

Package ‘eigencore’

July 25, 2026

Title Certified Partial Eigenvalue and Singular Value Computation

Version 1.0.1

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 2026-07-24 22:20:02 UTC

Contents

adjoint	3
alpha_beta	3
as_operator	4
auto	5
backward_error	5
both_ends	6
center	6
certificate	7
check_adjoint	8
compose	8
crossprod_operator	9
diagnostics	9
eigen_problem	10
eigs	11
eigs_sym	11
eig_full	12
eig_partial	13
euclidean	15
general	15
generalized_schur	16
generalized_svd	17
golub_kahan	17
hermitian	18
lanczos	18
largest	19
largest_imaginary	19
largest_magnitude	20
largest_real	20
left_vectors	20
linear_operator	21
lobpcg	22
nearest	23
plan_solver	23
randomized	24
residuals	24
right_vectors	25
scale_cols	25
scale_rows	26
shifted_cholesky_preconditioner	26
shifted_diagonal_preconditioner	27
shifted_tridiagonal_preconditioner	27
shift_invert	28
smallest	28
smallest_imaginary	29
smallest_magnitude	29
smallest_real	29

<i>adjoint</i>	3
solve.eigencore_eigen_problem	30
solve.eigencore_svd_problem	31
svds	32
svd_partial	32
svd_problem	34
symmetric_operator	34
values	35
vectors	36
Index	37

adjoint	<i>Return the adjoint operator.</i>
---------	-------------------------------------

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	<i>Extract homogeneous generalized coordinates.</i>
------------	---

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	<i>Automatic solver choice.</i>
------	---------------------------------

Description

Automatic solver choice.

Usage

auto()

Value

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

backward_error	<i>Extract backward-error diagnostics.</i>
----------------	--

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	<i>Target both algebraic ends.</i>
-----------	------------------------------------

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	<i>Center an operator by rows or columns.</i>
--------	---

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	<i>Extract a result certificate.</i>
-------------	--------------------------------------

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	<i>Check an operator adjoint identity.</i>
---------------	--

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	<i>Compose two operators.</i>
---------	-------------------------------

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 $A \circ B$.

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	<i>Define an eigenproblem.</i>
---------------	--------------------------------

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	<i>RSpectra-compatible eigen shim.</i>
------	--

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	<i>RSpectra-compatible symmetric eigen shim.</i>
----------	--

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 <code>solve.eigencore_eigen_problem()</code> .

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 <code>general()</code> 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	<i>Compute a partial eigendecomposition.</i>
-------------	--

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

<code>A</code>	Matrix or eigencore operator.
<code>k</code>	Number of eigenpairs to compute.
<code>target</code>	Eigencore eigenvalue target descriptor.
<code>B</code>	Optional metric matrix or operator for generalized problems.
<code>method</code>	Solver method descriptor.
<code>tol</code>	Convergence and certification tolerance.
<code>maxit</code>	Optional iteration limit.
<code>vectors</code>	Whether to compute vectors.
<code>seed</code>	Optional random seed for stochastic solver components.
<code>certify</code>	Whether to compute certification diagnostics.
<code>allow_dense_fallback</code>	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	<i>Euclidean vector space descriptor.</i>
-----------	---

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 $\mathbb{R}^{\text{dim}}$ or $\mathbb{C}^{\text{dim}}$.

general	<i>General operator structure descriptor.</i>
---------	---

Description

General operator structure descriptor.

Usage

```
general()
```

Value

An eigencore_structure descriptor for general operators.

generalized_schur	<i>Compute a dense generalized Schur decomposition</i>
-------------------	--

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

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

Description

Hermitian/symmetric operator structure descriptor.

Usage

```
hermitian()
```

Value

An eigencore_structure descriptor marking an operator as Hermitian / symmetric.

lanczos	<i>Hermitian Lanczos method descriptor.</i>
---------	---

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	<i>Target the largest algebraic values.</i>
---------	---

Description

Target the largest algebraic values.

Usage

```
largest()
```

Value

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

largest_imaginary	<i>Target the largest imaginary part.</i>
-------------------	---

Description

Target the largest imaginary part.

Usage

```
largest_imaginary()
```

Value

An eigencore_target descriptor selecting values by largest $\text{Im}(x)$ (ARPACK LI).

largest_magnitude	<i>Target the largest values by magnitude.</i>
-------------------	--

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	<i>Target the largest real part.</i>
--------------	--------------------------------------

Description

Target the largest real part.

Usage

```
largest_real()
```

Value

An eigencore_target descriptor selecting values by largest $\text{Re}(x)$ (ARPACK LR).

left_vectors	<i>Extract left singular vectors or left eigenvectors.</i>
--------------	--

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.

<code>linear_operator</code>	<i>Create a block-native linear operator.</i>
------------------------------	---

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	<i>Target values nearest a shift.</i>
---------	---------------------------------------

Description

Target values nearest a shift.

Usage

```
nearest(sigma)
```

Arguments

sigma Numeric shift used to rank values by distance $|x - \text{sigma}|$.

Value

An eigencore_target descriptor for nearest-to-sigma selection.

plan_solver	<i>Plan a solver for a problem.</i>
-------------	-------------------------------------

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	<i>Randomized SVD method descriptor.</i>
------------	--

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	<i>Extract residual diagnostics.</i>
-----------	--------------------------------------

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	<i>Extract right singular vectors.</i>
---------------	--

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	<i>Scale operator columns.</i>
------------	--------------------------------

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	<i>Shift-invert method descriptor.</i>
--------------	--

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	<i>Target the smallest algebraic values.</i>
----------	--

Description

Target the smallest algebraic values.

Usage

```
smallest()
```

Value

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

smallest_imaginary	<i>Target the smallest imaginary part.</i>
--------------------	--

Description

Target the smallest imaginary part.

Usage

smallest_imaginary()

Value

An eigencore_target descriptor selecting values by smallest $\text{Im}(x)$ (ARPACK SI).

smallest_magnitude	<i>Target the smallest values by magnitude.</i>
--------------------	---

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	<i>Target the smallest real part.</i>
---------------	---------------------------------------

Description

Target the smallest real part.

Usage

smallest_real()

Value

An eigencore_target descriptor selecting values by smallest $\text{Re}(x)$ (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

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

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

Value

An `eigencore_svd_result`.

svds	<i>RSpectra-compatible SVD shim.</i>
------	--------------------------------------

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	<i>Compute a partial singular-value decomposition.</i>
-------------	--

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

<code>A</code>	Matrix or eigencore operator.
<code>rank</code>	Number of singular values to compute.
<code>target</code>	Eigencore singular-value target descriptor.
<code>method</code>	Solver method descriptor.
<code>tol</code>	Convergence and certification tolerance.
<code>vectors</code>	Which singular-vector sides to compute.
<code>seed</code>	Optional random seed for stochastic solver components.
<code>certify</code>	Whether to compute certification diagnostics.
<code>allow_dense_fallback</code>	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	<i>Define an SVD problem.</i>
-------------	-------------------------------

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	<i>Mark an operator as symmetric/Hermitian.</i>
--------------------	---

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	<i>Extract computed values.</i>
--------	---------------------------------

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	<i>Extract eigenvectors.</i>
---------	------------------------------

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

adjoint, [3](#)
alpha_beta, [3](#)
as_operator, [4](#)
auto, [5](#)

backward_error, [5](#)
both_ends, [6](#)

center, [6](#)
certificate, [7](#)
check_adjoint, [8](#)
compose, [8](#)
crossprod_operator, [9](#)

diagnostics, [9](#)

eig_full, [12](#)
eig_partial, [13](#)
eig_partial(), [11](#), [30](#)
eigen_problem, [10](#)
eigen_problem(), [30](#)
eigs, [11](#)
eigs_sym, [11](#)
euclidean, [15](#)

general, [15](#)
general(), [12](#), [21](#)
generalized_schur, [16](#)
generalized_svd, [17](#)
golub_kahan, [17](#)

hermitian, [18](#)
hermitian(), [21](#)

lanczos, [18](#)
largest, [19](#)
largest_imaginary, [19](#)
largest_magnitude, [20](#)
largest_real, [20](#)
left_vectors, [20](#)
linear_operator, [21](#)

lobpcg, [22](#)

nearest, [23](#)

plan_solver, [23](#)
plan_solver(), [10](#), [34](#)

randomized, [24](#)
residuals, [24](#)
right_vectors, [25](#)

scale_cols, [25](#)
scale_rows, [26](#)
shift_invert, [28](#)
shift_invert(), [10](#)
shifted_cholesky_preconditioner, [26](#)
shifted_diagonal_preconditioner, [27](#)
shifted_tridiagonal_preconditioner, [27](#)
smallest, [28](#)
smallest_imaginary, [29](#)
smallest_magnitude, [29](#)
smallest_real, [29](#)
solve(), [10](#), [30](#), [31](#), [34](#)
solve.eigencore_eigen_problem, [30](#)
solve.eigencore_eigen_problem(), [12](#)
solve.eigencore_svd_problem, [31](#)
stats::residuals(), [24](#)
svd_partial, [32](#)
svd_partial(), [31](#), [32](#)
svd_problem, [34](#)
svd_problem(), [31](#)
svds, [32](#)
symmetric_operator, [34](#)

values, [35](#)
vectors, [36](#)