadings are scaled to the span of the individual cloud for readability.

Usage

```
plotBiplot(
  x,
  table = 1,
  comps = c(1, 2),
  top = 8,
  group = NULL,
  title = "Biplot"
)
```

Arguments

x	an mgcca object.
table	table name or index into x\$cor\$Y for the variable arrows. Default 1.
comps	the two components to display. Default c(1, 2).
top	number of variables (by correlation radius) to draw. Default 8.
group	optional factor to colour individuals by.
title	plot title.

Value

A ggplot object.

See Also

[plotIndividuals](#), [plotVariables](#)

Examples

```
data(cardiovascular)
u <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))
sel <- u[1:150]
mk <- function(d, cols = NULL) {
  m <- as.matrix(d[rownames(d) %in% sel, , drop = FALSE])
  if (!is.null(cols)) m <- m[, cols, drop = FALSE]
  storage.mode(m) <- "double"
  m
}
```

```

}
X <- list(methylation = mk(X1, 1:20), clinical = mk(X2), other = mk(X3))
fit <- mgcca(X, nfac = 2, method = "penalized", lambda = rep(0.1, 3))

plotBiplot(fit, table = "clinical", top = 4)

```

plotIndividuals

Plot individuals on the shared components

Description

Scatter plot of the shared canonical components Y (one point per individual), optionally coloured by a grouping factor. A modern, colourblind-safe replacement for `plotInds` that returns a ggplot object you can further customise.

Usage

```

plotIndividuals(
  x,
  group = NULL,
  comps = c(1, 2),
  label = FALSE,
  ellipse = TRUE,
  level = 0.95,
  point_size = 2.2,
  title = "Individuals"
)

```

Arguments

<code>x</code>	an mgcca object (from <code>mgcca_results</code> or <code>mgcca(collect = TRUE)</code>).
<code>group</code>	optional factor to colour individuals by. If it carries names they are matched to the row names of Y; otherwise it is assumed to be in the same order as the individuals.
<code>comps</code>	the two components to display. Default <code>c(1, 2)</code> .
<code>label</code>	logical; label points with individual IDs. Default <code>FALSE</code> .
<code>ellipse</code>	logical; when a group is given, overlay a 95% confidence ellipse per group (groups with fewer than 4 individuals, and the NA group, are skipped). Makes the separation legible when points overlap. Default <code>TRUE</code> .
<code>level</code>	confidence level for the ellipses. Default 0.95.
<code>point_size</code>	point size. Default 2.2.
<code>title</code>	plot title.

Value

A ggplot object.

See Also

[plotVariables](#), [plotAVE](#)

Examples

```
data(cardiovascular)
u <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))
sel <- u[1:150]
mk <- function(d, cols = NULL) {
  m <- as.matrix(d[rownames(d) %in% sel, , drop = FALSE])
  if (!is.null(cols)) m <- m[, cols, drop = FALSE]
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = mk(X1, 1:20), clinical = mk(X2), other = mk(X3))
fit <- mgcca(X, nfac = 2, method = "penalized", lambda = rep(0.1, 3))

plotIndividuals(fit)

## coloured by a grouping factor, with per-group confidence ellipses
grp <- factor(rep(c("a", "b"), length.out = nrow(fit$Y)))
names(grp) <- rownames(fit$Y)
plotIndividuals(fit, group = grp, title = "Individuals by group")
```

plotInds

Base-R scatter plot of the individuals on the shared components

Description

Draws the individuals in the plane of two shared canonical components, optionally coloured by a grouping factor and labelled with the individual identifiers. This is the original base-graphics plot; [plotIndividuals](#) is the modern replacement and returns a ggplot object instead of drawing on the current device.

Usage

```
plotInds(
  x,
  group,
  ax1 = 1,
  ax2 = 2,
  col.list,
  print.labels = FALSE,
  cex.label = 0.8,
  pos.leg = "bottomright",
  main = NULL,
  ...
)
```

Arguments

<code>x</code>	an object of class 'mgcca'
<code>group</code>	optional factor giving the group of each individual, in the row order of <code>x\$Y</code> . When omitted every individual is put in a single group and no legend is drawn.
<code>ax1, ax2</code>	the components on the horizontal and vertical axes. Defaults 1 and 2.
<code>col.list</code>	vector of colours, one per level of group. When omitted a fixed built-in palette is used, so repeated calls give the same colours and no random number is drawn; it must be at least as long as the number of levels.
<code>print.labels</code>	logical; when TRUE the individual names are written instead of plotting points (using wordcloud to spread the labels out when that package is available). Default FALSE.
<code>cex.label</code>	character expansion for those labels. Default 0.8.
<code>pos.leg</code>	position of the legend, as in legend . Default "bottomright".
<code>main</code>	plot title. Default NULL.
<code>...</code>	further graphical arguments passed to plot .

Value

No return value; called for the plot it draws on the current graphics device.

See Also

[plotIndividuals](#), [plotVars](#)

Examples

```
data(cardiovascular)
u <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))
sel <- u[1:150]
mk <- function(d, cols = NULL) {
  m <- as.matrix(d[rownames(d) %in% sel, , drop = FALSE])
  if (!is.null(cols)) m <- m[, cols, drop = FALSE]
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = mk(X1, 1:20), clinical = mk(X2), other = mk(X3))
fit <- mgcca(X, nfac = 2, method = "penalized", lambda = rep(0.1, 3))

plotInds(fit, main = "Individuals")

## coloured by an arbitrary two-level grouping
grp <- factor(rep(c("a", "b"), length.out = nrow(fit$Y)))
plotInds(fit, group = grp, col.list = c("steelblue", "tomato"))
```

plotLoadings

Barplot of the top variable loadings for one component

Description

For a chosen table and shared component, shows the variables most correlated with that component as a horizontal bar chart (signed correlation, coloured by sign). Complements the correlation circle of [plotVariables](#) and scales to omics tables via `top`.

Usage

```
plotLoadings(x, table = 1, comp = 1, top = 15, title = NULL)
```

Arguments

<code>x</code>	an mgcca object.
<code>table</code>	table name or index into <code>x\$corsY</code> . Default 1.
<code>comp</code>	component to display. Default 1.
<code>top</code>	number of top variables (by absolute correlation). Default 15.
<code>title</code>	plot title.

Value

A ggplot object.

See Also

[plotVariables](#), [getSignif](#)

Examples

```
data(cardiovascular)
u <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))
sel <- u[1:150]
mk <- function(d, cols = NULL) {
  m <- as.matrix(d[rownames(d) %in% sel, , drop = FALSE])
  if (!is.null(cols)) m <- m[, cols, drop = FALSE]
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = mk(X1, 1:20), clinical = mk(X2), other = mk(X3))
fit <- mgcca(X, nfac = 2, method = "penalized", lambda = rep(0.1, 3))

plotLoadings(fit, table = "methylation", comp = 1, top = 10)
plotLoadings(fit, table = "clinical", comp = 2)
```

plotScores*Scatter of one table's canonical scores*

Description

Plots the per-table scores (X_j projected onto its canonical weights) for one table on two components – each table's own view of the individuals. Requires the fit to have been computed with `scores = TRUE`.

Individuals that are not part of the chosen table have no score in it (NA) and are left out of the plot, with a message saying how many. They are not drawn at the origin, which would put them where a genuinely average measured individual sits. Use [plotIndividuals](#) on `Y` for a view that covers everyone.

Usage

```
plotScores(  
  x,  
  table = 1,  
  comps = c(1, 2),  
  group = NULL,  
  ellipse = TRUE,  
  level = 0.95,  
  title = NULL  
)
```

Arguments

<code>x</code>	an mgcca object.
<code>table</code>	table name or index into <code>x\$scores</code> . Default 1.
<code>comps</code>	the two components to display. Default <code>c(1, 2)</code> .
<code>group</code>	optional factor to colour by (matched to individuals as in plotIndividuals).
<code>ellipse, level</code>	passed through to draw per-group confidence ellipses.
<code>title</code>	plot title. Default: the table name.

Value

A ggplot object.

See Also

[plotIndividuals](#), [plotBiplot](#)

Examples

```
data(cardiovascular)
u <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))
sel <- u[1:150]
mk <- function(d, cols = NULL) {
  m <- as.matrix(d[rownames(d) %in% sel, , drop = FALSE])
  if (!is.null(cols)) m <- m[, cols, drop = FALSE]
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = mk(X1, 1:20), clinical = mk(X2), other = mk(X3))
## per-table scores are only produced when scores = TRUE
fit <- mgcca(X, nfac = 2, method = "penalized", lambda = rep(0.1, 3),
  scores = TRUE)

## individuals absent from the clinical table are dropped, with a message
plotScores(fit, table = "clinical")
```

plotSree

Scree plot of the shared-component eigenvalues

Description

Bar plot of the eigenvalues (inner-model AVE) associated with each shared component.

Usage

```
plotSree(x, title = "Eigenvalues")
```

Arguments

x	an mgcca object.
title	plot title. Default "Eigenvalues".

Value

A ggplot object.

See Also

[plotAVE](#)

Examples

```
data(cardiovascular)
u <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))
sel <- u[1:150]
mk <- function(d, cols = NULL) {
  m <- as.matrix(d[rownames(d) %in% sel, , drop = FALSE])
  if (!is.null(cols)) m <- m[, cols, drop = FALSE]
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = mk(X1, 1:20), clinical = mk(X2), other = mk(X3))
fit <- mgcca(X, nfac = 3, method = "penalized", lambda = rep(0.1, 3))

plotScree(fit)
```

plotVariables

Correlation circle of variables for one table

Description

For a chosen table, draws the correlation of each variable with the two shared components as a correlation circle (arrows within the unit circle). Only the top variables with the largest correlation radius are drawn as labelled arrows, so the plot stays readable even for omics-scale tables; the rest are shown as faint points.

Use `plotVariables` to look at a single table in detail, as a **ggplot2** correlation circle, and `plotVars` to compare several tables at once – one base-graphics panel per table, drawn through **made4**, with the same chosen variables highlighted in each.

Usage

```
plotVariables(x, table = 1, comps = c(1, 2), top = 10, title = NULL)
```

Arguments

<code>x</code>	an mgcca object.
<code>table</code>	table to display: a name or an index into <code>x\$corsY</code> . Default 1.
<code>comps</code>	the two components to display. Default <code>c(1, 2)</code> .
<code>top</code>	number of top variables (by correlation radius) to draw as labelled arrows. Default 10.
<code>title</code>	plot title. Default: the table name.

Value

A ggplot object.

See Also

[plotVars](#), [plotIndividuals](#), [getSignif](#)

Examples

```
data(cardiovascular)
u <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))
sel <- u[1:150]
mk <- function(d, cols = NULL) {
  m <- as.matrix(d[rownames(d) %in% sel, , drop = FALSE])
  if (!is.null(cols)) m <- m[, cols, drop = FALSE]
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = mk(X1, 1:20), clinical = mk(X2), other = mk(X3))
fit <- mgcca(X, nfac = 2, method = "penalized", lambda = rep(0.1, 3))

## the clinical table has few variables: label them all
plotVariables(fit, table = "clinical", top = 6)

## on an omics-scale table only the leading variables are labelled
plotVariables(fit, table = 1, top = 5)
```

plotVars

Base-R correlation plots of the variables, one panel per table

Description

Lays out one panel per selected table and, in each, plots the correlation of that table's variables with two shared canonical components (via `made4::plotgenes`). Variables named in `var` are highlighted, which makes it easy to follow a handful of variables of interest across tables. This is the original base-graphics plot; [plotVariables](#) is the modern per-table replacement and returns a `ggplot` object.

Use `plotVars` to compare several tables at once – one base-graphics panel per table, drawn through **made4**, with the same chosen variables highlighted in each – and [plotVariables](#) to look at a single table in detail, as a **ggplot2** correlation circle.

Requires the **made4** package (Suggests).

Usage

```
plotVars(
  x,
  var = NA,
  axes = c(1, 2),
  var.col = "red",
  var.lab = FALSE,
  bg.var.col = "gray",
```

```

    nlab = 0,
    df = NA,
    layout = NA,
    tit = "",
    tit.pos = 0,
    tit.cex = 1,
    ...
)

```

Arguments

<code>x</code>	an object of class 'mgcca'
<code>var</code>	character vector of variable names to highlight in every panel. Default NA, i.e. highlight nothing.
<code>axes</code>	the two components to plot. Default <code>c(1, 2)</code> .
<code>var.col</code>	colour(s) for the highlighted variables: either a single colour or one per element of <code>var</code> . Default "red".
<code>var.lab</code>	logical; write the names of the highlighted variables next to their points. Default FALSE.
<code>bg.var.col</code>	colour(s) for the remaining ("background") variables: either a single colour or one per selected table. Default "gray".
<code>nlab</code>	number of the most extreme variables that <code>made4::plotgenes</code> labels in each panel. Default 0.
<code>df</code>	tables to plot: a vector of positions in <code>x\$corsY</code> or of table names. The default NA means all tables.
<code>layout</code>	an explicit matrix passed to layout . By default (NA) a layout is chosen from the number of tables.
<code>tit</code>	panel title. Default "".
<code>tit.pos</code>	title line position, as line in title . Default 0.
<code>tit.cex</code>	title character expansion. Default 1.
<code>...</code>	further arguments passed to <code>made4::plotgenes</code> .

Value

No return value; called for the plots it draws on the current graphics device. When some names in `var` match no variable in any table, a note and a summary table are printed to the console.

See Also

[plotVariables](#), [plotInds](#)

Examples

```
if (requireNamespace("made4", quietly = TRUE)) {
  data(cardiovascular)
  u <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))
  sel <- u[1:150]
  mk <- function(d, cols = NULL) {
    m <- as.matrix(d[rownames(d) %in% sel, , drop = FALSE])
    if (!is.null(cols)) m <- m[, cols, drop = FALSE]
    storage.mode(m) <- "double"
    m
  }
  X <- list(methylation = mk(X1, 1:20), clinical = mk(X2), other = mk(X3))
  fit <- mgcca(X, nfac = 2, method = "penalized", lambda = rep(0.1, 3))

  ## one table, highlighting two clinical variables
  plotVars(fit, df = 2, var = c("HDL", "Gluc"),
           var.col = c("red", "blue"), var.lab = TRUE)
}
```

predict.mgcca

Project new individuals onto the shared canonical space

Description

Applies the per-table weights learned by [mgcca](#) to new individuals, returning their canonical scores per table. The new tables are centred/scaled with the *training* parameters, so the projection is consistent with the fit. Requires a fit produced with `scores = TRUE` (weights) and `scale = TRUE` (scaling parameters).

Usage

```
## S3 method for class 'mgcca'
predict(object, newdata, ...)
```

Arguments

<code>object</code>	an mgcca object.
<code>newdata</code>	a named list of new tables (individuals x variables) with the same variable columns used at fit time; tables not seen at fit time are ignored. Only the tables present are projected.
<code>...</code>	ignored.

Value

A named list of score matrices (new individuals x components), one per projected table.

See Also

[mgcca](#), [plotScores](#)

Examples

```
data(cardiovascular)
ids <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))
num <- function(d, keep, cols = seq_len(ncol(d))) {
  m <- as.matrix(d[rownames(d) %in% keep, cols, drop = FALSE])
  storage.mode(m) <- "double"
  m
}
train <- ids[1:150]
test <- ids[151:180]
X <- list(methylation = num(X1, train, 1:20), clinical = num(X2, train),
          other = num(X3, train))
Xn <- list(methylation = num(X1, test, 1:20), clinical = num(X2, test),
          other = num(X3, test))

## weights (scores = TRUE) and scaling (scale = TRUE) are what a
## projection needs; both are on by default except for scores.
fit <- mgcca(X, nfac = 2, method = "penalized", lambda = rep(0.1, 3),
             scores = TRUE)
proj <- predict(fit, newdata = Xn)
vapply(proj, nrow, integer(1))
head(proj$clinical)
```

print.ListMAE

Print method for 'ListMAE' objects

Description

Compact print for a MultiAssayExperiment-like list, as returned by [getTables](#): reports the class, the number of assays and their names.

Usage

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

Arguments

x Object of class "ListMAE", as returned by [getTables](#).

... Further arguments (ignored).

Value

Invisibly the assay names of x; called for the side effect of printing the summary.

See Also

[getTables](#), which produces the object printed here.

Examples

```

if (requireNamespace("MultiAssayExperiment", quietly = TRUE)) {
  data(cardiovascular)
  sel <- rownames(X2)[1:20]
  a1 <- t(as.matrix(X1[rownames(X1) %in% sel, 1:8, drop = FALSE]))
  a2 <- t(as.matrix(X2[rownames(X2) %in% sel, , drop = FALSE]))
  mae <- MultiAssayExperiment::MultiAssayExperiment(
    MultiAssayExperiment::ExperimentList(
      list(methylation = a1, clinical = a2)))

  tabs <- getTables(mae)
  class(tabs)          # "ListMAE"
  tabs                 # dispatches to print.ListMAE()
}

```

print.mgcca

Print an mgcca object

Description

One-screen overview of the fit. The spectral gap line is the external gap $\mu_L - \mu_{L+1}$ of the mgcca operator, relative to μ_L : it says whether the leading L -dimensional subspace is numerically identified at all. A gap at machine zero is flagged NOT IDENTIFIED – the solver converged, but the directions it returned are arbitrary (see [mgcca](#)).

Usage

```

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

```

Arguments

```

x          an mgcca object.
...        ignored.

```

Value

Called for its side effect (the overview is written to the console). Returns x invisibly.

Examples

```

data(cardiovascular)
ids <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))[1:150]
num <- function(d, cols = seq_len(ncol(d))) {
  m <- as.matrix(d[rownames(d) %in% ids, cols, drop = FALSE])
  storage.mode(m) <- "double"
  m
}

```

```

}
X <- list(methylation = num(X1, 1:20), clinical = num(X2), other = num(X3))

fit <- mgcca(X, nfac = 2, method = "penalized", lambda = rep(0.1, 3),
            scores = TRUE)
print(fit)

```

```
print.mgcca_block_profile
```

Print a block contribution profile

Description

Prints the per-query, per-block table built by [mgcca_block_profile](#), preceded by a one-line header with the number of queries and blocks and the grouping the sensitivity was run with. When no grouping was supplied, a closing note says so: the between-group columns are undefined there, not missing.

Usage

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

Arguments

x	An object of class "mgcca_block_profile".
...	ignored.

Value

Called for its side effect (the profile is written to the console). Returns x invisibly.

See Also

[mgcca_block_profile](#)

Examples

```
## A minimal stand-in for an ungrouped mgcca_sensitivity() result; see
## mgcca_block_profile() for why the example does not fit one here.
sens <- structure(
  list(by_block = data.frame(
    query = rep(c("age", "bmi"), each = 3),
    block = rep(c("methylation", "clinical", "other"), 2),
    S_total = c(0.62, 0.25, 0.13, 0.31, 0.44, 0.25),
    share_total = c(0.62, 0.25, 0.13, 0.31, 0.44, 0.25),
    stringsAsFactors = FALSE),
    groups = NULL,
    settings = list(grouped = FALSE),

```

```

    provenance = list(file = NA_character_, datasets = NULL),
    call = NULL),
    class = "mgcca_sensitivity")

print(mgcca_block_profile(sens))

```

```
print.mgcca_lambda
```

Print a lambda selection

Description

Print a lambda selection

Usage

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

Arguments

`x` an mgcca_lambda object, from [mgcca_select_lambda](#).
`...` ignored.

Value

`x`, invisibly. Called for the one-screen summary it writes to the console: the criterion, the selected penalty, the held-out cross-block agreement that justifies it, and how many candidates lie within one standard error of the best.

See Also

[mgcca_select_lambda](#)

Examples

```

data(cardiovascular, package = "mgcca")
ids <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))[1:150]
mk <- function(d, j = NULL) {
  m <- as.matrix(d[rownames(d) %in% ids, , drop = FALSE])
  if (!is.null(j)) m <- m[, j, drop = FALSE]
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = mk(X1, 1:20), clinical = mk(X2), cells = mk(X3))

sel <- mgcca_select_lambda(X, nfac = 2, gamma = c(0.01, 0.1, 1), K = 3,
                          seed = 1)

print(sel)

```

summary.mgcca	<i>Summarise an mgcca object</i>
---------------	----------------------------------

Description

Prints the model overview, the variance explained per table (outer-model AVE), the pairwise overlap between tables and, for each table and component, the top variables most correlated with the shared components.

Usage

```
## S3 method for class 'mgcca'
summary(object, top = 5, ...)
```

Arguments

object	an mgcca object.
top	number of top variables per component to list. Default 5.
...	ignored.

Details

The overlap block reports, for every pair of tables (j, k) , the number n_{jk} of individuals both tables hold and its normalised version $\alpha_{jk} = n_{jk}/n$ over the union, plus the dispersion of α across pairs. It is descriptive: nothing in the fit depends on it. It is worth reading because it is the structural quantity that decides how much the treatment of missing individuals can matter – with α flat and near 1 every sensible treatment agrees, and the choice only starts to bite when the pairs disagree about who they share. Two notes are printed when they apply: one when the dispersion of α is large, one when some pair shares very few individuals. Both are reporting heuristics with no inferential status.

Value

Called for its side effect (the overview, the per-table AVE, the pairwise overlap and the top variables are written to the console). Returns object invisibly.

Examples

```
data(cardiovascular)
ids <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))[1:150]
num <- function(d, cols = seq_len(ncol(d))) {
  m <- as.matrix(d[rownames(d) %in% ids, cols, drop = FALSE])
  storage.mode(m) <- "double"
  m
}
X <- list(methylation = num(X1, 1:20), clinical = num(X2), other = num(X3))

fit <- mgcca(X, nfac = 2, method = "penalized", lambda = rep(0.1, 3),
```

```

      scores = TRUE)
summary(fit, top = 3)

```

topVars

Variables most correlated with a shared component

Description

Lists, table by table, the variables at one end of a shared canonical component. By default it returns the topN variables with the largest correlation on that side; if pval.cut is supplied it returns instead every variable whose correlation is significant at that level and has the requested sign, which needs a fit carrying p-values.

Usage

```
topVars(x, axis = 1, end = "pos", topN = 5, pval.cut)
```

Arguments

x	an object of class 'mgcca'
axis	axis to take in to account to get the top values
end	which end of the component to look at: "pos" (default) for the most positively correlated variables, "neg" for the most negatively correlated ones.
topN	number of values to return with topVars if not pval.cut is introduced
pval.cut	significance level to cut data to return

Value

A named list with one character vector per table of x, holding the selected variable names. Without pval.cut each vector has the same length (topN, capped at the number of variables in the smallest table); with pval.cut the lengths vary and a vector may be empty.

See Also

[getSignif](#), [plotLoadings](#)

Examples

```

data(cardiovascular)
u <- Reduce(union, list(rownames(X1), rownames(X2), rownames(X3)))
sel <- u[1:150]
mk <- function(d, cols = NULL) {
  m <- as.matrix(d[rownames(d) %in% sel, , drop = FALSE])
  if (!is.null(cols)) m <- m[, cols, drop = FALSE]
  storage.mode(m) <- "double"
  m
}

```

```
X <- list(methylation = mk(X1, 1:20), clinical = mk(X2), other = mk(X3))
fit <- mgcca(X, nfac = 2, method = "penalized", lambda = rep(0.1, 3))

## three most positively correlated variables per table, first component
topVars(fit, axis = 1, topN = 3)

## and the negative end
topVars(fit, axis = 1, end = "neg", topN = 3)

## everything significant with a positive correlation
topVars(fit, axis = 1, pval.cut = 0.05)
```

Index

- * **datasets**
 - cardiovascular, [3](#)
- bdCreate_hdf5_matrix, [21](#)
- cardiovascular, [3](#)
- corTestP, [4](#)
- genCoords, [5](#)
- getSignif, [6](#), [11](#), [25](#), [50](#), [54](#), [62](#)
- getTables, [8](#), [9](#), [21](#), [57](#)
- impute.knn, [37](#), [38](#)
- layout, [55](#)
- legend, [49](#)
- matrixChr2Num, [8](#), [9](#)
- mgcca, [3](#), [4](#), [7](#), [8](#), [10](#), [13–15](#), [17](#), [22](#), [23](#), [25–27](#),
[29](#), [30](#), [33](#), [37](#), [56](#), [58](#)
- mgcca_associate, [13](#), [24](#)
- mgcca_audit, [14](#), [18](#), [41](#)
- mgcca_audit_load, [15](#), [17](#), [18](#)
- mgcca_block_profile, [19](#), [33](#), [42](#), [59](#)
- mgcca_import_hdf5, [12](#), [14](#), [20](#)
- mgcca_load, [11](#), [22](#), [30](#)
- mgcca_permtest, [13](#), [23](#)
- mgcca_rcpp, [10](#), [12](#), [25](#)
- mgcca_results, [4](#), [11–13](#), [22](#), [24](#), [47](#)
- mgcca_select_lambda, [26](#), [36](#), [60](#)
- mgcca_sensitivity, [11](#), [19](#), [20](#), [30](#), [34–36](#), [44](#)
- mgcca_stability, [11](#), [33](#), [34](#)
- mgccaImpute, [37](#)
- plot, [49](#)
- plot.mgcca, [38](#)
- plot.mgcca_audit, [39](#)
- plot.mgcca_block_profile, [42](#)
- plot.mgcca_sensitivity, [43](#)
- plot.mgcca_stability (mgcca_stability),
[34](#)
- plotAVE, [45](#), [48](#), [52](#)
- plotBiplot, [46](#), [51](#)
- plotIndividuals, [10–13](#), [38](#), [39](#), [46](#), [47](#), [48](#),
[49](#), [51](#), [54](#)
- plotInds, [25](#), [26](#), [47](#), [48](#), [55](#)
- plotLoadings, [50](#), [62](#)
- plotScores, [51](#), [56](#)
- plotScree, [45](#), [52](#)
- plotVariables, [7](#), [10](#), [11](#), [46](#), [48](#), [50](#), [53](#), [54](#),
[55](#)
- plotVars, [25](#), [26](#), [49](#), [53](#), [54](#), [54](#)
- predict.mgcca, [56](#)
- print.ListMAE, [57](#)
- print.mgcca, [58](#)
- print.mgcca_audit (mgcca_audit), [14](#)
- print.mgcca_block_profile, [59](#)
- print.mgcca_lambda, [60](#)
- print.mgcca_sensitivity
(mgcca_sensitivity), [30](#)
- print.mgcca_stability
(mgcca_stability), [34](#)
- summary, [26](#)
- summary.mgcca, [61](#)
- summary.mgcca_audit (mgcca_audit), [14](#)
- summary.mgcca_sensitivity
(mgcca_sensitivity), [30](#)
- summary.mgcca_stability
(mgcca_stability), [34](#)
- title, [55](#)
- topVars, [7](#), [25](#), [62](#)
- X1 (cardiovascular), [3](#)
- X2 (cardiovascular), [3](#)
- X3 (cardiovascular), [3](#)