Package 'genpca'

September 17, 2026

Type Package

Title Generalized Principal Component Analysis

Version 0.2.1

Description Generalized PCA and related matrix decompositions in weighted inner-product spaces. Methods are based on Allen, G. I., Grosenick, L., and Taylor, J. (2014) <doi:10.1080/01621459.2013.852978>, ``A generalized least-square matrix decomposition'', Journal of the American Statistical Association, 109(505), 145-159; and Abdi, H. (2007), ``Singular value decomposition (SVD) and generalized singular value decomposition'' <https://personal.utdallas.edu/~herve/Abdi-SVD2007-pretty.pdf>, in ``Encyclopedia of Measurement and Statistics'', 907-912.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Depends R (>= 4.1.0)

Imports Rcpp, eigencore (>= 1.0.3), FNN, Matrix, multivarious (>= 0.3.0), assertthat, methods, digest

LinkingTo Rcpp, RcppArmadillo, RcppEigen

Suggests testthat (>= 3.0.0), adjoin, irlba, matrixStats, clue, knitr, rmarkdown, ggplot2, albersdown, ragg, systemfonts

Config/testthat/edition 3

VignetteBuilder knitr

RoxygenNote 7.3.3

URL https://bbuchsbaum.github.io/genpca/, https://github.com/bbuchsbaum/genpca

BugReports https://github.com/bbuchsbaum/genpca/issues

Config/Needs/website albersdown

NeedsCompilation yes

Author Brad Buchsbaum [aut, cre, cph]

Maintainer Brad Buchsbaum <brad.buchsbaum@gmail.com>

Repository CRAN

Date/Publication 2026-09-16 23:40:02 UTC

Contents

geigen_cov 2
genpca 4
genpca_cov 9
genpls 11
genplsc 14
gmd_clear_cache 15
gpca_mle 16
gplssvd_op 18
mnpca_mrl 20
print.sfpca 23
reconstruct.genpca 24
reconstruct.sfpca 25
repair_metric 26
rpls 27
sfpca 29
transfer.cross_projector 32
truncate 33
truncate.genpca 34

Index **35**

`geigen_cov` *Generalized eigenproblem on a covariance matrix*

Description

Maximises $v'Cv$ subject to $v'Rv = 1$ and the additional constraint that v lies in the retained range of `R`. Successive components are R-orthogonal. If P is the orthogonal projector onto that range, the returned vectors satisfy $PCv = \lambda Rv$ and $V'RV = I$. For a full-rank `R` this is the usual equation $Cv = \lambda Rv$. It also holds for a singular `R` when `C` maps its retained range into itself. Otherwise the component of Cv outside the retained range need not vanish.

Usage

```
geigen_cov(
  C,
  R = NULL,
  ncomp = NULL,
  constraints_remedy = c("error", "ridge", "clip", "identity"),
  rank_rtol = 1e-06,
```

```
  metric_rtol = .metric_rtol_default(),
  verbose = FALSE
)
```

Arguments

C	A p x p symmetric matrix. Asymmetry beyond roundoff is an error; an indefinite C is allowed (the problem is still defined) and only produces a warning.
R	Variable-side constraint/metric. Can be: • NULL: identity matrix (standard PCA on C) • a numeric vector of length p: diagonal weights (must be non-negative) • a p x p symmetric PSD matrix: general metric/smoothing/structure penalties
ncomp	Number of components to return. Default is all positive eigenvalues.
constraints_remedy	What to do with an indefinite R: "error" (default), "ridge", "clip" or "identity"; a repair emits a genpca_metric_repaired warning. See genpca.
rank_rtol	Relative cutoff for component acceptance on the singular-value scale (components with d_j <= rank_rtol * d_1 are dropped). Default 1e-6.
metric_rtol	Relative tolerance for validating C and R and for detecting the numerical null space in an eigendecomposition of a general R. Every strictly positive diagonal weight is retained without a rank approximation. Default sqrt(.Machine$double.eps).
verbose	Logical. If TRUE, print progress messages. Default FALSE.

Details

This is a different estimator from the GMD of genpca_cov, which uses $R^{1/2}CR^{1/2}$. If C and R commute, their common eigenvectors can be ordered differently: the GMD weights variances by metric eigenvalues, whereas this estimator divides by them. With R = c * I, the directions and their ordering agree, but the eigenvalue scales differ unless c = 1.

C is validated for symmetry but may be indefinite (the generalized eigenproblem is still defined); a warning is issued when its minimum eigenvalue is below -metric_rtol * scale. R must be positive semi-definite; an indefinite R is subject to constraints_remedy.

Value

A plain list with the same components as genpca_cov (v, d, lambda, k, propv, cumv, R_rank) and method = "geigen". propv is relative to $\mathrm{tr}(R^{-1/2}CR^{-1/2})$ on the range of R.

See Also

genpca_cov

Examples

```
C <- cov(scale(iris[,1:4], center=TRUE, scale=FALSE))
w <- c(1, 1, 0.5, 2)
fit_gmd <- genpca_cov(C, R = w, ncomp = 2)
fit_geigen <- geigen_cov(C, R = w, ncomp = 2)
# different estimators: the singular values generally differ
rbind(gmd = fit_gmd$d, geigen = fit_geigen$d)

# With singular R, the equation is projected onto its retained range
C <- matrix(c(2, 1, 1, 2), 2)
R <- diag(c(1, 0))
fit <- geigen_cov(C, R, ncomp = 1)
P <- diag(c(1, 0))
P %*% C %*% fit$v - (R %*% fit$v) * fit$lambda
```

---

genpca *Generalised Principal Components Analysis (GPCA)*

Description

Implements the Generalised Least-Squares Matrix Decomposition of Allen, Grosenick & Taylor (2014) for data observed in a **row** inner-product space M and a **column** inner-product space A. Setting M = I_n, A = I_p recovers ordinary PCA.

Usage

```
genpca(
  X,
  A = NULL,
  M = NULL,
  ncomp = NULL,
  method = c("eigen", "auto", "spectra", "randomized", "deflation"),
  constraints_remedy = c("error", "ridge", "clip", "identity"),
  preproc = multivarious::pass(),
  threshold = 1e-06,
  maxit_deflation = 500L,
  use_cpp = TRUE,
  maxeig = 5000,
  warn_approx = TRUE,
  maxit_spectra = 1000,
  tol_spectra = 1e-09,
  rank_rtol = 1e-06,
  oversample = 20L,
  n_power = 1L,
  n_polish = 0L,
  jitter_metric = 1e-10,
  seed_randomized = 1234L,
```

```
  tol_polish_randomized = 1e-04,
  verbose = FALSE
)
```

Arguments

`X`	Numeric matrix n x p.
`A`	Column constraint: vector (implies diagonal), dense matrix, or sparse symmetric p x p PSD matrix. If `NULL`, defaults to identity.
`M`	Row constraint: vector (implies diagonal), dense matrix, or sparse symmetric n x n PSD matrix. If `NULL`, defaults to identity.
`ncomp`	Number of components to extract. Defaults to `min(dim(X))`. Must be positive.
`method`	Character string specifying the computation method. One of `"eigen"` (default, uses `gmdLA`), `"auto"` (heuristic choice among `"eigen"`, `"spectra"`, and `"randomized"`), `"spectra"` (iterative partial SVD of the metric-whitened data via **eigencore**, `gmd_spectra`), `"randomized"` (approximate randomized block solver `gmd_randomized`), or `"deflation"` (uses `gmd_deflationR` or `gmd_deflation_cpp`).
`constraints_remedy`	Character string specifying what to do with a supplied `A` or `M` that is not positive semi-definite (within a relative tolerance of `sqrt(.Machine$double.eps)`). Default `"error"`: reject the input. The alternatives repair it and emit a warning of class `genpca_metric_repaired` whose `report` field (see `repair_metric`) records the minimum eigenvalue before and after, the shift applied, the rank and the condition number: `"ridge"` (Gershgorin diagonal shift: add the smallest diagonal loading that restores positive definiteness, falling back to `Matrix::nearPD()` for small dense matrices), `"clip"` (spectral clip to the PSD cone by zeroing negative eigenvalues; this densifies the matrix and refuses sparse input larger than 2000 rows/cols, where `"ridge"` should be used instead), or `"identity"` (replace the matrix with the identity). An asymmetric metric is an error under every setting. Singular PSD metrics are valid input and are never repaired.
`preproc`	Pre-processing transformer object from the **multivarious** package (default `multivarious::pass()`). Use `multivarious::center()` for centered GPCA. See `?multivarious::prep` for options.
`threshold`	Convergence tolerance for the `"deflation"` method's inner loop. Default `1e-6`. Cutoffs are relative to the scale of the problem (the norm/singular-value floors scale with $\sqrt{\mathrm{tr}(X'MXA)}$), so results are invariant to rescaling `X`. The convergence check is on a squared step difference, so the resulting singular-vector accuracy scales like $\sqrt{\texttt{threshold}}$, not `threshold` itself.
`maxit_deflation`	Maximum iterations per component for the `"deflation"` method. Default `500`.
`use_cpp`	Logical. If `TRUE` (default) and package was compiled with C++ support, use faster C++ implementation for `method = "deflation"`. Fallback to R otherwise. (Ignored for `method = "eigen"` and `method = "spectra"`).
`maxeig`	For `method = "eigen"` and `method = "spectra"`: a positive definite general metric is factored exactly by Cholesky at any size, but a singular general metric (e.g. a graph Laplacian) needs a dense eigendecomposition of the metric,

which is refused when the metric has more than `maxeig` rows. The error names the alternatives (`method = "deflation"`, which only multiplies by the metric, or raising `maxeig`); `method = "auto"` routes such cases to deflation. Results are never approximated. Default `5000`.

`warn_approx` Deprecated and ignored: `method = "eigen"` no longer approximates anything.

`maxit_spectra` Retained for compatibility and currently unused: the **eigencore** partial SVD used by `method = "spectra"` is controlled by `tol_spectra` alone.

`tol_spectra` Convergence tolerance of the iterative solver when `method = "spectra"`. Default `1e-9`. This governs iteration only; rank decisions use `rank_rtol`.

`rank_rtol` Relative cutoff for component acceptance, on the scale of the singular values: component j is dropped when `d_j <= rank_rtol * d_1`. Applied by every method (for the eigen paths on d_j^2), so the number of components returned does not change when `X` is rescaled. Default `1e-6`. Metric validation uses a separate relative tolerance, `sqrt(.Machine$double.eps)`, for positive semi-definiteness and null-space detection for general metric eigendecompositions. Every strictly positive diagonal weight is retained in both the forward and inverse factors.

`oversample` Oversampling for `method = "randomized"` (sketch size = `ncomp + oversample`). Default `20`.

`n_power` Number of power iterations for `method = "randomized"`. Default `1`.

`n_polish` Number of optional block-polish iterations for `method = "randomized"`. Default `0`.

`jitter_metric` Relative Gram jitter for the candidate Cholesky preconditioner in `method = "randomized"`. The basis is checked in the original metric; a failed check uses rank-revealing orthonormalization instead. Default `1e-10`.

`seed_randomized`
Optional seed for `method = "randomized"`. Default `1234`. This fully determines the randomized backend's random stream: the C++ kernel seeds its own generator from this value rather than from R's `set.seed()`/`.Random.seed`, and calling `genpca()` with `method = "randomized"` does not alter the caller's `.Random.seed`. To reproduce a randomized fit, fix `seed_randomized`, not the R seed.

`tol_polish_randomized`
Relative tolerance used for early stopping of polish iterations in `method = "randomized"`. Set `0` to disable early stop. Default `1e-4`.

`verbose` Logical. If `TRUE`, print progress messages. Default `FALSE`.

Value

An object of class `c("genpca", "bi_projector")` inheriting from `multivarious::bi_projector`, with slots including:

u,v Left/right singular vectors scaled by the constraint metrics (MU, AV). These correspond to components in the original space's geometry. Use `components(fit)`.

ou,ov Orthonormal singular vectors in the constraint metric (U, V such that UT M U = I, VT AV = I). These are the core mathematical factors.

sdev Generalised singular values d_k. Note these are singular values of the metric-whitened data matrix, not standard deviations: with identity metrics and centering, `sdev = prcomp(X)$sdev * sqrt(nrow(X) - 1)`.

s Scores: the generalised principal components `z_k = X A ov_k = ou_k d_k` (Allen et al. 2014, Section 2.4). Identical to `project(fit, X)` on the training data. Use `scores(fit)`.

preproc The `multivarious` pre-processing object used.

A, M The constraint matrices used (potentially after coercion to sparse format).

propv Proportion of generalized variance explained by each component.

cumv Cumulative proportion of generalized variance explained.

Method

We compute the rank-ncomp factors UDVT that minimise

$$\|X - UDV^\top\|^2_{M,A} = \mathrm{tr}\big(M\,(X - UDV^\top)\,A\,(X - UDV^\top)^\top\big)$$

subject to UT M U = I, VT AV = I. (Allen et al., 2014). Five methods are available via the `method` argument:

- `"eigen"` (Default): Uses a one-shot eigen decomposition strategy based on `gmdLA`. It explicitly forms and decomposes a $p \times p$ or $n \times n$ matrix (depending on `n` vs `p`).
- `"auto"`: Chooses among `"eigen"`, `"spectra"`, and `"randomized"` using heuristics on shape, rank ratio (`ncomp / min(n,p)`), and constraint structure.
- `"spectra"`: Computes the top-k singular triplets of the metric-whitened data $F_M' X F_A$ (with $M = F_M F_M'$, $A = F_A F_A'$) as an implicit operator via the **eigencore** package, without forming the large intermediate matrix. Generally faster and uses less memory for large `n` or `p` when few components are requested.
- `"randomized"`: Uses a randomized block range finder and small projected eigendecomposition. This is an approximate low-pass method that is often much faster for wide dense matrices with sparse metrics when only top components are needed.
- `"deflation"`: Uses an iterative power/deflation algorithm. Can be slower but potentially uses less memory than `"eigen"` for very large dense problems where `ncomp` is small.

Backend Guidance

The default is `method = "eigen"`; `"auto"` is opt-in, not the default.

- Use `"eigen"` (the default) when you need a stable reference solution on small/medium problems.
- Use `"auto"` to let a heuristic pick among `"eigen"`, `"spectra"`, and `"randomized"` based on problem shape and constraint structure.
- Use `"spectra"` for larger matrix-free iterative solves where memory pressure is a concern.
- Use `"randomized"` for wide low-rank settings (`p >> n`) with sparse metrics when throughput matters most.
- Use `"deflation"` when you only need a few components and can tolerate iterative convergence behavior.

For pre-computed covariance matrices C = X'MX, see `genpca_cov` which performs GPCA directly on C with column constraint R (equivalent to A).

References

Allen, G. I., Grosenick, L., & Taylor, J. (2014). *A Generalized Least-Squares Matrix Decomposition.* Journal of the American Statistical Association, 109(505), 145-159. arXiv:1102.3074.

See Also

genpca_cov for GPCA on pre-computed covariance matrices, truncate.genpca, reconstruct.genpca, multivarious::bi_projector, multivarious::project, multivarious::scores, multivarious::components, multivarious::reconstruct.

Examples

```
if (requireNamespace("multivarious", quietly = TRUE)) {
  set.seed(123)
  X <- matrix(stats::rnorm(200 * 100), 200, 100)
  rownames(X) <- paste0("R", 1:200)
  colnames(X) <- paste0("C", 1:100)

  # Standard PCA (A=I, M=I, centered) - using default method="eigen"
 gpca_std_eigen <- genpca(X, ncomp = 5, preproc = multivarious::center(), verbose = FALSE)

  # Standard PCA using Spectra method (requires C++ build)
  # gpca_std_spectra <- try(genpca(X, ncomp = 5,
  #                                preproc = multivarious::center(),
  #                                method = "spectra", verbose = TRUE))
  # if (!inherits(gpca_std_spectra, "try-error")) {
  #    print(head(gpca_std_spectra$sdev))
  # }

  # Compare singular values with prcomp
  pr_std <- stats::prcomp(X, center = TRUE, scale. = FALSE)
  print("Eigen Method Sdev:")
  print(head(gpca_std_eigen$sdev))
  print("prcomp Sdev:")
  print(head(pr_std$sdev))
  print(paste("Total Var Explained (Eigen):",
              round(sum(gpca_std_eigen$propv) * 100), "%"))

  # Weighted column PCA (diagonal A, no centering)
  col_weights <- stats::runif(100, 0.5, 1.5)
  gpca_weighted <- genpca(X, A = col_weights, ncomp = 3,
                          preproc = multivarious::pass(), verbose = FALSE)
  print("Weighted GPCA Sdev:")
  print(gpca_weighted$sdev)
  print(head(components(gpca_weighted)))
}
```

genpca_cov *Generalized PCA on a covariance matrix (GMD form)*

Description

Performs Generalized PCA directly on a pre-computed covariance matrix `C = X'MX` with a single variable-side metric `R`, following Allen et al.'s GMD: the eigendecomposition of $R^{1/2}CR^{1/2}$ mapped back with $V = R^{-1/2}Z$, so that $V'RV = I$. With `C = X'MX` and `R = A` this matches `genpca(X, M = M, A = A)` exactly. This is useful when you already have `C` or when `X` is too large to store but `C` is manageable.

Usage

```
genpca_cov(
  C,
  R = NULL,
  ncomp = NULL,
  method = c("gmd", "geigen"),
  constraints_remedy = c("error", "ridge", "clip", "identity"),
  rank_rtol = 1e-06,
  metric_rtol = .metric_rtol_default(),
  tol = NULL,
  verbose = FALSE
)
```

Arguments

`C`	A p x p symmetric positive semi-definite covariance matrix, typically `C = X'MX`. Asymmetry beyond roundoff and indefiniteness beyond `metric_rtol` are errors.
`R`	Variable-side constraint/metric. Can be: • NULL: identity matrix (standard PCA on C) • a numeric vector of length p: diagonal weights (must be non-negative) • a p x p symmetric PSD matrix: general metric/smoothing/structure penalties
`ncomp`	Number of components to return. Default is all positive eigenvalues.
`method`	Deprecated. `"gmd"` (default) is this function; `"geigen"` forwards to `geigen_cov` with a warning.
`constraints_remedy`	Deprecated here (GMD requires PSD input and stops otherwise); forwarded to `geigen_cov` when `method = "geigen"`.
`rank_rtol`	Relative cutoff for component acceptance on the singular-value scale (components with `d_j <= rank_rtol * d_1` are dropped). Default 1e-6.

`metric_rtol`	Relative tolerance for validating `C` and `R` and for detecting the numerical null space in an eigendecomposition of a general `R`. Every strictly positive diagonal weight is retained without a rank approximation. Default `sqrt(.Machine$double.eps)`.
`tol`	Deprecated; use `rank_rtol` and `metric_rtol`.
`verbose`	Logical. If TRUE, print progress messages. Default FALSE.

Details

The generalized eigenproblem $Cv = \lambda Rv$ is a different estimator (it maximises $v'Cv$ subject to $v'Rv = 1$, which is generally gives different components from the GMD) and lives in its own function, `geigen_cov`. `method = "geigen"` is accepted here for one release and forwards to it with a deprecation warning.

Value

A plain list (**not** a **multivarious** `bi_projector`) with components:

v p x k matrix of loadings (R-orthonormal eigenvectors)

d Singular values (square root of eigenvalues lambda)

lambda Eigenvalues (variances under the R-metric)

k Number of components returned

propv Proportion of variance explained by each component (total variance is $\mathrm{tr}(CR)$, Allen et al. Corollary 5)

cumv Cumulative proportion of variance explained

R_rank Rank of the constraint matrix R

method `"gmd"`

Because this is a plain list rather than a `bi_projector`, the `multivarious` generics `scores()`, `components()`, and `reconstruct()` do not apply to it; index `$v`/`$d` directly, or use `genpca` when you need the full projector interface on a data matrix rather than a pre-computed covariance matrix.

References

Allen, G. I., Grosenick, L., & Taylor, J. (2014). A Generalized Least-Squares Matrix Decomposition. Journal of the American Statistical Association, 109(505), 145-159.

See Also

`geigen_cov` for the generalized eigenproblem $Cv = \lambda Rv$, `genpca` for the two-sided GPCA on data matrices, `genpls` for generalized partial least squares

Examples

```
# Standard PCA on a covariance (no constraint)
C <- cov(scale(iris[,1:4], center=TRUE, scale=FALSE))
fit0 <- genpca_cov(C, R=NULL, ncomp=3)
print(fit0$d[1:3])       # first 3 singular values
print(fit0$propv[1:3])   # variance explained by first 3 components
```

```
# Equivalence with genpca()
set.seed(123)
X <- matrix(rnorm(50 * 10), 50, 10)
M_diag <- runif(50, 0.5, 1.5)  # row weights
A_diag <- runif(10, 0.5, 2)    # column weights
fit_gpca <- genpca(X, M = M_diag, A = A_diag, ncomp = 5,
                   preproc = multivarious::pass())
C <- crossprod(X, diag(M_diag) %*% X)  # C = X'MX
fit_cov <- genpca_cov(C, R = A_diag, ncomp = 5)
all.equal(fit_gpca$sdev, fit_cov$d, tolerance = 1e-10)

# Variable weights via a diagonal metric (iris covariance, 4 variables)
C_iris <- cov(scale(iris[,1:4], center=TRUE, scale=FALSE))
w <- c(1, 1, 0.5, 2)
fitW <- genpca_cov(C_iris, R = w, ncomp=3)
print(fitW$d[1:3])
```

---

genpls *Generalized PLS via Implicit Operator (PLS-SVD / GPLSSVD)*

Description

Canonical (two-block) generalized PLS using sparse-friendly implicit matrix-vector products. Solves the SVD of the operator $S = Xe'Ye$ without materializing $Xe = Mx^{1/2}XAx^{1/2}$ or $Ye = My^{1/2}YAy^{1/2}$.

Usage

```
genpls(
  X,
  Y,
  Ax = NULL,
  Ay = NULL,
  Mx = NULL,
  My = NULL,
  ncomp = 2,
  preproc_x = multivarious::pass(),
  preproc_y = multivarious::pass(),
  svd_backend = c("eigencore", "irlba", "RSpectra"),
  svd_opts = list(tol = 1e-07, maxitr = 1000),
  constraints_remedy = c("error", "ridge", "clip", "identity"),
  verbose = FALSE
)
```

Arguments

X	Numeric or Matrix, n x p.
Y	Numeric or Matrix, n x q. Must have same n as X.
Ax	Column metric for X (W_X): vector/diagonal/matrix; NULL means identity.
Ay	Column metric for Y (W_Y): vector/diagonal/matrix; NULL means identity.
Mx	Row metric for X (M_X): vector/diagonal/matrix; NULL means identity.
My	Row metric for Y (M_Y): vector/diagonal/matrix; NULL means identity.
ncomp	Number of components to extract (rank-k). Default 2.
preproc_x, preproc_y	Optional multivarious preprocessors (e.g., center()). Defaults to multivarious::pass() (no-op).
svd_backend	Character, one of "eigencore" (default) or "irlba" for the iterative SVD. This choice only matters for larger problems: whenever both X and Y have at most 64 columns after preprocessing, the operator materializes S densely and computes a direct svd(), ignoring svd_backend entirely (see gplssvd_op()).
svd_opts	List of options: tol for both backends and maxitr for irlba only. An incomplete eigencore solve raises an error of class genpca_solver_nonconvergence; no unchecked fit is returned.
constraints_remedy	What to do with a metric that is not positive semi-definite: "error" (default), "ridge", "clip" or "identity"; repairs emit a genpca_metric_repaired warning. See genpca().
verbose	Logical; print brief progress messages.

Details

This follows the GPLSSVD/PLS-SVD formulation (Beaton, eqs. 10–14): the top ncomp singular triplets of S are computed by iterative SVD on the linear maps v -> S v and u -> S^T u, implemented with metric Cholesky multiplies/solves when possible. Works with dense or sparse Matrix inputs and constraint metrics.

Returns a multivarious::cross_projector with X-/Y-weights (vx, vy) chosen to provide natural projection of new data (X %*% vx, Y %*% vy). Additional GPLSSVD quantities are attached to the object for access: singular values d, generalized weights p, q, variable scores fi, fj, and row latent variables lx, ly.

genpls() maximizes the covariance between latent variables of X and Y under the (Mx, Ax, My, Ay) metrics by computing the SVD of $S = Xe'Ye$, where $Xe = Mx^{1/2}XAx^{1/2}$ and $Ye = My^{1/2}YAy^{1/2}$.

project() on a fitted object returns latent variables in the ambient (original data) metric, i.e. X \%*\% vx = X W_X p. This differs from the attached lx = Mx^{1/2} X W_X p, which lives in the row-whitened metric, by the factor $Mx^{1/2}$: lx and project(fit, X) are equal only when Mx = I. New-row projection necessarily uses project()'s ambient-metric convention, because a training-row metric Mx has no natural extension to out-of-sample rows.

Metric naming. genpls()'s row/column metric arguments (Mx, My for rows; Ax, Ay for columns) follow the same M/A convention as genpca(). Internally they are forwarded to gplssvd_op(), which uses the names XLW/YLW (left/row weights, i.e. Mx/My) and XRW/YRW (right/column weights, i.e. Ax/Ay).

Value

An object of class `c("genpls", "cross_projector", "projector")` with:

vx, vy X- and Y- projection weights (stored in cross_projector) such that `project(fit, X) = X \%*\% vx` and `project(fit, Y, source = "Y") = Y \%*\% vy` recover the latent variables in the ambient (non-whitened) metric. Algebraically `vx = W_X p = fi \%*\% diag(1/d)` and `vy = W_Y q = fj \%*\% diag(1/d)` (see Details).

d singular values of $S = Xe'Ye$ (attached field)

p, q generalized weights $W_X^{-1/2}u$, $W_Y^{-1/2}v$ (attached)

fi, fj variable/component scores $W_X pD$, $W_Y qD$ (attached)

lx, ly row latent variables $M_X^{1/2}XW_X p$, $M_Y^{1/2}YW_Y q$ (attached)

metrics the supplied metrics (attached)

ncomp Number of components actually extracted. The underlying operator may return fewer than the requested `ncomp` (e.g. when `ncomp` exceeds `min(ncol(X), ncol(Y))`); this field reflects the actual count, not the request.

backend The `svd_backend` value passed in (for reference only; see the `svd_backend` argument for when it is actually used).

preproc_x, preproc_y The fitted `multivarious` preprocessing objects for `X` and `Y`, stored on the `cross_projector`.

References

Beaton, D. (2020). Generalized eigen, singular value, and partial least squares decompositions: The GSVD package. (Eqs. 10-14). arXiv:2010.14734.

Examples

```
if (requireNamespace("multivarious", quietly = TRUE)) {
  set.seed(1)
  n <- 100; p <- 40; q <- 30
  X <- matrix(rnorm(n*p), n, p)
  Y <- matrix(rnorm(n*q), n, q)
  w <- runif(n); w <- w/sum(w)
  Mx <- My <- Matrix::Diagonal(x = w)
  fit <- genpls(X, Y, Mx = Mx, My = My, ncomp = 2,
                preproc_x = multivarious::center(),
                preproc_y = multivarious::center())
  fit$d  # singular values
}
```

genplsc *Canonical Generalized PLS (alias)*

Description

Convenience alias for `genpls()`; computes canonical generalized PLS (PLS-SVD/GPLSSVD). See `?genpls` for full documentation.

Usage

```
genplsc(
  X,
  Y,
  Ax = NULL,
  Ay = NULL,
  Mx = NULL,
  My = NULL,
  ncomp = 2,
  preproc_x = multivarious::pass(),
  preproc_y = multivarious::pass(),
  svd_backend = c("eigencore", "irlba", "RSpectra"),
  svd_opts = list(tol = 1e-07, maxitr = 1000),
  constraints_remedy = c("error", "ridge", "clip", "identity"),
  verbose = FALSE
)
```

Arguments

`X`	Numeric or Matrix, n x p.
`Y`	Numeric or Matrix, n x q. Must have same n as `X`.
`Ax`	Column metric for X (W_X): vector/diagonal/matrix; `NULL` means identity.
`Ay`	Column metric for Y (W_Y): vector/diagonal/matrix; `NULL` means identity.
`Mx`	Row metric for X (M_X): vector/diagonal/matrix; `NULL` means identity.
`My`	Row metric for Y (M_Y): vector/diagonal/matrix; `NULL` means identity.
`ncomp`	Number of components to extract (rank-k). Default 2.
`preproc_x, preproc_y`	Optional `multivarious` preprocessors (e.g., `center()`). Defaults to `multivarious::pass()` (no-op).
`svd_backend`	Character, one of `"eigencore"` (default) or `"irlba"` for the iterative SVD. This choice only matters for larger problems: whenever both `X` and `Y` have at most 64 columns after preprocessing, the operator materializes `S` densely and computes a direct `svd()`, ignoring `svd_backend` entirely (see `gplssvd_op()`).
`svd_opts`	List of options: `tol` for both backends and `maxitr` for irlba only. An incomplete eigencore solve raises an error of class `genpca_solver_nonconvergence`; no unchecked fit is returned.

constraints_remedy
: What to do with a metric that is not positive semi-definite: "error" (default), "ridge", "clip" or "identity"; repairs emit a genpca_metric_repaired warning. See genpca().

verbose
: Logical; print brief progress messages.

Value

An object of class c("genpls", "cross_projector", "projector") with the same structure as genpls() returns (X-/Y-weights vx/vy, singular values d, generalized weights p/q, scores fi/fj, latent variables lx/ly, ncomp, and backend); see ?genpls for the definition of each slot.

References

Beaton, D. (2020). Generalized eigen, singular value, and partial least squares decompositions: The GSVD package. (Eqs. 10-14). arXiv:2010.14734.

See Also

genpls()

Examples

```
set.seed(1)
X <- matrix(rnorm(60 * 5), 60, 5)
Y <- matrix(rnorm(60 * 4), 60, 4)
fit <- genplsc(X, Y, ncomp = 2,
               preproc_x = multivarious::center(),
               preproc_y = multivarious::center())
fit$d
```

---

gmd_clear_cache *Clear internal cache for matrix decompositions*

Description

Clears the internal cache used by generalized matrix decomposition functions. This can be useful to free up memory or when working with different datasets.

Usage

```
gmd_clear_cache()
```

Value

Invisibly returns TRUE after clearing the cache.

Examples

```
# Clear the internal cache
gmd_clear_cache()
```

`gpca_mle` *Experimental penalized-ML estimation of GPCA metrics*

Description

Alternates between GPCA factor estimation and penalized maximum-likelihood updates of the row/column metric matrices (M, A) under a Gaussian matrix-normal error model with a low-rank mean. Each iteration performs three exact block minimizations of a single penalized objective:

1. Fit GPCA with current `A`, `M`: the GMD theorem makes this the best rank-`ncomp` fit in the (M, A) norm, so it exactly minimizes the residual term.
2. Update $\Sigma_r = EAE^T/p + \lambda I$ (the exact block minimizer under the ridge penalty), set `M = solve(Sigma_r)`.
3. Update $\Sigma_c = E^T ME/n + \lambda I$ using the *updated* `M` (sequential flip-flop, Dutilleul 1999), set `A = solve(Sigma_c)`.

Usage

```
gpca_mle(
  X,
  ncomp = min(dim(X)),
  max_iter = 20,
  lambda = 0.001,
  scale_fix = c("none", "trace", "det"),
  tol = 1e-04,
  method = "eigen",
  constraints_remedy = "error",
  preproc = multivarious::pass(),
  verbose = FALSE,
  ...
)
```

Arguments

`X`	Numeric matrix (n x p).
`ncomp`	Rank to extract at each GPCA step.
`max_iter`	Maximum outer alternations (default 20).
`lambda`	Ridge penalty weight (default 1e-3). Part of the objective (MAP interpretation), not just a numerical safeguard: it shrinks both covariances toward a multiple of the identity and pins the row/column scale split during iteration. Must be non-negative; with `lambda = 0` the objective loses strict convexity in the scale direction and covariances may become singular.

`scale_fix` Optional post-hoc reparameterization of the `c * Sigma_r, Sigma_c / c` split at exit. One of `"none"` (default: keep the penalized optimum), `"trace"` (row covariance scaled to mean diagonal 1) or `"det"` (row covariance scaled to determinant 1). Applied as a joint reciprocal rescale, so the fitted covariance $\Sigma_r \otimes \Sigma_c$ and the unpenalized likelihood are unchanged, but the penalized objective generally decreases; see Details.

`tol` Relative tolerance on successive penalized log-likelihood change (default 1e-4) for early stopping.

`method` GPCA method passed to `genpca` (defaults to "eigen").

`constraints_remedy` Passed to `genpca`; defaults to "error". The learned metrics are inverses of positive definite matrices, so no repair fires in practice.

`preproc` Pre-processing transformer; defaults to `multivariate::pass()`.

`verbose` Logical; if TRUE, prints iteration diagnostics.

`...` Additional arguments forwarded to `genpca`.

Details

The objective is the matrix-normal log-likelihood with a low-rank mean and an inverse-Wishart-style ridge penalty $\lambda\,(p\,\mathrm{tr}\,\Sigma_r^{-1} + n\,\mathrm{tr}\,\Sigma_c^{-1})$ (a MAP estimate). Because every block update is an exact minimizer of this one objective, `loglik_path` is monotone non-decreasing up to numerical noise. The penalty also resolves the $c\,\Sigma_r, \Sigma_c/c$ scale indeterminacy, so the converged metrics are the penalized optimum and no rescaling is needed (`scale_fix = "none"`, the default). `scale_fix = "trace"` or `"det"` additionally applies a joint reciprocal rescale at exit (row covariance normalized, factor absorbed into the column covariance). The unpenalized matrix-normal likelihood is invariant to that rescale, but the penalty $p\lambda\,\mathrm{tr}(M) + n\lambda\,\mathrm{tr}(A)$ is not, so the rescaled metrics are no longer the penalized optimum; the returned `loglik` is always evaluated at the returned metrics and `loglik_rescale_delta` reports how far the rescale moved it. The algorithm stops when the relative change in the penalized log-likelihood falls below `tol` or `max_iter` is reached. Increase `lambda` or reduce `ncomp` if iterations become unstable. The objective is not identifiable with `lambda = 0`.

Value

A list with elements `fit` (a `genpca` fit computed with the returned metrics), `A`, `M` (learned SPD metrics), `loglik` (the penalized log-likelihood evaluated at the returned `M`, `A` and `fit`), `loglik_unpenalized` (the same without the `lambda` penalty), `loglik_rescale_delta` (the change in the penalty contribution caused solely by reciprocal metric rescaling; exactly zero for `scale_fix = "none"`), `loglik_refit_delta` (the remaining change from the last path value to `loglik`, including final refitting and numerical objective reevaluation), and `loglik_path` (the penalized log-likelihood after each outer iteration; monotone non-decreasing up to numerical noise, since every block update exactly minimizes the shared penalized objective). Values omit additive constants and include the `lambda` penalty, so they are comparable across iterations and across runs with the same `lambda`, but not across different `lambda` values.

References

Dutilleul, P. (1999). *The MLE algorithm for the matrix normal distribution*. Journal of Statistical Computation and Simulation, 64(2), 105-123.

Examples

```
if (requireNamespace("multivarious", quietly = TRUE)) {
  set.seed(123)
  X <- matrix(rnorm(40), 8, 5)
  res <- gpca_mle(X, ncomp = 2, max_iter = 5, lambda = 1e-3,
                  scale_fix = "trace", verbose = FALSE)
  # Learned metrics are SPD and match dimensions
  dim(res$A); dim(res$M)
  res$loglik_path
}
```

`gplssvd_op` *Generalized PLS-SVD via Implicit Operator (memory-safe)*

Description

Compute the top-k singular triplets of $S = Xe'Ye$ without materializing the whitened matrices $Xe = Mx^{1/2}XWx^{1/2}$, $Ye = My^{1/2}YWy^{1/2}$ when doing so would densify sparse data. When the whitening is sparsity-preserving (identity/diagonal metrics) or the data are dense, the whitened blocks are precomputed once so each matrix-vector product in the iterative SVD costs two multiplies.

Usage

```
gplssvd_op(
  X,
  Y,
  XLW = NULL,
  YLW = NULL,
  XRW = NULL,
  YRW = NULL,
  k = 2,
  center = FALSE,
  scale = FALSE,
  svd_backend = c("eigencore", "irlba", "RSpectra"),
  svd_opts = list(tol = 1e-07, maxitr = 1000),
  constraints_remedy = c("error", "ridge", "clip", "identity")
)
```

Arguments

X	n x I matrix (numeric or Matrix)
Y	n x J matrix (numeric or Matrix)
XLW	Row metric for X (M_X): NULL/identity, numeric length-n, diagonalMatrix, or PSD Matrix
YLW	Row metric for Y (M_Y)
XRW	Column metric for X (W_X)
YRW	Column metric for Y (W_Y)
k	Number of components. If k exceeds min(ncol(X), ncol(Y)), a warning is issued and k is silently truncated to that maximum.
center, scale	Logical; pre-center/scale columns of X, Y before metrics
svd_backend	One of "eigencore" (default) or "irlba"; "RSpectra" is accepted as a deprecated alias of "eigencore". Ignored whenever both ncol(X) <= 64 and ncol(Y) <= 64, in which case S is materialized densely and solved with base::svd().
svd_opts	List of options for the backend: tol (both backends) and maxitr (irlba only; the eigencore partial SVD has no iteration cap). An incomplete eigencore solve raises genpca_solver_nonconvergence; try a less stringent tol if the requested accuracy cannot be reached.
constraints_remedy	What to do with a metric that is not positive semi-definite: "error" (default), "ridge", "clip" or "identity"; repairs emit a genpca_metric_repaired warning. See genpca().

Details

Naming map: this function names its metrics XLW/YLW (left/row weights) and XRW/YRW (right/column weights); these correspond to Mx/My (row metrics) and Ax/Ay (column metrics) in genpca()'s and genpls()'s M/A convention.

Value

A list with elements:

d Length-k numeric vector of singular values of $S = Xe'Ye$.

u I x k matrix; left singular vectors of S (orthonormal in the Euclidean metric).

v J x k matrix; right singular vectors of S (orthonormal in the Euclidean metric).

p I x k matrix of generalized X-weights, $p = W_X^{-1/2}u$.

q J x k matrix of generalized Y-weights, $q = W_Y^{-1/2}v$.

fi I x k matrix of X-variable scores, $F_i = W_X pD$ (columns of p rescaled by the singular values).

fj J x k matrix of Y-variable scores, $F_j = W_Y qD$.

lx N x k matrix of X row latent variables, $L_x = M_X^{1/2} X W_X p$.

ly N x k matrix of Y row latent variables, $L_y = M_Y^{1/2} Y W_Y q$.

k Integer; number of components actually returned (may be less than the requested k if it exceeded min(I, J)).

dims A list `list(N, I, J)` with the row count `N` and column counts `I = ncol(X)`, `J = ncol(Y)`.

center A list `list(X, Y)` of the length-`I` / length-`J` column means subtracted from `X`/`Y` (all zero when `center = FALSE`).

scale A list `list(X, Y)` of the length-`I` / length-`J` column scale factors divided out of `X`/`Y` (all one when `scale = FALSE`).

References

Beaton, D. (2020). Generalized eigen, singular value, and partial least squares decompositions: The GSVD package. arXiv:2010.14734.

Abdi, H. (2007). Partial least square regression PLS-Regression. In N. Salkind (Ed.), *Encyclopedia of Measurement and Statistics*. Thousand Oaks, CA: Sage.

Examples

```
set.seed(1)
X <- matrix(rnorm(40 * 6), 40, 6)
Y <- matrix(rnorm(40 * 4), 40, 4)
op <- gplssvd_op(X, Y, k = 2, center = TRUE)
round(op$d, 3)
```

`mnpca_mrl` *Matrix-Normal PCA via Maximum Regularized Likelihood*

Description

Fits a rank-`ncomp` matrix factorization under matrix-normal noise with sparse row/column precision matrices:

$$Y = XW^\top + E, \quad E \sim \mathcal{MN}(0, \Omega, \Sigma)$$

using alternating block updates:

1. Alternating least squares updates for `X`, `W` with fixed precisions (`Theta_row = Omega^{-1}`, `Theta_col = Sigma^{-1}`).
2. Graphical-lasso style precision updates for `Theta_row` and `Theta_col` using ADMM, warm starts, and optional block screening.

The precision updates are solved on a residual-free scatter surrogate (rather than the exact residual `E = Y - XW^T`), so the two block updates do not jointly minimize a single shared objective at each step; as a result the reported `objective_path` is a convergence diagnostic, not a monotone objective trace (see Details and the `objective_path` return value).

Usage

```
mnpca_mrl(
  Y,
  ncomp = min(dim(Y)),
  lambda_row = 0.05,
  lambda_col = 0.05,
  max_outer = 25,
  max_inner = 5,
  tol = 1e-04,
  eps_ridge = 1e-08,
  jitter = 1e-06,
  center = TRUE,
  update_precisions = TRUE,
  warm_start = TRUE,
  gl_maxit = 200,
  gl_tol = 1e-04,
  gl_rho = 1,
  penalize_diagonal = FALSE,
  block_screen = TRUE,
  scale_fix = c("trace", "none"),
  sparsify_threshold = 1e-08,
  as_sparse_precision = TRUE,
  verbose = FALSE
)
```

Arguments

Y	Numeric matrix (n x p).
ncomp	Target rank (r).
lambda_row	L1 penalty for row precision (Theta_row).
lambda_col	L1 penalty for column precision (Theta_col).
max_outer	Maximum number of outer BCD iterations.
max_inner	Maximum ALS steps per outer iteration.
tol	Relative tolerance used for ALS and objective convergence checks.
eps_ridge	Ridge added to small r x r normal-equation systems.
jitter	Small diagonal jitter used in covariance/precision updates.
center	Logical; center columns of Y before fitting.
update_precisions	Logical; if FALSE, keeps identity precisions and runs weighted ALS only.
warm_start	Logical; warm start precision updates from previous iterate.
gl_maxit	Maximum ADMM iterations per graphical-lasso subproblem.
gl_tol	ADMM convergence tolerance for graphical-lasso subproblems.
gl_rho	ADMM augmented Lagrangian parameter.

penalize_diagonal
: Logical; whether to penalize diagonal precision entries in L1 term. Default FALSE.

block_screen
: Logical; use thresholded connected components to solve precision subproblems blockwise.

scale_fix
: One of "trace" or "none". "trace" normalizes each precision to mean diagonal 1 after updates.

sparsify_threshold
: Off-diagonal magnitude threshold used to set tiny precision entries to zero after each graphical-lasso solve.

as_sparse_precision
: Logical; store precisions as sparse matrices when many entries are zero.

verbose
: Logical; print iteration diagnostics.

Details

The covariance updates use low-rank correction identities and avoid explicit construction of E = Y - XW^T.

This function returns a plain S3 list of class "mnpca_mrl", not a **multivarious** bi_projector/cross_projector; the scores()/components()/reconstruct() generics used elsewhere in this package do not apply here. Use the returned $X, $W, and $fitted fields directly.

This implementation follows the maximum regularized likelihood (MRL) formulation of MN-PCA, combining low-rank factor updates with sparse precision estimation in row and column spaces, via alternating surrogate updates rather than joint minimization of a single objective at every step.

The main optimization target is:

$$\frac{1}{2}\mathrm{tr}\left(\Theta_c(Y - XW^\top)^\top \Theta_r (Y - XW^\top)\right) - \frac{p}{2}\log|\Theta_r| - \frac{n}{2}\log|\Theta_c| + n\lambda_r\|\Theta_r\|_1 + p\lambda_c\|\Theta_c\|_1$$

with $\Theta_r \succ 0, \Theta_c \succ 0$. When update_precisions = FALSE, the method reduces to weighted low-rank approximation with fixed identity precisions.

Because the ALS factor updates and the graphical-lasso precision updates are solved against different surrogates of this target (the precision step uses a residual-free scatter matrix rather than the exact residual), the outer alternation is best understood as alternating surrogate updates with a relative-objective-change convergence heuristic, not classical block coordinate descent on a single monotone objective.

Value

An object of class "mnpca_mrl" with components:

X, W Estimated low-rank factors (n x r, p x r).

Theta_row, Theta_col Estimated row/column precision matrices.

fitted Reconstructed matrix on input scale.

fitted_centered Reconstructed centered matrix used in optimization.

residual_centered Centered residual matrix.

objective_path Objective value evaluated after each outer iteration's block updates, before any trace rescaling of the precisions. Because the factor updates (ALS) and precision updates (graphical lasso on a residual-free scatter surrogate) target different surrogates rather than a single shared objective, this path is a convergence heuristic and is **not guaranteed to be monotone**.

iterations Number of outer iterations used.

converged Logical; `TRUE` if the relative change in `objective_path` fell below `tol` before `max_outer` was reached. This is a convergence heuristic based on relative objective change, not a guarantee of a local optimum.

center Column centering vector (or `NULL`).

call Matched call.

References

Zhang, C., Gai, K., & Zhang, S. (2024). *Matrix normal PCA for interpretable dimension reduction and graphical noise modeling*. Pattern Recognition, 154, 110591. doi:10.1016/j.patcog.2024.110591

Friedman, J., Hastie, T., & Tibshirani, R. (2008). *Sparse inverse covariance estimation with the graphical lasso*. Biostatistics, 9(3), 432-441.

Examples

```
set.seed(123)
n <- 20; p <- 12; r <- 3
Y <- matrix(rnorm(n * p), n, p)
fit <- mnpca_mrl(
  Y,
  ncomp = r,
  lambda_row = 0.08,
  lambda_col = 0.08,
  max_outer = 6,
  max_inner = 6,
  verbose = FALSE
)
dim(fit$X)
dim(fit$W)
length(fit$objective_path)
```

`print.sfpca` *Print an sfpca fit*

Description

Prints a one-line summary of an `sfpca` object: number of components, dimensions, and singular values.

Usage

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

Arguments

x	An sfpca object.
...	Ignored.

Value

x, invisibly.

---

reconstruct.genpca *Reconstruct data from a genpca fit*

Description

Reconstructs (an approximation of) the original data from a genpca fit as ou[, comp] %*% diag(d[comp]) %*% t(ov[, comp]), followed by the inverse of the preprocessing transform. With all components and full rank this recovers the original data.

Usage

```
## S3 method for class 'genpca'
reconstruct(
  x,
  comp = 1:multivarious::ncomp(x),
  rowind = NULL,
  colind = NULL,
  ...
)
```

Arguments

x	A genpca object.
comp	Integer vector of components to use (default: all).
rowind	Optional integer vector of rows to reconstruct (default: all).
colind	Optional integer vector of columns to reconstruct (default: all). The inverse preprocessing transform is applied to the selected columns.
...	Ignored.

Value

A numeric matrix of dimension length(rowind) x length(colind).

See Also

genpca(), truncate.genpca()

Examples

```
X <- matrix(rnorm(60), 15, 4)
fit <- genpca(X, ncomp = 4, preproc = multivarious::center())
max(abs(reconstruct(fit) - X)) # ~ 0 at full rank
```

`reconstruct.sfpca` *Reconstruct data from an sfpca fit*

Description

Reconstructs the rank-K sfpca model as UDV', using the stored (non-orthogonal) factors directly.

Usage

```
## S3 method for class 'sfpca'
reconstruct(
  x,
  comp = seq_len(multivarious::ncomp(x)),
  rowind = NULL,
  colind = NULL,
  ...
)
```

Arguments

x	An sfpca object.
comp	Integer vector of components to use (default: all).
rowind	Optional integer vector of rows to reconstruct (default: all).
colind	Optional integer vector of columns to reconstruct (default: all).
...	Ignored.

Details

sfpca components are Euclidean unit vectors but are **not** mutually orthogonal, so $V'V \neq I$. The inherited reconstruct.bi_projector() method reconstructs through the Moore-Penrose pseudoinverse of the loadings (scores \%*\% pinv(V)), which for non-orthogonal V does **not** return the rank-comp model UDV' that sfpca() actually fits and deflates with. This method instead computes scores(x)[rowind, comp] \%*\% t(components(x)[colind, comp]), i.e. UDV' restricted to the requested rows/columns/components. sfpca() does no preprocessing, so no inverse transform is applied.

Value

A numeric matrix of dimension `length(rowind) x length(colind)`, the rank-`length(comp)` reconstruction UDV' using sfpca's stored non-orthogonal factors.

See Also

`sfpca()`

`repair_metric` *Repair a metric matrix explicitly*

Description

Returns a positive (semi)definite version of `A` together with a diagnostic report of what was done. This is the explicit counterpart of the `constraints_remedy` argument of `genpca()`: nothing in the package repairs a metric silently, and this function lets you inspect the repair before using the result.

Usage

```
repair_metric(
  A,
  method = c("ridge", "clip", "identity"),
  rtol = .metric_rtol_default(),
  name = "A",
  diag_maxn = 2000L
)
```

Arguments

`A`	A square symmetric matrix (base matrix or `Matrix`). Asymmetry beyond round-off is an error (see the relative asymmetry test in `symmetrize_or_stop()`); it is not something a PSD repair should hide.
`method`	`"ridge"` adds a diagonal loading that makes the matrix positive definite (Gershgorin-based shift, falling back to `Matrix::nearPD()` for small dense matrices); `"clip"` projects onto the PSD cone by zeroing negative eigenvalues (dense eigendecomposition; refuses large sparse input); `"identity"` replaces an indefinite matrix by the identity (the report still describes the input).
`rtol`	Relative tolerance: eigenvalues above `-rtol * scale(A)` count as non-negative for `"ridge"` and `"identity"`, which then return the matrix unchanged. `"clip"` always removes negative eigenvalues, regardless of this tolerance (up to reconstruction roundoff). Default `sqrt(.Machine$double.eps)`.
`name`	Label used in messages.
`diag_maxn`	Largest dimension for which the report computes the full spectrum (minimum eigenvalue, rank, condition number); above it an iterative minimum eigenvalue estimate and a Gershgorin bound are reported, with rank and condition number unavailable.

Value

The repaired matrix (a `Matrix`), with attribute `"repair_report"` of class `"metric_repair_report"`: a list with `name`, `method`, `changed`, `n`, `min_eigenvalue_before`, `min_eigenvalue_after`, `gershgorin_bound_before`, `shift` (diagonal loading added by `"ridge"`, `NA` for `"clip"`), `rank`, `condition_number` and `rtol`.

See Also

`genpca()` (argument `constraints_remedy`)

Examples

```
A <- matrix(c(1, 2, 2, 1), 2)           # eigenvalues 3 and -1
B <- repair_metric(A, method = "ridge")
attr(B, "repair_report")
C <- repair_metric(A, method = "clip")
eigen(as.matrix(C))$values
```

---

rpls *Regularised / Generalised Partial Least Squares (RPLS / GPLS)*

Description

Implements the algorithm of Allen *et al.* (2013) for supervised dimension-reduction with optional sparsity (ℓ_1) or ridge (ℓ_2) penalties **and** the generalised extension that operates in a user-supplied quadratic form Q.

Usage

```
rpls(
  X,
  Y,
  K = 2,
  lambda = 0.1,
  penalty = c("l1", "ridge"),
  Q = NULL,
  nonneg = FALSE,
  preproc_x = multivarious::pass(),
  preproc_y = multivarious::pass(),
  tol = 1e-06,
  maxiter = 200,
  verbose = FALSE,
  ...
)
```

Arguments

X	Numeric matrix $(n \times p)$ — predictors.
Y	Numeric matrix $(n \times q)$ — responses.
K	Integer, number of latent factors to extract. Default 2.
lambda	Scalar or length-K numeric vector of penalties.
penalty	Either "l1" (lasso) or "ridge".
Q	Optional positive-(semi)definite $p \times p$ matrix inducing *generalised* PLS. NULL means identity.
nonneg	Logical, force non-negative loadings when penalty = "l1". Note: This option is currently ignored when penalty = "ridge".
preproc_x, preproc_y	Optional **multivarious** preprocessing objects (see fit_transform). By default they pass the data through unchanged using pass().
tol	Relative tolerance for the inner iterations convergence check. Default 1e-6.
maxiter	Maximum number of inner iterations per component. Default 200.
verbose	Logical; print progress messages during component extraction. Default FALSE.
...	Further arguments (e.g., custom stopping criteria if implemented) are stored in the returned object (they are not used by fit_rpls).

Details

Unlike genpls(), which handles separate row and column metrics (Mx, Ax, My, Ay) with a Gram–Schmidt orthogonalisation step, rpls() uses a single metric Q and the simpler penalised updates of Allen et al.

Value

An object of class c("rpls","cross_projector","projector") with at least the elements

vx $p \times K$ matrix of X-loadings.

vy $q \times K$ matrix of Y-loadings.

ncomp Number of components actually extracted (may be < K).

penalty Penalty type used ("l1" or "ridge").

lambda The lambda value(s) used.

tol The convergence tolerance used.

maxiter The maximum number of inner iterations used per component.

nonneg Logical flag for the non-negativity constraint on l1.

Q_used Logical; TRUE if a custom Q metric was supplied to induce generalised RPLS, FALSE for standard (identity-metric) RPLS. Note the field is named Q_used, not Q: the metric matrix itself is not retained on the returned object.

verbose The verbose flag used.

preproc_x, preproc_y Pre-processing transforms used.

`rpls` objects store no row (X-)scores: the factors z_k are computed internally during fitting to drive deflation and are then discarded, so use `multivarious::project()` on `X` to obtain scores for any rows of interest.

The object supports `predict()`, `project()`, `transfer()`, `coef()` and other **multivarious** generics.

Method

The routine follows Algorithm 1 of Allen *et al.* (2013, *Stat. Anal. Data Min.*, 6 : 302–314) — see the paper for details. Briefly, with $C = X^\top Y$ the cross-product of the (preprocessed) blocks, each component maximises

$$\max_{u,v} \; v^\top Q C u - \lambda\, P(v)$$

with $Q = I_p$ for standard RPLS. The alternating updates are: $u \leftarrow C^\top Q v / \|C^\top Q v\|_2$, then a penalised (possibly non-negative) regression for v, normalised in the Q-norm.

References

Allen, G. I., Peterson, C., Vannucci, M., & Maletić-Savatić, M. (2013). *Regularized Partial Least Squares with an Application to NMR Spectroscopy.* **Statistical Analysis and Data Mining, 6(4)**, 302-314. DOI:10.1002/sam.11169.

Examples

```
# Generate sample data
set.seed(123)
n <- 50
p <- 20
q <- 10
X <- matrix(rnorm(n * p), n, p)
Y <- X[, 1:5] %*% matrix(rnorm(5 * q), 5, q) + matrix(rnorm(n * q), n, q)

# Fit regularized PLS with L1 penalty
fit_l1 <- rpls(X, Y, K = 3, lambda = 0.1, penalty = "l1")
print(fit_l1)

# Fit regularized PLS with ridge penalty
fit_ridge <- rpls(X, Y, K = 3, lambda = 0.1, penalty = "ridge")
print(fit_ridge)
```

`sfpca` *Sparse and Functional Principal Components Analysis (SFPCA) with Spatial Coordinates*

Description

Performs Sparse and Functional PCA on a data matrix, allowing for both sparsity and smoothness in the estimated principal components. Penalty parameters left `NULL` are selected automatically (see Details). The spatial smoothness penalty is constructed based on provided spatial coordinates.

Usage

```
sfpca(
  X,
  K,
  spat_cds,
  lambda_u = NULL,
  lambda_v = NULL,
  alpha_u = NULL,
  alpha_v = NULL,
  Omega_u = NULL,
  penalty_u = "l1",
  penalty_v = "l1",
  nlambda = 10,
  lambda_min_ratio = 0.01,
  knn = min(6, ncol(X) - 1),
  max_iter = 100,
  tol = 1e-06,
  verbose = FALSE,
  uthresh = NULL,
  vthresh = NULL
)
```

Arguments

X	A numeric data matrix of dimensions n (observations/time points) by p (variables/space).
K	The number of principal components to estimate.
spat_cds	A matrix of spatial coordinates for each column of X (variables). Each row corresponds to a spatial dimension (e.g., x, y, z), and each column corresponds to a variable. Note the orientation: this is `dimensions x variables`, so `ncol(spat_cds)` must equal `ncol(X)` – the transpose of the layout a coordinate data frame usually has. For a one-dimensional axis (a spectrum, a transect) pass `matrix(coords, nrow = 1)`.
lambda_u	Sparsity penalty parameter for u. If NULL, selected per component by BIC along a regularization path (see Details).
lambda_v	Sparsity penalty parameter for v. If NULL, selected per component by BIC along a regularization path (see Details).
alpha_u	Smoothness penalty parameter for u. If NULL, defaults to `1 / lambda_max(Omega_u)` (see Details).
alpha_v	Smoothness penalty parameter for v. If NULL, defaults to `1 / lambda_max(Omega_v)` (see Details).
Omega_u	A positive semi-definite matrix for smoothness penalty on u. If NULL, defaults to second differences penalty (sparse matrix). Unlike `Omega_u`, there is no corresponding `Omega_v` argument: the column-side smoothness penalty is always built internally from `spat_cds` (via `knn`); supplying a custom `Omega_v` is not currently supported.

penalty_u	The penalty function for u. Either "l1" (lasso, the default) or "scad".
penalty_v	The penalty function for v. Either "l1" (lasso, the default) or "scad".
nlambda	Number of values on the regularization path used for BIC selection of lambda_u/lambda_v when they are NULL. Default 10.
lambda_min_ratio	Smallest path value as a fraction of the closed-form lambda_max, on a log-spaced grid. Default 1e-2.
knn	Number of nearest neighbours for constructing Omega_v. Default min(6, ncol(X) - 1).
max_iter	Maximum number of iterations for the alternating optimization. Default 100.
tol	Tolerance for convergence of the rank-1 objective. Default 1e-6.
verbose	Logical; if TRUE, prints progress messages.
uthresh	Deprecated and ignored; lambda_u is now selected by BIC.
vthresh	Deprecated and ignored; lambda_v is now selected by BIC.

Details

Each rank-1 problem is solved by alternating solves of the penalized quadratic subproblems (via C++ coordinate descent) followed by rescaling onto the smoothness-metric ball, in the constraint form of Allen & Weylandt (2019). For the convex "l1" penalty with subproblems solved to tolerance (the internal exact_inner = TRUE path, used by the monotonicity test) the objective is monotonically non-decreasing; the default inexact path tightens the inner tolerance to a floor before it may declare convergence, reproducing the same terminal iterates but without an every-iteration monotonicity guarantee (it may also stop at max_iter).

When lambda_u or lambda_v is NULL it is selected per component by a BIC-style criterion along a regularization path. For the convex "l1" penalty lambda_max = max(abs(b)) is, in closed form, the smallest value whose subproblem solution is exactly zero (at x = 0 the S x term vanishes, so the KKT condition |b_j| <= lambda does not depend on S); where b is the matrix-vector product with the other factor fixed at the SVD initializer. For the non-convex "scad" penalty the same value anchors the path but is not a global-optimality threshold. nlambda values are laid log-spaced down to lambda_min_ratio * lambda_max, coordinate descent is warm-started along the path, and the value minimizing log(RSS / (n p)) + df * log(n p) / (n p) is chosen, with df the support size of the solution and RSS the one-sided rank-1 residual sum of squares with the opposite factor held fixed (a selection heuristic, not the BIC of the fully alternated rank-1 model). The all-zero solution (at lambda_max) is a legitimate candidate: if no rank-1 structure justifies its degrees of freedom, the component is returned as exactly zero with d = 0.

When alpha_u or alpha_v is NULL it defaults to 1 / lambda_max(Omega), so the roughest direction of the smoothness penalty is weighted exactly as strongly as the identity term. This makes the default invariant to the scaling of Omega and bounds the condition number of every subproblem system I + alpha * Omega by 2.

Value

An object of class c("sfpca", "bi_projector") from the **multivarious** framework. Use multivarious::scores() for the sample scores (UD), multivarious::components() for the sparse loadings V, multivarious::sdev() for d_k, and multivarious::reconstruct() for the rank-K approximation. ov (like components())

holds the sparse right factors V; ou holds the left factors U. The selected penalty parameters are stored as lambda_u, lambda_v, alpha_u, and alpha_v. For backward compatibility the pre-0.1 list fields $d (singular values) and $u (left factors) remain readable but emit a deprecation warning; use sdev() and scores()/$ou instead.

Important: unlike genpca(), the columns of U (ou) and V (ov) are Euclidean unit-norm but are **not** mutually orthogonal across components – sfpca() extracts each rank-1 term from a constraint-form subproblem rather than a joint SVD, so U'U != I and V'V != I in general. Consequently multivarious::sdev() here is *not* the singular values of X; it is the per-component captured covariance $d_k = u_k' X_k v_k$, where X_k is the matrix after the preceding components have been deflated out (so the identity holds against X itself only for $k = 1$). This non-orthogonality is also why reconstruct() for "sfpca" objects uses the stored U, d, V factors directly (U D V') rather than SVD-based identities such as the Moore-Penrose pseudoinverse of the loadings, which would not reproduce the fitted model for non-orthogonal V (see reconstruct.sfpca()).

References

Allen, G. I., & Weylandt, M. (2019). Sparse and functional principal components analysis. In *2019 IEEE Data Science Workshop (DSW)* (pp. 11-16). doi:10.1109/DSW.2019.8755778. Also available as arXiv:1309.2895, first posted in 2013 and revised through 2019; the preprint and the DSW paper are the same work, which is why both years appear in the literature.

See Also

genpca() for the shared **multivarious** verbs; multivarious::bi_projector.

Examples

```
library(Matrix)
set.seed(123)
# Smooth temporal factor, sparse spatial factor
n <- 100  # Number of time points
p <- 50   # Number of spatial locations
u <- sin(seq(0, 2 * pi, length.out = n))
v <- c(rnorm(10), rep(0, p - 10))
X <- 8 * tcrossprod(u / sqrt(sum(u^2)), v / sqrt(sum(v^2))) +
  matrix(rnorm(n * p, sd = 0.2), n, p)
spat_cds <- matrix(runif(p * 3), nrow = 3, ncol = p)  # 3D coordinates
result <- sfpca(X, K = 1, spat_cds = spat_cds)
multivarious::sdev(result)                  # captured covariance (BIC-tuned)
sum(multivarious::components(result) != 0)  # sparse spatial loading
```

---

transfer.cross_projector

Compatibility wrapper for multivarious::transfer

Description

Provides a transfer method that accepts source/target arguments used in the tests. The method forwards to multivarious's transfer method which expects from/to.

Usage

```
## S3 method for class 'cross_projector'
transfer(x, new_data, from = NULL, to = NULL, opts = list(), ...)
```

Arguments

x	A cross_projector object.
new_data	New data to transfer.
from	Source space ("X" or "Y").
to	Target space ("X" or "Y").
opts	Options list passed to multivarious::transfer.
...	Additional arguments. Legacy parameters source and target are accepted here for backwards compatibility but are deprecated; use from and to instead.

Value

Matrix with transferred data.

Examples

```
# Generate sample data
set.seed(123)
n <- 50
X <- matrix(rnorm(n * 10), n, 10)
Y <- matrix(rnorm(n * 8), n, 8)

# Create a cross projector using rpls
fit <- rpls(X, Y, K = 2)

# Transfer new X data to Y space
new_X <- matrix(rnorm(10 * 10), 10, 10)
transferred <- transfer(fit, new_X, from = "X", to = "Y")
```

---

truncate *Truncate a projection to fewer components*

Description

Re-exported from **multivarious**. See truncate for details.

Usage

```
truncate(x, ncomp)
```

Arguments

x	A projection object
ncomp	Number of components to retain

Value

Truncated projection object

`truncate.genpca` *Truncate a genpca fit to fewer components*

Description

Returns a new genpca object retaining only the first ncomp components. All component-indexed slots (v, s, sdev, ov, ou, u, propv, cumv) are sliced consistently; the preprocessing object and constraint matrices are carried over unchanged.

Usage

```
## S3 method for class 'genpca'
truncate(x, ncomp)
```

Arguments

x	A genpca object.
ncomp	Number of components to retain (a positive integer no larger than ncomp(x)).

Value

A genpca object with ncomp components.

See Also

genpca(), reconstruct.genpca()

Examples

```
X <- matrix(rnorm(60), 15, 4)
fit <- genpca(X, ncomp = 4)
fit2 <- truncate(fit, 2)
multivarious::ncomp(fit2)
```

Index

fit_transform, *28*

geigen_cov, 2, *9, 10*
genpca, *3*, 4, *9, 10*
genpca(), *12*, *15*, *19*, *25–27*, *32*, *34*
genpca_cov, *3*, *7, 8*, 9
genpls, *10*, 11
genpls(), *15*
genplsc, 14
gmd_clear_cache, 15
gpca_mle, 16
gplssvd_op, 18

mnpca_mrl, 20

print.sfpca, 23

reconstruct.genpca, *8*, 24
reconstruct.genpca(), *34*
reconstruct.sfpca, 25
repair_metric, *5*, 26
rpls, 27

sfpca, *23*, *25*, 29
sfpca(), *26*

transfer.cross_projector, 32
truncate, *33*, 33
truncate.genpca, *8*, 34
truncate.genpca(), *25*