Package: amatrix (via r-universe)

July 2, 2026

Type Package

Title Backend-Agnostic Matrix Extensions

Version 0.1.0

Description Matrix-compatible S4 classes with backend-dispatch hooks for accelerated execution and predictable CPU fallback.

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

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

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.3)

Imports methods, Matrix

Suggests testthat (>= 3.0.0), callr, ggplot2, irlba, knitr, pkgload, rmarkdown, rprojroot, withr, amatrix.mlx, amatrix.arrayfire, amatrix.opencl, amatrix.metal

Additional_repositories https://bbuchsbaum.r-universe.dev

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 7.3.3

Config/Needs/website albersdown

Repository https://bbuchsbaum.r-universe.dev

Date/Publication 2026-07-02 13:18:49 UTC

RemoteUrl https://github.com/bbuchsbaum/amatrix

RemoteRef HEAD

RemoteSha aa1a39a22def2841186cc195dd1b4c7e0d5ff8e2

Contents

`%*%,adgCMatrix,ANY-method`

Matrix multiplication for adgCMatrix

Description

Dispatches `%*%` through the amatrix backend for sparse `adgCMatrix` objects on the left-hand side, preserving GPU residency metadata across the operation.

Usage

```
## S4 method for signature 'adgCMatrix,ANY'
x %*% y

## S4 method for signature 'adgCMatrix,matrix'
x %*% y

## S4 method for signature 'adgCMatrix,Matrix'
x %*% y

## S4 method for signature 'adgCMatrix,dgeMatrix'
x %*% y

## S4 method for signature 'adgCMatrix,dgCMatrix'
x %*% y

## S4 method for signature 'adgCMatrix,adgeMatrix'
x %*% y

## S4 method for signature 'adgCMatrix,adgCMatrix'
x %*% y
```

Arguments

x	An `adgCMatrix`.
y	A matrix-like object: `matrix`, `Matrix`, `adgeMatrix`, `adgCMatrix`, or `ANY`.

Value

An `adgeMatrix` or `adgCMatrix` containing the product, with backend metadata inherited from `x`.

Examples

```
sp <- as(matrix(c(1, 0, 0, 2), 2, 2), "dgCMatrix")
A  <- adgCMatrix(sp)
B  <- matrix(1:4, 2, 2)
A %*% B
```

addmm *Scaled matrix multiply with optional bias: alpha*(A%*%B) + beta*C*

Description

Scaled matrix multiply with optional bias: alpha*(A%*%B) + beta*C

Usage

```
addmm(A, B, C = NULL, alpha = 1, beta = 1)
```

Arguments

A	n×p adgeMatrix or plain matrix.
B	p×k numeric matrix.
C	n×k numeric matrix or NULL (treated as zeros).
alpha	Scalar multiplier for A%*%B (default 1).
beta	Scalar multiplier for C (default 1).

Value

adgeMatrix if A is resident, otherwise plain matrix.

adgCMatrix *Create a backend-aware sparse matrix*

Description

Converts a sparse or dense matrix to an adgCMatrix with the specified backend, policy, and precision. This is the primary user-facing constructor for sparse amatrix objects.

Usage

```
adgCMatrix(
  x,
  mode = NULL,
  backend = NULL,
  preferred_backend = NULL,
  policy = NULL,
  precision = NULL
)
```

Arguments

x	A dgCMatrix, other sparseMatrix, or base R matrix.
mode	Single string shortcut passed to .amatrix_resolve_mode(). Pass NULL to use the individual arguments. mode = "fast" prefers an available fast-capable accelerator automatically, with CPU fallback.
backend	Alias for preferred_backend; ignored when preferred_backend is non-NULL.
preferred_backend	Single string naming the preferred compute backend.
policy	Single string; one of "auto", "cpu", "mlx", "metal", "arrayfire", "opencl".
precision	Single string; "strict" or "fast".

Value

An adgCMatrix with the data from x and the requested backend metadata.

Examples

```
m <- matrix(c(1, 0, 0, 2), nrow = 2)
S <- adgCMatrix(m)
S
```

---

adgCMatrix-class *Sparse column-compressed matrix with backend-dispatch metadata*

Description

adgCMatrix extends both aMatrix and Matrix::dgCMatrix, adding backend-dispatch slots to a compressed-column sparse double-precision matrix.

See Also

adgCMatrix for the user-facing constructor, adgeMatrix-class for the dense counterpart

adgeMatrix *Create a backend-aware dense matrix*

Description

Converts a base R matrix or `Matrix::dgeMatrix` to an `adgeMatrix` with the specified backend, policy, and precision. This is the primary user-facing constructor for dense amatrix objects.

Usage

```
adgeMatrix(
  x,
  mode = NULL,
  backend = NULL,
  preferred_backend = NULL,
  policy = NULL,
  precision = NULL
)
```

Arguments

x	A base R `matrix`, `dgeMatrix`, or any `denseMatrix` coercible to `dgeMatrix`.
mode	Single string shortcut accepted by `.amatrix_resolve_mode()`; used to set `backend`, `policy`, and `precision` together. Pass `NULL` to use the individual arguments instead. In particular, `mode = "fast"` requests reduced precision and prefers an available fast-capable accelerator automatically, with CPU fallback when none is available.
backend	Alias for `preferred_backend`; ignored when `preferred_backend` is non-`NULL`.
preferred_backend	Single string naming the preferred compute backend, e.g. `"cpu"`, `"mlx"`, or `"metal"`.
policy	Single string; one of `"auto"`, `"cpu"`, `"mlx"`, `"metal"`, `"arrayfire"`, `"opencl"`.
precision	Single string; `"strict"` for full double-precision accuracy or `"fast"` to allow reduced precision on GPU backends.

Value

An `adgeMatrix` with the data from `x` and the requested backend metadata.

Examples

```
m <- matrix(1:6, nrow = 2)
A <- adgeMatrix(m)
A
```

adgeMatrix-class *Dense general matrix with backend-dispatch metadata*

Description

adgeMatrix extends both aMatrix and Matrix::dgeMatrix, adding backend-dispatch slots to a column-major dense double-precision matrix. All arithmetic generics dispatch through the amatrix backend system rather than directly to BLAS.

See Also

adgeMatrix for the user-facing constructor, adgCMatrix for the sparse counterpart

adlgCMatrix-class *Sparse logical matrix with backend-dispatch metadata*

Description

adlgCMatrix extends both aMatrix and Matrix::lgCMatrix, adding backend-dispatch slots to a compressed-column sparse logical matrix.

adlgeMatrix-class *Dense logical matrix with backend-dispatch metadata*

Description

adlgeMatrix extends both aMatrix and Matrix::lgeMatrix, adding backend-dispatch slots to a column-major dense logical matrix.

am_argreduce *Row and column argmax/argmin*

Description

Return the index of the maximum or minimum value in each row or column.

Usage

```
am_rowargmax(x)

am_rowargmin(x)

am_colargmax(x)

am_colargmin(x)
```

Arguments

x	A numeric matrix or adgeMatrix.

Value

An integer vector of indices.

am_ewise_inplace *In-place elementwise operation on a resident handle*

Description

Applies an elementwise arithmetic operation between the handle's resident matrix and either a scalar or another resident handle, replacing the handle's device buffer with the result.

Usage

```
am_ewise_inplace(h, rhs, op)
```

Arguments

h	A resident_handle.
rhs	A length-1 numeric scalar, or a resident_handle with identical dimensions to h.
op	Character string. Arithmetic operator: "+", "-", "*", or "/".

Value

h, invisibly. The handle is modified in place.

See Also

am_sweep_inplace, resident_handle

Examples

```
# requires a backend with residency support (e.g. MLX, OpenCL)
```

---

am_qr *QR decomposition of an amatrix object*

Description

Computes the QR decomposition of a matrix or aMatrix, routing to a backend-specific implementation when available.

Usage

```
am_qr(x, ...)
```

Arguments

x	A matrix or aMatrix object.
...	Additional arguments passed to the underlying QR routine.

Value

An object of class amDenseQR (or a wrapped sparse QR for adgCMatrix input) containing the factorisation components.

Examples

```
m <- adgeMatrix(matrix(rnorm(12), 4, 3))
qr_obj <- am_qr(m)
qr.R(qr_obj)
```

am_scatter_mean *Scatter mean by group labels*

Description

Compute the mean of rows of `x` grouped by integer `labels`.

Usage

```
am_scatter_mean(x, labels, K)
```

Arguments

`x`	A numeric matrix or `adgeMatrix`.
`labels`	Integer vector of group labels (1-based).
`K`	Number of groups.

Value

A `K`-by-`ncol(x)` matrix of group means.

am_sweep *Backend-dispatched sweep*

Description

Apply a function to each row or column of a matrix, dispatching to the preferred GPU backend when available.

Usage

```
am_sweep(x, MARGIN, STATS, FUN = "+")
```

Arguments

`x`	A numeric matrix or `adgeMatrix`.
`MARGIN`	1 for rows, 2 for columns.
`STATS`	Numeric vector of statistics to apply.
`FUN`	Operation: `"+"`, `"-"`, `"*"`, or `"/"`.

Value

A matrix of the same dimensions as `x`.

See Also

`am_sweep_inplace`

am_sweep_inplace *In-place broadcast sweep on a resident handle*

Description

Applies a row-wise or column-wise arithmetic operation between the resident matrix and a statistics vector, mutating the handle in place. Equivalent to `sweep(as.matrix(h), MARGIN, STATS, FUN)` but avoids downloading the matrix to host.

Usage

```
am_sweep_inplace(h, MARGIN, STATS, FUN = "+")
```

Arguments

`h`	A `resident_handle`.
`MARGIN`	Integer. `1L` to sweep across rows (one value per row), `2L` to sweep across columns.
`STATS`	Numeric vector of length equal to the number of rows or columns selected by `MARGIN`.
`FUN`	Character string. Arithmetic operator to apply: `"+"`, `"-"`, `"*"`, or `"/"`. Default `"+"`.

Value

`h`, invisibly. The handle is modified in place; the underlying device buffer is replaced with the sweep result.

See Also

`resident_handle`, `am_ewise_inplace`

Examples

```
# requires a backend with residency support (e.g. MLX, OpenCL)
```

aMatrix-class *Virtual base class for backend-aware matrices*

Description

aMatrix is the abstract base from which all concrete amatrix classes inherit. It carries backend-dispatch metadata that controls which compute backend (CPU, GPU, etc.) is used for operations on the matrix.

Slots

preferred_backend Single string naming the preferred compute backend; one of "cpu", "mlx", "metal", or "arrayfire".

policy Single string controlling dispatch policy; one of "auto", "cpu", "mlx", "metal", or "arrayfire".

precision Single string; either "strict" (double precision, exact results) or "fast" (backend may use lower precision).

object_id Non-empty string uniquely identifying this object within the session; used for caching and residency tracking.

src_id String recording the object_id of the object this was derived from, or "" for originals.

finalizer_env Environment used to manage GPU-resident memory and deferred host-copy state.

amatrix_backend_capabilities *Query the capabilities of a registered backend*

Description

Returns the unique capability strings advertised by the named backend, as reported by its capabilities() function.

Usage

```
amatrix_backend_capabilities(name)
```

Arguments

name	Character string. Name of a registered backend.

Value

Character vector of capability identifiers (e.g. "matmul", "svd").

See Also

amatrix_backend_features, amatrix_backend_status

Examples

```
amatrix_backend_capabilities("cpu")
```

---

amatrix_backend_features

Query the features of a registered backend

Description

Returns the unique feature strings advertised by the named backend, as reported by its `features()` function. Features describe optional capabilities such as sparse residency or deferred execution.

Usage

```
amatrix_backend_features(name)
```

Arguments

name	Character string. Name of a registered backend.

Value

Character vector of feature identifiers.

See Also

amatrix_backend_capabilities, amatrix_backend_status

Examples

```
amatrix_backend_features("cpu")
```

`amatrix_backend_health_probe` *Run a canary health probe against a registered backend*

Description

Executes a small matmul round-trip against the named backend and compares the result to the base R reference. On success the backend is marked `healthy`; on failure it is marked `unhealthy:<reason>`. Subsequent calls to `amatrix_backend_status()` reflect the recorded health.

Usage

```
amatrix_backend_health_probe(name, tol = NULL)
```

Arguments

`name`	Character string. Name of a registered backend.
`tol`	Numeric. Residual tolerance for the probe, default `1e-8` (float64) or `1e-4` (if the backend only supports fast precision).

Details

The probe is intentionally tiny (10x10 double-precision matmul) so it completes in milliseconds even on cold GPU. It is not a benchmark; it is a liveness check.

Value

Invisibly, the health record as a list with elements `status`, `reason`, `timestamp`.

See Also

`amatrix_backend_status`, `amatrix_fallback_log`

Examples

```
amatrix_backend_health_probe("cpu")
```

amatrix_backend_matrix *Tabulate dispatch plans across multiple operations*

Description

Runs amatrix_backend_plan for each requested operation and returns the results as a single data.frame, one row per operation. Useful for inspecting which backend will be used across an entire workload.

Usage

```
amatrix_backend_matrix(
  x,
 ops = c("matmul", "crossprod", "tcrossprod", "ewise", "rowSums", "colSums", "solve",
    "chol", "qr", "svd", "eigen", "diag"),
  y_map = list()
)
```

Arguments

x	An aMatrix object.
ops	Character vector of operation names. Defaults to the twelve standard operations.
y_map	Named list mapping operation names to right-hand-side objects. Use to supply a y argument for binary operations such as "matmul".

Value

A data.frame with one row per operation and columns:

op Character. Operation name.

precision Character. Precision mode.

pinned_backend Character. Backend to which x is GPU-resident, or NA.

preferred Character. Preference order string.

chosen Character. Selected backend.

chosen_path Character. "resident" or "cold".

resident_reuse Logical. Whether the resident path is active.

cpu_fallback Logical. Whether CPU was chosen despite not being first preference.

candidate_summary Character. Compact flag string for all candidates.

See Also

amatrix_backend_plan, amatrix_execution_info

Examples

```
m <- adgeMatrix(matrix(1:6, 2, 3))
amatrix_backend_matrix(m, ops = c("matmul", "crossprod"))
```

`amatrix_backend_names` *List names of all registered backends*

Description

Returns the names of every backend currently in the session registry. When optional backends are enabled (the default), this also attempts to auto-register any installed optional backend packages before returning the list.

Usage

```
amatrix_backend_names()
```

Value

Character vector of registered backend names, sorted alphabetically. Always includes at least `"cpu"`.

See Also

`amatrix_backend_status`, `amatrix_register_backend`

Examples

```
amatrix_backend_names()
```

`amatrix_backend_plan` *Compute the dispatch plan for a single operation*

Description

Evaluates each candidate backend in preference order and returns a structured plan describing which backend was chosen and why each candidate was accepted or rejected. The plan respects GPU residency, precision compatibility, and calibration thresholds.

Usage

```
amatrix_backend_plan(x, op, y = NULL)
```

Arguments

x	An aMatrix object.
op	Character string naming the operation, e.g. "matmul", "crossprod", "svd".
y	Right-hand-side aMatrix or NULL. Used for binary operations such as "matmul" to check compatibility and calibration workload.

Value

A named list with elements:

op Character. The requested operation.

pinned_backend Character or NULL. Backend to which x is currently GPU-resident.

preferred Character vector. Backends evaluated in order.

requested_precision Character. Precision mode of x.

chosen Character. Name of the chosen backend.

chosen_path Character. Either "resident" or "cold".

candidates List of per-candidate evaluation records, each a named list with logical flags for registered, available, precision_compatible, supported_cold, supported_resident, calibration_ok, supported, and chosen.

See Also

amatrix_backend_matrix, amatrix_explain, amatrix_execution_info

Examples

```
m <- adgeMatrix(matrix(1:6, 2, 3))
amatrix_backend_plan(m, "matmul")
```

---

amatrix_backend_precision_modes

Query the precision modes supported by a registered backend

Description

Returns the precision mode strings advertised by the named backend. Valid values are "strict" (double precision) and "fast" (single/mixed precision).

Usage

```
amatrix_backend_precision_modes(name)
```

Arguments

name	Character string. Name of a registered backend.

Value

Character vector of precision mode identifiers, a subset of `c("strict", "fast")`.

See Also

amatrix_backend_capabilities, amatrix_backend_status

Examples

```
amatrix_backend_precision_modes("cpu")
```

`amatrix_backend_status` *Summarise the status of registered backends*

Description

Returns a data.frame with one row per backend describing its availability, supported precision modes, features, capabilities, and whether it supports GPU residency.

Usage

```
amatrix_backend_status(names = NULL)
```

Arguments

`names`	Character vector of backend names to query. When `NULL` (default) all registered backends are included, with optional backends auto-registered first if possible.

Value

A data.frame with columns:

name Character. Backend identifier.

available Logical. Whether the backend reports itself as available on this machine.

precision_modes Character. Comma-separated precision modes (`"strict"`, `"fast"`).

features Character. Comma-separated feature strings.

residency_capable Logical. Whether the backend supports GPU-resident matrix storage.

capabilities Character. Comma-separated operation capability strings.

See Also

amatrix_backend_names, amatrix_register_backend

Examples

```
amatrix_backend_status()
amatrix_backend_status("cpu")
```

`amatrix_benchmark_report` *Report amatrix benchmark status across ops and backends*

Description

Reads a machine-local benchmark baseline CSV (a table of recorded per-op cold and warm timings), if one is present, together with the cached calibration in the user cache directory, and returns a structured data.frame surfacing per-op cold vs warm timings and the currently-calibrated dispatch thresholds.

Usage

```
amatrix_benchmark_report(baseline_path = file.path("tools", "baseline.csv"))
```

Arguments

`baseline_path`	Path to a benchmark baseline CSV of recorded per-op timings. Defaults to a `baseline.csv` looked up relative to the current working directory. Pass `NULL` to skip baseline reading entirely and return only calibration data.

Details

This is the user-facing honesty surface for Track 4's speed contract: users can see (a) which backends are calibrated on their machine, (b) cold-start vs warm-run ratios per op, and (c) where the dispatcher will currently route.

Value

A list with two elements:

baseline data.frame with columns `op`, `size`, `backend`, `cold_ms`, `warm_ms`, `warm_vs_cold_ratio`, `speedup_vs_cpu`. Rows with missing cold OR warm data use `NA` for the missing variant. Empty when the baseline file is absent.

calibration data.frame with columns `backend`, `op`, `threshold_elements`, `gpu_wins`. Rows come from the cached calibration; empty when no calibration is available.

See Also

`amatrix_calibrate`, `amatrix_calibration_info`

Examples

```
## Not run:
rep <- amatrix_benchmark_report()
head(rep$baseline)
head(rep$calibration)

## End(Not run)
```

`amatrix_bind_resident` *Bind an amatrix object to resident backend storage*

Description

Upload a dense or sparse matrix to a residency-capable backend and return the corresponding `aMatrix` object with a live resident binding. This is primarily useful for repeated GPU work where paying the upload cost once is preferable to relying on cold-path dispatch.

Usage

```
amatrix_bind_resident(x, backend = NULL, op = NULL, y = NULL)
```

Arguments

`x`	An `adgeMatrix`, `adgCMatrix`, base matrix, or sparse Matrix object.
`backend`	Backend name, `"auto"`, or `NULL`. When left `NULL` or set to `"auto"`, `amatrix` picks the first residency-capable accelerator backend that supports the requested resident operation.
`op`	Optional operation name such as `"matmul"` used when selecting an automatic resident backend.
`y`	Optional rhs object used when checking resident-op support for automatic backend selection.

Value

An `adgeMatrix` or `adgCMatrix` with a live resident binding on `backend`. When no suitable accelerator backend is available in automatic mode, returns `x` unchanged.

amatrix_cache_max_size *Get or set the model cache maximum size*

Description

`amatrix_cache_max_size` returns the current limit. `amatrix_set_cache_max_size` changes the limit and immediately evicts the least-recently-used entries if the cache exceeds the new bound. When `max_size` is `Inf` (the default) the cache grows without bound.

Usage

```
amatrix_cache_max_size()

amatrix_set_cache_max_size(max_size)
```

Arguments

`max_size`	Positive numeric scalar or `Inf`; the maximum number of factorizations to retain in the model cache.

Value

`amatrix_cache_max_size` returns a length-1 numeric giving the current limit. `amatrix_set_cache_max_size` returns the new limit invisibly.

Examples

```
old <- amatrix_cache_max_size()
amatrix_set_cache_max_size(10)
amatrix_cache_max_size()
amatrix_set_cache_max_size(old)
```

amatrix_calibrate *Calibrate GPU dispatch thresholds for this machine*

Description

Runs micro-benchmarks for each (op, backend, size) combination and derives the minimum matrix element count at which each GPU backend reliably outperforms CPU. Results are stored in the current session and optionally persisted to disk for reuse in future sessions.

Usage

```
amatrix_calibrate(
  backend = NULL,
 ops = c("gemm", "gemv", "crossprod", "rowSums", "colSums", "qr", "chol", "solve",
    "svd"),
 sizes = list(c(64L, 32L), c(128L, 64L), c(256L, 128L), c(512L, 256L), c(1024L, 512L)),
  sparse_densities = c(0.01, 0.05, 0.2),
  n_reps = 10L,
  margin = 0.1,
  persist = TRUE,
  quiet = FALSE
)
```

Arguments

backend	Character vector of backend names to benchmark. Defaults to all registered non-CPU backends that report `available = TRUE`.
ops	Character vector of operations to benchmark. Supported values: `"matmul"` (alias for `"gemm"`), `"gemm"`, `"gemv"`, `"spmv"`, `"spmm"`, `"crossprod"`, `"rowSums"`, `"colSums"`, `"qr"`, `"chol"`, `"solve"`, `"svd"`.
sizes	List of integer vectors of length 2 giving `c(nrow, ncol)` test matrix dimensions.
sparse_densities	Numeric vector of target fill densities used when benchmarking `"spmv"` and `"spmm"`.
n_reps	Positive integer. Number of timed repetitions per benchmark cell, after 2 warm-up repetitions.
margin	Non-negative numeric less than 1. Fraction by which GPU median time must beat CPU to count as a GPU win (default `0.10` means GPU must be at least 10% faster).
persist	Logical. If `TRUE` (default), save calibration to the user cache directory so future sessions load it automatically.
quiet	Logical. Suppress progress messages.

Value

Invisibly, a list with elements `version`, `calibrated_at` (POSIXct), `thresholds` (nested list keyed by backend then op), and `results` (data.frame of all benchmark measurements).

See Also

`amatrix_calibration_info`, `amatrix_backend_plan`

amatrix_calibration_info

Retrieve the current calibration state

Description

Returns the calibration data stored in the current session. If no calibration has been run yet, the function attempts to load a previously persisted calibration from the user cache directory. Returns `NULL` when no calibration is available.

Usage

```
amatrix_calibration_info()
```

Value

A list as returned by `amatrix_calibrate`, or `NULL` if no calibration data exists for this session.

See Also

`amatrix_calibrate`

Examples

```
cal <- amatrix_calibration_info()
is.null(cal) # TRUE when no calibration has been run
```

amatrix_compile_product

Compile a reusable matrix-product plan

Description

Prepares a fixed left operand for repeated products, choosing and binding a resident accelerator backend when beneficial. The returned object is a callable function, so repeated right-hand sides can be applied without rethinking backend selection each time.

Usage

```
amatrix_compile_product(
  x,
  op = c("matmul", "crossprod", "tcrossprod"),
  backend = "auto",
  precision = amatrix_default_precision(),
  policy = amatrix_default_policy()
)
```

Arguments

x	Fixed left operand.
op	Product primitive: "matmul", "crossprod", or "tcrossprod".
backend	Backend name or "auto".
precision	Precision to use when wrapping base matrices.
policy	Policy to use when wrapping base matrices.

Value

A callable am_product_plan. Calling the plan with materialize = "matrix" returns a base matrix directly, which is useful for internal algorithms that immediately need host matrix data.

---

amatrix_default_policy

Get the session-level default dispatch policy

Description

Returns the dispatch policy used when an aMatrix object does not specify its own policy. The policy controls which backend is preferred for operations on new matrices.

Usage

```
amatrix_default_policy()
```

Value

Character string, one of "auto", "cpu", "mlx", "metal", "arrayfire", or "opencl".

See Also

amatrix_set_default_policy, amatrix_default_precision

Examples

```
amatrix_default_policy()
```

amatrix_default_precision *Get the session-level default precision mode*

Description

Returns the precision mode used when constructing new `aMatrix` objects that do not specify their own precision.

Usage

```
amatrix_default_precision()
```

Value

Character string, either `"strict"` (double precision) or `"fast"` (single/mixed precision).

See Also

`amatrix_set_default_precision`, `amatrix_default_policy`

Examples

```
amatrix_default_precision()
```

amatrix_dispatch_op *Low-level backend dispatch for a single operation*

Description

Resolves the best available backend for `op` on `x`, attempts the GPU-resident path when applicable, and falls back to the cold path (materializing `x` to host) if needed. If the chosen backend does not implement `method`, the `fallback` function is called instead.

Usage

```
amatrix_dispatch_op(x, op, method = op, y = NULL, args = list(), fallback)
```

Arguments

x	An aMatrix object.
op	Character string. Operation key used for backend selection (e.g. "matmul", "svd").
method	Character string. Name of the backend list element to call. Defaults to op; override when the backend method name differs from the operation key.
y	Right-hand-side aMatrix or NULL. Passed to the backend method and used during backend selection.
args	Named list of additional arguments forwarded to the backend method on the cold path.
fallback	Zero-argument function called when the chosen backend does not implement method.

Value

The result of the backend method, or the result of fallback() if the method is unavailable.

See Also

amatrix_backend_plan, amatrix_materialize_host

---

amatrix_execution_info

Collect full dispatch information for an aMatrix object

Description

Returns a snapshot of the dispatch state for an aMatrix, including residency, preferred backend, policy, precision, and the per-operation dispatch matrix for a set of operations.

Usage

```
amatrix_execution_info(
  x,
  ops = c("matmul", "crossprod", "tcrossprod", "ewise", "rowSums", "colSums"),
  y_map = list()
)
```

Arguments

x	An aMatrix object.
ops	Character vector of operation names to include in the dispatch matrix. Default covers the six core operations.
y_map	Named list mapping operation names to right-hand-side objects used when planning binary operations such as "matmul".

Value

A named list with elements:

object_id Character. Internal object identifier.

preferred_backend Character. Preferred backend slot value.

pinned_backend Character or NULL. Backend to which the object is currently GPU-resident.

policy Character. Dispatch policy slot value.

precision Character. Precision mode ("strict" or "fast").

residency data.frame. Output of amatrix_residency_info.

plans data.frame. Output of amatrix_backend_matrix.

See Also

amatrix_backend_plan, amatrix_backend_matrix, amatrix_explain

---

amatrix_explain *Explain dispatch decisions for an aMatrix operation*

Description

Prints a human-readable diagnostic showing which backend was chosen for op on x, the accept/reject status of every candidate backend, and actionable suggestions for improving performance.

Usage

```
amatrix_explain(x, op, y = NULL)
```

Arguments

x	An aMatrix object.
op	Character string. Operation to explain, e.g. "matmul", "crossprod", "svd".
y	Right-hand-side aMatrix or NULL. Supply for binary operations to include workload-specific advice.

Value

Invisibly, the dispatch plan list from amatrix_backend_plan. Called primarily for its printed output.

See Also

amatrix_backend_plan, amatrix_backend_matrix, amatrix_execution_info

Examples

```
m <- adgeMatrix(matrix(1:12, 3, 4))
amatrix_explain(m, "matmul")
```

amatrix_fallback_log *Return the amatrix backend fallback log*

Description

The fallback log records every runtime fall-through from a preferred backend to the CPU reference path. A non-empty log after a clean conformance run is a stop-ship condition: it means a backend claimed support for an op it cannot actually execute, so the result silently came from a different backend than the one that was requested.

Usage

```
amatrix_fallback_log()
```

Value

A data.frame with columns `timestamp`, `op`, `from_backend`, `to_backend`, `reason`. Zero rows means no fallbacks have been recorded.

See Also

`amatrix_fallback_log_reset`, `amatrix_backend_health_probe`

Examples

```
amatrix_fallback_log()
amatrix_fallback_log_reset()
```

amatrix_fallback_log_reset *Clear the amatrix backend fallback log*

Description

Resets the fallback log to empty. Typically called at the start of a test block to isolate the assertion that the log is empty after a clean run.

Usage

```
amatrix_fallback_log_reset()
```

Value

Invisibly, `NULL`.

See Also

amatrix_fallback_log

---

amatrix_gc *Free stale GPU residency entries and optionally flush the model cache*

Description

Scans the residency registry and removes entries whose backend no longer reports the associated device buffer as present. Such stale entries arise when a backend is unloaded or the device is reset between sessions. Optionally flushes all cached matrix factors (QR, Cholesky, SVD) from the session model cache.

Usage

```
amatrix_gc(cache = FALSE)
```

Arguments

cache	Logical. If TRUE, also remove all model-cache entries. Default FALSE.

Value

Invisibly, a list with two integer elements:

dead_entries Number of stale residency slots removed.

cache_entries_cleared Number of model-cache entries flushed (0 when cache = FALSE).

See Also

amatrix_memory_stats, amatrix_residency_info

Examples

```
amatrix_gc()
amatrix_gc(cache = TRUE)
```

amatrix_gpu_status *GPU backend status: why am I (not) on the GPU?*

Description

One row per known GPU backend with the state of every gate between "installed" and "computing on the GPU": package installed, backend registered, device available, health, and the registry's recorded reason when something is off.

Usage

```
amatrix_gpu_status()
```

Value

A data frame with columns `backend`, `package`, `installed`, `registered`, `available`, `health`, and `reason`.

See Also

`amatrix_use_gpu`, `amatrix_backend_status`, `amatrix_explain`

Examples

```
amatrix_gpu_status()
```

amatrix_materialize_host *Force materialization of an aMatrix to a host Matrix object*

Description

Downloads any GPU-resident data and returns a standard `Matrix`-package object on the host. For `adgeMatrix` inputs the result is a `dgeMatrix`; for `adgCMatrix` inputs the result is a `dgCMatrix`; for `aTransposeView` the transposed dense host matrix is returned. Host-only objects are returned unchanged.

Usage

```
amatrix_materialize_host(x)
```

Arguments

`x`	An `aMatrix` object (`adgeMatrix`, `adgCMatrix`, or `aTransposeView`).

Value

A dgeMatrix, dgCMatrix, or the original object if no materialization is needed.

See Also

amatrix_residency_info, amatrix_gc

Examples

```
m <- adgeMatrix(matrix(1:6, 2, 3))
host <- amatrix_materialize_host(m)
class(host)
```

---

amatrix_memory_stats *Report GPU residency and model cache usage*

Description

Returns a snapshot of GPU device memory usage and model cache occupancy for the current session. Backends that expose a memory_usage() method contribute device byte counts; backends without that method show NA for byte fields.

Usage

```
amatrix_memory_stats()
```

Value

An object of class amatrix_memory_stats, which is a list with two components:

residency data.frame with one row per registered backend and columns backend (character), resident_objects (integer count of GPU-resident R objects), bytes_used (numeric, device bytes in use, or NA), and bytes_total (numeric, total device capacity, or NA).

model_cache List with n_entries (integer, number of cached matrix factors) and max_size (integer or Inf, the cache size limit).

See Also

amatrix_gc, amatrix_residency_info

Examples

```
stats <- amatrix_memory_stats()
print(stats)
```

amatrix_prepare_operands

Prepare operands for a repeated matrix product

Description

Converts inputs to `amatrix` wrappers when needed, chooses a residency-capable accelerator backend in automatic mode, and binds the operands so repeated products reuse the resident fast path.

Usage

```
amatrix_prepare_operands(
  x,
  y,
  op = c("matmul", "crossprod", "tcrossprod"),
  backend = "auto",
  precision = amatrix_default_precision(),
  policy = amatrix_default_policy()
)
```

Arguments

`x`	Left operand.
`y`	Right operand.
`op`	Product primitive: `"matmul"`, `"crossprod"`, or `"tcrossprod"`.
`backend`	Backend name or `"auto"`.
`precision`	Precision to use when wrapping base matrices.
`policy`	Policy to use when wrapping base matrices.

Value

A list with elements `x`, `y`, and `backend`.

amatrix_register_backend

Register a backend with the amatrix dispatch system

Description

Adds a named backend to the session backend registry. The backend must be a named list containing all required callable fields. Once registered, the backend is available for dispatch by any `aMatrix` object whose `preferred_backend` or `policy` slot names it.

Usage

```
amatrix_register_backend(name, backend, overwrite = FALSE)
```

Arguments

name	Character string. Unique identifier for the backend (e.g. "mlx", "opencl").
backend	Named list implementing the backend contract. Required fields: capabilities, features, precision_modes (each a zero-argument function returning a character vector), available (zero-argument logical function), supports, matmul, crossprod, tcrossprod, ewise, rowSums, colSums.
overwrite	Logical. Allow replacement of an existing registration with the same name. Default FALSE.

Value

Invisibly, name.

See Also

amatrix_backend_names, amatrix_backend_status

Examples

```
# Minimal no-op backend for illustration only
noop <- list(
  capabilities   = function() character(),
  features       = function() character(),
  precision_modes = function() "strict",
  available      = function() FALSE,
  supports       = function(op, x, y = NULL) FALSE,
  matmul         = function(x, y) x,
  crossprod      = function(x, y = NULL) x,
  tcrossprod     = function(x, y = NULL) x,
  ewise          = function(x, y, op) x,
  rowSums        = function(x) numeric(nrow(x)),
  colSums        = function(x) numeric(ncol(x))
)
amatrix_register_backend("noop_test", noop, overwrite = TRUE)
```

---

amatrix_release_resident

Release GPU-resident data held by an amatrix object

Description

Frees any device-resident buffer associated with x and drops its residency-registry binding, leaving the host copy as the authoritative storage. This gives long-lived GPU pipelines explicit control over device memory instead of waiting for garbage collection to reclaim resident handles.

Usage

```
amatrix_release_resident(x)
```

Arguments

x	An aMatrix object (for example an adgeMatrix or adgCMatrix). Non-amatrix inputs are ignored.

Details

The object remains fully usable afterwards: its data is served from the host copy and is re-uploaded to the device on the next GPU operation if needed. On CPU-only sessions, or for any object that currently holds no device buffer, this is a safe no-op.

Value

Invisibly, TRUE if a resident binding was released, and FALSE otherwise (including the CPU-only no-op case).

Examples

```
A <- adgeMatrix(matrix(1:6, 2, 3))
# On a CPU-only session there is no device buffer, so this is a no-op:
released <- amatrix_release_resident(A)
released
```

---

amatrix_residency_info

Query GPU residency state of an aMatrix object

Description

Returns a single-row data.frame describing whether x is currently uploaded to a GPU backend and, if so, which backend holds it and whether that binding is still live (the device buffer still exists).

Usage

```
amatrix_residency_info(x)
```

Arguments

x	An aMatrix object.

Value

A data.frame with one row and columns:

backend Character. Backend name, or NA when not resident.

resident_key Character. Internal device buffer key, or NA.

pinned_backend Character. Backend name when the binding is confirmed live, otherwise NA.

live Logical. TRUE when the backend still holds the buffer identified by resident_key.

See Also

amatrix_materialize_host, amatrix_memory_stats

Examples

```
m <- adgeMatrix(matrix(1:4, 2, 2))
amatrix_residency_info(m)
```

---

amatrix_resident_backend_for

Choose a residency-capable accelerator backend for a hot path

---

Description

Returns the first available non-CPU backend that can keep x resident for the requested operation. This is intended for package authors who want repeated work to stay on the fastest available accelerator without hardcoding backend names such as "metal" or "mlx".

Usage

```
amatrix_resident_backend_for(x, op = NULL, y = NULL)
```

Arguments

x	An aMatrix.
op	Optional operation name such as "matmul".
y	Optional rhs object used when checking resident-op support.

Value

A backend name, or NULL when no residency-capable accelerator is available for the requested operation.

amatrix_set_default_policy

Set the session-level default dispatch policy

Description

Sets the dispatch policy applied to new `aMatrix` objects that do not specify their own policy. The change affects all subsequent matrix constructions in the current session.

Usage

```
amatrix_set_default_policy(policy)
```

Arguments

`policy`	Character string. Must be one of `"auto"`, `"cpu"`, `"mlx"`, `"metal"`, `"arrayfire"`, or `"opencl"`.

Value

Invisibly, `policy`.

See Also

`amatrix_default_policy`, `amatrix_set_default_precision`

Examples

```
old <- amatrix_default_policy()
amatrix_set_default_policy("auto")
amatrix_set_default_policy(old) # restore
```

amatrix_set_default_precision

Set the session-level default precision mode

Description

Sets the precision mode applied to new `aMatrix` objects that do not specify their own precision. Use `"strict"` for reproducible double-precision results and `"fast"` for maximum GPU throughput with single/mixed precision.

Usage

```
amatrix_set_default_precision(precision)
```

Arguments

precision	Character string. Must be one of "strict" or "fast".

Value

Invisibly, precision.

See Also

amatrix_default_precision, amatrix_set_default_policy

Examples

```
old <- amatrix_default_precision()
amatrix_set_default_precision("strict")
amatrix_set_default_precision(old) # restore
```

---

amatrix_use_gpu *Enable GPU acceleration for this session*

Description

Finds, enables, and health-checks an installed GPU backend, then adopts it as the session default for "fast"-precision work. On Apple Silicon with **amatrix.mlx** installed this is usually unnecessary: MLX probing is on by default and activates on first use. Call this for the opt-in backends (**amatrix.opencl**, **amatrix.arrayfire**, **amatrix.metal**), to force a specific backend, or to get an explicit confirmation line.

Usage

```
amatrix_use_gpu(backend = NULL, quiet = FALSE)
```

Arguments

backend	Optional backend name ("mlx", "metal", "arrayfire", "opencl"). Default NULL tries the automatic preference order and adopts the first healthy one.
quiet	Logical; suppress the status messages. Default FALSE.

Details

GPU backends compute in float32 ("fast" precision, conformance tolerance ~1e-4); "strict" float64 work always stays on the CPU reference backend regardless of this setting.

Side effect: on success this sets the session default precision to "fast" (and, when backend is given explicitly, the session default policy to that backend) so subsequent matrices route to the GPU without per-object arguments. Undo with amatrix_set_default_precision("strict") / amatrix_set_default_policy("auto").

Value

Invisibly, the name of the enabled backend, or FALSE if no GPU backend could be enabled.

See Also

amatrix_gpu_status, amatrix_backend_status, amatrix_set_default_precision

Examples

```
status <- amatrix_gpu_status()
if (interactive()) amatrix_use_gpu()
```

---

amatrix_warm *Warm up GPU backends to eliminate cold-start latency*

Description

Pre-compiles GPU kernels by running tiny dummy operations through each requested backend. Call once before timed work to pay JIT compilation costs upfront. Errors are silently swallowed; warming never alters numerical state.

Usage

```
amatrix_warm(
  backend = NULL,
  ops = c("matmul", "crossprod", "qr", "chol"),
  size = c(64L, 64L),
  quiet = FALSE
)
```

Arguments

backend	Character vector of backend names to warm, or NULL to warm all non-CPU backends currently registered.
ops	Character vector of operation names to trigger. Recognised values: "matmul", "crossprod", "tcrossprod", "qr", "chol", "svd", "solve".
size	Integer vector of length 2 giving the dimensions c(nrow, ncol) of the dummy matrices used during warming.
quiet	Logical; suppress progress messages when TRUE.

Value

An invisible named list, one entry per backend, each a list with elements warmed (logical) and elapsed_ms (numeric milliseconds, or NA when unavailable).

See Also

amatrix_backend_names

Examples

```
results <- amatrix_warm(quiet = TRUE)
```

---

amChol-class *Cholesky factorization result*

Description

Stores the upper-triangular Cholesky factor R of a symmetric positive-definite adgeMatrix, as returned by chol_factor. When the factor is resident on a GPU backend the host-side @factor slot may be an empty zero-row matrix; use chol_solve rather than accessing slots directly.

Slots

factor Numeric matrix; the upper-triangular factor R such that t(R) %*% R equals the source matrix. May be matrix(numeric(0), 0, 0) when the factor lives only on the device.

factor_obj Either an adgeMatrix holding the GPU-resident factor, or NULL for CPU-only factors.

source_id Character string; the object_id of the source adgeMatrix.

precision Character string; "strict" or "fast".

backend Character string; the backend that computed the factorization.

See Also

chol_factor, chol_solve, chol_logdet

---

amLU-class *LU factorization result for general square matrices*

Description

Stores the original square matrix for use with LAPACK's DGESV routine. Unlike amChol, which caches the explicit triangular factor, amLU retains A and delegates factorization to base::solve at solve time.

Slots

A Numeric square matrix; the original matrix passed to lu_factor.

source_id Character string; the object_id of the source adgeMatrix, or NA for base matrices.

precision Character string; "strict" or "fast", or NA for base matrices.

backend Character string; the preferred backend of the source object, or NA for base matrices.

See Also

lu_factor, lu_solve

---

amSVD-class *Truncated SVD factorization result*

Description

Stores the rank-k truncated singular value decomposition of an aMatrix as returned by svd_factor. Left and right singular vectors are kept as base R matrices so they are always host-accessible; optional adgeMatrix copies of t(u) and v may be present for GPU-accelerated projection.

Slots

u Numeric matrix of left singular vectors; nrow(u) equals the number of rows of the source matrix and ncol(u) equals k.

d Numeric vector of singular values of length k, in descending order.

v Numeric matrix of right singular vectors; nrow(v) equals the number of columns of the source matrix and ncol(v) equals k.

k Integer; the requested rank.

method Character string; one of "exact", "rsvd", or "subspace".

engine Character string identifying the low-level solver used (e.g., "exact_svd", "irlba_svdr", "backend_rsvd").

source_id Character string; the object_id of the source aMatrix.

precision Character string; "strict" or "fast".

backend Character string; the backend that computed the SVD.

ut_am Either an adgeMatrix holding t(u) for GPU matrix-multiply routing, or NULL on CPU paths.

v_am Either an adgeMatrix holding v for GPU matrix-multiply routing, or NULL on CPU paths.

See Also

svd_factor, svd_project, svd_reconstruct

`array_lm` *Fit linear models with array-shaped response*

Description

Wraps `many_lm` to accept a response `Y` with more than two dimensions (e.g. a 3-D array). The trailing dimensions are collapsed into columns for fitting and optionally restored in the output.

Usage

```
array_lm(
  X,
  Y,
  weights = NULL,
  intercept = FALSE,
  include_fitted = FALSE,
  include_residuals = FALSE,
  cache = TRUE,
  method = c("normal", "qr"),
  restore_array = TRUE
)
```

Arguments

`X`	Numeric matrix or `adgeMatrix` of predictors, shape `[n, p]`.
`Y`	Numeric array or matrix of responses. The first dimension must equal `n` (observations). Additional dimensions are treated as independent response variables.
`weights`	Optional numeric vector of length `n` with non-negative observation weights.
`intercept`	Logical. When `TRUE`, a column of ones is prepended to `X` before fitting.
`include_fitted`	Logical. When `TRUE`, fitted values are stored in the returned object.
`include_residuals`	Logical. When `TRUE`, residuals are stored in the returned object.
`cache`	Logical. When `TRUE`, the design-matrix factorization is cached for reuse.
`method`	Solver: `"normal"` or `"qr"`.
`restore_array`	Logical. When `TRUE` (default), `rss`, `sigma2`, fitted values, and residuals are reshaped to match the original trailing dimensions of `Y`.

Value

An object of class `"am_array_lm_fit"`, a named list containing the same fields as `many_lm` plus:

response_dims Integer vector of trailing dimensions of `Y`.

rss Array or vector of residual sums of squares.

sigma2 Array or vector of residual variances.

See Also

many_lm, lm_fit

Examples

```
X <- matrix(rnorm(50), nrow = 10)
Y <- array(rnorm(10 * 3 * 4), dim = c(10, 3, 4))
fit <- array_lm(X, Y)
dim(fit$sigma2)
```

---

as.matrix,adgeMatrix-method

Coerce amatrix objects to base R types

Description

Convert adgeMatrix, adgCMatrix, or aTransposeView objects to base R matrix, numeric vector, or array by materializing the host copy.

Usage

```
## S4 method for signature 'adgeMatrix'
as.matrix(x, ...)

## S3 method for class 'adgeMatrix'
as.matrix(x, ...)

## S4 method for signature 'adgCMatrix'
as.matrix(x, ...)

## S3 method for class 'adgCMatrix'
as.matrix(x, ...)

## S4 method for signature 'aTransposeView'
as.matrix(x, ...)

## S3 method for class 'aTransposeView'
as.matrix(x, ...)

## S4 method for signature 'amChol'
as.matrix(x, ...)

## S4 method for signature 'KronMatrix'
as.matrix(x, ...)

## S4 method for signature 'adgeMatrix'
```

```
as.numeric(x, ...)

## S4 method for signature 'adgeMatrix'
as.vector(x, mode = "any")

## S4 method for signature 'adgeMatrix'
as.array(x, ...)

## S4 method for signature 'adgCMatrix'
as.array(x, ...)
```

Arguments

x	An adgeMatrix, adgCMatrix, or aTransposeView.
...	Further arguments passed to the corresponding base R coercion function.
mode	Storage mode string passed to as.vector.

Value

A plain R matrix, numeric vector, or array containing the materialized host data.

Examples

```
A <- adgeMatrix(matrix(1:6, 2, 3))
as.matrix(A)
```

---

as_adgCMatrix *Coerce an object to adgCMatrix*

Description

Converts a sparse or dense matrix-like object to an adgCMatrix with the requested backend metadata.

Usage

```
as_adgCMatrix(
  x,
  mode = NULL,
  backend = NULL,
  preferred_backend = NULL,
  policy = NULL,
  precision = NULL
)
```

Arguments

x	A dgCMatrix, other sparseMatrix, or base R matrix.
mode	Single string shortcut; see adgCMatrix.
backend	Alias for preferred_backend.
preferred_backend	Single string; preferred compute backend.
policy	Single string dispatch policy.
precision	Single string; "strict" or "fast".

Value

An adgCMatrix.

---

as_adgeMatrix *Coerce an object to adgeMatrix*

Description

Converts a matrix-like object or a resident_handle to an adgeMatrix. When x is a resident_handle, ownership of the GPU-resident buffer is transferred to the new adgeMatrix.

Usage

```
as_adgeMatrix(
  x,
  mode = NULL,
  backend = NULL,
  preferred_backend = NULL,
  policy = NULL,
  precision = NULL
)
```

Arguments

x	A resident_handle, base R matrix, dgeMatrix, or any denseMatrix.
mode	Single string shortcut; see adgeMatrix.
backend	Alias for preferred_backend.
preferred_backend	Single string; preferred compute backend.
policy	Single string dispatch policy.
precision	Single string; "strict" or "fast".

Value

An adgeMatrix.

Examples

```
m <- matrix(1:6, nrow = 2)
A <- as_adgeMatrix(m)
dim(A)
```

`as_adgeMatrix.resident_handle` *Convert a resident handle back to an adgeMatrix*

Description

Creates an adgeMatrix with the resident key still bound. By default the GPU data is materialized to a host copy. If `defer_host = TRUE`, the host copy is not materialized until first host access.

Usage

```
as_adgeMatrix.resident_handle(h, ..., defer_host = FALSE)
```

Arguments

`h`	A `resident_handle`.
`...`	Reserved for future use.
`defer_host`	When `TRUE`, return a deferred-host `adgeMatrix` that materializes lazily. Deferred-host objects are not process-serializable unless materialized before persistence; after `saveRDS()`/`readRDS()` they fail cleanly instead of returning sentinel data.

Value

An `adgeMatrix`.

`aTransposeView-class` *Lazy transpose view of an adgeMatrix*

Description

`aTransposeView` is a zero-copy structural view representing the transpose of an `adgeMatrix`. It carries no independent dense host storage; the underlying data lives in `source`. The transposed matrix is materialized on demand via `as.matrix()` or `amatrix_materialize_host()`.

Slots

`source` The originating `adgeMatrix`; kept alive by this reference.

`Dim` Integer vector of length 2 giving the transposed dimensions `c(ncol_src, nrow_src)`.

`Dimnames` List of length 2 with transposed dimnames.

batch_chol *Batch Cholesky factorization*

Description

Factorize B symmetric positive-definite matrices in parallel. Each matrix is dispatched through the same backend as `chol_factor`, so MLX GPU acceleration applies to every element when available.

Usage

```
batch_chol(A)
```

Arguments

`A`	A list of square numeric matrices, or a 3-D array `[n, n, B]`.

Value

A list of `amChol` objects, one per input matrix.

See Also

`chol_factor`, `batch_solve`

batch_crossprod *Batch crossproduct*

Description

Compute `t(A_b) %*% A_b` for each matrix in a batch.

Usage

```
batch_crossprod(A)
```

Arguments

`A`	A list of numeric matrices, or a 3-D array `[n, p, B]`.

Value

A list of `p x p` crossproduct matrices.

See Also

`batch_chol`

batch_solve *Batch triangular solve*

Description

Solve B linear systems `A_b x_b = B_b` where each `A_b` is represented by its Cholesky factor from `batch_chol`.

Usage

```
batch_solve(Ls, B)
```

Arguments

`Ls`	A list of `amChol` objects (output of `batch_chol`).
`B`	A list of right-hand-side matrices/vectors, or a 3-D array `[n, k, B]`. Length / third dimension must match `Ls`.

Value

A list of solution matrices (or vectors when each rhs is a vector).

See Also

`batch_chol`, `chol_solve`

block_lanczos *Block Lanczos SVD via block Krylov iteration*

Description

Computes a truncated SVD using a block Lanczos bidiagonalization. Each Krylov step issues one GPU GEMM per block rather than sequential GEMVs, significantly reducing kernel-launch overhead on accelerated backends.

Usage

```
block_lanczos(
  A,
  nv = 5L,
  nu = nv,
  block_size = NULL,
  n_steps = NULL,
  mode = "fast",
  backend = NULL
```

```
)

block_svd(
  A,
  k,
  block_size = NULL,
  n_steps = NULL,
  mode = "fast",
  backend = NULL
)
```

Arguments

A	Numeric matrix, adgeMatrix, or adgCMatrix. Plain matrices are coerced to adgeMatrix using mode and backend.
nv	Number of right singular vectors to return.
nu	Number of left singular vectors to return. Defaults to nv.
block_size	Integer block width for the Krylov iteration. When NULL (default), a size is chosen automatically based on nv and nu.
n_steps	Number of Krylov steps. When NULL (default), chosen automatically.
mode	Execution mode passed to adgeMatrix() when coercing plain matrices.
backend	Backend name passed to adgeMatrix() when coercing plain matrices. Ignored when A is already an aMatrix.
k	Number of singular values/vectors. Alias for nv = nu = k.

Value

A named list with components:

u Numeric matrix [m, nu]: left singular vectors.

d Numeric vector of length min(nu, nv): singular values in decreasing order.

v Numeric matrix [n, nv]: right singular vectors.

iter Integer: number of Krylov steps performed.

mprod Integer: total matrix-vector products issued.

See Also

rsvd, block_svd

Examples

```
A <- matrix(rnorm(200), nrow = 20)
res <- block_lanczos(A, nv = 3L)
length(res$d)
```

chol,adgCMatrix-method

Cholesky factorization for adgCMatrix

Description

Compute the Cholesky factorization of a sparse symmetric positive-definite adgCMatrix.

Usage

```
## S4 method for signature 'adgCMatrix'
chol(x, ...)
```

Arguments

x	A symmetric positive-definite adgCMatrix.
...	Further arguments passed to Matrix::chol.

Value

An adgCMatrix or sparse Cholesky factor object.

chol,adgeMatrix-method

Cholesky factorization for adgeMatrix

Description

Compute the Cholesky factor of a symmetric positive-definite adgeMatrix, dispatching through the amatrix backend.

Usage

```
## S4 method for signature 'adgeMatrix'
chol(x, ...)
```

Arguments

x	A symmetric positive-definite adgeMatrix.
...	Further arguments passed to the backend.

Value

An adgeMatrix containing the upper triangular Cholesky factor.

Examples

```
S <- adgeMatrix(crossprod(matrix(rnorm(9), 3, 3)) + 3 * diag(3))
R <- chol(S)
```

`chol_diag` *Extract the diagonal of a Cholesky factor*

Description

Returns the diagonal of the upper-triangular matrix `R` stored in an `amChol` object.

Usage

```
chol_diag(factor)
```

Arguments

`factor`	An `amChol` object from `chol_factor`.

Value

Numeric vector of length equal to the matrix dimension.

Examples

```
m <- crossprod(matrix(rnorm(16), 4, 4)) + diag(4)
A <- adgeMatrix(m)
fac <- chol_factor(A)
chol_diag(fac)
```

`chol_factor` *Compute the Cholesky factorization of an adgeMatrix*

Description

Computes the upper-triangular Cholesky factor `R` of a symmetric positive-definite `adgeMatrix` `X` such that `t(R) %*% R == X`. Results are cached by `object_id`; repeated calls with the same object return the cached factor.

Usage

```
chol_factor(X)
```

Arguments

X	An adgeMatrix that is symmetric positive definite.

Value

An amChol object.

Examples

```
m <- crossprod(matrix(rnorm(16), 4, 4)) + diag(4)
A <- adgeMatrix(m)
fac <- chol_factor(A)
fac
```

---

chol_logdet *Log-determinant from a Cholesky factor*

Description

Computes log(det(X)) from the Cholesky factor of a symmetric positive-definite matrix X as 2 * sum(log(diag(R))), which avoids forming the full determinant and is numerically stable.

Usage

```
chol_logdet(factor)
```

Arguments

factor	An amChol object from chol_factor.

Value

Scalar double; the log-determinant of the source matrix.

Examples

```
m <- crossprod(matrix(rnorm(16), 4, 4)) + diag(4)
A <- adgeMatrix(m)
fac <- chol_factor(A)
chol_logdet(fac)
```

chol_solve — *Solve a linear system using a Cholesky factor*

Description

Solves `X %*% x = B` where `X` is the symmetric positive-definite matrix whose Cholesky factorization is stored in `factor`. Dispatches to a GPU backend when the factor was computed in `"fast"` precision and a device-resident factor is available.

Usage

```
chol_solve(factor, B)
```

Arguments

`factor`	An `amChol` object from `chol_factor`.
`B`	Numeric vector or matrix; the right-hand side. The number of rows must equal the dimension of the factor.

Value

Numeric vector or matrix `x` satisfying `X %*% x == B`. Returns a vector when `B` is a vector.

Examples

```
m <- crossprod(matrix(rnorm(16), 4, 4)) + diag(4)
A <- adgeMatrix(m)
fac <- chol_factor(A)
b <- rnorm(4)
x <- chol_solve(fac, b)
```

chol_solve_batches — *Solve many right-hand-side batches with one Cholesky factor*

Description

This is the same operation as repeatedly calling `chol_solve(factor, B[[i]])`, but it packs all RHS batches into one wide solve and then splits the result. BLAS/GPU backends generally amortize launch and dispatch overhead much better on one wide RHS than on many small independent solves.

Usage

```
chol_solve_batches(factor, B)
```

Arguments

factor	An amChol object from chol_factor.
B	A list of RHS vectors/matrices, or a 3-D array [n, k, batch].

Value

A list of solution vectors/matrices.

---

correlation *Compute a correlation matrix*

Description

Computes the sample correlation matrix of the columns of X, optionally with observation weights and column-blocked covariance accumulation.

Usage

```
correlation(X, center = TRUE, weights = NULL, block_size = NULL)
```

Arguments

X	Numeric matrix or adgeMatrix of shape [n, p].
center	Logical; when TRUE (default) column means are subtracted before computing covariances.
weights	Numeric vector of length n, or NULL for unweighted correlation.
block_size	Positive integer or NULL. When non-NULL, covariances are accumulated in blocks of this many columns to limit memory usage.

Value

An adgeMatrix of shape [p, p]: the sample correlation matrix with diagonal entries set to 1.

See Also

covariance

Examples

```
X <- matrix(rnorm(50), nrow = 10)
R <- correlation(X)
round(diag(as.matrix(R)), 6)
```

cov2cor,adgeMatrix-method

Covariance-to-correlation methods for amatrix objects

Description

Bridge `cov2cor()` through Matrix's covariance-to-correlation methods so standard workflows such as `cov2cor(crossprod(X))` keep working when `crossprod()` preserves an amatrix class.

Usage

```
## S4 method for signature 'adgeMatrix'
cov2cor(V)

## S4 method for signature 'adgCMatrix'
cov2cor(V)
```

Arguments

`V`	A square `adgeMatrix` or `adgCMatrix`.

Value

A base R correlation matrix, matching `stats::cov2cor()` on the corresponding host matrix.

Examples

```
X <- adgeMatrix(matrix(1:9 + 0, 3, 3))
cov2cor(crossprod(X))
```

covariance

Backend-dispatched covariance matrix

Description

Computes the (possibly weighted) sample or population covariance matrix of `X`. Sparse inputs use a memory-efficient path; dense inputs use a fused GPU kernel when available, otherwise fall back to CPU centering followed by a GPU cross-product.

Usage

```
covariance(X, center = TRUE, sample = TRUE, weights = NULL, block_size = NULL)
```

Arguments

X	Numeric matrix, adgeMatrix, or sparse adgCMatrix / sparseMatrix, shape [n, p].
center	Logical. When TRUE (default), columns are mean-centred before computing the cross-product.
sample	Logical. When TRUE (default), divides by n - 1 (sample covariance); when FALSE, divides by n.
weights	Optional numeric vector of length n with non-negative observation weights. When supplied, a weighted covariance is computed.
block_size	Optional integer. When set, the cross-product is computed in column-blocks of this size to limit peak memory. Ignored for sparse inputs.

Value

An adgeMatrix of shape [p, p].

See Also

many_lm

Examples

```
X <- matrix(rnorm(200), nrow = 40)
C <- covariance(X)
dim(C)
```

---

crossprod,adgCMatrix,missing-method

Cross-product methods for adgCMatrix

Description

Compute $t(x)\% * \%y$ (crossprod) or $x\% * \%t(y)$ (tcrossprod) for sparse adgCMatrix objects, dispatching through the amatrix backend.

Usage

```
## S4 method for signature 'adgCMatrix,missing'
crossprod(x, y = NULL, ...)

## S4 method for signature 'adgCMatrix,ANY'
crossprod(x, y = NULL, ...)

## S4 method for signature 'adgCMatrix,matrix'
crossprod(x, y = NULL, ...)
```

```
## S4 method for signature 'adgCMatrix,Matrix'
crossprod(x, y = NULL, ...)

## S4 method for signature 'adgCMatrix,dgeMatrix'
crossprod(x, y = NULL, ...)

## S4 method for signature 'adgCMatrix,dgCMatrix'
crossprod(x, y = NULL, ...)

## S4 method for signature 'adgCMatrix,adgeMatrix'
crossprod(x, y = NULL, ...)

## S4 method for signature 'adgCMatrix,adgCMatrix'
crossprod(x, y = NULL, ...)

## S4 method for signature 'adgCMatrix,missing'
tcrossprod(x, y = NULL, ...)

## S4 method for signature 'adgCMatrix,ANY'
tcrossprod(x, y = NULL, ...)

## S4 method for signature 'adgCMatrix,matrix'
tcrossprod(x, y = NULL, ...)

## S4 method for signature 'adgCMatrix,Matrix'
tcrossprod(x, y = NULL, ...)

## S4 method for signature 'adgCMatrix,dgeMatrix'
tcrossprod(x, y = NULL, ...)

## S4 method for signature 'adgCMatrix,dgCMatrix'
tcrossprod(x, y = NULL, ...)

## S4 method for signature 'adgCMatrix,adgeMatrix'
tcrossprod(x, y = NULL, ...)

## S4 method for signature 'adgCMatrix,adgCMatrix'
tcrossprod(x, y = NULL, ...)
```

Arguments

x	An adgCMatrix.
y	A matrix-like object or NULL/missing for the symmetric form.
...	Further arguments passed to the backend.

Value

An adgeMatrix or adgCMatrix containing the result.

Examples

```
sp <- as(matrix(c(1, 0, 0, 2, 1, 0), 3, 2), "dgCMatrix")
A  <- adgCMatrix(sp)
crossprod(A)
```

---

crossprod,adgeMatrix,ANY-method

Cross-product methods for adgeMatrix

Description

Compute $t(x)\%*\%y$ (`crossprod`) or $x\%*\%t(y)$ (`tcrossprod`) for `adgeMatrix` objects, dispatching through the amatrix backend to preserve GPU residency.

Usage

```
## S4 method for signature 'adgeMatrix,ANY'
crossprod(x, y = NULL, ...)

## S4 method for signature 'adgeMatrix,missing'
crossprod(x, y = NULL, ...)

## S4 method for signature 'adgeMatrix,ANY'
tcrossprod(x, y = NULL, ...)

## S4 method for signature 'adgeMatrix,missing'
tcrossprod(x, y = NULL, ...)
```

Arguments

`x`	An `adgeMatrix`.
`y`	A matrix-like object, or `NULL` for the symmetric form $t(x)\%*\%x$ or $x\%*\%t(x)$.
`...`	Further arguments passed to the underlying backend operation.

Value

An `adgeMatrix` containing the result.

Examples

```
A <- adgeMatrix(matrix(rnorm(12), 4, 3))
crossprod(A)
tcrossprod(A)
```

crossprod_add_diag *Cross-product plus diagonal perturbation*

Description

Computes $X^T X + \lambda I$ (scalar `lambda`) or $X^T X + \mathrm{diag}(\lambda)$ (vector `lambda`) in a single fused call.

Usage

```
crossprod_add_diag(X, lambda)
```

Arguments

`X`	Numeric matrix or `adgeMatrix` of shape `[n, p]`.
`lambda`	Scalar or numeric vector of length `p`; diagonal perturbation to add to the cross-product.

Value

An `adgeMatrix` of shape `[p, p]`: the perturbed cross-product.

See Also

`crossprod_weighted`

Examples

```
X <- matrix(rnorm(20), nrow = 5)
crossprod_add_diag(X, lambda = 0.1)
```

crossprod_weighted *Weighted cross-product X'WX*

Description

Computes $X^T \mathrm{diag}(w) X$, a `p` x `p` weighted cross-product. A GPU-resident fast path is used when available.

Usage

```
crossprod_weighted(X, w)
```

Arguments

`X`	Numeric matrix or `adgeMatrix` of shape `[n, p]`.
`w`	Positive numeric vector of length `n`; observation weights.

Value

An adgeMatrix of shape [p, p].

See Also

tcrossprod_weighted, xty_weighted

Examples

```
X <- matrix(rnorm(20), nrow = 5)
w <- runif(5)
crossprod_weighted(X, w)
```

---

dist_matrix *GPU-accelerated pairwise distance matrix*

Description

Computes the pairwise distance matrix between rows of X and Y. The dominant cost (row inner-products via am_tcrossprod) is dispatched to the active GPU backend (ArrayFire or MLX); norm computation and final transforms run on CPU where they are O(mp + np) — negligible versus the O(mnp) GEMM.

Usage

```
dist_matrix(
  X,
  Y = NULL,
  method = c("euclidean", "sqeuclidean", "cosine"),
  tile_size = NULL
)
```

Arguments

X	Numeric matrix or adgeMatrix, shape [m, p].
Y	Numeric matrix or adgeMatrix, shape [n, p], or NULL to compute pairwise distances within X (returns [m, m] matrix).
method	One of "euclidean" (default), "sqeuclidean", or "cosine".
tile_size	Integer row-block size for tiled computation, or NULL (default) to auto-tile when nrow(X) > 50000 (self-distance only). Set explicitly to process any size in row-blocks; useful when GPU memory is limited. Not supported for method = "cosine".

Value

Numeric matrix [m, n] of pairwise distances.

See Also

kernel_matrix

Examples

```
X <- matrix(rnorm(30), nrow = 6)
D <- dist_matrix(X)
dim(D)
```

---

dot *Inner product of two vectors or matrices*

Description

Computes the element-wise inner product `sum(x * y)`, equivalent to `as.numeric(t(x) %*% y)` for vectors.

Usage

```
dot(x, y)
```

Arguments

x	A numeric vector, matrix, or `aMatrix`.
y	A numeric vector, matrix, or `aMatrix` conformable with `x`.

Value

A single numeric scalar.

Examples

```
dot(1:4, 4:1)
```

eigen,adgCMatrix-method

Eigendecomposition for adgCMatrix

Description

Compute eigenvalues and eigenvectors of a sparse `adgCMatrix`, dispatching through the amatrix backend.

Usage

```
## S4 method for signature 'adgCMatrix'
eigen(x, symmetric, only.values = FALSE, EISPACK = FALSE)
```

Arguments

`x`	An `adgCMatrix`.
`symmetric`	Logical; whether `x` is symmetric. Auto-detected when missing.
`only.values`	Logical; if `TRUE` only eigenvalues are returned.
`EISPACK`	Ignored; retained for signature compatibility.

Value

A list with components `values` and `vectors`.

eigen,adgeMatrix-method

Eigendecomposition for adgeMatrix

Description

Compute eigenvalues and eigenvectors of an `adgeMatrix`, dispatching through the amatrix backend for symmetric matrices when GPU acceleration is available. A fallback method for plain `matrix` preserves base R behaviour after the generic is promoted to S4.

Usage

```
## S4 method for signature 'adgeMatrix'
eigen(x, symmetric, only.values = FALSE, EISPACK = FALSE)
```

Arguments

x	An adgeMatrix or plain matrix.
symmetric	Logical indicating whether x is symmetric. When missing, symmetry is detected automatically from the host copy.
only.values	Logical; if TRUE only eigenvalues are returned.
EISPACK	Ignored; retained for signature compatibility.

Value

A list with components values (numeric vector) and vectors (matrix, omitted when only.values = TRUE).

Examples

```
S <- adgeMatrix(crossprod(matrix(rnorm(9), 3, 3)))
ev <- eigen(S, symmetric = TRUE)
length(ev$values)
```

---

eigh *Symmetric eigendecomposition*

Description

Computes eigenvalues and eigenvectors of a real symmetric matrix by dispatching to the active backend via eigen with symmetric = TRUE.

Usage

```
eigh(x)
```

Arguments

x	A real symmetric numeric matrix, adgeMatrix, or other object accepted by eigen.

Value

A list with components values (numeric vector of eigenvalues in ascending order) and vectors (numeric matrix whose columns are the corresponding eigenvectors).

See Also

rsvd

Examples

```
S <- crossprod(matrix(rnorm(25), nrow = 5))
ev <- eigh(adgeMatrix(S))
length(ev$values)
```

---

ewise *Element-wise operations*

Description

Apply an element-wise arithmetic operation to one or two matrices, dispatching to the preferred GPU backend when available.

Usage

```
ewise(op, e1, e2 = NULL)
```

Arguments

op	Character string: "+", "-", "*", or "/".
e1	A numeric matrix or adgeMatrix.
e2	A numeric matrix, adgeMatrix, or NULL for unary ops.

Value

A matrix of the same dimensions as e1.

---

gemm *Generalised matrix multiply (BLAS DGEMM interface)*

Description

Computes alpha * op(A) %*% op(B) + beta * C, where op(X) = t(X) when the corresponding trans flag is TRUE. Routes internally to the most efficient resident operation for the chosen transpose combination.

Usage

```
gemm(A, B, C = NULL, alpha = 1, beta = 1, transA = FALSE, transB = FALSE)
```

Arguments

A	A matrix or aMatrix.
B	A matrix or aMatrix.
C	Optional matrix or aMatrix to add after scaling; NULL omits the addition term.
alpha	Numeric scalar multiplier for op(A) %*% op(B). Default 1.0.
beta	Numeric scalar multiplier for C. Default 1.0.
transA	Logical; transpose A before multiplying. Default FALSE.
transB	Logical; transpose B before multiplying. Default FALSE.

Value

A matrix of dimensions nrow(op(A)) by ncol(op(B)).

Examples

```
A <- adgeMatrix(matrix(1:6, 2, 3))
B <- adgeMatrix(matrix(1:6, 2, 3))
gemm(A, B, transA = TRUE)          # t(A) %*% B
gemm(A, B, transB = TRUE)          # A %*% t(B)
```

---

irlba *GPU-accelerated truncated SVD via irlba*

Description

Wraps irlba::irlba() with an adgeMatrix input so that every Lanczos matrix-vector product routes through the amatrix GPU dispatch path. The matrix A is kept resident on device; consecutive matvecs in the Lanczos loop avoid host round-trips.

Usage

```
irlba(
  A,
  nv = 5,
  nu = nv,
  ...,
  mode = "fast",
  backend = NULL,
  implementation = c("compat", "block"),
  block_size = NULL,
  n_steps = NULL
)
```

Arguments

A	A matrix, adgeMatrix, or adgCMatrix. Plain matrices are coerced via adgeMatrix(A, mode=mode, backend=backend).
nv	Number of right singular vectors.
nu	Number of left singular vectors. Defaults to nv.
...	Additional arguments forwarded to irlba::irlba(). fastpath is always forced to FALSE — the C fastpath bypasses S4 dispatch and cannot be GPU-accelerated.
mode	Execution mode passed to adgeMatrix() when coercing. "fast" permits float32 and enables GPU routing. Ignored if A is already an adgeMatrix or adgCMatrix.
backend	Backend name (e.g. "mlx", "arrayfire"). Ignored if A is already an amatrix object.
implementation	Lanczos implementation to use. "compat" preserves the current irlba::irlba() wrapper behavior. "block" routes to block_lanczos for a GEMM-oriented approximation.
block_size	Block size passed to am_block_lanczos() when implementation = "block". Defaults to a small MLX-friendly block size derived from the requested rank.
n_steps	Number of block Krylov steps passed to am_block_lanczos() when implementation = "block".

Details

The hot loop in irlba is two matrix-vector products per Lanczos step: A %*% v and w %*% A. Both route through am_matmul() when A is an adgeMatrix, giving GPU acceleration on the dominant cost. Orthogonalization, svd(B), and convergence tests remain on CPU where they belong (the subspace dimension work is always small).

Do **not** pass mult= — it is deprecated in irlba and forces a non-standard dispatch path. Pass an adgeMatrix instead.

Value

Same structure as irlba::irlba(): a list with components d, u, v, iter, mprod.

See Also

adgeMatrix, svd_factor

---

irlba_native *GPU-native truncated SVD via Lanczos bidiagonalization*

Description

Implements Golub-Kahan Lanczos bidiagonalization directly in ArrayFire C, keeping all matvecs and CGS2 reorthogonalization on the GPU. Only 2*work scalars and the final basis matrices cross PCIe per restart; no per-step host transfers.

Usage

```
irlba_native(
  A,
  nv = 5L,
  nu = nv,
  tol = sqrt(.Machine$double.eps),
  maxit = 100L,
  work = max(nv + 20L, 3L * nv),
  v0 = NULL,
  mode = "fast",
  backend = NULL
)
```

Arguments

A	A matrix or adgeMatrix. Coerced if necessary.
nv	Number of singular values/vectors to compute.
nu	Number of left singular vectors (default = nv).
tol	Convergence tolerance.
maxit	Maximum number of restarts.
work	Size of the Lanczos subspace per restart. Larger values converge in fewer restarts at the cost of more memory and work per restart. Default is max(nv + 20L, 3L * nv).
v0	Optional starting vector (length ncol(A)).
mode, backend	Passed to adgeMatrix() when coercing.

Details

Compared to am_irlba, which routes each Lanczos matvec through S4 dispatch, this function:

- eliminates S4 overhead on the hot path
- replaces k sequential GEMVs for reorthogonalization with one GEMM
- uploads A once and never re-uploads it across restarts

Value

A list with components d, u, v, iter, mprod, compatible with irlba::irlba().

See Also

irlba, adgeMatrix

kernel_matrix *GPU-accelerated pairwise kernel matrix*

Description

Computes the pairwise kernel matrix between rows of X and Y. The expensive am_tcrossprod is GPU-dispatched; element-wise transforms (exp, sqrt, pow) run on CPU.

Usage

```
kernel_matrix(
  X,
  Y = NULL,
  kernel = c("linear", "rbf", "polynomial", "cosine", "laplacian"),
  sigma = 1,
  degree = 2L,
  coef = 0,
  preferred_backend = NULL,
  zero_diag = FALSE
)
```

Arguments

X	Numeric matrix or adgeMatrix, shape [m, p].
Y	Numeric matrix or adgeMatrix, shape [n, p], or NULL.
kernel	Kernel type string (see Details).
sigma	Bandwidth for "rbf" and "laplacian".
degree	Polynomial degree for "polynomial".
coef	Constant term for "polynomial": $(\text{coef} + x^\top y)^{\text{degree}}$.
preferred_backend	Optional backend name to override the default dispatch (e.g., "mlx", "opencl").
zero_diag	When TRUE and Y is NULL, set the diagonal of the kernel matrix to zero.

Details

Kernels:

linear $k(x, y) = x^\top y$

rbf $k(x, y) = \exp(-\|x - y\|^2/(2\sigma^2))$

polynomial $k(x, y) = (\text{coef} + x^\top y)^{\text{degree}}$

cosine $k(x, y) = (x^\top y)/(\|x\|\|y\|)$

laplacian $k(x, y) = \exp(-\|x - y\|/\sigma)$

Value

Numeric matrix [m, n] of kernel values.

See Also

dist_matrix

---

kron *Eager Kronecker product*

Description

Computes the Kronecker product $A \otimes B$ and returns the result as an adgeMatrix. Accepts plain matrices or any aMatrix subclass. For a lazy variant that avoids forming the full product see kron_matrix.

Usage

```
kron(A, B)
```

Arguments

A, B	Matrices or aMatrix objects.

Value

An adgeMatrix of dimension (nrow(A)*nrow(B)) x (ncol(A)*ncol(B)).

See Also

kron_matrix

---

kron_matrix *Construct a lazy Kronecker product*

Description

Creates a KronMatrix that stores the factor matrices A and B without materializing the full Kronecker product. Standard operations such as %*%, crossprod, solve, and determinant are available and exploit the Kronecker structure.

Usage

```
kron_matrix(A, B)
```

Arguments

A	Numeric matrix or object coercible via `as.matrix()`; the left Kronecker factor.
B	Numeric matrix or object coercible via `as.matrix()`; the right Kronecker factor.

Value

A `KronMatrix` of implicit dimensions `c(nrow(A) * nrow(B), ncol(A) * ncol(B))`.

Examples

```
A <- matrix(1:4, 2, 2)
B <- diag(3)
K <- kron_matrix(A, B)
dim(K)
as.matrix(K)
```

---

kronecker-methods *Kronecker product of backend-aware matrices*

Description

S4 methods for `kronecker` and the `%x%` operator that keep the result as an amatrix. Without them, `kronecker(A, B)` and `A %x% B` dispatch to the **Matrix** methods for the parent `dgeMatrix` / `dgCMatrix` classes and silently demote to a plain (non-amatrix) result, discarding backend-dispatch metadata.

Usage

```
## S4 method for signature 'adgeMatrix,adgeMatrix'
kronecker(X, Y, FUN = "*", make.dimnames = FALSE, ...)

## S4 method for signature 'adgeMatrix,adgCMatrix'
kronecker(X, Y, FUN = "*", make.dimnames = FALSE, ...)

## S4 method for signature 'adgCMatrix,adgeMatrix'
kronecker(X, Y, FUN = "*", make.dimnames = FALSE, ...)

## S4 method for signature 'adgCMatrix,adgCMatrix'
kronecker(X, Y, FUN = "*", make.dimnames = FALSE, ...)

## S4 method for signature 'adgeMatrix,matrix'
kronecker(X, Y, FUN = "*", make.dimnames = FALSE, ...)

## S4 method for signature 'matrix,adgeMatrix'
kronecker(X, Y, FUN = "*", make.dimnames = FALSE, ...)
```

```
## S4 method for signature 'adgCMatrix,matrix'
kronecker(X, Y, FUN = "*", make.dimnames = FALSE, ...)

## S4 method for signature 'matrix,adgCMatrix'
kronecker(X, Y, FUN = "*", make.dimnames = FALSE, ...)
```

Arguments

X, Y	Kronecker factors. At least one is an aMatrix subclass; the other may be an amatrix, a base matrix, or a **Matrix** object.
FUN	Function (or its name) applied to the outer products; passed to the underlying **Matrix** method. Defaults to "*".
make.dimnames	Logical; construct dimnames from the factors. Passed to the underlying method.
...	Further arguments passed to the underlying method.

Details

The product itself is computed by **Matrix**'s own Kronecker methods on the materialized host contents, so values are identical to base::kronecker() on the dense contents. The result is re-wrapped as an adgCMatrix when it is sparse and an adgeMatrix otherwise, inheriting the preferred backend, policy, and precision of the first amatrix operand.

Value

An adgeMatrix (dense) or adgCMatrix (sparse).

Examples

```
A <- adgeMatrix(matrix(1:4, 2, 2))
B <- adgeMatrix(diag(2))
kronecker(A, B)
A %x% B
```

KronMatrix-class *Lazy Kronecker product of two matrices*

Description

KronMatrix stores the two factor matrices A (m x n) and B (p x q) without forming the full (mp x nq) Kronecker product. Matrix-vector and matrix-matrix products are evaluated using the vec-permutation identity (A x B) vec(X) = vec(B X t(A)), keeping memory use at O(mn + pq) rather than O(mnpq).

Slots

A Numeric matrix; the left factor of the Kronecker product.

B Numeric matrix; the right factor of the Kronecker product.

See Also

kron_matrix

---

lm_fit *Fit a single linear model*

Description

Solves the ordinary least-squares problem $Y \approx X\beta$ for a single design matrix X and one or more response columns Y.

Usage

```
lm_fit(
  X,
  Y,
  intercept = FALSE,
  include_fitted = TRUE,
  include_residuals = TRUE,
  cache = TRUE,
  method = c("normal", "qr")
)
```

Arguments

X	Numeric matrix or adgeMatrix of predictors, shape [n, p].
Y	Numeric matrix, vector, or adgeMatrix of responses, shape [n, q].
intercept	Logical. When TRUE, a column of ones is prepended to X before fitting.
include_fitted	Logical. When TRUE, fitted values are included in the returned object.
include_residuals	Logical. When TRUE, residuals are included in the returned object.
cache	Logical. When TRUE, the $X^T X$ or QR factorization is cached for reuse across calls sharing the same X.
method	Solver method: "normal" (normal equations, default) or "qr" (QR decomposition).

Value

An object of class "lm_fit", a named list containing:

coefficients adgeMatrix of shape [p, q].

fitted.values adgeMatrix of shape [n, q], or NULL when include_fitted = FALSE.

residuals adgeMatrix of shape [n, q], or NULL when include_residuals = FALSE.

rank Integer model rank.

df.residual Residual degrees of freedom.

See Also

many_lm, array_lm

Examples

```
X <- matrix(rnorm(50), nrow = 10)
y <- rnorm(10)
fit <- lm_fit(X, y)
coef(fit)
```

---

lm_loo_cv *Leave-one-out cross-validation for linear models*

Description

Computes exact leave-one-out (LOO) prediction errors by refitting the model n times, each time dropping one observation. Uses qr_downdate internally to avoid recomputing the full factorization from scratch.

Usage

```
lm_loo_cv(X, y, method = "qr", ...)
```

Arguments

X	Numeric matrix or adgeMatrix of predictors, shape [n, p]. No intercept column is added automatically.
y	Numeric vector or single-column matrix of responses, length n.
method	Character string passed to am_qr controlling the QR algorithm. Currently only "qr" is supported.
...	Additional arguments forwarded to am_qr.

Value

A named list with two elements:

residuals Numeric vector of length n: LOO prediction errors $y_i - \hat{y}_i^{(-i)}$.

mse Scalar mean squared LOO error.

See Also

lm_fit, many_lm

Examples

```
## Not run:
X <- adgeMatrix(matrix(rnorm(50), nrow = 10))
y <- rnorm(10)
cv <- lm_loo_cv(X, y)
cv$mse

## End(Not run)
```

---

lu_factor *Store a general square matrix for LU-based solving*

Description

Wraps a square numeric matrix or adgeMatrix in an amLU object. The actual LU decomposition is performed by base::solve at solve time via LAPACK's DGESV.

Usage

```
lu_factor(A)
```

Arguments

A	A square numeric matrix or adgeMatrix.

Value

An amLU object.

Examples

```
m <- matrix(c(2, 1, 5, 3), nrow = 2)
fac <- lu_factor(m)
fac
```

lu_solve — *Solve a linear system using an LU factor*

Description

Solves `A %*% x = B` where `A` is the square matrix stored in `factor`, delegating to `base::solve` (LAPACK `DGESV`).

Usage

```
lu_solve(factor, B)
```

Arguments

`factor`	An `amLU` object from `lu_factor`.
`B`	Numeric vector or matrix; the right-hand side. The number of rows must equal the dimension of `factor@A`.

Value

Numeric vector or matrix `x` satisfying `A %*% x == B`. Returns a vector when `B` is a vector or single-column matrix.

Examples

```
m <- matrix(c(2, 1, 5, 3), nrow = 2)
fac <- lu_factor(m)
b <- c(1, 2)
lu_solve(fac, b)
```

many_lm — *Fit multiple linear models against a shared design matrix*

Description

The flagship batch regression function. Solves $Y \approx XB$ where `Y` is a matrix whose columns are independent response variables. When no GPU backend is active the normal-equations or QR path runs on CPU. With an active GPU backend the QR path dispatches the factorization to the device and keeps intermediate results resident to minimise host round-trips.

Usage

```
many_lm(
  X,
  Y,
  weights = NULL,
  intercept = FALSE,
  include_fitted = FALSE,
  include_residuals = FALSE,
  cache = TRUE,
  method = c("normal", "qr")
)
```

Arguments

X	Numeric matrix or adgeMatrix of predictors, shape [n, p].
Y	Numeric matrix or adgeMatrix of responses, shape [n, q]. Each column is fitted independently against X.
weights	Optional numeric vector of length n with non-negative observation weights. When supplied, weighted least squares is used.
intercept	Logical. When TRUE, a column of ones is prepended to X before fitting.
include_fitted	Logical. When TRUE, fitted values are stored in the returned object.
include_residuals	Logical. When TRUE, residuals are stored in the returned object.
cache	Logical. When TRUE, the design-matrix factorization is cached for reuse when X is the same across calls.
method	Solver: "normal" (normal equations) or "qr" (QR decomposition). Ignored when weights is non-NULL and the weighted path is selected.

Value

An object of class "am_many_lm_fit", a named list containing:

coefficients adgeMatrix of shape [p, q].

fitted.values adgeMatrix [n, q] or NULL.

residuals adgeMatrix [n, q] or NULL.

rss Numeric vector of length q: residual sums of squares.

sigma2 Numeric vector of length q: residual variances.

rank Integer model rank.

df.residual Residual degrees of freedom.

See Also

lm_fit, array_lm

Examples

```
X <- matrix(rnorm(100), nrow = 20)
Y <- matrix(rnorm(60), nrow = 20)
fit <- many_lm(X, Y, method = "qr")
dim(coef(fit))
```

`mat_fun` *Matrix functions via symmetric eigendecomposition*

Description

Apply an elementwise function to the eigenvalues of a symmetric positive definite matrix and reconstruct the result: $f(X) = Q\,\mathrm{diag}(f(\lambda))\,Q^T$.

Usage

```
mat_sqrt(X)

mat_pow(X, p)

mat_log(X)
```

Arguments

`X`	Symmetric positive definite numeric matrix or `adgeMatrix` of shape `[p, p]`.
`p`	Numeric scalar exponent (used by `mat_pow` only).

Value

An `adgeMatrix` of shape `[p, p]`: the matrix function applied to `X`.

Examples

```
S <- crossprod(matrix(rnorm(16), 4)) + diag(4)
mat_sqrt(S)
mat_log(S)
mat_pow(S, -1)
```

matmul *Matrix multiplication*

Description

Multiplies two matrices, routing to an accelerated backend when available. Plain numeric vectors supplied as y are promoted to a column matrix and the result is dropped back to a vector.

Usage

```
matmul(x, y)
```

Arguments

x	A matrix or aMatrix object.
y	A matrix, aMatrix object, or numeric vector.

Value

A matrix (or numeric vector when y was a vector) of dimensions nrow(x) by ncol(y).

Examples

```
A <- adgeMatrix(matrix(1:6, 2, 3))
B <- adgeMatrix(matrix(1:6, 3, 2))
matmul(A, B)
```

matmul-methods *Matrix multiplication for adgeMatrix*

Description

Dispatches %*% through the amatrix backend for dense adgeMatrix objects on the left-hand side, preserving GPU residency across the operation.

Usage

```
## S4 method for signature 'adgeMatrix,matrix'
x %*% y

## S4 method for signature 'adgeMatrix,Matrix'
x %*% y

## S4 method for signature 'adgeMatrix,dgeMatrix'
x %*% y
```

```
## S4 method for signature 'adgeMatrix,dgCMatrix'
x %*% y

## S4 method for signature 'adgeMatrix,adgeMatrix'
x %*% y

## S4 method for signature 'adgeMatrix,adgCMatrix'
x %*% y

## S4 method for signature 'numeric,adgeMatrix'
x %*% y
```

Arguments

x	An adgeMatrix, numeric vector, or matrix.
y	A matrix-like object: matrix, Matrix, adgeMatrix, adgCMatrix, or ANY.

Value

An adgeMatrix (or numeric vector when y is a vector and x is adgeMatrix), with the same backend metadata as x.

Examples

```
A <- adgeMatrix(matrix(1:6, 2, 3))
B <- matrix(1:3, 3, 1)
A %*% B
```

---

pairwise_sqdist_argmin

Nearest-centroid assignment via fused squared-distance computation

---

Description

Computes $D[i,k] = \|x_i\|^2 - 2x_i^\top c_k + \|c_k\|^2$ and returns $\arg\min_k D[i,k]$ for each row i, 1-indexed. GPU path avoids host round-trips by chaining resident operations.

Usage

```
pairwise_sqdist_argmin(X, Ct, x_norms = NULL, c_norms = NULL)
```

Arguments

X	n×p adgeMatrix or plain matrix (query points).
Ct	p×K numeric matrix holding the centroids **transposed**: nrow(Ct) is the feature dimension p and each *column* is a centroid. Pass t(centroids) when your centroids are stored k×p with one centroid per row. A dimension mismatch (ncol(X) != nrow(Ct)) errors, but a square k == p matrix passed untransposed cannot be detected and will silently assign to the wrong (column) centroids.
x_norms	Optional n-vector of precomputed $\|x_i\|^2$. Computed if NULL.
c_norms	Optional K-vector of precomputed $\|c_k\|^2$. Computed if NULL.

Value

Integer vector of length n, 1-indexed nearest centroid per row.

---

pca_coef *Project and reconstruct data using a truncated SVD*

Description

Convenience wrapper that calls svd_project followed by svd_reconstruct, yielding the rank-k least-squares approximation of Y in the column space of the original matrix.

Usage

```
pca_coef(factor, Y)
```

Arguments

factor	An amSVD object from svd_factor.
Y	Numeric matrix or vector to project and reconstruct. Must have nrow(Y) equal to the number of rows of the original source matrix.

Value

Numeric matrix with the same dimensions as Y, containing the rank-k approximation.

Examples

```
m <- matrix(rnorm(30), nrow = 6)
A <- adgeMatrix(m)
fac <- svd_factor(A, k = 2L)
approx <- pca_coef(fac, m)
```

qr_downdate *QR downdate after removing one row*

Description

Updates a QR factorization to reflect the removal of a single row from the original matrix. The current implementation refits from the reduced matrix; a Givens-rotation path is planned for a future release.

Usage

```
qr_downdate(qr_factor, row_idx, X = NULL)
```

Arguments

qr_factor	An amQR factor object returned by am_qr(), or any object for which a method is defined.
row_idx	Positive integer index of the row to remove.
X	The original numeric matrix or adgeMatrix used to compute qr_factor. Required for amQR factors because they do not store the source matrix.

Value

An updated amQR factor with row row_idx excluded.

See Also

am_qr, qr_info, lm_loo_cv

Examples

```
## Not run:
X <- adgeMatrix(matrix(rnorm(40), nrow = 8))
qf <- am_qr(X)
qf2 <- qr_downdate(qf, row_idx = 3L, X = X)
qr_info(qf2)$dim

## End(Not run)
```

qr_info *Inspect an amQR factorization object*

Description

Returns a named list of metadata fields describing an amQR factor produced by am_qr().

Usage

```
qr_info(qr)
```

Arguments

qr	An amQR object.

Value

A named list with the following elements:

rank Integer effective rank of the factored matrix.

dim Integer vector of length 2: c(nrow, ncol) of the source matrix.

thin Logical; TRUE when a thin (economy) QR was computed.

pivoted Logical; TRUE when column pivoting was used.

pivot Integer permutation vector, or NULL when unpivoted.

representation Character string describing the internal storage format.

backend_ops Character string naming the backend that owns any resident buffers, or NULL.

backend Character string: the preferred backend.

precision Character string: "strict" or "fast".

method Character string: QR algorithm used.

compact_factor_available Logical.

compact_factor_source Character string or NULL.

compact_factor_materialized Logical.

q_materialized Logical.

r_materialized Logical.

See Also

am_qr, qr_downdate

Examples

```
X <- adgeMatrix(matrix(rnorm(30), nrow = 6))
qf <- am_qr(X)
info <- qr_info(qf)
info$rank
```

quad_form *Evaluate a quadratic form using a Cholesky factor*

Description

Computes `t(v) %*% solve(X) %*% v` (for a vector `v`) or `t(V) %*% solve(X) %*% V` (for a matrix `V`) efficiently via the Cholesky factor of `X`, without forming the inverse.

Usage

```
quad_form(factor, v)
```

Arguments

`factor`	An `amChol` object from `chol_factor`.
`v`	Numeric vector or matrix. For a vector, the result is a scalar; for a matrix with `p` columns, the result is a `p` x `p` matrix.

Value

Scalar double (when `v` is a vector) or numeric matrix of dimensions `ncol(v)` x `ncol(v)` containing the quadratic form.

Examples

```
m <- crossprod(matrix(rnorm(16), 4, 4)) + diag(4)
A <- adgeMatrix(m)
fac <- chol_factor(A)
v <- rnorm(4)
quad_form(fac, v)
```

resident_handle *Create a mutable GPU-resident handle*

Description

Wraps an `adgeMatrix` or plain matrix in a lightweight mutable environment that holds a GPU-resident buffer key. Unlike `adgeMatrix`, the handle can be updated in place, making it suitable for iterative algorithms that would otherwise incur per-step S4 object allocation overhead. The handle owns its resident key and releases the device buffer when garbage collected.

Usage

```
resident_handle(x, backend = NULL)
```

Arguments

x	An adgeMatrix or plain matrix. If x is already GPU-resident on backend, the existing device buffer is reused without re-uploading.
backend	Character string. Name of the backend to use. Defaults to x@preferred_backend for adgeMatrix inputs and "cpu" for plain matrices. The backend must support GPU residency.

Value

A resident_handle environment with fields backend_name, resident_key, dim, dimnames, policy, precision, and active.

See Also

am_sweep_inplace, rh_rowSums, rh_colSums

Examples

```
m <- adgeMatrix(matrix(runif(12), 3, 4), preferred_backend = "cpu")
# resident_handle requires a backend with residency support (e.g. MLX, OpenCL)
```

---

rh_colSums *Column sums of a GPU-resident handle*

Description

Computes column sums of the matrix stored in a resident_handle, using a GPU-resident reduction when the backend supports it. Falls back to base::colSums on the materialized matrix when no resident reduction is available.

Usage

```
rh_colSums(h)
```

Arguments

h	A resident_handle.

Value

Numeric vector of length ncol(h).

See Also

rh_rowSums, am_sweep_inplace

Examples

```
# requires a backend with residency support (e.g. MLX, OpenCL)
```

---

rh_rowSums *Row sums of a GPU-resident handle*

Description

Computes row sums of the matrix stored in a `resident_handle`, using a GPU-resident reduction when the backend supports it to avoid a round-trip download. Falls back to `base::rowSums` on the materialized matrix when no resident reduction is available.

Usage

```
rh_rowSums(h)
```

Arguments

h	A `resident_handle`.

Value

Numeric vector of length `nrow(h)`.

See Also

`rh_colSums`, `am_sweep_inplace`

Examples

```
# requires a backend with residency support (e.g. MLX, OpenCL)
```

`ridge_fit` *Fit a single ridge regression model*

Description

Solves the penalized least-squares problem $\min_\beta \|Y - X\beta\|^2 + \lambda\|\beta\|^2$ for a single penalty value `lambda`.

Usage

```
ridge_fit(
  X,
  Y,
  lambda,
  intercept = FALSE,
  penalize_intercept = FALSE,
  include_fitted = TRUE,
  include_residuals = TRUE,
  cache = TRUE
)
```

Arguments

`X`	Numeric matrix or `adgeMatrix` of predictors, shape `[n, p]`.
`Y`	Numeric matrix, vector, or `adgeMatrix` of responses, shape `[n, q]`.
`lambda`	Non-negative scalar ridge penalty.
`intercept`	Logical; when `TRUE` a column of ones is prepended to `X` before fitting.
`penalize_intercept`	Logical; when `FALSE` (default) the intercept coefficient is excluded from the penalty.
`include_fitted`	Logical; include fitted values in the result.
`include_residuals`	Logical; include residuals in the result.
`cache`	Logical; cache $X^T X$ for reuse across calls sharing the same `X`.

Value

An object of class `"ridge_fit"`, a named list containing:

coefficients `adgeMatrix` of shape `[p, q]`.

fitted.values `adgeMatrix` of shape `[n, q]`, or `NULL` when `include_fitted = FALSE`.

residuals `adgeMatrix` of shape `[n, q]`, or `NULL` when `include_residuals = FALSE`.

lambda The penalty value used.

rank Integer model rank.

df.residual Residual degrees of freedom.

See Also

ridge_path, lm_fit

Examples

```
X <- matrix(rnorm(50), nrow = 10)
y <- rnorm(10)
fit <- ridge_fit(X, y, lambda = 1)
coef(fit)
```

---

ridge_path *Compute a ridge regression solution path*

Description

Fits ridge regression for every penalty value in `lambdas` via a single thin SVD of `X`, returning coefficients for all penalties at once.

Usage

```
ridge_path(X, Y, lambdas, k = NULL, ...)
```

Arguments

`X`	Numeric matrix or `adgeMatrix` of predictors, shape `[n, p]`.
`Y`	Numeric matrix, vector, or `adgeMatrix` of responses, shape `[n, q]`.
`lambdas`	Positive numeric vector of ridge penalty values. Must satisfy `all(lambdas > 0)`.
`k`	Integer or `NULL`. Number of singular values to retain in the truncated SVD. When `NULL`, defaults to `min(nrow(X), ncol(X))`.
`...`	Additional arguments forwarded to `svd_factor`.

Value

An object of class `"ridge_path"`, a named list containing:

coef Numeric array of shape `[p, q, length(lambdas)]`; coefficient matrix for each penalty.

lambdas The input penalty vector.

svd The `amSVD` factor object used internally.

k Integer number of singular values actually used.

See Also

ridge_fit, svd_factor

Examples

```
X <- matrix(rnorm(60), nrow = 15)
y <- rnorm(15)
path <- ridge_path(X, y, lambdas = c(0.1, 1, 10))
dim(path$coef)
```

`rowmeans` *Row and column means*

Description

Compute row or column means of a matrix or `aMatrix`, dispatching to an accelerated backend when one is available.

Usage

```
rowmeans(x, na.rm = FALSE)

colmeans(x, na.rm = FALSE)
```

Arguments

`x`	A matrix or `aMatrix` object.
`na.rm`	Logical; if `TRUE`, `NA` values are excluded before averaging. Default `FALSE`.

Value

A numeric vector of length `nrow(x)` (`rowmeans`) or `ncol(x)` (`colmeans`).

Examples

```
m <- matrix(1:12, 3, 4)
rowmeans(m)
colmeans(m)
```

rowscale *Row and column diagonal scaling*

Description

Scale each row or column of a matrix by a numeric vector, equivalent to left- or right-multiplying by a diagonal matrix. `rowscale` computes `diag(d) %*% X` (row i scaled by `d[i]`); `colscale` computes `X %*% diag(d)` (column j scaled by `d[j]`).

Usage

```
rowscale(X, d)

colscale(X, d)
```

Arguments

`X`	A matrix or `aMatrix` object.
`d`	Numeric vector of scale factors. Length must equal `nrow(X)` for `rowscale` and `ncol(X)` for `colscale`.

Value

A matrix or `aMatrix` of the same dimensions as `X`.

Examples

```
m <- matrix(1:6, 2, 3)
rowscale(m, c(2, 0.5))
colscale(m, c(1, 2, 3))
```

rowsums *Row and column sums*

Description

Compute row or column sums of a matrix or `aMatrix`, dispatching to an accelerated backend when one is available.

Usage

```
rowsums(x, na.rm = FALSE, dims = 1L)

colsums(x, na.rm = FALSE, dims = 1L)
```

Arguments

x	A matrix or aMatrix object.
na.rm	Logical; if TRUE, missing values are removed before summing. Default FALSE.
dims	Integer; the number of dimensions to regard as rows (for rowsums) or columns (for colsums). Default 1L.

Value

A numeric vector of length nrow(x) (rowsums) or ncol(x) (colsums).

Examples

```
m <- adgeMatrix(matrix(1:12, 3, 4))
rowsums(m)
colsums(m)
```

---

rowSums,adgCMatrix-method

Row and column summary methods for adgCMatrix

Description

Compute row or column sums and means for a sparse adgCMatrix, dispatching through the amatrix backend when GPU acceleration is available.

Usage

```
## S4 method for signature 'adgCMatrix'
rowSums(x, na.rm = FALSE, dims = 1L)

## S4 method for signature 'adgCMatrix'
colSums(x, na.rm = FALSE, dims = 1L)

## S4 method for signature 'adgCMatrix'
rowMeans(x, na.rm = FALSE, dims = 1L)

## S4 method for signature 'adgCMatrix'
colMeans(x, na.rm = FALSE, dims = 1L)
```

Arguments

x	An adgCMatrix.
na.rm	Logical; if TRUE, NA values are ignored.
dims	Integer; dimensions to sum over (passed to the backend).

Value

A numeric vector of length equal to the number of rows or columns.

Examples

```
sp <- as(matrix(c(1, 0, 2, 0, 3, 0), 2, 3), "dgCMatrix")
A  <- adgCMatrix(sp)
rowSums(A)
colMeans(A)
```

---

rowSums,adgeMatrix-method

Row and column summary methods for adgeMatrix

Description

Compute row or column sums and means for an `adgeMatrix`, dispatching through the amatrix backend when GPU acceleration is available.

Usage

```
## S4 method for signature 'adgeMatrix'
rowSums(x, na.rm = FALSE, dims = 1L)

## S4 method for signature 'adgeMatrix'
colSums(x, na.rm = FALSE, dims = 1L)

## S4 method for signature 'adgeMatrix'
rowMeans(x, na.rm = FALSE, dims = 1L)

## S4 method for signature 'adgeMatrix'
colMeans(x, na.rm = FALSE, dims = 1L)
```

Arguments

`x`	An `adgeMatrix`.
`na.rm`	Logical; if `TRUE`, `NA` values are ignored.
`dims`	Integer; dimensions to sum over (passed to the backend).

Value

A numeric vector of length equal to the number of rows or columns.

Examples

```
A <- adgeMatrix(matrix(1:12, 3, 4))
rowSums(A)
colMeans(A)
```

`rsvd` *GPU-native randomized SVD (Halko et al. 2011)*

Description

Computes a truncated SVD via randomized projection entirely on the GPU. All QR, matmul, and SVD steps stay on device; a single `mlx_eval` materializes the results. Falls back to `irlba::svdr` on CPU if no GPU backend with rsvd support is active.

Usage

```
rsvd(x, k, n_oversamples = 10L, n_iter = 2L, ...)
```

Arguments

`x`	An `adgeMatrix` or plain numeric matrix.
`k`	Number of singular values/vectors to compute.
`n_oversamples`	Extra columns for the random projection (default 10). Increasing this improves accuracy at modest cost.
`n_iter`	Number of power-iteration passes (default 2). More passes give better accuracy for matrices with slowly decaying spectra.
`...`	Ignored (for forward compatibility).

Value

A list with components `u` (m x k), `d` (length-k singular values, decreasing), and `v` (n x k).

References

Halko, N., Martinsson, P. G., & Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2), 217-288.

See Also

`block_lanczos`, `eigh`

Examples

```
A <- matrix(rnorm(200), nrow = 20)
res <- rsvd(A, k = 3L)
length(res$d)
```

segment_mean *Segment mean by group labels*

Description

Compute the mean of rows of `x` grouped by integer `labels`, dispatching to GPU when available.

Usage

```
segment_mean(x, labels, K)
```

Arguments

`x`	A numeric matrix or `adgeMatrix`.
`labels`	Integer vector of group labels (1-based).
`K`	Number of groups.

Value

A `K`-by-`ncol(x)` matrix of group means.

See Also

`segment_sum`, `am_scatter_mean`

segment_sum *Segment sum by group labels*

Description

Sum rows of `x` grouped by integer `labels`, dispatching to GPU when available.

Usage

```
segment_sum(x, labels, K)
```

Arguments

`x`	A numeric matrix or `adgeMatrix`.
`labels`	Integer vector of group labels (1-based).
`K`	Number of groups.

Value

A `K`-by-`ncol(x)` matrix of group sums.

See Also

segment_mean, am_scatter_mean

---

sinkhorn *Doubly-stochastic scaling via Sinkhorn-Knopp iterations*

Description

Alternates row and column normalization until the matrix is approximately doubly stochastic. When the chosen backend supports resident broadcast and reduction kernels, the hot loop stays on device via resident_handle and returns a deferred adgeMatrix bound to the resident result.

Usage

```
sinkhorn(
  A,
  max_iter = 200L,
  tol = 1e-08,
  check_every = 5L,
  eps = 1e-15,
  mode = "fast",
  backend = NULL,
  return_info = FALSE
)
```

Arguments

A	A dense numeric matrix or adgeMatrix. Sparse inputs are not yet supported in this surface.
max_iter	Maximum number of Sinkhorn iterations.
tol	Convergence tolerance on the maximum row/column sum error.
check_every	Check convergence every check_every iterations.
eps	Floor applied to row/column sums before division.
mode	Execution mode used when coercing a plain matrix. Default "fast" allows accelerated backends to use lower precision.
backend	Backend name used when coercing a plain matrix. Ignored when A is already an adgeMatrix.
return_info	When TRUE, return convergence metadata alongside the scaled matrix.

Value

By default, an adgeMatrix. With return_info = TRUE, a list containing result, iterations, converged, row_error, col_error, backend, and method.

See Also

dist_matrix

Examples

```
A <- abs(matrix(rnorm(16), nrow = 4)) + 0.1
S <- sinkhorn(A, max_iter = 50L)
# Row sums should be close to 1
rowSums(as.matrix(S))
```

---

solve,adgCMatrix,missing-method

Solve a linear system for adgCMatrix

Description

Compute the solution to $ax = b$ or the inverse of a when b is missing, for a sparse adgCMatrix coefficient matrix.

Usage

```
## S4 method for signature 'adgCMatrix,missing'
solve(a, b, ...)

## S4 method for signature 'adgCMatrix,ANY'
solve(a, b, ...)
```

Arguments

a	An adgCMatrix coefficient matrix.
b	A matrix or vector right-hand side, or missing for inversion.
...	Further arguments passed to the backend.

Value

An adgeMatrix or adgCMatrix containing the solution.

solve,adgeMatrix,missing-method *Solve a linear system for adgeMatrix*

Description

Compute the solution to $ax = b$ or the matrix inverse of `a` when `b` is missing, dispatching through the amatrix backend.

Usage

```
## S4 method for signature 'adgeMatrix,missing'
solve(a, b, ...)

## S4 method for signature 'adgeMatrix,ANY'
solve(a, b, ...)
```

Arguments

`a`	An `adgeMatrix` coefficient matrix.
`b`	A matrix or vector right-hand side, or missing for matrix inversion.
`...`	Further arguments passed to the backend.

Value

An `adgeMatrix` (or numeric vector when `b` is a plain vector) containing the solution.

Examples

```
A <- adgeMatrix(crossprod(matrix(rnorm(9), 3, 3)) + 3 * diag(3))
solve(A)
```

solve_triangular *Solve a triangular linear system*

Description

Solves `R %*% x = B` (or `t(R) %*% x = B` when `lower = TRUE`) for `x`, where `R` is a triangular matrix. Dispatches to a GPU backend when `R` is an `amChol` or `adgeMatrix` with a live resident key and a capable backend.

Usage

```
solve_triangular(R, B, lower = FALSE)
```

Arguments

R	An amChol, adgeMatrix, or numeric matrix holding the triangular factor. Upper triangular by default.
B	Numeric vector or matrix; the right-hand side.
lower	Logical scalar; FALSE (default) treats R as upper triangular, TRUE treats it as lower triangular.

Value

Numeric vector or matrix x satisfying R %*% x == B (or its transpose variant). Returns a vector when B is a vector or single-column matrix.

Examples

```
R <- chol(crossprod(matrix(rnorm(16), 4, 4)) + diag(4))
b <- rnorm(4)
x <- solve_triangular(R, b)
```

---

svd-methods *Singular value decomposition for adgeMatrix*

Description

Compute the singular value decomposition of an adgeMatrix, dispatching through the amatrix backend. A fallback method for plain matrix objects is also provided to preserve base R behaviour after the generic is promoted to S4.

Usage

```
## S4 method for signature 'adgeMatrix'
svd(x, nu = min(dim(x)), nv = min(dim(x)), LINPACK = FALSE, ...)
```

Arguments

x	An adgeMatrix or plain matrix.
nu	Number of left singular vectors to compute.
nv	Number of right singular vectors to compute.
LINPACK	Ignored; retained for signature compatibility.
...	Further arguments passed to the backend.

Value

A list with components d (singular values), u (left singular vectors, nrow(x) by nu), and v (right singular vectors, ncol(x) by nv).

Examples

```
A <- adgeMatrix(matrix(rnorm(12), 4, 3))
s <- svd(A)
length(s$d)
```

---

svd,adgCMatrix-method *Singular value decomposition for adgCMatrix*

Description

Compute the singular value decomposition of a sparse adgCMatrix, dispatching through the amatrix backend.

Usage

```
## S4 method for signature 'adgCMatrix'
svd(x, nu = min(dim(x)), nv = min(dim(x)), LINPACK = FALSE, ...)
```

Arguments

x	An adgCMatrix.
nu	Number of left singular vectors to compute.
nv	Number of right singular vectors to compute.
LINPACK	Ignored; retained for signature compatibility.
...	Further arguments passed to the backend.

Value

A list with components d, u, and v.

---

svd_factor *Compute a truncated SVD of an aMatrix*

Description

Computes a rank-k truncated singular value decomposition of X, dispatching to the most suitable algorithm and backend based on matrix size, requested rank, and available hardware.

Usage

```
svd_factor(
  X,
  k = min(dim(X)),
  method = c("auto", "exact", "rsvd", "subspace"),
  n_oversamples = .amatrix_svd_factor_default_oversamples(k),
  n_iter = 2L
)
```

Arguments

X	An aMatrix (typically adgeMatrix or adgCMatrix).
k	Positive integer; the number of singular values and vectors to compute. Defaults to min(dim(X)).
method	One of "auto" (default), "exact", "rsvd", or "subspace". "auto" selects the algorithm based on matrix dimensions and the available backend.
n_oversamples	Non-negative integer; extra random vectors used to stabilize randomized algorithms. Ignored for method = "exact".
n_iter	Non-negative integer; number of power iterations for the subspace method. Ignored for method = "exact".

Value

An amSVD object containing slots u, d, v, and metadata.

Examples

```
m <- matrix(rnorm(30), nrow = 6)
A <- adgeMatrix(m)
fac <- svd_factor(A, k = 3L)
fac
```

---

svd_project *Project new data onto SVD left singular vectors*

Description

Computes t(U) %*% Y, where U is the matrix of left singular vectors stored in factor. Routes through a GPU backend when the factor was computed in "fast" precision and a device copy of t(U) is available.

Usage

```
svd_project(factor, Y)
```

Arguments

factor	An amSVD object from svd_factor.
Y	Numeric matrix or vector with nrow(Y) equal to the number of rows of the original source matrix.

Value

Numeric matrix of dimensions k x ncol(Y) containing the projected coordinates.

Examples

```
m <- matrix(rnorm(30), nrow = 6)
A <- adgeMatrix(m)
fac <- svd_factor(A, k = 2L)
coords <- svd_project(fac, m)
```

---

svd_reconstruct *Reconstruct data from SVD coordinates*

Description

Computes V %*% (Z / d), mapping coordinates in the rank-k SVD subspace back to the original column space. Routes through a GPU backend when the factor was computed in "fast" precision and a device copy of V is available.

Usage

```
svd_reconstruct(factor, Z)
```

Arguments

factor	An amSVD object from svd_factor.
Z	Numeric matrix or vector with nrow(Z) equal to factor@k.

Value

Numeric matrix of dimensions ncol(X) x ncol(Z), where X is the original source matrix.

Examples

```
m <- matrix(rnorm(30), nrow = 6)
A <- adgeMatrix(m)
fac <- svd_factor(A, k = 2L)
coords <- svd_project(fac, m)
approx <- svd_reconstruct(fac, coords)
```

sym *Symmetrise a matrix*

Description

Returns `(x + t(x)) / 2`, enforcing exact symmetry. Handles both dense `aMatrix` and sparse `adgCMatrix` inputs.

Usage

```
sym(x)
```

Arguments

`x`	A square matrix or `aMatrix` object.

Value

A symmetric matrix or `aMatrix` of the same class and dimensions as `x`.

Examples

```
m <- matrix(c(1, 2, 3, 4), 2, 2)
sym(m)
```

tcrossprod_weighted *Weighted outer cross-product XWX'*

Description

Computes $X\mathrm{diag}(w)X^T$, an `n` x `n` weighted outer cross-product. A GPU-resident fast path is used when available.

Usage

```
tcrossprod_weighted(X, w)
```

Arguments

`X`	Numeric matrix or `adgeMatrix` of shape `[n, p]`.
`w`	Positive numeric vector of length `n`; observation weights.

Value

An `adgeMatrix` of shape `[n, n]`.

See Also

crossprod_weighted, xty_weighted

Examples

```
X <- matrix(rnorm(20), nrow = 5)
w <- runif(5)
tcrossprod_weighted(X, w)
```

---

trace *Matrix trace*

Description

Returns the trace (sum of diagonal elements) of a square matrix or aMatrix.

Usage

```
trace(x)
```

Arguments

x	A square matrix, sparse sparseMatrix, or aMatrix.

Value

A single numeric scalar equal to the sum of diagonal elements.

Examples

```
trace(diag(1:4))
```

---

trace_estim *Stochastic trace estimator (Hutchinson)*

Description

Estimates $\mathrm{tr}(A)$ or $\mathrm{tr}(A^{-1})$ using Hutchinson's method with k Rademacher probe vectors. Supply solve_fn to estimate the trace of an inverse without forming it explicitly.

Usage

```
trace_estim(A = NULL, k = 30L, seed = NULL, solve_fn = NULL, n = NULL)
```

Arguments

A	Square matrix or aMatrix; required when solve_fn is NULL.
k	Integer number of Rademacher probe vectors. Default 30L.
seed	Optional integer random seed for reproducibility.
solve_fn	Optional function function(V) that returns A^{-1} %*% V; use this to estimate $\mathrm{tr}(A^{-1})$ without materialising the inverse.
n	Integer dimension of the matrix; required when solve_fn is supplied.

Value

A single numeric scalar estimate of the trace.

Examples

```
A <- crossprod(matrix(rnorm(25), 5, 5)) + 5 * diag(5)
trace_estim(A, k = 50L, seed = 1L)
```

---

with_amatrix *Evaluate code with temporary amatrix defaults*

Description

Temporarily overrides the session-default dispatch policy and/or precision mode for the duration of code, then restores the previous values on exit, even when code errors.

Usage

```
with_amatrix(policy = NULL, precision = NULL, code)
```

Arguments

policy	Optional temporary policy. Must be one of "auto", "cpu", "mlx", "metal", or "arrayfire".
precision	Optional temporary precision. Must be either "strict" or "fast".
code	Expression to evaluate under the temporary defaults.

Value

The result of evaluating code.

See Also

adgeMatrix, amatrix_set_default_policy, amatrix_set_default_precision

Examples

```
with_amatrix(policy = "auto", precision = "fast", {
  adgeMatrix(matrix(1:4, nrow = 2))
})
```

`wls_fit` *Fit a weighted least squares model*

Description

Solves the weighted least-squares problem $\min_\beta \sum_i w_i (y_i - x_i^T \beta)^2$ using either the normal equations or a QR decomposition of the weight-scaled design.

Usage

```
wls_fit(
  X,
  Y,
  weights,
  intercept = FALSE,
  include_fitted = TRUE,
  include_residuals = TRUE,
  cache = TRUE,
  method = c("normal", "qr")
)
```

Arguments

`X`	Numeric matrix or `adgeMatrix` of predictors, shape `[n, p]`.
`Y`	Numeric matrix, vector, or `adgeMatrix` of responses, shape `[n, q]`.
`weights`	Positive numeric vector of length `n`; observation weights.
`intercept`	Logical; when `TRUE` a column of ones is prepended to `X` before fitting.
`include_fitted`	Logical; include fitted values in the result.
`include_residuals`	Logical; include residuals in the result.
`cache`	Logical; cache intermediate factorizations for reuse across calls sharing the same `X` and `weights`.
`method`	Solver method: `"normal"` (weighted normal equations, default) or `"qr"` (QR on the weight-scaled design).

Value

An object of class "wls_fit", a named list containing:

coefficients adgeMatrix of shape [p, q].

fitted.values adgeMatrix of shape [n, q], or NULL when include_fitted = FALSE.

residuals adgeMatrix of shape [n, q], or NULL when include_residuals = FALSE.

rank Integer model rank.

df.residual Residual degrees of freedom.

See Also

lm_fit, crossprod_weighted

Examples

```
X <- matrix(rnorm(50), nrow = 10)
y <- rnorm(10)
w <- runif(10, 0.5, 2)
fit <- wls_fit(X, y, weights = w)
coef(fit)
```

---

woodbury_logdet *Log-determinant via the Woodbury matrix determinant lemma*

Description

Computes $\log|A + UCV|$ using the matrix determinant lemma in $O(nk^2 + k^3)$ time, reusing an existing Cholesky factor of A.

Usage

```
woodbury_logdet(A_factor, U, V = NULL, C_inv = NULL)
```

Arguments

A_factor	An amChol object from chol_factor(), or a square numeric matrix that is automatically Cholesky-factored.
U	Numeric matrix of shape [n, k]; low-rank left factor.
V	Numeric matrix of shape [k, n]; low-rank right factor. Defaults to t(U) (symmetric update).
C_inv	Numeric matrix of shape [k, k]; inverse of the central factor C. Defaults to diag(k).

Value

A length-1 numeric: $\log|A + UCV|$.

See Also

woodbury_solve, chol_factor

Examples

```
A <- crossprod(matrix(rnorm(25), 5)) + diag(5)
U <- matrix(rnorm(10), 5, 2)
ld <- woodbury_logdet(A, U)
is.finite(ld)
```

`woodbury_solve` *Solve a linear system using the Woodbury matrix identity*

Description

Computes $(A + UCV)^{-1}b$ in $O(nk^2 + k^3)$ time using the Woodbury matrix identity, avoiding an $O(n^3)$ refactorisation of the updated matrix.

Usage

```
woodbury_solve(A_factor, U, b, V = NULL, C_inv = NULL)
```

Arguments

`A_factor`	An `amChol` object from `chol_factor()`, or a square numeric matrix that is automatically Cholesky-factored.
`U`	Numeric matrix of shape `[n, k]`; low-rank left factor.
`b`	Numeric matrix of shape `[n, rhs]`; right-hand side(s).
`V`	Numeric matrix of shape `[k, n]`; low-rank right factor. Defaults to `t(U)` (symmetric update).
`C_inv`	Numeric matrix of shape `[k, k]`; inverse of the central factor C. Defaults to `diag(k)` (pure rank-k update with $C = I$).

Value

Numeric matrix of shape `[n, rhs]`: the solution $(A + UCV)^{-1}b$.

See Also

woodbury_logdet, chol_factor

Examples

```
A <- crossprod(matrix(rnorm(25), 5)) + diag(5)
U <- matrix(rnorm(10), 5, 2)
b <- rnorm(5)
x <- woodbury_solve(A, U, b)
length(x)
```

`xty_weighted` *Weighted cross-product X'Wy*

Description

Computes $X^T \mathrm{diag}(w) y$, a `p` x `k` weighted cross-product between `X` and response matrix `y`. A GPU-resident fast path is used when available.

Usage

```
xty_weighted(X, w, y)
```

Arguments

`X`	Numeric matrix or `adgeMatrix` of shape `[n, p]`.
`w`	Positive numeric vector of length `n`; observation weights.
`y`	Numeric vector or matrix of shape `[n, k]`; response(s).

Value

An `adgeMatrix` of shape `[p, k]`.

See Also

`crossprod_weighted`, `tcrossprod_weighted`

Examples

```
X <- matrix(rnorm(20), nrow = 5)
w <- runif(5)
y <- rnorm(5)
xty_weighted(X, w, y)
```

Index