Package 'eigencore'

July 23, 2026

Title Certified Partial Eigenvalue and Singular Value Computation

Version 1.0.0

Author Bradley Buchsbaum [aut, cre, cph]

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

Description Computes the top-k singular triplets or eigenpairs of large sparse and structured matrices: the computation behind principal component analysis on big sparse data, spectral embeddings, and low-rank approximation. Every result carries a numerical certificate with residuals, a backward-error bound, orthogonality loss, and a pass/fail flag, and bounds that can only be estimated are reported as such rather than passed. Centered, scaled, and composed operators are solved through native 'C++' kernels without forming dense matrices. Drop-in replacements for the 'RSpectra' interface are included.

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

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

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.1)

Imports Matrix, methods

Suggests bench, knitr, rmarkdown, RSpectra, irlba, rsvd, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

SystemRequirements C++17

Config/Needs/website albersdown

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-07-23 14:10:02 UTC

Contents

`adjoint` *Return the adjoint operator.*

Description

Return the adjoint operator.

Usage

```
adjoint(x, ...)
```

Arguments

`x`	Operator-like object.
`...`	Additional arguments passed to methods.

Value

An `eigencore_operator` representing the adjoint map.

`alpha_beta` *Extract homogeneous generalized coordinates.*

Description

Generalized dense-pencil and generalized Schur results store eigenvalues in homogeneous form as `alpha / beta`; generalized SVD results use the same alpha/beta convention for generalized singular values. `alpha_beta()` exposes those coordinates along with the finite/infinite/undefined classification computed by eigencore.

Usage

```
alpha_beta(x, ...)
```

Arguments

x	An eigencore result with homogeneous alpha and beta fields.
...	Reserved for future methods.

Value

A list containing alpha, beta, and any available values, classification, finite, infinite, and undefined fields. Results that record how the finite/infinite/undefined labels were decided also include a classification_policy list with the policy name, the tolerance, the per-coordinate zero thresholds, and the pencil norms used for norm-scaled classification.

Examples

```
A <- diag(c(2, 3, 0))
B <- diag(c(1, 0, 0))
fit <- eig_full(A, B = B, structure = general())
alpha_beta(fit)$classification
```

---

as_operator *Convert an object to an eigencore operator.*

Description

Convert an object to an eigencore operator.

Usage

```
as_operator(x, ...)
```

Arguments

x	Object to convert.
...	Additional arguments passed to methods.

Value

An eigencore_operator representation of x.

Examples

```
op <- as_operator(diag(c(3, 2, 1)))
op$dim
op$structure$kind
```

auto *Automatic solver choice.*

Description

Automatic solver choice.

Usage

```
auto()
```

Value

An `eigencore_method` descriptor that lets the planner choose a solver based on problem structure.

backward_error *Extract backward-error diagnostics.*

Description

Extract backward-error diagnostics.

Usage

```
backward_error(x, ...)
```

Arguments

`x`	An eigencore result object.
`...`	Reserved for future methods.

Value

A numeric vector of per-pair or per-triplet backward-error estimates.

both_ends *Target both algebraic ends.*

Description

Target both algebraic ends.

Usage

```
both_ends(k_low, k_high)
```

Arguments

k_low	Number of values to select from the smallest algebraic end.
k_high	Number of values to select from the largest algebraic end.

Value

An `eigencore_target` descriptor selecting both algebraic ends (ARPACK `BE`).

center *Center an operator by rows or columns.*

Description

Center an operator by rows or columns.

Usage

```
center(
  A,
  rows = FALSE,
  columns = TRUE,
  row_means = NULL,
  col_means = NULL,
  name = NULL
)
```

Arguments

A	Operator-like object.
rows	Whether to subtract row means.
columns	Whether to subtract column means.
row_means	Optional row means. Required for matrix-free row centering when they cannot be derived without densifying.

col_means	Optional column means. Required for matrix-free column centering when they cannot be derived without densifying.
name	Optional label for the centered operator.

Value

An eigencore_operator representing the centered linear map.

`certificate` *Extract a result certificate.*

Description

Extract a result certificate.

Usage

```
certificate(x, ...)
```

Arguments

x	An eigencore result object.
...	Reserved for future methods.

Value

The eigencore_certificate object stored on x, or NULL if the result does not carry a certificate field.

Examples

```
fit <- eig_partial(diag(c(3, 2, 1)), k = 1, target = largest())
cert <- certificate(fit)
cert$passed
cert$max_residual
```

check_adjoint *Check an operator adjoint identity.*

Description

Check an operator adjoint identity.

Usage

```
check_adjoint(A, trials = 20, tol = 1e-12, seed = NULL)
```

Arguments

A	Operator-like object.
trials	Number of random block trials.
tol	Relative tolerance for each adjoint identity check.
seed	Optional random seed for reproducible trials.

Value

An `eigencore_adjoint_check` list with pass/fail status, tolerance, maximum relative error, per-trial errors, and trial count.

compose *Compose two operators.*

Description

Compose two operators.

Usage

```
compose(A, B, name = NULL)
```

Arguments

A	Left operator-like object.
B	Right operator-like object.
name	Optional label for the composed operator.

Value

An `eigencore_operator` representing the composition `A %*% B`.

crossprod_operator *Create A^* A as an operator.*

Description

Create A^* A as an operator.

Usage

```
crossprod_operator(A, name = NULL)
```

Arguments

A	Operator-like object with an adjoint implementation.
name	Optional label for the cross-product operator.

Value

A Hermitian `eigencore_operator` representing `A^* A`.

diagnostics *Extract diagnostics.*

Description

Extract diagnostics.

Usage

```
diagnostics(x, ...)
```

Arguments

x	An eigencore result object.
...	Reserved for future methods.

Value

A named list of diagnostic fields, including residuals, backward errors, orthogonality diagnostics, iteration counts, method/plan metadata, warnings, and any available left-eigenvector diagnostics.

Examples

```
fit <- eig_partial(diag(c(3, 2, 1)), k = 1, target = largest())
d <- diagnostics(fit)
d$nconv
d$method
```

`eigen_problem` *Define an eigenproblem.*

Description

Define an eigenproblem.

Usage

```
eigen_problem(
  A,
  metric = NULL,
  structure = NULL,
  target = largest(),
  transform = NULL
)
```

Arguments

`A`	Matrix or operator defining the linear map.
`metric`	Optional metric operator for generalized eigenproblems.
`structure`	Optional structure descriptor; defaults to the operator structure.
`target`	Eigencore target descriptor.
`transform`	Optional transform method such as `shift_invert()`.

Value

An `eigencore_eigen_problem` object containing the operator, optional metric, structure, target, and transform metadata consumed by `plan_solver()` and `solve()`.

Examples

```
A <- diag(c(4, 3, 2, 1))
P <- eigen_problem(A, target = largest())
fit <- solve(P, k = 2)
values(fit)
```

eigs *RSpectra-compatible eigen shim.*

Description

RSpectra-compatible eigen shim.

Usage

```
eigs(A, k, which = "LM", opts = list(), ...)
```

Arguments

A	Matrix or eigencore operator.
k	Number of eigenpairs to compute.
which	RSpectra-style target selector.
opts	Compatibility options list; currently accepted for API compatibility and not interpreted directly.
...	Additional arguments passed to eig_partial().

Value

A list compatible with RSpectra::eigs(), including values, vectors, convergence counts, operation counts, certificate diagnostics, and left/right vector fields when available.

Examples

```
A <- diag(c(5, 4, 3, 2, 1))
A[1, 2] <- 0.5
res <- eigs(A, k = 2, which = "LM")
res$values
```

eigs_sym *RSpectra-compatible symmetric eigen shim.*

Description

RSpectra-compatible symmetric eigen shim.

Usage

```
eigs_sym(A, k, which = "LA", opts = list(), ...)
```

Arguments

A	Matrix or eigencore operator.
k	Number of eigenpairs to compute.
which	RSpectra-style target selector.
opts	Compatibility options list; currently accepted for API compatibility and not interpreted directly.
...	Additional arguments passed to solve.eigencore_eigen_problem().

Value

A list compatible with RSpectra::eigs_sym(), including values, vectors, convergence counts, operation counts, certificate diagnostics, and eigencore diagnostics.

Examples

```
A <- diag(c(5, 4, 3, 2, 1))
res <- eigs_sym(A, k = 2, which = "LA")
res$values
```

---

eig_full *Compute a full dense eigendecomposition*

Description

eig_full() is the dense/full eigencore surface for standard and generalized eigenproblems. Sparse and operator inputs are not silently densified.

Usage

```
eig_full(
  A,
  B = NULL,
  structure = NULL,
  vectors = TRUE,
  tol = 1e-08,
  allow_dense_fallback = c("auto", "never", "always"),
  ...
)
```

Arguments

A	Base dense square matrix.
B	Optional base dense square matrix for generalized problems.
structure	Optional structure descriptor. Use general() to force the general-pencil path for dense generalized inputs.

vectors	Whether to return right eigenvectors.
tol	Certification tolerance.
allow_dense_fallback	Reserved dense fallback policy. Sparse/operator inputs still fail unless a future issue explicitly opens an opt-in dense fallback contract for eig_full().
...	Reserved for future options.

Value

An eigencore_eigen_result. Dense general-pencil results additionally carry alpha, beta, classification, classification_policy, left_vectors (left generalized eigenvectors satisfying w^H A = lambda w^H B), and conditioning. For real pencils the decomposition runs through the expert LAPACK driver DGGEVX with balancing, so conditioning contains reciprocal condition numbers rconde/rcondv and the balanced pencil norms abnrm/bbnrm. Complex pencils use ZGGEV (R's bundled LAPACK subset has no ZGGEVX), so they return left vectors but conditioning$available is FALSE.

Examples

```
A <- diag(c(1, 4, 9))
B <- diag(c(1, 2, 3))
fit <- eig_full(A, B = B)
values(fit)
certificate(fit)$passed

# Force the dense general-pencil path when alpha/beta diagnostics matter.
pencil <- eig_full(A, B = B, structure = general())
pencil$alpha
pencil$beta
pencil$classification
```

---

eig_partial *Compute a partial eigendecomposition.*

Description

Compute a partial eigendecomposition.

Usage

```
eig_partial(
  A,
  k,
  target = largest(),
  B = NULL,
  method = auto(),
  tol = 1e-08,
  maxit = NULL,
  vectors = TRUE,
```

```
  seed = NULL,
  certify = TRUE,
  allow_dense_fallback = c("auto", "never", "always")
)
```

Arguments

A	Matrix or eigencore operator.
k	Number of eigenpairs to compute.
target	Eigencore eigenvalue target descriptor.
B	Optional metric matrix or operator for generalized problems.
method	Solver method descriptor.
tol	Convergence and certification tolerance.
maxit	Optional iteration limit.
vectors	Whether to compute vectors.
seed	Optional random seed for stochastic solver components.
certify	Whether to compute certification diagnostics.
allow_dense_fallback	Dense fallback policy.

Value

An `eigencore_eigen_result` containing computed values, optional vectors, certificate diagnostics, method/plan metadata, and convergence diagnostics.

Examples

```
A <- diag(c(5, 4, 3, 2, 1))
A[1, 2] <- A[2, 1] <- 0.1
fit <- eig_partial(A, k = 2, target = largest())
values(fit)
certificate(fit)$passed

# Generalized SPD problem A x = lambda B x
B <- diag(c(2, 1, 1, 1, 1))
gfit <- eig_partial(A, B = B, k = 2, target = smallest())
values(gfit)
```

euclidean *Euclidean vector space descriptor.*

Description

Euclidean vector space descriptor.

Usage

```
euclidean(dim, dtype = "double")
```

Arguments

dim	Dimension of the space.
dtype	Scalar type (currently only "double").

Value

An eigencore_space descriptor for the Euclidean space R^dim or C^dim.

general *General operator structure descriptor.*

Description

General operator structure descriptor.

Usage

```
general()
```

Value

An eigencore_structure descriptor for general operators.

generalized_schur *Compute a dense generalized Schur decomposition*

Description

generalized_schur() is eigencore's dense QZ surface for general matrix pencils A x = lambda B x. It computes the generalized Schur pair S, T and, when requested, left/right Schur vectors Q, Z such that A = Q S Z* and B = Q T Z* for complex inputs, with transpose replacing conjugate-transpose for real inputs. Sparse and operator inputs are not silently densified.

Usage

```
generalized_schur(A, B, sort = NULL, vectors = TRUE, ...)
```

Arguments

A	Base dense square matrix.
B	Base dense square matrix with the same dimension as A.
sort	Optional LAPACK sorting class. Use NULL or "none" for no sorting, "finite" to move finite generalized eigenvalues first, "infinite" to move beta-zero nonzero-alpha eigenvalues first. Custom predicates and undefined alpha-zero/beta-zero sorting are not part of the public contract.
vectors	Whether to compute Schur vectors Q and Z.
...	Reserved for future options.

Value

A classed generalized Schur result with fields S, T, Q, Z, alpha, beta, values, classification, sdim, method, plan, and certificate.

Examples

```
A <- matrix(c(0, -1, 1, 0), 2, 2)
B <- diag(2)
qz <- generalized_schur(A, B)
values(qz)

pencil <- generalized_schur(diag(c(2, 3, 0)), diag(c(1, 0, 0)),
                            sort = "infinite")
pencil$classification
```

generalized_svd *Compute a dense generalized singular value decomposition*

Description

generalized_svd() is eigencore's dense GSVD compatibility surface for matrix pairs with the same number of columns. The current native path uses LAPACK dggsvd through R's native LAPACK interface, so it requires a linked LAPACK that still provides that deprecated routine. Sparse/operator inputs are not silently densified, and complex GSVD remains explicit future scope until a complex GSVD driver is available through the eigencore native layer.

Usage

```
generalized_svd(A, B, tol = 1e-08, ...)
```

Arguments

A	Base dense real matrix with m rows and n columns.
B	Base dense real matrix with p rows and n columns.
tol	Finite non-negative reconstruction and orthogonality certification tolerance.
...	Reserved for future options.

Value

An eigencore_gsvd_result with fields alpha, beta, values, classification, U, V, Q, D1, D2, R, zero_R, A_factor, B_factor, k, l, rank, method, plan, and certificate.

Examples

```
A <- matrix(c(1, 2, 3, 3, 2, 1), nrow = 2, byrow = TRUE)
B <- matrix(1:9, nrow = 3)
fit <- generalized_svd(A, B)
alpha_beta(fit)$values
certificate(fit)$passed
```

golub_kahan *Golub-Kahan bidiagonalization method descriptor.*

Description

Golub-Kahan bidiagonalization method descriptor.

Usage

```
golub_kahan(max_subspace = NULL, reorthogonalize = TRUE)
```

Arguments

`max_subspace`	Optional maximum Krylov subspace size.
`reorthogonalize`	Whether to apply full two-sided reorthogonalization. `FALSE` selects the native one-sided small-side policy where supported, with final acceptance still controlled by the exact two-sided certificate.

Value

An `eigencore_method` descriptor selecting Golub-Kahan bidiagonalization.

`hermitian` *Hermitian/symmetric operator structure descriptor.*

Description

Hermitian/symmetric operator structure descriptor.

Usage

```
hermitian()
```

Value

An `eigencore_structure` descriptor marking an operator as Hermitian / symmetric.

`lanczos` *Hermitian Lanczos method descriptor.*

Description

Hermitian Lanczos method descriptor.

Usage

```
lanczos(
  max_subspace = NULL,
  max_restarts = NULL,
  block = 1L,
  reorthogonalize = TRUE
)
```

Arguments

max_subspace	Optional maximum active Krylov subspace size m. Must be at least k + 1. The native thick-restart path keeps the active basis bounded by this value across restart cycles.
max_restarts	Optional non-negative integer giving the maximum number of thick-restart cycles allowed before stopping with whatever has converged. Default 100L.
block	Native block size. 1L selects the scalar thick-restart path; values greater than one select the native block Krylov prototype where supported.
reorthogonalize	Whether to apply full reorthogonalization. The native path always reorthogonalizes (DGKS x2) and ignores this flag; it is preserved for the R reference solver's public API.

Value

An eigencore_method descriptor selecting Lanczos iteration.

---

largest *Target the largest algebraic values.*

Description

Target the largest algebraic values.

Usage

```
largest()
```

Value

An eigencore_target descriptor selecting the largest algebraic eigenvalues or singular values.

---

largest_imaginary *Target the largest imaginary part.*

Description

Target the largest imaginary part.

Usage

```
largest_imaginary()
```

Value

An eigencore_target descriptor selecting values by largest Im(x) (ARPACK LI).

largest_magnitude *Target the largest values by magnitude.*

Description

Target the largest values by magnitude.

Usage

```
largest_magnitude()
```

Value

An `eigencore_target` descriptor selecting values with the largest modulus (ARPACK `LM`).

largest_real *Target the largest real part.*

Description

Target the largest real part.

Usage

```
largest_real()
```

Value

An `eigencore_target` descriptor selecting values by largest `Re(x)` (ARPACK `LR`).

left_vectors *Extract left singular vectors or left eigenvectors.*

Description

For SVD results this returns the left singular vectors `U`. For nonsymmetric and dense general-pencil eigen results this returns left eigenvectors when the solver computed them (for example, the dense general-pencil `eig_full()` path, which computes left generalized eigenvectors satisfying `w^H A = lambda w^H B`).

Usage

```
left_vectors(x, ...)
```

Arguments

x	An eigencore SVD or eigen result object.
...	Reserved for future methods.

Value

A matrix of left singular vectors or left eigenvectors, or NULL when the result does not contain a left-vector field.

---

linear_operator *Create a block-native linear operator.*

Description

Create a block-native linear operator.

Usage

```
linear_operator(
  dim,
  apply,
  apply_adjoint = NULL,
  dtype = "double",
  structure = general(),
  name = NULL,
  metadata = list()
)
```

Arguments

dim	Integer vector of length two giving row and column dimensions.
apply	Function implementing block multiplication by the operator.
apply_adjoint	Optional function implementing block multiplication by the adjoint operator.
dtype	Scalar character type label, currently "double" or "complex".
structure	Eigencore structure descriptor such as general() or hermitian().
name	Optional operator label used in plans and diagnostics.
metadata	Optional list of implementation metadata.

Value

An eigencore_operator list containing dimensions, apply callbacks, scalar type, structure metadata, a display name, and implementation metadata.

Examples

```
A <- diag(c(3, 2, 1))
op <- linear_operator(
  dim = dim(A),
  apply = function(X, alpha = 1, beta = 0, Y = NULL) {
    Z <- alpha * (A %*% X)
    if (is.null(Y) || beta == 0) Z else Z + beta * Y
  },
  structure = hermitian(),
  metadata = list(frobenius_norm = sqrt(sum(A^2)))
)
fit <- eig_partial(op, k = 1, target = largest())
values(fit)
```

---

lobpcg *LOBPCG method descriptor.*

Description

LOBPCG method descriptor.

Usage

```
lobpcg(maxit = 200L, preconditioner = NULL, constraints = NULL)
```

Arguments

`maxit`	Maximum LOBPCG iterations.
`preconditioner`	Optional function taking a residual block and returning a preconditioned block with the same dimensions.
`constraints`	Optional matrix whose columns span a subspace to deflate. Iterates are kept orthogonal to this subspace in the Euclidean or generalized `B` inner product. Native constrained LOBPCG is not promoted yet; constrained problems use the honest reference path.

Value

An `eigencore_method` descriptor selecting LOBPCG. Built-in standard Hermitian dense/CSC operators may use a native prototype; unsupported cases route to the reference prototype.

nearest *Target values nearest a shift.*

Description

Target values nearest a shift.

Usage

```
nearest(sigma)
```

Arguments

sigma	Numeric shift used to rank values by distance `|x - sigma|`.

Value

An `eigencore_target` descriptor for nearest-to-`sigma` selection.

plan_solver *Plan a solver for a problem.*

Description

Plan a solver for a problem.

Usage

```
plan_solver(problem, ...)
```

Arguments

problem	Eigencore eigen or SVD problem object.
...	Additional planning arguments passed to methods.

Value

An `eigencore_plan` list describing the requested problem, chosen method label, target, planner reasons, fallback label, and control metadata used by solver dispatch.

Examples

```
A <- diag(c(4, 3, 2, 1))
plan <- plan_solver(eigen_problem(A), k = 2)
plan$method
plan$reasons
```

randomized *Randomized SVD method descriptor.*

Description

Randomized SVD method descriptor.

Usage

```
randomized(
  oversample = 10,
  n_iter = 2,
  block = NULL,
  normalizer = c("qr", "lu", "none"),
  refine = TRUE
)
```

Arguments

`oversample`	Number of extra samples beyond the requested rank.
`n_iter`	Number of subspace-iteration refinement passes.
`block`	Optional block size.
`normalizer`	Basis normalizer to use (`"qr"`, `"lu"`, or `"none"`).
`refine`	Whether to refine with a certified Lanczos pass.

Value

An `eigencore_method` descriptor selecting randomized SVD.

residuals *Extract residual diagnostics.*

Description

Methods for the `stats::residuals()` generic: return the per-pair (or per-triplet) residual norms stored in an eigencore result or certificate.

Usage

```
## S3 method for class 'eigencore_eigen_result'
residuals(object, ...)

## S3 method for class 'eigencore_svd_result'
residuals(object, ...)

## S3 method for class 'eigencore_certificate'
residuals(object, ...)
```

Arguments

object	An eigencore result or certificate object.
...	Reserved for future methods.

---

right_vectors *Extract right singular vectors.*

Description

Extract right singular vectors.

Usage

```
right_vectors(x, ...)
```

Arguments

x	An eigencore SVD or nonsymmetric eigen result object.
...	Reserved for future methods.

Value

A matrix of right singular vectors or right eigenvectors, or NULL when the result does not contain a right-vector field.

---

scale_cols *Scale operator columns.*

Description

Scale operator columns.

Usage

```
scale_cols(A, weights, name = NULL)
```

Arguments

A	Operator-like object.
weights	Numeric vector of column weights.
name	Optional label for the scaled operator.

Value

An eigencore_operator representing column-wise right scaling of A.

scale_rows *Scale operator rows.*

Description

Scale operator rows.

Usage

```
scale_rows(A, weights, name = NULL)
```

Arguments

A	Operator-like object.
weights	Numeric vector of row weights.
name	Optional label for the scaled operator.

Value

An eigencore_operator representing row-wise left scaling of A.

shifted_cholesky_preconditioner *Shifted Cholesky preconditioner.*

Description

Shifted Cholesky preconditioner.

Usage

```
shifted_cholesky_preconditioner(A, shift = 0)
```

Arguments

A	Symmetric positive semidefinite or positive definite matrix.
shift	Non-negative diagonal shift added before factorization.

Value

A typed preconditioner function mapping residual blocks to preconditioned blocks.

shifted_diagonal_preconditioner *Shifted diagonal preconditioner.*

Description

Shifted diagonal preconditioner.

Usage

```
shifted_diagonal_preconditioner(A, shift = 0)
```

Arguments

A	Diagonal matrix or numeric vector containing the diagonal entries.
shift	Non-negative diagonal shift added before inversion.

Value

A typed native-backed preconditioner function mapping residual blocks to preconditioned blocks.

shifted_tridiagonal_preconditioner *Shifted tridiagonal preconditioner.*

Description

Shifted tridiagonal preconditioner.

Usage

```
shifted_tridiagonal_preconditioner(A, shift = 0)
```

Arguments

A	Real symmetric tridiagonal matrix.
shift	Non-negative diagonal shift added to the tridiagonal system.

Value

A typed preconditioner function mapping residual blocks to preconditioned blocks.

shift_invert *Shift-invert method descriptor.*

Description

Shift-invert method descriptor.

Usage

```
shift_invert(sigma, solve = NULL, factorization = NULL)
```

Arguments

sigma	Shift value sigma.
solve	Optional user-supplied solve operator for (A - sigma B).
factorization	Optional precomputed factorization handle.

Value

An eigencore_method descriptor selecting shift-invert.

smallest *Target the smallest algebraic values.*

Description

Target the smallest algebraic values.

Usage

```
smallest()
```

Value

An eigencore_target descriptor selecting the smallest algebraic eigenvalues or singular values.

smallest_imaginary *Target the smallest imaginary part.*

Description

Target the smallest imaginary part.

Usage

```
smallest_imaginary()
```

Value

An `eigencore_target` descriptor selecting values by smallest `Im(x)` (ARPACK `SI`).

smallest_magnitude *Target the smallest values by magnitude.*

Description

Target the smallest values by magnitude.

Usage

```
smallest_magnitude()
```

Value

An `eigencore_target` descriptor selecting values with the smallest modulus (ARPACK `SM`).

smallest_real *Target the smallest real part.*

Description

Target the smallest real part.

Usage

```
smallest_real()
```

Value

An `eigencore_target` descriptor selecting values by smallest `Re(x)` (ARPACK `SR`).

`solve.eigencore_eigen_problem` *Solve a planned eigenproblem.*

Description

S3 method that runs the planned solver for an eigenproblem built by `eigen_problem()`. Most users call `eig_partial()`, which constructs the problem and dispatches here; call `solve()` directly when you want to build a problem once and reuse or inspect it. Returns a certified partial eigendecomposition.

Usage

```
## S3 method for class 'eigencore_eigen_problem'
solve(
  a,
  b,
  k,
  method = auto(),
  tol = 1e-08,
  maxit = NULL,
  vectors = TRUE,
  certify = TRUE,
  allow_dense_fallback = c("auto", "never", "always"),
  ...
)
```

Arguments

`a`	Eigencore eigen problem object.
`b`	Unused second argument reserved by the base `solve()` generic.
`k`	Number of eigenpairs to compute.
`method`	Solver method descriptor.
`tol`	Convergence and certification tolerance.
`maxit`	Optional iteration limit.
`vectors`	Whether to compute vectors.
`certify`	Whether to compute certification diagnostics.
`allow_dense_fallback`	Dense fallback policy.
`...`	Reserved for future solver options.

Value

An `eigencore_eigen_result`.

`solve.eigencore_svd_problem` *Solve a planned SVD problem.*

Description

S3 method that runs the planned solver for an SVD problem built by `svd_problem()`. Most users call `svd_partial()`, which constructs the problem and dispatches here; call `solve()` directly when you want to build a problem once and reuse or inspect it. Returns a certified partial singular-value decomposition.

Usage

```
## S3 method for class 'eigencore_svd_problem'
solve(
  a,
  b,
  rank,
  method = auto(),
  tol = 1e-08,
  vectors = c("both", "left", "right", "none"),
  certify = TRUE,
  allow_dense_fallback = c("auto", "never", "always"),
  ...
)
```

Arguments

`a`	Eigencore SVD problem object.
`b`	Unused second argument reserved by the base `solve()` generic.
`rank`	Number of singular values to compute.
`method`	Solver method descriptor.
`tol`	Convergence and certification tolerance.
`vectors`	Which singular-vector sides to compute.
`certify`	Whether to compute certification diagnostics.
`allow_dense_fallback`	Dense fallback policy.
`...`	Reserved for future solver options.

Value

An `eigencore_svd_result`.

svds *RSpectra-compatible SVD shim.*

Description

RSpectra-compatible SVD shim.

Usage

```
svds(A, k, nu = k, nv = k, opts = list(), ...)
```

Arguments

A	Matrix or eigencore operator.
k	Number of singular values to compute.
nu	Number of left singular vectors requested.
nv	Number of right singular vectors requested.
opts	Compatibility options list; currently accepted for API compatibility and not interpreted directly.
...	Additional arguments passed to svd_partial().

Value

A list compatible with RSpectra::svds(), including d, optional u and v, convergence counts, operation counts, certificate diagnostics, and eigencore diagnostics.

Examples

```
set.seed(1)
X <- matrix(rnorm(60), 10, 6)
res <- svds(X, k = 2)
res$d
```

svd_partial *Compute a partial singular-value decomposition.*

Description

Compute a partial singular-value decomposition.

Usage

```
svd_partial(
  A,
  rank,
  target = largest(),
  method = auto(),
  tol = 1e-08,
  vectors = c("both", "left", "right", "none"),
  seed = NULL,
  certify = TRUE,
  allow_dense_fallback = c("auto", "never", "always")
)
```

Arguments

A	Matrix or eigencore operator.
rank	Number of singular values to compute.
target	Eigencore singular-value target descriptor.
method	Solver method descriptor.
tol	Convergence and certification tolerance.
vectors	Which singular-vector sides to compute.
seed	Optional random seed for stochastic solver components.
certify	Whether to compute certification diagnostics.
allow_dense_fallback	Dense fallback policy.

Value

An `eigencore_svd_result` containing singular values, optional left and right singular vectors, certificate diagnostics, method/plan metadata, and convergence diagnostics.

Examples

```
set.seed(1)
X <- matrix(rnorm(60), 10, 6)
fit <- svd_partial(X, rank = 3)
values(fit)
certificate(fit)$passed
```

svd_problem *Define an SVD problem.*

Description

Define an SVD problem.

Usage

```
svd_problem(A, domain = NULL, codomain = NULL, target = largest())
```

Arguments

A	Matrix or operator defining the rectangular linear map.
domain	Optional domain-space descriptor.
codomain	Optional codomain-space descriptor.
target	Eigencore singular-value target descriptor.

Value

An eigencore_svd_problem object containing the operator, domain and codomain descriptors, and singular-value target consumed by plan_solver() and solve().

Examples

```
set.seed(1)
X <- matrix(rnorm(40), 8, 5)
S <- svd_problem(X, target = largest())
fit <- solve(S, rank = 2)
values(fit)
```

symmetric_operator *Mark an operator as symmetric/Hermitian.*

Description

Mark an operator as symmetric/Hermitian.

Usage

```
symmetric_operator(A, validate = TRUE, tol = 1e-10)
```

Arguments

A	Operator-like object.
validate	Whether to check the adjoint identity before marking the operator symmetric.
tol	Relative tolerance for the adjoint check.

Value

An eigencore_operator with Hermitian structure metadata.

---

values *Extract computed values.*

Description

Extract computed values.

Usage

```
values(x, ...)
```

Arguments

x	An eigencore result object.
...	Reserved for future methods.

Value

A numeric or complex vector of computed eigenvalues, singular values, or generalized singular values.

Examples

```
fit <- eig_partial(diag(c(3, 2, 1)), k = 2, target = largest())
values(fit)
```

`vectors` *Extract eigenvectors.*

Description

Extract eigenvectors.

Usage

```
vectors(x, ...)
```

Arguments

`x`	An eigencore eigen result object.
`...`	Reserved for future methods.

Value

A matrix whose columns are computed right eigenvectors, or `NULL` when vectors were not requested or are unavailable.

Examples

```
fit <- eig_partial(diag(c(3, 2, 1)), k = 2, target = largest())
dim(vectors(fit))
```

Index