Package 'mgcca'

November 20, 2026

Type Package

Title Generalized Canonical Correlation Analysis with Missing Individuals

Version 0.99.8

Date 2026-11-12

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), impute, SummarizedExperiment, grid, methods, stats, utils

LinkingTo Rcpp, RcppEigen, Rhdf5lib, BigDataStatMeth, BH

Suggests BiocStyle, knitr, rmarkdown, testthat (>= 3.0.0), MultiAssayExperiment, Biobase, GenomicRanges, IRanges, BiocGenerics, RSpectra, GenomeInfoDb, curatedTCGAData, 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

git_url https://git.bioconductor.org/packages/mgcca

git_branch devel

git_last_commit d796172

git_last_commit_date 2026-11-12

Repository Bioconductor 3.24

Date/Publication 2026-11-19

Author Dolors Pelegri-Siso [aut, cre] (ORCID:
<https://orcid.org/0000-0002-5993-3003>),
Isaac Subirana [aut],
Juan R. Gonzalez [aut] (ORCID: <https://orcid.org/0000-0003-3267-2146>)

Maintainer Dolors Pelegri-Siso <dolors.pelegri@isglobal.org>

Contents

mgcca-package 4
cardiovascular 4
corTestP 6
genCoords 6
getSignif 8
getTables 9
matrixChr2Num 10
mgcca 11
mgccaAssociate 14
mgccaAudit 15
mgccaAuditLoad 19
mgccaBlockProfile 20
mgccaImportHDF5 21
mgccaImpute 22
mgccaLambdaScaleHDF5 24
mgccaLoad 25
mgccaPermtest 27
mgccaResults 28
mgccaSelectLambda 30
mgccaSensitivity 34
mgccaStability 37
mgcca_block_gram 41
mgcca_cor_ave_rcpp 42
mgcca_eigen_rcpp 42
mgcca_file_is_open_rcpp 43
mgcca_getK_rcpp 43
mgcca_hold_file_rcpp 44
mgcca_lambda_gram_rcpp 44
mgcca_list_group_rcpp 45
mgcca_rcpp 46
mgcca_read_attrs_rcpp 47
mgcca_read_audit_rcpp 47
mgcca_read_blocks_rcpp 48
mgcca_read_dimensions_rcpp 48
mgcca_read_manifest_rcpp 49

mgcca_read_rownames_rcpp . . . 49
mgcca_release_file_rcpp . . . 50
mgcca_save_audit_rcpp . . . 50
mgcca_scores_rcpp . . . 51
mgcca_scores_svd_rcpp . . . 51
mgcca_solve_rank_guard_rcpp . . . 52
mgcca_svd_rcpp . . . 52
mgcca_write_attrs_rcpp . . . 53
mgcca_write_manifest_rcpp . . . 54
mgcca_XKX_rcpp . . . 54
plot.mgcca . . . 55
plot.mgcca_audit . . . 55
plot.mgcca_block_profile . . . 58
plot.mgcca_sensitivity . . . 59
plotAVE . . . 61
plotBiplot . . . 62
plotIndividuals . . . 63
plotLoadings . . . 64
plotScores . . . 65
plotScree . . . 66
plotVariables . . . 67
predict.mgcca . . . 68
print.ListMAE . . . 69
print.mgcca . . . 70
print.mgcca_block_profile . . . 71
print.mgcca_lambda . . . 72
reliability_block_inputs . . . 73
reliability_gramc_block_hdf5 . . . 73
reliability_gramc_block_mem . . . 74
reliability_gram_block . . . 74
reliability_gram_hdf5 . . . 75
reliability_proj_overlap . . . 75
reliability_query . . . 76
reliability_read_full_test . . . 76
reliability_reference_fit . . . 77
reliability_sensitivity_gram . . . 77
reliability_sensitivity_gram_matrix . . . 78
reliability_subspace_fit . . . 78
reliability_subspace_overlap . . . 79
reliability_tcrossprod_hdf5 . . . 80
summary.mgcca . . . 80
topVars . . . 81

Index **83**

`mgcca-package` *mgcca: Generalized Canonical Correlation Analysis with missing individuals*

Description

Generalized Canonical Correlation Analysis for tables with missing individuals (whole missing rows; van de Velden and Bijmolt, 2006) for multi-block omics integration, running its numerical core in C++ on HDF5 files via the BigDataStatMeth API. Missing cells within a measured individual are outside the estimator's scope and must be handled before import. See `mgcca` to get started.

Details

See `vignette("mgcca_example", package = "mgcca")` for a worked analysis and `vignette("mgcca_reliability", package = "mgcca")` for the sensitivity and stability tools.

Exported names follow one convention: `mgcca_*` for the core workflow (the fit and the analyses built on it), and camelCase for the helpers (data preparation, extraction and plotting).

Author(s)

Maintainer: Dolors Pelegri-Siso <dolors.pelegri@isglobal.org> (ORCID)

Authors:

- Isaac Subirana
- Juan R. Gonzalez <juanr.gonzalez@isglobal.org> (ORCID)

See Also

Useful links:

- https://isglobal-brge.github.io/mgcca/
- https://github.com/isglobal-brge/mgcca
- http://brge.isglobal.org
- Report bugs at https://github.com/isglobal-brge/mgcca/issues

`cardiovascular` *Simulated cardiovascular multi-table example data*

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 `mgccaResults` 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 — *Subset a MultiAssayExperiment object by specific genomic coordinates*

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

multiassayexperiment	a MultiAssayExperiment object whose assays carry annotation (rowRanges, or rowData for the assay pointed at by meth.index).
gen.coords	a GRanges object holding *one* region. Several ranges are an error: pass them one at a time.
meth.index	is the index where the methylation data (or a SummarizedExperiment non Ranged) is located inside the mae.
col.name	name of the rowData column of the assay pointed at by meth.index that holds the feature position, in the same coordinate system as gen.coords.
seqname.col	name of the rowData column of that same assay that holds the chromosome, so that features at the same position on another chromosome are dropped. The default NULL looks for a column called (case-insensitively) seqnames, seqname, chr, chrom or chromosome; 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
}
```

`getSignif` *Variables significantly correlated with the shared components*

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

`x`	an object of class 'mgcca'
`df`	tables to inspect: a vector of positions in `x$pval.cor` or of table names. The default `NA` means all tables.
`pval.cut`	significance level. A variable is reported when any of its component p-values is strictly below this value. Default 0.05.
`...`	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` *Convert a character table in a list of tables to numeric*

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,
  gamma = 1,
  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 `MultiAssayExperiment`, an `ExpressionSet`/ `SummarizedExperiment`, 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 `NULL`, mgcca fits *in memory* 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 `MultiAssayExperiment`, a `SummarizedExperiment` or an `ExpressionSet` (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` *is* the choice of backend – there is no automatic fallback by size.

`group` Input HDF5 group. Default `"MGCCA_IN"`.

`datasets` Optional dataset names (subset/order).

`nfac` Number of shared components. Default 2.

`scale` If TRUE (default), column-center+scale each table before analysis, with base-R `scale()` semantics (sd with the n-1 divisor), over the individuals present in each table. On the file-backed path (`filename` given) the scaling is performed *inside* 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 (`filename = NULL`) it is done in RAM on the materialised tables.

`method` Inversion method: `"solve"` (SPD Cholesky), `"penalized"` (penalty from `lambda`, computed when it is left unset), or `"geninv"`/`"ginv"` (Moore-Penrose pseudoinverse). On the covariance route (see `route`), `"solve"` needs an ordinary inverse of the table's cross-product $X_j^\top K_j X_j$. A table whose variables are not linearly independent on its measured individuals (at least as many variables as measured individuals when the columns are centred, or a constant or repeated variable) has none, and is refused with an error naming it rather than inverted from rounding error.

`lambda` The penalty for `method = "penalized"`, either given or computed. Pass a numeric vector (one value per table, or one value recycled to all of them) to set it yourself; the fit then uses exactly those values. Leave it `NULL` (the default) and mgcca computes the scale-matched penalty from the data: $\lambda_j = s_j$, where s_j is the mean positive eigenvalue of table j's Gram, which is the scale of the eigenvalues the penalty shrinks. A fixed absolute λ does not hold the amount of regularisation constant as a table gets wider, so the scale-matched value is the one that is comparable across tables of very different widths. The strength of that computed penalty is set by `gamma`. The values actually used are reported with a message and saved in the file's manifest, and they are the same numbers `mgccaSelectLambda` reports for the same `gamma`. On the file-backed path the computation reads each table in column chunks, so it never holds a table in RAM; see `mgccaLambdaScaleHDF5`.

`gamma` How strong the *computed* penalty should be, as a multiple of each table's own scale: the penalty used is $\lambda_j = \gamma s_j$. Default 1. Raise it to regularise harder, lower it to regularise less; because it multiplies each table's own s_j, one `gamma` means the same amount of regularisation in every table however wide they are, which a single absolute λ cannot do. It applies only when the penalty is computed: with an explicit `lambda`, or with any method other than `"penalized"`, there is nothing for it to scale and it is ignored with a warning. Use `mgccaSelectLambda` to choose it by held-out cross-block agreement instead of by hand.

`scores` If TRUE, also compute per-table weights and scores. A table's scores are standardised over the individuals *present in that table*, and individuals absent from it get `NA` (`NaN` in the HDF5 file) rather than a score of zero. The returned weights carry that rescaling (see `mgccaResults`). Only the shared components `Y` are defined for every individual of the union.

`route` Per-table algebra route: `"auto"` (default) picks per table by `min(n, p)` (covariance `X'X}` when `\code{p < n}, Gram \code{XX'` when `p >= n`); `"cov"` or `"dual"` force one route for all tables. The Gram/dual route is the one that scales to `p >> n` (e.g. full methylation) without ever forming a `p x p` matrix.

threads	Optional thread count.
overwrite	If TRUE (default), overwrite the file/datasets on import.
collect	If TRUE (default), read the results back from HDF5 and return an in-memory object of class "mgcca" (via mgccaResults) ready for plotIndividuals / plotVariables / getSignif; the HDF5 descriptor is kept on its "desc" attribute. With collect = FALSE the lightweight descriptor list is returned instead (results still live in the HDF5 file). On the in-memory path (filename = NULL) collect = FALSE has no meaning – there is no file to leave the results in – and is refused with a clear error.
outputs	Which components to collect when collect = TRUE. NULL (default) collects all of them – exactly the behaviour of every previous version. A character subset of "Y", "blocks_present", "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 collect = FALSE. See mgccaResults.

Value

With collect = TRUE (default), an object of class "mgcca" (see mgccaResults); 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 mgccaLoad, and the reliability analyses (mgccaSensitivity, mgccaStability) 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 $n_{present} - 2$ 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

mgccaResults, plotIndividuals, mgccaImportHDF5, 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 mgccaLoad()).

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)
```

`mgccaAssociate` *Associate the shared components with external phenotypes*

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

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

Arguments

`x`	an `mgcca` object (from `mgcca` / `mgccaResults`).
`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`, `mgccaPermtest`

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

`mgccaAudit` *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

```
mgccaAudit(
  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, ...)

mgcca_audit(...)
```

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 MultiAssayExperiment, a SummarizedExperiment, an ExpressionSet, a path to an HDF5 file holding the blocks, or the descriptor list (filename, group, datasets) returned by mgccaImportHDF5. 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 ($individuals). 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: ess_min (minimum effective count, default 30, applied to stratum ESS and to pair support), availability_min (minimum realised availability, default 0.05) and max_weight (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 availability_audit group of that same file, so the certificate travels with the data, and read it back whole with mgccaAuditLoad. 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 saveRDS-able.
x	An object of class "mgcca_audit", as returned by mgccaAudit().
object	An object of class "mgcca_audit", as returned by mgccaAudit().
...	Ignored by print() and summary(). For mgcca_audit(): the arguments of mgccaAudit(), passed on unchanged.

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.

Former name

`mgcca_audit()` is the name this function had before version 0.99.8. It still works, without a warning, so that code and published analyses that call it keep running: it passes its arguments to `mgccaAudit()` unchanged and returns the same result. It will be removed in a later release; new code should call `mgccaAudit()`.

See Also

`mgcca` for the estimator this audit deliberately does not touch, and `mgccaAuditLoad` 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 <- mgccaAudit(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(mgccaAudit(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))))
mgccaAudit(blocks, L = 2)$graph$component
```

`mgccaAuditLoad` *Read a stored availability audit back from HDF5*

Description

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

Usage

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

Arguments

`file`	Path to the HDF5 file holding the audit.
`group`	Group the audit was written to. Default `"availability_audit"`, which is where `mgccaAudit()` puts it.
`check_inputs`	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 `mgccaAudit`.

See Also

`mgccaAudit`

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 <- mgccaImportHDF5(blocks, filename = h5, overwriteFile = TRUE)

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

---

mgccaBlockProfile *Per-block contribution profile of a sensitivity result*

Description

Reformats the block-level decomposition that `mgccaSensitivity` 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

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

Arguments

`x`	An object of class `"mgcca_sensitivity"`.
`query`	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

`mgccaSensitivity`

Examples

```
## mgccaSensitivity() 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 mgccaSensitivity()
## 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 <- mgccaBlockProfile(sens)
bp$profile

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

`mgccaImportHDF5` *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

```
mgccaImportHDF5(
  x,
  filename,
  group = "MGCCA_IN",
  datasets = NULL,
  overwriteDataset = FALSE,
  overwriteFile = FALSE
)
```

Arguments

`x`	Input data: a length-1 character path to an existing HDF5 file, a `MultiAssayExperiment`, an `ExpressionSet`/ `SummarizedExperiment`, or a named `list` of matrices.
`filename`	Target HDF5 file. Ignored (and may be missing) when `x` is itself an HDF5 path.
`group`	HDF5 group to write the tables under. Default `"MGCCA_IN"`.

datasets	Optional character vector of dataset names. When x is an existing HDF5 file, restricts/orders the tables to use; otherwise names the written datasets (defaults to the names of x).
overwriteDataset	Logical, overwrite datasets if present. Default FALSE.
overwriteFile	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 <- mgccaImportHDF5(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)
```

---

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

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

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]])
}
```

`mgccaLambdaScaleHDF5` *Scale-matched block scale from HDF5, over column chunks*

Description

Computes the per-block scale s_j of the scale-matched penalty parameterisation for tables held in an HDF5 file, reading each block in column chunks so that neither the block nor its scaled copy is ever held in memory. The penalty is $\lambda_j = \gamma s_j$, with $s_j = \mathrm{tr}(G_j)/\mathrm{rank}(G_j)$ the mean positive eigenvalue of the Gram G_j of the scaled block: the scale of the eigenvalues the penalty actually shrinks. A fixed absolute λ does not hold the amount of regularisation constant as p_j grows, which is why the dimensionless γ is the comparable quantity.

Usage

```
mgccaLambdaScaleHDF5(
  filename,
  group = "MGCCA_IN",
  datasets = NULL,
  gamma = 1,
  scale = TRUE,
  chunk_cols = 5000,
  rank_tol = 1e-08,
  sd_tol = 1e-08
)
```

Arguments

`filename`	Path to the HDF5 file holding the tables.
`group`	HDF5 group holding the tables. Default `"MGCCA_IN"`, the group `mgccaImportHDF5` writes.
`datasets`	Optional character vector naming the tables to use, in order. Defaults to every dataset in `group`.
`gamma`	Numeric, the dimensionless ridge: finite positive values, length 1 (recycled) or one value per table. Default 1.
`scale`	Logical, whether to centre and scale the columns before forming the Gram. Default `TRUE`, the scale-matched rule.
`chunk_cols`	Number of variables read per chunk, capped at the block width. Default 5000.
`rank_tol`	Relative tolerance for the numerical rank of G_j: an eigenvalue counts when it exceeds `rank_tol` times the largest. Default `1e-8`.
`sd_tol`	Columns whose standard deviation is at or below this are dropped. Default `1e-8`.

Details

The rule applied is the package's own, as used by `mgccaSelectLambda`: each column is centred by its mean and scaled by its standard deviation, and a column whose centre or scale is non-finite, or whose scale is at or below `sd_tol`, is dropped. A column holding any non-finite value therefore drops out, and the result is finite.

Memory is one column chunk plus one $n \times n$ Gram per block, so `chunk_cols` sets the footprint directly: at `n` individuals a chunk costs `n * chunk_cols * 8` bytes. The result is invariant to `chunk_cols` up to floating-point summation order.

Because `sd_tol` is a threshold, the function reports how far the kept/dropped decision ran from it (`sd_min_kept`, `sd_max_dropped` in `details`) and warns when either sits within a factor of `1e-6` of `sd_tol`, where the decision would be at the mercy of the last bits of a standard deviation.

Value

A list with

`lambda` named numeric, `gamma * s`, ready for `mgcca`'s `lambda` argument.

`s` named numeric, the per-block scale s_j.

`gamma` the gamma used, one value per table.

`details` a data frame with one row per table: the sizes `n`, `p`, `n_kept`, `n_dropped`, the chunking `chunk_cols`, `n_chunks`, and the diagnostics `sym_err`, `sd_min_kept`, `sd_max_dropped`.

See Also

`mgccaSelectLambda` for choosing `gamma` by held-out cross-block agreement, and `mgccaImportHDF5` for writing tables into the file this function reads.

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 <- mgccaImportHDF5(X, filename = h5, overwriteFile = TRUE)

sm <- mgccaLambdaScaleHDF5(desc$filename, desc$group, desc$datasets)
sm$lambda
sm$details

## the same scales, read in narrower chunks
sm2 <- mgccaLambdaScaleHDF5(desc$filename, desc$group, desc$datasets,
                            chunk_cols = 3)
max(abs(sm2$s - sm$s) / abs(sm$s))
unlink(h5)
```

`mgccaLoad` *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 mgccaResults (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), mgccaLoad 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

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

mgcca_load(...)
```

Arguments

file	Path to the HDF5 results file.
group	HDF5 group holding the results. Default "FINAL_RESULTS".
...	For mgcca_load() only: the arguments of mgccaLoad(), passed on unchanged.

Value

An object of class "mgcca" (see mgccaResults), with the reconstructed provenance descriptor on attr(., "desc").

Former name

mgcca_load() is the name this function had before version 0.99.8. It still works, without a warning, so that code and published analyses that call it keep running: it passes its arguments to mgccaLoad() unchanged and returns the same result. It will be removed in a later release; new code should call mgccaLoad().

See Also

mgcca, mgccaResults

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

`mgccaPermtest` *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

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

Arguments

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

`mgccaAssociate`

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 <- mgccaPermtest(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 <- mgccaPermtest(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)
```

`mgccaResults` *Collect mgcca results from HDF5 into an in-memory 'mgcca' object*

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 `plotIndividuals`, `plotVariables`, `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

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

Arguments

x	Either the descriptor list returned by `mgcca()` / `mgcca_rcpp()` (with elements `filename`, `datasets`, `final_group`), or a length-1 character path to the HDF5 file.
datasets	Optional table names to read. Defaults to `x$datasets` when `x` is a descriptor, otherwise required.
final_group	HDF5 group holding the results. Default `x$final_group` or `"FINAL_RESULTS"`.
scores, pval	Whether to read the per-table scores / p-values (they exist only if `mgcca()` was called with `scores = TRUE` / the defaults). Default: infer from the descriptor, otherwise `TRUE`.
tmp_group	HDF5 group holding the intermediates, where the per-table presence masks `K` live. Only used for the overlap report; if the group is absent the report is `NULL` and nothing else changes. Default `"MGCCA_TMP"`.
outputs	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"`, `"blocks_present"`, `"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 `$corsY` on a partial object is `NULL` rather than an error. What was requested is recorded on `attr(., "outputs")`.

Value

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

`blocks_present` is an integer vector over the individuals of `Y`, named by them and in the same order: the number of tables in which each individual was measured (K_{ii}, at least 1). It comes from the presence masks the fit used and matches the pattern of `NA` in `scores`: an individual with `blocks_present = k` has a score in exactly `k` tables. It is `NULL` when the file no longer carries the masks (they live in `tmp_group`).

`weights` are the per-table coefficients that turn a table's variables (standardised, when the fit used `scale = TRUE`) into its scores, one column per component. They are rescaled: each column of

a table's coefficient matrix B_j is divided by the standard deviation of that table's scores over the individuals measured in it, so that every returned `scores` column has standard deviation 1 over the measured individuals. The coefficients before this rescaling stay in the HDF5 file, at `FINAL_RESULTS/A/<table>` (an in-memory fit does not keep them). `scaling` holds the centre and scale each variable was standardised with, when the fit used `scale = TRUE`.

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`, `plotIndividuals`, `plotVariables`

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 <- mgccaResults(desc)
dim(res$Y)
table(res$blocks_present)
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 <- mgccaResults(desc, outputs = "Y")
identical(resY$Y, res$Y)
names(Filter(Negate(is.null), resY))
unlink(h5)
```

`mgccaSelectLambda` *Choose the ridge penalty by held-out cross-block agreement*

Description

Selects the penalty `lambda` 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. Left without a `lambda`, `mgcca(method = "penalized")` computes the scale-matched penalty at the `gamma` it is given (default 1); this function is the way to choose `gamma` by held-out cross-block agreement.

Usage

```
mgccaSelectLambda(
  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
)

mgcca_select_lambda(...)
```

Arguments

`x`	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 `mgcca`.
`nfac`	Number of shared components the penalty is being chosen for. Default 2. The selected `lambda` is conditional on it.
`gamma`	Candidate grid for the *dimensionless* ridge (`scale_matched = TRUE`), where $\lambda_j = \gamma s_j$ and $s_j = \mathrm{tr}(G_j)/\mathrm{rank}(G_j)$ is the mean positive Gram eigenvalue of block j. Default `10^seq(-1, 1, by = 0.25)`.
`lambda`	Candidate grid of absolute penalties (`scale_matched = FALSE`), applied identically to every block. Default `c(0.05, 0.2, 0.5, 1, 2, 5, 10, 30, 100, 300, 1000)` – the grid of the reference calibration.
`scale_matched`	If `TRUE` (default) the grid is over `gamma` 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 `lambda` does not.
`K`	Number of folds. Default 5.
`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 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 nfac + 3.
rank_tol	Relative tolerance for the numerical rank of G_j when scale_matched = TRUE. Default 1e-8.
sd_tol	Columns whose training-fold standard deviation is below this are dropped inside the fold. Default 1e-8.
...	For mgcca_select_lambda() only: the arguments of mgccaSelectLambda(), passed on unchanged.

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 scale_matched = FALSE).
s	the per-block scales s_j on the full data (NULL when scale_matched = FALSE).
agreement	the mean held-out cross-block RV at the selection – the diagnostic that justifies it.
grid	the candidate trace: the tuning value, cv_mean, cv_se, n_cells (how many (fold, pair) cells were scorable), in_1se 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 $B_j = X_j'(X_jX_j' + \lambda_j I)^{-1}Y$ 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.

Former name

`mgcca_select_lambda()` is the name this function had before version 0.99.8. It still works, without a warning, so that code and published analyses that call it keep running: it passes its arguments to `mgccaSelectLambda()` unchanged and returns the same result. It will be removed in a later release; new code should call `mgccaSelectLambda()`.

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 <- mgccaSelectLambda(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)
```

`mgccaSensitivity` *Sensitivity of the shared subspace to a participant-level query*

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

```
mgccaSensitivity(
  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 `mgccaLoad` 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 mgccaSensitivity().
...	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.

The blocks are balanced before the subspace is read. Block j enters the layer's operator through $G_j R_j$, with G_j its participant Gram (standardised over its measured rows, zero for unmeasured participants) and $R_j = (G_j+\lambda_j I)^{-1}$. With $D = \mathrm{diag}(K_{ii}^{-1/2})$, K_{ii} the number of blocks participant i was measured in, each block's summand is $A_j = DG_jR_jD$. Its top-L ceiling $c_{j,L}$ is the sum of the L largest eigenvalues of A_j, the most that block could contribute to any L-dimensional subspace, and its weight is $w_j = Jc_{j,L}^{-1} / \sum_h c_{h,L}^{-1}$, so the weights average one and $w_j c_{j,L}$ is the same for every block. The retained subspace is the top-L eigenspace of $\sum_j w_j A_j$: no block steers it through its scale alone. The weights depend on the blocks and on who was measured where, never on the query, and are recomputed on every call. In the first-order sensitivity they are held fixed, like the ridges, so the reported sensitivities are conditional on them. A block whose summand has numerical rank

below L, or whose ceiling is not positive, has no defined weight and is refused with an error naming it. The weights and ceilings are returned in `balancing`.

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, where the ridge came from and the block weights – 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), `balancing` (a data frame, one row per block, with the top-L `ceiling` $c_{j,L}$, the balancing `weight` w_j and the numerical `rank` of the block's summand), `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

`mgccaBlockProfile` to reformat the per-block decomposition, `mgccaStability` 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 <- mgccaSensitivity(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)
```

`mgccaStability` *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

```
mgccaStability(
  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 mgccaSensitivity 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 mgccaSensitivity.

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 mgccaSensitivity.

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 mgccaStability().

... — 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 ratio of the smallest to the largest singular value of the reference frame restricted to those shared participants (its reciprocal condition number).

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 this ratio equal to one, and restricting it to a random share of its rows keeps the ratio near one: being a ratio, it does not shrink with the fraction retained, so a value well below one signals genuine ill-conditioning.

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 `mu`, 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

`mgccaSensitivity` for the query-based companion, and `mgccaSelectLambda` – 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
## mgccaSensitivity(), 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 <- mgccaStability(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)
```

`mgcca_block_gram` *Present-only, feature-streamed participant block Gram (reliability layer, K1)*

Description

Computes the participant Gram $G = Z^\top Z$ (`n_pr x n_pr`) of one HDF5-backed block, where Z standardizes each feature (row) to mean 0 and unit variance using the **present participants only** and the $(n_{pr} - 1)$ sample-SD denominator. Features are streamed in chunks so the full $p \times n_{pr}$ standardized matrix is never materialized (only the active chunk and the `n_pr x n_pr` Gram are retained). This is the first kernel (K1) of the mgcca reliability-layer C++/HDF5 port; it reproduces the sealed R oracle (`34_methyl_gram_dense.R`) within scale-aware tolerance.

Usage

```
mgcca_block_gram(
  file,
  group,
  dataset,
  present_ids,
  chunk = 4096L,
  var_eps = 1e-08,
  out = NULL
)
```

Arguments

`file`, `group`, `dataset`	HDF5 location of the block, stored R-view `p x N` (features x all-participants) with participant IDs as colnames.
`present_ids`	Character vector of present participant IDs in the frozen order; selects the columns that enter the Gram.
`chunk`	Integer number of features per streamed chunk (default 4096).
`var_eps`	Degenerate-feature threshold on the per-feature SD (default 1e-8, the oracle value); `sd <= var_eps` stops with the feature index.
`out`	Optional `list(file, group, dataset, compression)` to persist G to HDF5 entirely in C++ (the file is created if missing, participant dimnames carried, compression default 0).

Details

Block-absent participants must NEVER enter centring/scaling/cross products as observed zero rows (the frozen zero-padding regression); pass the true present-only participant set in `present_ids`. Degenerate (constant) features trigger a hard stop reporting their index, matching the oracle convention (no silent variance floor, drop, or rescale).

Value

A list with G (n_pr x n_pr, participant dimnames), ids, p_eff, n_pr, N_all, chunk, n_chunks, last_chunk, eigen_threads, an invariants list (e_trace, e_center, sym_err, neg_mass, min_eig_ratio, num_rank), and a per-chunk ledger.

---

mgcca_cor_ave_rcpp *MGCCA corsY / p-values / AVE stage over HDF5*

Description

MGCCA corsY / p-values / AVE stage over HDF5

Usage

```
mgcca_cor_ave_rcpp(
  filename,
  tmp_group,
  datasets,
  nfac,
  final_group = "FINAL_RESULTS"
)
```

Value

A list with the stage descriptor (filename), or NULL on error. The results (corsY, pval, AVE) are written into final_group of the HDF5 file.

---

mgcca_eigen_rcpp *MGCCA eigen stage (Ksum, MKsum05, eigen, Y) over HDF5*

Description

MGCCA eigen stage (Ksum, MKsum05, eigen, Y) over HDF5

Usage

```
mgcca_eigen_rcpp(
  filename,
  tmp_group,
  datasets,
  nfac,
  final_group = "FINAL_RESULTS",
  threads = NULL
)
```

Value

A list with the stage descriptor (filename, nfac, eig_values, Y_path), or NULL on error. The shared components Y are written into final_group of the HDF5 file.

`mgcca_file_is_open_rcpp` *Whether a handle still holds an HDF5 file open*

Description

Whether a handle still holds an HDF5 file open

Usage

```
mgcca_file_is_open_rcpp(handle)
```

Arguments

`handle`	An external pointer returned by `mgcca_hold_file_rcpp`.

Value

`TRUE` while the handle holds an open file, `FALSE` once it has been released.

`mgcca_getK_rcpp` *MGCCA getK stage (build padded X and indicator diagonal K) over HDF5*

Description

MGCCA getK stage (build padded X and indicator diagonal K) over HDF5

Usage

```
mgcca_getK_rcpp(filename, in_group, datasets, out_group = "MGCCA_TMP")
```

Value

A list with the stage descriptor (`filename`, `out_group`, `datasets`, `m`, `rn`), or `NULL` on error. The union-padded tables and their indicator diagonals are written into `out_group` of the HDF5 file.

mgcca_hold_file_rcpp *Open one HDF5 file and keep it open*

Description

Opens `filename` and returns the open handle, which stays open until it is released. The file is not read and not written.

Usage

```
mgcca_hold_file_rcpp(filename)
```

Arguments

`filename`	Path to the HDF5 file, exactly as every operation of the same analysis will name it.

Value

An external pointer to the open file.

mgcca_lambda_gram_rcpp *Participant Gram of each scaled block, accumulated over HDF5 column chunks*

Description

Reads every dataset named in `datasets` in column chunks and returns the n by n Gram of the scaled block, so neither the block nor its scaled copy is ever held dense. The scaling is the one the package's penalty rule applies: centre by `colMeans`, scale by `sd`, and drop any column whose centre or scale is non-finite or whose scale is at or below `sd_tol`.

Usage

```
mgcca_lambda_gram_rcpp(
  filename,
  group,
  datasets,
  chunk_cols,
  do_scale = TRUE,
  sd_tol = 1e-08,
  return_stats = FALSE
)
```

Arguments

filename	Path to the HDF5 file.
group	Group holding the datasets.
datasets	Names of the datasets to measure, in the order wanted.
chunk_cols	Number of variables per chunk; capped at the block width.
do_scale	Whether to centre and scale the columns before accumulating.
sd_tol	Columns with a scale at or below this are dropped.
return_stats	Whether to return the per-column centre and scale of the kept columns.

Value

A list with one entry per dataset, each a list holding the Gram G, the sizes n, p, n_kept, n_dropped, the chunking chunk_cols, n_chunks, last_chunk, and the diagnostics sym_err, sd_min_kept, sd_max_dropped (plus center and scale when return_stats is TRUE).

---

mgcca_list_group_rcpp *List the immediate children of an HDF5 group*

Description

Returns the names of the objects (datasets and subgroups) directly under group, without depending on the rhdf5 package. Used by mgccaLoad to auto-discover table names in files written before the provenance manifest existed.

Usage

```
mgcca_list_group_rcpp(filename, group)
```

Arguments

filename	Path to the HDF5 file.
group	Group whose children to list.

Value

Character vector of child object names.

mgcca_rcpp *MGCCA over HDF5 (single-call C++ orchestrator)*

Description

MGCCA over HDF5 (single-call C++ orchestrator)

Usage

```
mgcca_rcpp(
  filename,
  in_group,
  datasets,
  nfac,
  inv_method,
  lambda = NULL,
  scores = FALSE,
  scale = TRUE,
  route = "auto",
  tmp_group = "MGCCA_TMP",
  final_group = "FINAL_RESULTS",
  threads = NULL
)
```

Arguments

filename	HDF5 file with input tables already written under in_group.
in_group	Input group (default "MGCCA_IN").
datasets	Character vector of dataset names inside in_group.
nfac	Number of shared components.
inv_method	1 = solve (SPD Cholesky), 2 = penalized (+lambda I), 3 = geninv/ginv (pseudoinverse).
lambda	Numeric vector (length = #tables) for inv_method = 2.
scores	If TRUE, also compute per-table weights and scores.
scale	If TRUE (default), column-center+scale each table in-HDF5 (out-of-core, base-R scale() semantics) before the analysis, reading the raw tables from in_group and never loading them fully into RAM.
route	Per-table algebra route: "auto" (default) picks per table by min(n, p) – covariance X'X} (p x p) when \code{p < n},Gram \code{XX' (n x n) when p >= n; "cov" forces the covariance route for all tables; "dual" forces the Gram route.
tmp_group	Group for intermediates (default "MGCCA_TMP").
final_group	Group for results (default "FINAL_RESULTS").
threads	Optional thread count.

Value

A descriptor list (filename, datasets, nfac, m, eigenvalues, route, route_dual, final_group). Results are written under final_group.

See Also

mgcca, mgccaResults

`mgcca_read_attrs_rcpp` *Read all HDF5 attributes from a dataset or group*

Description

Returns every attribute (except BigDataStatMeth's internal bookkeeping attribute) as a named list. When `dataset` is the empty string the attributes are read from the group `group`; otherwise from the dataset `group/dataset`.

Usage

```
mgcca_read_attrs_rcpp(filename, group, dataset)
```

Arguments

`filename`	Path to the HDF5 file.
`group`	Group path (source group, or the parent group of the dataset).
`dataset`	Dataset name, or "" to read from the group itself.

Value

Named list of attribute values.

`mgcca_read_audit_rcpp` *Read a stored availability audit from one HDF5 file in a single open*

Description

Reads the attributes of `group` and every table of `tables` that the file carries, through one open of `filename`.

Usage

```
mgcca_read_audit_rcpp(filename, group, tables)
```

Arguments

`filename`	Path to the HDF5 file holding the audit.
`group`	Group the audit was written to.
`tables`	Names of the audit tables to read. A name the file does not carry comes back as `NULL`, so the caller reports what is missing rather than failing on the first gap.

Value

A list with `attrs` (the group's attributes) and `tables` (one matrix per table, with its dimnames), named after `tables`.

`mgcca_read_blocks_rcpp`
Read several datasets of one HDF5 file in a single open

Description

Returns the values and the participant row names of every dataset named in `datasets`, read through one open of `filename`.

Usage

```
mgcca_read_blocks_rcpp(filename, group, datasets)
```

Arguments

`filename`	Path to the HDF5 file.
`group`	Group holding the datasets.
`datasets`	Names of the datasets to read, in the order they are wanted.

Value

A list with `values` (one matrix per dataset) and `rownames` (one character vector per dataset, empty where the file carries none), both named after `datasets`.

`mgcca_read_dimensions_rcpp`
Read the dimensions of several datasets of one HDF5 file in a single open

---

Description

Reports the shape each dataset has as R sees it. A dataset the file does not carry is reported as `0` by `0` rather than raised as an error, so a caller sizing a choice over a set of datasets is not stopped by one that is absent.

Usage

```
mgcca_read_dimensions_rcpp(filename, group, datasets)
```

Arguments

`filename`	Path to the HDF5 file.
`group`	Group holding the datasets.
`datasets`	Names of the datasets to measure.

Value

A list with the numeric vectors `nrow` and `ncol`, one entry per dataset, named after `datasets`.

mgcca_read_manifest_rcpp

Read a provenance manifest from one HDF5 file in a single open

Description

Reads the attributes of `group` and the attributes of every dataset under `dataset_group`, through one open of `filename`. A `dataset_group` the file does not carry yields no per-dataset attributes rather than an error, since a file can hold a run-level manifest and no per-table one.

Usage

```
mgcca_read_manifest_rcpp(filename, group, dataset_group)
```

Arguments

`filename`	Path to the HDF5 file.
`group`	Group the run-level attributes belong to.
`dataset_group`	Group holding the datasets whose attributes are wanted, or `""` to read none.

Value

A list with `group` (the run-level attributes) and `datasets` (one named list of attribute values per dataset, named after the datasets).

mgcca_read_rownames_rcpp

Read the row names of several datasets of one HDF5 file in a single open

Description

Reads the participant identifiers of every dataset named in `datasets` without reading one value of any of them.

Usage

```
mgcca_read_rownames_rcpp(filename, group, datasets)
```

Arguments

`filename`	Path to the HDF5 file.
`group`	Group holding the datasets.
`datasets`	Names of the datasets whose row names are wanted.

Value

A list of character vectors named after `datasets`, each empty where the file carries no row names for that dataset.

mgcca_release_file_rcpp *Release a held HDF5 file handle*

Description

Closes the file the handle holds and invalidates the handle. A handle that holds nothing is left alone, so this can be called more than once.

Usage

```
mgcca_release_file_rcpp(handle)
```

Arguments

handle	An external pointer returned by mgcca_hold_file_rcpp.

Value

Invisibly NULL; called for the side effect of closing the file.

mgcca_save_audit_rcpp *Save an availability audit into one HDF5 file in a single open*

Description

Creates the audit group if it is missing, writes every table of tables as a double dataset with its dimnames, and writes attrs as attributes of the group. The file is opened once for the whole operation and closed once at the end, so no part of the save depends on reopening a file this process already holds.

Usage

```
mgcca_save_audit_rcpp(filename, group, tables, attrs)
```

Arguments

filename	Path to the HDF5 file; it is created if it does not exist, and never truncated, so the blocks the audit describes survive.
group	Group to write the audit into.
tables	Named list of double matrices with dimnames, one per audit table. An existing dataset of the same name is replaced, whatever its shape.
attrs	Named list of manifest values, written as attributes of group.

Value

Invisibly NULL; called for the side effect of writing the audit into the HDF5 file.

mgcca_scores_rcpp *MGCCA scores stage (A=B_j, weights, scores) over HDF5*

Description

MGCCA scores stage (A=B_j, weights, scores) over HDF5

Usage

```
mgcca_scores_rcpp(
  filename,
  tmp_group,
  datasets,
  nfac,
  final_group = "FINAL_RESULTS"
)
```

Value

A list with the stage descriptor (`filename`), or `NULL` on error. The per-table weights and scores are written into `final_group` of the HDF5 file.

mgcca_scores_svd_rcpp *MGCCA dual scores stage (A=V diag(w_A) U'Y, weights, scores) over HDF5*

Description

MGCCA dual scores stage (A=V diag(w_A) U'Y, weights, scores) over HDF5

Usage

```
mgcca_scores_svd_rcpp(
  filename,
  tmp_group,
  datasets,
  inv_method,
  nfac,
  lambda = NULL,
  final_group = "FINAL_RESULTS"
)
```

Value

A list with the stage descriptor (`filename`), or `NULL` on error. The per-table weights and scores are written into `final_group` of the HDF5 file.

`mgcca_solve_rank_guard_rcpp` *Refuse method = "solve" on a table whose cross-product has no inverse*

Description

Checks every table that the covariance route would take (the route `mgcca_rcpp` picks from `route`) and stops, naming the table, when its cross-product $X_j^\top K_j X_j$ has no ordinary inverse. Tables on the Gram route are not checked. Called by `mgcca` after the import and before the fit, for `method = "solve"` only.

Usage

```
mgcca_solve_rank_guard_rcpp(
  filename,
  group,
  datasets,
  scale = TRUE,
  route = "auto"
)
```

Arguments

`filename`	Path to the HDF5 file.
`group`	Group holding the imported tables.
`datasets`	Names of the tables, in fit order.
`scale`	Whether the fit centres and scales the tables.
`route`	Per-table algebra route, as given to `mgcca_rcpp`.

Value

`NULL` when every checked table has an inverse; otherwise an error naming the first table that has none.

`mgcca_svd_rcpp` *MGCCA dual SVD stage: M_j = U diag(w) U' from SVD(X_j)*

Description

MGCCA dual SVD stage: M_j = U diag(w) U' from SVD(X_j)

Usage

```
mgcca_svd_rcpp(
  filename,
  tmp_group,
  datasets,
  inv_method,
  lambda = NULL,
  svd_method = "full",
  threads = NULL
)
```

Value

A list with the stage descriptor (`filename`, `tmp_group`, `datasets`, `inv_method`, `svd_method`), or `NULL` on error. The per-table dual projectors are written into `tmp_group` of the HDF5 file.

`mgcca_write_attrs_rcpp` *Write HDF5 attributes to a dataset or group*

Description

Writes each named element of `attrs` as an HDF5 attribute. When `dataset` is the empty string the attributes are attached to the group `group`; otherwise they are attached to the dataset `group/dataset`. Supports scalar and vector character/integer/double/logical values (logical is stored as integer 0/1, since HDF5 has no native boolean).

Usage

```
mgcca_write_attrs_rcpp(filename, group, dataset, attrs)
```

Arguments

`filename`	Path to the HDF5 file.
`group`	Group path (target group, or the parent group of the dataset).
`dataset`	Dataset name, or "" to target the group itself.
`attrs`	Named list of attribute values.

Value

Invisibly `NULL`; called for the side effect of writing the attributes into the HDF5 file.

`mgcca_write_manifest_rcpp` *Write a provenance manifest into one HDF5 file in a single open*

Description

Writes `attrs` as attributes of `group` and, for each element of `dataset_attrs`, that element as attributes of the dataset named after it under `dataset_group`. The run-level facts and the per-table facts of one manifest therefore reach the file through a single open.

Usage

```
mgcca_write_manifest_rcpp(filename, group, attrs, dataset_group, dataset_attrs)
```

Arguments

`filename`	Path to the HDF5 file.
`group`	Group the run-level attributes belong to.
`attrs`	Named list of run-level attribute values.
`dataset_group`	Group holding the datasets the per-table attributes belong to, or `""` when there are none.
`dataset_attrs`	Named list, one named list of attribute values per dataset, named after the datasets.

Value

Invisibly `NULL`; called for the side effect of writing the manifest into the HDF5 file.

`mgcca_XKX_rcpp` *MGCCA XKX stage (Mgram=X'X, inverse, Mi=X xkx X') over HDF5*

Description

MGCCA XKX stage (Mgram=X'X, inverse, Mi=X xkx X') over HDF5

Usage

```
mgcca_XKX_rcpp(
  filename,
  tmp_group,
  datasets,
  inv_method,
  lambda = NULL,
  threads = NULL
)
```

Value

A list with the stage descriptor (`filename`, `tmp_group`, `datasets`, `inv_method`), or `NULL` on error. The results themselves (per-table `Mgram`, its inverse and `Mi`) are written into `tmp_group` of the HDF5 file.

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 *Plot an availability audit*

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

mgccaAudit

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 <- mgccaAudit(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

`mgccaBlockProfile`

Examples

```
## A minimal stand-in for an ungrouped mgccaSensitivity() result; see
## mgccaBlockProfile() 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(mgccaBlockProfile(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

`mgccaSensitivity`, 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 <- mgccaSensitivity(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$corsY` 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, in a colourblind-safe palette. 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

`x`	an `mgcca` object (from `mgccaResults` or `mgcca(collect = TRUE)`).
`group`	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.
`comps`	the two components to display. Default `c(1, 2)`.
`label`	logical; label points with individual IDs. Default FALSE.
`ellipse`	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 TRUE.
`level`	confidence level for the ellipses. Default 0.95.
`point_size`	point size. Default 2.2.
`title`	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")
```

---

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

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

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

---

plotScree *Scree plot of the shared-component eigenvalues*

Description

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

Usage

```
plotScree(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.

Usage

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

Arguments

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

Value

A `ggplot` object.

See Also

`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)
```

`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

`object`	an `mgcca` object.
`newdata`	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.
`...`	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 `mgccaBlockProfile`, 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

`mgccaBlockProfile`

Examples

```
## A minimal stand-in for an ungrouped mgccaSensitivity() result; see
## mgccaBlockProfile() 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(mgccaBlockProfile(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 `mgccaSelectLambda`.
`...`	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

`mgccaSelectLambda`

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

reliability_block_inputs

Per-block inputs for the sealed sensitivity kernel (phase R3b)

Description

Rr_j = R_j D V_L and Sm_j = R_j D V_R. Kept in C++ so the algebra lives in one place; R only dispatches.

Usage

```
reliability_block_inputs(Rj, D, V, L)
```

Value

A list with the two per-block input matrices `Rr` ($R_j D V_L$) and `Sm` ($R_j D V_R$).

reliability_gramc_block_hdf5

*K3b variant-C Gram-C block (HDF5-in/HDF5-out via BigDataStatMeth): H*G[O,O]*H embedded*

Description

K3b variant-C Gram-C block (HDF5-in/HDF5-out via BigDataStatMeth): H*G[O,O]*H embedded

Usage

```
reliability_gramc_block_hdf5(
  file,
  in_group,
  ds_in,
  out_group,
  ds_out,
  present,
  comp = 0L
)
```

Value

A list with the `file` and the `path` of the centred Gram-C block written into the HDF5 file.

reliability_gramc_block_mem

K3b variant-C Gram-C block (in-memory, for unit smoke of the centring math)

Description

K3b variant-C Gram-C block (in-memory, for unit smoke of the centring math)

Usage

```
reliability_gramc_block_mem(Gsid, present)
```

Value

A numeric matrix: the centred Gram-C block $HG[O,O]H$ embedded back into the full participant indexing.

reliability_gram_block

Present-only participant Gram of one block, out-of-core (reliability phase R1)

Description

Present-only participant Gram of one block, out-of-core (reliability phase R1)

Usage

```
reliability_gram_block(
  filename,
  in_group,
  dataset,
  out_group,
  block_size = 0L,
  threads = NULL,
  standardize = TRUE
)
```

Arguments

block_size	0 lets BigDataStatMeth choose; a positive value FORCES that block size, so the blocked path can be exercised on small fixtures.

Value

A list with `filename`, the `path` of the Gram written into the HDF5 file, the participant `ids`, the `block_size` actually used and whether the block was `standardized`.

`reliability_gram_hdf5` *K1: present-only, feature-streamed participant block Gram (HDF5)*

Description

K1: present-only, feature-streamed participant block Gram (HDF5)

Usage

```
reliability_gram_hdf5(
  file,
  group,
  dataset,
  present_ids,
  chunk,
  var_eps = 1e-08,
  out_file = "",
  out_group = "",
  out_dataset = "",
  out_compression = 0L
)
```

Value

A list with the participant Gram `G` and its `ids`, the effective sizes (`p_eff`, `n_pr`, `N_all`), the streaming layout (`chunk`, `n_chunks`, `last_chunk`, `eigen_threads`) and the `invariants` / `ledger` diagnostics of the accumulation.

`reliability_proj_overlap` *K3b projector overlap trPP = tr(Pa Pb)/L*

Description

K3b projector overlap trPP = tr(Pa Pb)/L

Usage

```
reliability_proj_overlap(Pa, Pb, L)
```

Value

A single number: $tr(P_a P_b)/L$, the projector overlap of the two subspaces.

`reliability_query` *Alignment functional and first-order perturbation coefficients (phase R3)*

Description

Alignment functional and first-order perturbation coefficients (phase R3)

Usage

```
reliability_query(V, mu, zstar, L)
```

Value

A list with the alignment functional `T`, the first-order perturbation coefficients `C` and the per-component weights `w`.

`reliability_read_full_test` *K3a orientation check: read an HDF5 dataset via the EXACT helper the kernel uses (mgcca::read_full) so that a deliberately asymmetric fixture can demonstrate that no transpose is applied along the way.*

Description

K3a orientation check: read an HDF5 dataset via the EXACT helper the kernel uses (mgcca::read_full) so that a deliberately asymmetric fixture can demonstrate that no transpose is applied along the way.

Usage

```
reliability_read_full_test(file, group, dataset)
```

Value

A numeric matrix: the HDF5 dataset read back in the R view through the kernel's own reader.

reliability_reference_fit

Weighted operator, eigenbasis and per-block resolvents (reliability phase R2)

Description

Weighted operator, eigenbasis and per-block resolvents (reliability phase R2)

Usage

```
reliability_reference_fit(Glist, present, lambda, L, balance = FALSE)
```

Arguments

balance	If TRUE, weight each block's summand by its top-L balancing weight before the sum (see mgccaSensitivity); if FALSE, every weight is one.

Value

A list with the operator S, its eigenbasis V and eigenvalues mu, the availability weights D, the per-block resolvents Rlist, the spectral gap at the retained rank and that rank L; and, per block, the balancing weights w (all one without balancing), the top-L ceilings cL and the numerical ranks rank_num of the summands (NA and -1 without balancing).

reliability_sensitivity_gram

K2: methylation sensitivity accumulators from the sealed K1 Gram (participant-space)

Description

K2: methylation sensitivity accumulators from the sealed K1 Gram (participant-space)

Usage

```
reliability_sensitivity_gram(
  file,
  group,
  dataset,
  mids_to_fit,
  Rr,
  Sm,
  C,
  alpha,
  present,
  prm,
  grp,
  neg_tol = 1e-08
)
```

Value

A list with the sensitivity accumulators `trtH` and `accX`, the symmetry residual `sym_err` and the sizes `n_pr`, `n_meth`, `n_grp`.

`reliability_sensitivity_gram_matrix`

K2: the same accumulators with the Gram supplied as a matrix

Description

The sensitivity kernel reads one block's Gram once per query, so a run of Q queries over J blocks read the same Q x J datasets out of the same file, opening and closing it every time. The Grams are participant by participant – small whatever the width of the block behind them – so the caller reads them once and passes them here. The arithmetic is the same body as `reliability_sensitivity_gram`: only where the Gram came from differs.

Usage

```
reliability_sensitivity_gram_matrix(
  G,
  mids_to_fit,
  Rr,
  Sm,
  C,
  alpha,
  present,
  prm,
  grp,
  neg_tol = 1e-08
)
```

Arguments

`G`	The present-only participant Gram of the block.

Value

A list with the sensitivity accumulators `trtH` and `accX`, the symmetry residual `sym_err` and the sizes `n_pr`, `n_meth`, `n_grp`.

`reliability_subspace_fit`

K3a: top-L balanced subspace refit from block Grams (participant-space, dense Eigen core)

Description

K3a: top-L balanced subspace refit from block Grams (participant-space, dense Eigen core)

Usage

```
reliability_subspace_fit(
  file,
  group,
  gram_ds,
  masks,
  L,
  gamma,
  eps = 1e-08,
  rank_tol = 1e-08,
  sym_tol = 1e-08,
  pad_tol = 1e-08,
  return_B = TRUE
)
```

Value

A list with the refitted basis `V` and projector `P`, the retained rank `L`, the balancing weights `lambda`, the spectrum `mu` and its `gap`, the `eligible` participants and the numerical validity flags (`solve_ok`, `symSbal`, `P_idem`, `eig_min`, `n_clipped`, ...).

---

reliability_subspace_overlap

K3b subspace overlap ovl_rows (frozen rank_tol): mean cos^2 of principal angles; NA if rank-deficient

---

Description

K3b subspace overlap ovl_rows (frozen rank_tol): mean cos^2 of principal angles; NA if rank-deficient

Usage

```
reliability_subspace_overlap(Va, Vb)
```

Value

A single number: the mean squared cosine of the principal angles between the two subspaces, or `NA` if either basis is rank-deficient.

`reliability_tcrossprod_hdf5` *K3b variant-A small-block Gram via the BigDataStatMeth C++ tcrossprod API: G = Z Z' (participant Gram). Z (in_group/in_ds) = the subset-standardised block table (participants x features, R view). G is written to out_group/out_ds (n x n), isSymmetric – the same pattern the estimator's own run_svd / run_XKX phases use (mgcca_phases.h). BigDataStatMeth::tcrossprod is adaptive rather than unconditionally out-of-core: it preloads the table in memory when that takes at most about 20 otherwise. The small-block Gram path is reached through the C++ API alone; no R-side BigDataStatMeth algebra dispatch takes part in it. Returns file, path, nrow, ncol, block_size, threads_requested (n = participants; for binding + the parity report).*

Description

K3b variant-A small-block Gram via the BigDataStatMeth C++ tcrossprod API: G = Z Z' (participant Gram). Z (in_group/in_ds) = the subset-standardised block table (participants x features, R view). G is written to out_group/out_ds (n x n), isSymmetric – the same pattern the estimator's own run_svd / run_XKX phases use (mgcca_phases.h). BigDataStatMeth::tcrossprod is adaptive rather than unconditionally out-of-core: it preloads the table in memory when that takes at most about 20 otherwise. The small-block Gram path is reached through the C++ API alone; no R-side BigDataStatMeth algebra dispatch takes part in it. Returns file, path, nrow, ncol, block_size, threads_requested (n = participants; for binding + the parity report).

Usage

```
reliability_tcrossprod_hdf5(
  file,
  in_group,
  in_ds,
  out_group,
  out_ds,
  threads = NULL
)
```

Value

A list with `file`, the `path` of the Gram written into the HDF5 file, its `nrow` and `ncol`, the `block_size` and `threads_requested` (for binding and the parity report).

`summary.mgcca` *Summarise an mgcca object*

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, 4
∗ **internal**
mgcca-package, 4
mgcca_block_gram, 41
mgcca_cor_ave_rcpp, 42
mgcca_eigen_rcpp, 42
mgcca_file_is_open_rcpp, 43
mgcca_getK_rcpp, 43
mgcca_hold_file_rcpp, 44
mgcca_lambda_gram_rcpp, 44
mgcca_list_group_rcpp, 45
mgcca_rcpp, 46
mgcca_read_attrs_rcpp, 47
mgcca_read_audit_rcpp, 47
mgcca_read_blocks_rcpp, 48
mgcca_read_dimensions_rcpp, 48
mgcca_read_manifest_rcpp, 49
mgcca_read_rownames_rcpp, 49
mgcca_release_file_rcpp, 50
mgcca_save_audit_rcpp, 50
mgcca_scores_rcpp, 51
mgcca_scores_svd_rcpp, 51
mgcca_solve_rank_guard_rcpp, 52
mgcca_svd_rcpp, 52
mgcca_write_attrs_rcpp, 53
mgcca_write_manifest_rcpp, 54
mgcca_XKX_rcpp, 54
reliability_block_inputs, 73
reliability_gram_block, 74
reliability_gram_hdf5, 75
reliability_gramc_block_hdf5, 73
reliability_gramc_block_mem, 74
reliability_proj_overlap, 75
reliability_query, 76
reliability_read_full_test, 76
reliability_reference_fit, 77
reliability_sensitivity_gram, 77
reliability_sensitivity_gram_matrix, 78
reliability_subspace_fit, 78
reliability_subspace_overlap, 79
reliability_tcrossprod_hdf5, 80

bdCreate_hdf5_matrix, *22*

cardiovascular, 4
corTestP, 6

genCoords, 6
getSignif, 8, *13*, *28*, *64*, *67*, *82*
getTables, 9, *10*, *22*, *69*

impute.knn, *22*, *23*

matrixChr2Num, *9*, 10
mgcca, *4*, *5*, *8*, *9*, 11, *14–16*, *18*, *22*, *25–28*, *30*, *31*, *33*, *34*, *37*, *47*, *52*, *68*, *70*
mgcca-package, 4
mgcca_audit *(*mgccaAudit*)*, 15
mgcca_block_gram, 41
mgcca_cor_ave_rcpp, 42
mgcca_eigen_rcpp, 42
mgcca_file_is_open_rcpp, 43
mgcca_getK_rcpp, 43
mgcca_hold_file_rcpp, 44
mgcca_lambda_gram_rcpp, 44
mgcca_list_group_rcpp, 45
mgcca_load *(*mgccaLoad*)*, 25
mgcca_rcpp, *11*, *13*, *28*, 46, *52*
mgcca_read_attrs_rcpp, 47
mgcca_read_audit_rcpp, 47
mgcca_read_blocks_rcpp, 48
mgcca_read_dimensions_rcpp, 48
mgcca_read_manifest_rcpp, 49
mgcca_read_rownames_rcpp, 49
mgcca_release_file_rcpp, 50
mgcca_save_audit_rcpp, 50
mgcca_scores_rcpp, 51
mgcca_scores_svd_rcpp, 51
mgcca_select_lambda *(*mgccaSelectLambda*)*, 30
mgcca_solve_rank_guard_rcpp, 52
mgcca_svd_rcpp, 52
mgcca_write_attrs_rcpp, 53
mgcca_write_manifest_rcpp, 54
mgcca_XKX_rcpp, 54
mgccaAssociate, 14, *27*

mgccaAudit, 15, *19*, *57*
mgccaAuditLoad, *16*, *18*, 19
mgccaBlockProfile, 20, *37*, *58*, *71*
mgccaImportHDF5, *13*, *16*, 21, *24, 25*
mgccaImpute, 22
mgccaLambdaScaleHDF5, *12*, 24
mgccaLoad, *13*, 25, *34*
mgccaPermtest, *14*, 27
mgccaResults, *6*, *12–14*, *26*, 28, *47*, *63*
mgccaSelectLambda, *12*, *24, 25*, 30, *40*, *72*
mgccaSensitivity, *13*, *20*, 34, *38–40*, *60*, *77*
mgccaStability, *13*, *37*, 37

plot.mgcca, 55
plot.mgcca_audit, 55
plot.mgcca_block_profile, 58
plot.mgcca_sensitivity, 59
plot.mgcca_stability *(*mgccaStability*)*, 37
plotAVE, 61, *63*, *66*
plotBiplot, 62, *66*
plotIndividuals, *11*, *13, 14*, *28*, *30*, *55*, *62*, 63, *65–67*
plotLoadings, 64, *82*
plotScores, 65, *68*
plotScree, *61*, 66
plotVariables, *8*, *11*, *13*, *28*, *30*, *62–64*, 67
predict.mgcca, 68
print.ListMAE, 69
print.mgcca, 70
print.mgcca_audit *(*mgccaAudit*)*, 15
print.mgcca_block_profile, 71
print.mgcca_lambda, 72
print.mgcca_sensitivity *(*mgccaSensitivity*)*, 34
print.mgcca_stability *(*mgccaStability*)*, 37

reliability_block_inputs, 73
reliability_gram_block, 74
reliability_gram_hdf5, 75
reliability_gramc_block_hdf5, 73
reliability_gramc_block_mem, 74
reliability_proj_overlap, 75
reliability_query, 76
reliability_read_full_test, 76
reliability_reference_fit, 77
reliability_sensitivity_gram, 77
reliability_sensitivity_gram_matrix, 78
reliability_subspace_fit, 78
reliability_subspace_overlap, 79
reliability_tcrossprod_hdf5, 80

summary, *30*
summary.mgcca, 80
summary.mgcca_audit *(*mgccaAudit*)*, 15
summary.mgcca_sensitivity *(*mgccaSensitivity*)*, 34
summary.mgcca_stability *(*mgccaStability*)*, 37

topVars, *8*, *28*, 81

X1 *(*cardiovascular*)*, 4
X2 *(*cardiovascular*)*, 4
X3 *(*cardiovascular*)*, 4