Package 'riemtan'

November 10, 2025

Title Riemannian Metrics for Symmetric Positive Definite Matrices

Version 0.2.5

Description Implements various Riemannian metrics for symmetric positive definite matrices, including AIRM (Affine Invariant Riemannian Metric, <doi:10.1007/s11263-005-3222-z>), Log-Euclidean (<doi:10.1002/mrm.20965>), Euclidean, Log-Cholesky (<doi:10.1137/18M1221084>), and Bures-Wasserstein metrics (<doi:10.1016/j.exmath.2018.01.002>). Provides functions for computing logarithmic and exponential maps, vectorization, and statistical operations on the manifold of positive definite matrices.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

LinkingTo Rcpp, RcppEigen

Depends R (>= 4.3.0)

Imports arrow, future, furrr, jsonlite, Matrix, methods, R6, purrr, MASS, matrixStats, Rcpp (>= 1.0.0)

Suggests progressr, testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

URL https://nicoesve.github.io/riemtan/

BugReports https://github.com/nicoesve/riemtan/issues

Config/testthat/edition 3

Maintainer Nicolas Escobar <nescoba@iu.edu>

NeedsCompilation yes

Author Nicolas Escobar [aut, cre] (ORCID: <https://orcid.org/0009-0006-0800-5692>), Jaroslaw Harezlak [ths]

Repository CRAN

Date/Publication 2025-11-10 21:20:02 UTC

Contents

`airm_exp` *Compute the AIRM Exponential*

Description

This function computes the Riemannian exponential map for the Affine-Invariant Riemannian Metric (AIRM).

Usage

```
airm_exp(sigma, v, validate = FALSE)
```

Arguments

`sigma`	A symmetric positive-definite matrix of class `dppMatrix`, representing the reference point.
`v`	A tangent vector of class `dspMatrix`, to be mapped back to the manifold at `sigma`.
`validate`	A logical value indicating whether to validate input arguments. Default is FALSE.

Value

A symmetric positive-definite matrix of class `dppMatrix`.

Examples

```
if (requireNamespace("Matrix", quietly = TRUE)) {
  library(Matrix)
  sigma <- diag(2) |>
    Matrix::nearPD() |>
    _$mat |>
    Matrix::pack()
  v <- diag(c(1, 0.5)) |>
    Matrix::symmpart() |>
    Matrix::pack()
  airm_exp(sigma, v)
}
```

`airm_log` *Compute the AIRM Logarithm*

Description

This function computes the Riemannian logarithmic map for the Affine-Invariant Riemannian Metric (AIRM).

Usage

```
airm_log(sigma, lambda, validate = FALSE)
```

Arguments

`sigma`	A symmetric positive-definite matrix of class `dppMatrix`, representing the reference point.
`lambda`	A symmetric positive-definite matrix of class `dppMatrix`, representing the target point.
`validate`	A logical value indicating whether to validate input arguments. Default is FALSE.

Value

A symmetric matrix of class `dspMatrix`, representing the tangent space image of `lambda` at `sigma`.

Examples

```
if (requireNamespace("Matrix", quietly = TRUE)) {
  library(Matrix)
  sigma <- diag(2) |>
    Matrix::nearPD() |>
    _$mat |>
    Matrix::pack()
  lambda <- diag(c(2, 3)) |>
    Matrix::nearPD() |>
    _$mat |>
```

```
    Matrix::pack()
  airm_log(sigma, lambda)
}
```

`airm_unvec` *Compute the Inverse Vectorization (AIRM)*

Description

Converts a vector back into a tangent matrix relative to a reference point using AIRM.

Usage

```
airm_unvec(sigma, w)
```

Arguments

`sigma`	A symmetric positive-definite matrix of class `dppMatrix`, representing the reference point.
`w`	A numeric vector, representing the vectorized tangent image.

Value

A symmetric matrix of class `dspMatrix`, representing the tangent vector.

Examples

```
if (requireNamespace("Matrix", quietly = TRUE)) {
  library(Matrix)
  sigma <- diag(2) |>
    Matrix::nearPD() |>
    _$mat |>
    Matrix::pack()
  w <- c(1, sqrt(2), 2)
  airm_unvec(sigma, w)
}
```

airm_vec *Compute the AIRM Vectorization of Tangent Space*

Description

Vectorizes a tangent matrix into a vector in Euclidean space using AIRM.

Usage

```
airm_vec(sigma, v)
```

Arguments

sigma	A symmetric positive-definite matrix of class dppMatrix, representing the reference point.
v	A symmetric matrix of class dspMatrix, representing a tangent vector.

Value

A numeric vector, representing the vectorized tangent image.

Examples

```
if (requireNamespace("Matrix", quietly = TRUE)) {
  library(Matrix)
  sigma <- diag(2) |>
    Matrix::nearPD() |>
    _$mat |>
    Matrix::pack()
  v <- diag(c(1, 0.5)) |>
    Matrix::symmpart() |>
    Matrix::pack()
  airm_vec(sigma, v)
}
```

bures_wasserstein_exp *Compute the Bures-Wasserstein Exponential*

Description

This function computes the Riemannian exponential map using the Bures-Wasserstein metric for symmetric positive-definite matrices. The map operates by solving a Lyapunov equation and then constructing the exponential.

Usage

```
bures_wasserstein_exp(sigma, v, validate = FALSE)
```

Arguments

sigma	A symmetric positive-definite matrix of class dppMatrix, representing the reference point.
v	A symmetric matrix of class dspMatrix, representing the tangent vector to be mapped.
validate	A logical value indicating whether to validate input arguments. Default is FALSE.

Value

A symmetric positive-definite matrix of class dppMatrix, representing the point on the manifold.

---

bures_wasserstein_log *Compute the Bures-Wasserstein Logarithm*

Description

This function computes the Riemannian logarithmic map using the Bures-Wasserstein metric for symmetric positive-definite matrices.

Usage

```
bures_wasserstein_log(sigma, lambda, validate = FALSE)
```

Arguments

sigma	A symmetric positive-definite matrix of class dppMatrix, representing the reference point.
lambda	A symmetric positive-definite matrix of class dppMatrix, representing the target point.
validate	A logical value indicating whether to validate input arguments. Default is FALSE.

Value

A symmetric matrix of class dspMatrix, representing the tangent space image of lambda at sigma.

bures_wasserstein_unvec *Compute the Bures-Wasserstein Inverse Vectorization*

Description

Compute the Bures-Wasserstein Inverse Vectorization

Usage

```
bures_wasserstein_unvec(sigma, w)
```

Arguments

sigma	A symmetric positive-definite matrix of class dppMatrix
w	A numeric vector representing the vectorized tangent image

Value

A symmetric matrix of class dspMatrix

bures_wasserstein_vec *Compute the Bures-Wasserstein Vectorization*

Description

Compute the Bures-Wasserstein Vectorization

Usage

```
bures_wasserstein_vec(sigma, v)
```

Arguments

sigma	A symmetric positive-definite matrix of class dppMatrix
v	A symmetric matrix of class dspMatrix

Value

A numeric vector representing the vectorized tangent image

compute_frechet_mean *Compute the Frechet Mean*

Description

This function computes the Frechet mean of a sample using an iterative algorithm with optional parallel processing.

Usage

```
compute_frechet_mean(
  sample,
  tol = 0.05,
  max_iter = 20,
  lr = 0.2,
  batch_size = 32,
  progress = FALSE
)
```

Arguments

sample	An object of class CSample containing the sample data.
tol	A numeric value specifying the tolerance for convergence. Default is 0.05.
max_iter	An integer specifying the maximum number of iterations. Default is 20.
lr	A numeric value specifying the learning rate. Default is 0.2.
batch_size	Integer. The number of samples to process in each batch during computation. Default is 32.
progress	Logical indicating whether to show progress during computation (default: FALSE). Requires progressr package.

Details

The function iteratively updates the reference point of the sample until the change in the reference point is less than the specified tolerance or the maximum number of iterations is reached. If the tangent images are not already computed, they will be computed before starting the iterations.

When parallel processing is enabled (via set_parallel_plan()), the relocate() function will use parallel processing for relocating tangent images in each iteration, which can significantly speed up computation for large samples.

Value

The computed Frechet mean as a dppMatrix object.

Examples

```
if (requireNamespace("Matrix", quietly = TRUE)) {
  library(Matrix)
  # Load the AIRM metric object
  data(airm)
  # Create a CSample object with example data
  conns <- list(
    diag(2) |> Matrix::nearPD() |> _$mat |> Matrix::pack(),
    diag(c(2, 3)) |> Matrix::nearPD() |> _$mat |> Matrix::pack()
  )
  sample <- CSample$new(conns = conns, metric_obj = airm)
  # Compute the Frechet mean
  compute_frechet_mean(sample, tol = 0.01, max_iter = 50, lr = 0.1)
}
```

---

configure_progress *Configure Progress Handlers*

Description

Convenience function to configure progressr handlers for the current session.

Usage

```
configure_progress(handler = "txtprogressbar", ...)
```

Arguments

handler	Character string specifying the handler type: • "progress": progress package style (if available) • "txtprogressbar": base R text progress bar • "cli": cli package style (if available) • "none": Disable progress reporting • NULL: Use progressr default handlers
...	Additional arguments passed to progressr::handlers()

Details

This is a convenience wrapper around progressr::handlers(). If progressr is not available, a warning is issued and the function returns NULL.

Value

Invisibly returns the configured handlers (via progressr::handlers()).

See Also

progressr::handlers()

Examples

```
## Not run:
# Use base R text progress bar
configure_progress("txtprogressbar")

# Use cli style (if cli package is installed)
configure_progress("cli")

# Disable progress reporting
configure_progress("none")

## End(Not run)
```

`create_parquet_backend` *Create ParquetBackend from Directory*

Description

Convenience function to create a ParquetBackend from a validated directory.

Usage

```
create_parquet_backend(data_dir, cache_size = 10, validate = TRUE)
```

Arguments

`data_dir`	Path to directory containing Parquet files and metadata
`cache_size`	Maximum number of matrices to cache in memory (default: 10)
`validate`	If TRUE, validates directory before creating backend (default: TRUE)

Value

A ParquetBackend object

Examples

```
## Not run:
backend <- create_parquet_backend("my_connectomes")
sample <- CSample$new(backend = backend, metric_obj = airm())

## End(Not run)
```

CSample *CSample Class*

Description

This class represents a sample of connectomes, with various properties and methods to handle their tangent and vectorized images.

Active bindings

`connectomes` Connectomes data

`tangent_images` Tangent images data

`vector_images` Vector images data

`sample_size` Sample size

`matrix_size` Matrix size

`mfd_dim` Manifold dimension

`is_centered` Centering status

`frechet_mean` Frechet mean

`riem_metric` Riemannian Metric used

`variation` Variation of the sample

`sample_cov` Sample covariance

`ref_point` Reference point for tangent or vectorized images

`distances` Distances to the Frechet mean

`batch_size` Batch size for compute_fmean

Methods

Public methods:

- `CSample$new()`
- `CSample$load_connectomes_batched()`
- `CSample$compute_tangents()`
- `CSample$compute_conns()`
- `CSample$compute_vecs()`
- `CSample$compute_unvecs()`
- `CSample$compute_fmean()`
- `CSample$change_ref_pt()`
- `CSample$center()`
- `CSample$compute_variation()`
- `CSample$compute_dists()`
- `CSample$compute_sample_cov()`
- `CSample$clone()`

Method `new()`**:** Initialize a CSample object

Usage:

```
CSample$new(
  conns = NULL,
  tan_imgs = NULL,
  vec_imgs = NULL,
  centered = NULL,
  ref_pt = NULL,
  metric_obj,
  batch_size = NULL,
  backend = NULL
)
```

Arguments:

`conns` A list of connectomes (default is NULL).

`tan_imgs` A list of tangent images (default is NULL).

`vec_imgs` A matrix whose rows are vectorized images (default is NULL).

`centered` Boolean indicating whether tangent or vectorized images are centered (default is NULL).

`ref_pt` A connectome (default is identity)

`metric_obj` Object of class `rmetric` representing the Riemannian metric used.

`batch_size` The batch size for compute_fmean (default is 32).

`backend` A DataBackend object (ListBackend or ParquetBackend). If NULL and conns is provided, a ListBackend will be created automatically.

Returns: A new `CSample` object.

Method `load_connectomes_batched()`**:** Load connectomes in parallel batches from ParquetBackend. This method is particularly useful for large Parquet-backed datasets where loading all matrices at once would be memory-prohibitive.

Usage:

```
CSample$load_connectomes_batched(
  indices = NULL,
  batch_size = 50,
  progress = FALSE
)
```

Arguments:

`indices` Optional vector of indices to load. If NULL (default), loads all matrices.

`batch_size` Number of matrices to load per batch (default: 50). Larger batches use more memory but may be faster.

`progress` Logical indicating whether to show progress (default: FALSE).

Details: This method only works when the CSample was initialized with a ParquetBackend. It loads matrices in parallel batches, clearing the cache between batches to manage memory usage. Sequential loading is used for ListBackend.

Returns: A list of dppMatrix objects.

Examples:

```
\dontrun{
# Create CSample with ParquetBackend
backend <- create_parquet_backend("my_data")
sample <- CSample$new(backend = backend, metric_obj = airm)

# Load first 100 matrices in batches of 20
conns <- sample$load_connectomes_batched(indices = 1:100, batch_size = 20)
}
```

Method `compute_tangents()`**:** This function computes the tangent images from the connectomes.

Usage:

```
CSample$compute_tangents(ref_pt = default_ref_pt(private$p), progress = FALSE)
```

Arguments:

`ref_pt` A reference point, which must be a `dppMatrix` object (default is `default_ref_pt`).

`progress` Logical indicating whether to show progress (default: FALSE).

Details: Error if `ref_pt` is not a `dppMatrix` object or if `conns` is not specified.

Returns: None

Method `compute_conns()`**:** This function computes the connectomes from the tangent images.

Usage:

```
CSample$compute_conns(progress = FALSE)
```

Arguments:

`progress` Logical indicating whether to show progress (default: FALSE).

Details: Error if tangent images are not specified.

Returns: None

Method `compute_vecs()`**:** This function computes the vectorized tangent images from the tangent images.

Usage:

```
CSample$compute_vecs(progress = FALSE)
```

Arguments:

`progress` Logical indicating whether to show progress (default: FALSE).

Details: Error if tangent images are not specified.

Returns: None

Method `compute_unvecs()`**:** This function computes the tangent images from the vector images.

Usage:

```
CSample$compute_unvecs(progress = FALSE)
```

Arguments:

`progress` Logical indicating whether to show progress (default: FALSE).

Details: Error if `vec_imgs` is not specified.

Returns: None

Method `compute_fmean()`**:** This function computes the Frechet mean of the sample.

Usage:

```
CSample$compute_fmean(
  tol = 0.001,
  max_iter = 100,
  lr = 0.2,
  batch_size = NULL,
  progress = FALSE
)
```

Arguments:

`tol` Tolerance for the convergence of the mean (default is 0.05).

`max_iter` Maximum number of iterations for the computation (default is 20).

`lr` Learning rate for the optimization algorithm (default is 0.2).

`batch_size` The batch size (default is the instance's batch_size).

`progress` Logical indicating whether to show progress (default: FALSE).

Returns: None

Method `change_ref_pt()`**:** This function changes the reference point for the tangent images.

Usage:

```
CSample$change_ref_pt(new_ref_pt, progress = FALSE)
```

Arguments:

`new_ref_pt` A new reference point, which must be a `dppMatrix` object.

`progress` Logical indicating whether to show progress (default: FALSE).

Details: Error if tangent images have not been computed or if `new_ref_pt` is not a `dppMatrix` object.

Returns: None

Method `center()`**:** Center the sample

Usage:

```
CSample$center()
```

Details: This function centers the sample by computing the Frechet mean if it is not already computed, and then changing the reference point to the computed Frechet mean. Error if tangent images are not specified. Error if the sample is already centered.

Returns: None. This function is called for its side effects.

Method `compute_variation()`**:** Compute Variation

Usage:

```
CSample$compute_variation()
```

Details: This function computes the variation of the sample. It first checks if the vector images are null, and if so, it computes the vectors, computing first the tangent images if necessary. If the sample is not centered, it centers the sample and recomputes the vectors. Finally, it calculates the variation as the mean of the sum of squares of the vector images. Error if `vec_imgs` is not specified.

Returns: None. This function is called for its side effects.

Method `compute_dists()`**:** Compute distances

Usage:

```
CSample$compute_dists()
```

Details: This function computes the distances of the elements of the sample to the Frechet mean. It first checks if the vector images are null, and if so, it computes the vectors, computing first the tangent images if necessary. If the sample is not centered, it centers the sample and recomputes the vectors. Finally, it calculates the distances as the Euclidean norms of the vector images. Error if `vec_imgs` is not specified.

Returns: None. This function is called for its side effects.

Method `compute_sample_cov()`**:** Compute Sample Covariance

Usage:

```
CSample$compute_sample_cov()
```

Details: This function computes the sample covariance matrix for the vector images. It first checks if the vector images are null, and if so, it computes the vectors, computing first the tangent images if necessary.

Returns: None. This function is called for its side effects.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CSample$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `CSample$load_connectomes_batched`
## ------------------------------------------------

## Not run:
# Create CSample with ParquetBackend
backend <- create_parquet_backend("my_data")
sample <- CSample$new(backend = backend, metric_obj = airm)

# Load first 100 matrices in batches of 20
conns <- sample$load_connectomes_batched(indices = 1:100, batch_size = 20)

## End(Not run)
```

CSuperSample *CSuperSample Class*

Description

This class represents a collection of CSample objects (samples of connectomes), providing methods to aggregate, analyze, and compute statistics across multiple samples sharing the same Riemannian metric.

Active bindings

`list_of_samples` The list of CSample objects aggregated in this super-sample.

`sample_size` The total number of connectomes in all samples.

`matrix_size` The size of the connectome matrices.

`mfd_dim` The dimension of the manifold.

`riem_metric` The Riemannian metric used by all samples.

`variation` The total variation of the aggregated sample.

`sample_cov` The sample covariance matrix of the aggregated sample.

`full_sample` The aggregated CSample object containing all connectomes.

`frechet_mean` The Frechet mean of the aggregated sample.

`Within` The within-group covariance matrix (W).

`Total` The total covariance matrix (T).

Methods

Public methods:

- `CSuperSample$new()`
- `CSuperSample$compute_variation()`
- `CSuperSample$compute_sample_cov()`
- `CSuperSample$gather()`
- `CSuperSample$compute_fmean()`
- `CSuperSample$compute_W()`
- `CSuperSample$compute_T()`
- `CSuperSample$clone()`

Method `new()`**:** Initialize a CSuperSample object

Usage:

```
CSuperSample$new(samples)
```

Arguments:

`samples` A list of CSample objects. All must use the same Riemannian metric.

Returns: A new `CSuperSample` object.

Method `compute_variation()`**:** Compute the total variation of the aggregated sample.

Usage:

```
CSuperSample$compute_variation()
```

Returns: None. This function is called for its side effects. The result is stored in the `variation` active binding.

Method `compute_sample_cov()`**:** Compute the sample covariance matrix of the aggregated sample.

Usage:

```
CSuperSample$compute_sample_cov()
```

Returns: None. This function is called for its side effects. The result is stored in the `sample_cov` active binding.

Method `gather()`**:** Gather all connectomes from the list of samples into a single CSample object.

Usage:

```
CSuperSample$gather()
```

Returns: None. This function is called for its side effects. The result is stored in the `full_sample` active binding.

Method `compute_fmean()`**:** Compute the Frechet mean of the aggregated sample.

Usage:

```
CSuperSample$compute_fmean(batch_size = NULL)
```

Arguments:

`batch_size` Optional batch size parameter passed to the underlying compute_fmean function.

Returns: None. This function is called for its side effects. The result is stored in the `frechet_mean` active binding.

Method `compute_W()`**:** Compute the within-group covariance matrix (W) for the samples.

Usage:

```
CSuperSample$compute_W()
```

Returns: None. This function is called for its side effects. The result is stored in the `Within` active binding.

Method `compute_T()`**:** Compute the total covariance matrix (T) for the samples.

Usage:

```
CSuperSample$compute_T()
```

Returns: None. This function is called for its side effects. The result is stored in the `Total` active binding.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CSuperSample$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

default_ref_pt *Default reference point*

Description

Default reference point

Usage

```
default_ref_pt(p)
```

Arguments

p	the dimension

Value

A diagonal matrix of the desired dimension

dexp *Differential of Matrix Exponential Map*

Description

Computes the differential of the matrix exponential map located at a point a, evaluated at x

Usage

```
dexp(a, x)
```

Arguments

a	A symmetric matrix of class dspMatrix
x	A symmetric matrix representing tangent vector of class dspMatrix

Value

A positive definite symmetric matrix representing the differential located at a and evaluated at x, of class dppMatrix

dlog *Differential of Matrix Logarithm Map*

Description

Computes the differential of the matrix logarithm map at a point Sigma, evaluated at H

Usage

```
dlog(sigma, h)
```

Arguments

sigma	A symmetric positive definite matrix of class dspMatrix
h	A symmetric matrix representing tangent vector of class dsyMatrix

Value

A symmetric matrix representing the differential evaluated at H of class dsyMatrix

euclidean_exp *Compute the Euclidean Exponential*

Description

Compute the Euclidean Exponential

Usage

```
euclidean_exp(sigma, v, validate = FALSE)
```

Arguments

sigma	A reference point.
v	A tangent vector to be mapped back to the manifold at sigma.
validate	A logical value indicating whether to validate input arguments. Default is FALSE.

Value

The point on the manifold corresponding to the tangent vector at sigma.

euclidean_log *Compute the Euclidean Logarithm*

Description

Compute the Euclidean Logarithm

Usage

```
euclidean_log(sigma, lambda, validate = FALSE)
```

Arguments

sigma	A reference point.
lambda	A point on the manifold.
validate	A logical value indicating whether to validate input arguments. Default is FALSE.

Value

The tangent space image of `lambda` at `sigma`.

euclidean_unvec *Compute the Inverse Vectorization (Euclidean)*

Description

Converts a vector back into a tangent matrix relative to a reference point using Euclidean metric.

Usage

```
euclidean_unvec(sigma, w)
```

Arguments

sigma	A symmetric matrix.
w	A numeric vector, representing the vectorized tangent image.

Value

A symmetric matrix, representing the tangent vector.

euclidean_vec *Vectorize at Identity Matrix (Euclidean)*

Description

Converts a symmetric matrix into a vector representation.

Usage

```
euclidean_vec(sigma, v)
```

Arguments

sigma	A symmetric matrix.
v	A vector.

Value

A numeric vector, representing the vectorized tangent image.

get_n_workers *Get Current Number of Parallel Workers*

Description

Returns the number of parallel workers configured in the current future plan.

Usage

```
get_n_workers()
```

Value

Integer specifying the number of workers, or 1 if sequential processing is enabled.

See Also

set_parallel_plan()

Examples

```
## Not run:
set_parallel_plan("multisession", workers = 4)
get_n_workers()  # Returns 4

set_parallel_plan("sequential")
get_n_workers()  # Returns 1

## End(Not run)
```

`half_underscore` *Half-underscore operation for use in the log-Cholesky metric*

Description

Half-underscore operation for use in the log-Cholesky metric

Usage

```
half_underscore(x)
```

Arguments

`x`	A symmetric matrix (object of class dsyMatrix)

Value

The strictly lower triangular part of the matrix, plus half its diagonal part

---

`id_matr` *Create an Identity Matrix*

Description

Create an Identity Matrix

Usage

```
id_matr(sigma)
```

Arguments

`sigma`	A matrix.

Value

An identity matrix of the same dimensions as `sigma`.

is_parallel_enabled *Check if Parallel Processing is Enabled*

Description

Checks whether parallel processing is currently enabled based on the future plan.

Usage

```
is_parallel_enabled()
```

Value

Logical value: `TRUE` if parallel processing is enabled, `FALSE` if sequential.

See Also

`set_parallel_plan()`, `should_parallelize()`

Examples

```
## Not run:
# Check current status
is_parallel_enabled()

# Enable parallel processing
set_parallel_plan("multisession")
is_parallel_enabled()  # Returns TRUE

# Disable parallel processing
set_parallel_plan("sequential")
is_parallel_enabled()  # Returns FALSE

## End(Not run)
```

is_progress_available *Check if Progress Reporting is Available*

Description

Checks whether the progressr package is available and can be used for progress reporting.

Usage

```
is_progress_available()
```

Value

Logical value: TRUE if progressr is available, FALSE otherwise.

Examples

```
if (is_progress_available()) {
  message("Progress reporting is available")
}
```

`ListBackend` *ListBackend Class*

Description

Backend implementation using in-memory list storage. This wraps the existing list-based storage mechanism for backwards compatibility.

Super class

`riemtan::DataBackend` -> `ListBackend`

Methods

Public methods:

- `ListBackend$new()`
- `ListBackend$get_matrix()`
- `ListBackend$get_all_matrices()`
- `ListBackend$length()`
- `ListBackend$get_dimensions()`
- `ListBackend$clone()`

Method `new()`**:** Initialize a ListBackend

Usage:

```
ListBackend$new(matrices)
```

Arguments:

`matrices` A list of dppMatrix objects

Method `get_matrix()`**:** Get a specific matrix by index

Usage:

```
ListBackend$get_matrix(i)
```

Arguments:

`i` Integer index

Returns: A dppMatrix object

Method `get_all_matrices()`**:** Get all matrices

Usage:

```
ListBackend$get_all_matrices()
```

Returns: A list of dppMatrix objects

Method `length()`**:** Get the number of matrices

Usage:

```
ListBackend$length()
```

Returns: Integer count

Method `get_dimensions()`**:** Get matrix dimensions

Usage:

```
ListBackend$get_dimensions()
```

Returns: Integer p (matrices are p x p)

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ListBackend$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`log_cholesky_exp` *Compute the Log-Cholesky Exponential*

Description

This function computes the Riemannian exponential map using the Log-Cholesky metric for symmetric positive-definite matrices. The map operates by transforming the tangent vector via Cholesky decomposition of the reference point.

Usage

```
log_cholesky_exp(sigma, v, validate = FALSE)
```

Arguments

`sigma`	A symmetric positive-definite matrix of class `dppMatrix`, representing the reference point.
`v`	A symmetric matrix of class `dspMatrix`, representing the tangent vector to be mapped.
`validate`	A logical value indicating whether to validate input arguments. Default is FALSE.

Value

A symmetric positive-definite matrix of class `dppMatrix`, representing the point on the manifold.

log_cholesky_log *Compute the Log-Cholesky Logarithm*

Description

This function computes the Riemannian logarithmic map using the Log-Cholesky metric for symmetric positive-definite matrices. The Log-Cholesky metric operates by transforming matrices via their Cholesky decomposition.

Usage

```
log_cholesky_log(sigma, lambda, validate = FALSE)
```

Arguments

sigma	A symmetric positive-definite matrix of class dppMatrix, representing the reference point.
lambda	A symmetric positive-definite matrix of class dppMatrix, representing the target point.
validate	A logical value indicating whether to validate input arguments. Default is FALSE.

Value

A symmetric matrix of class dspMatrix, representing the tangent space image of lambda at sigma.

log_cholesky_unvec *Compute the Log-Cholesky Inverse Vectorization*

Description

Compute the Log-Cholesky Inverse Vectorization

Usage

```
log_cholesky_unvec(sigma, w)
```

Arguments

sigma	A symmetric positive-definite matrix of class dppMatrix
w	A numeric vector representing the vectorized tangent image

Value

A symmetric matrix of class dspMatrix

log_cholesky_vec *Compute the Log-Cholesky Vectorization*

Description

Compute the Log-Cholesky Vectorization

Usage

```
log_cholesky_vec(sigma, v)
```

Arguments

sigma	A symmetric positive-definite matrix of class dppMatrix
v	A symmetric matrix of class dspMatrix

Value

A numeric vector representing the vectorized tangent image

log_euclidean_exp *Compute the Log-Euclidean Exponential*

Description

This function computes the Euclidean exponential map.

Usage

```
log_euclidean_exp(ref_pt, v, validate = FALSE)
```

Arguments

ref_pt	A reference point.
v	A tangent vector to be mapped back to the manifold at ref_pt.
validate	A logical value indicating whether to validate input arguments. Default is FALSE.

Value

The point on the manifold corresponding to the tangent vector at ref_pt.

log_euclidean_log *Compute the Log-Euclidean Logarithm*

Description

Compute the Log-Euclidean Logarithm

Usage

```
log_euclidean_log(sigma, lambda, validate = FALSE)
```

Arguments

sigma	A reference point.
lambda	A point on the manifold.
validate	A logical value indicating whether to validate input arguments. Default is FALSE.

Value

The tangent space image of `lambda` at `sigma`.

log_euclidean_unvec *Compute the Inverse Vectorization (Euclidean)*

Description

Converts a vector back into a tangent matrix relative to a reference point using Euclidean metric.

Usage

```
log_euclidean_unvec(sigma, w)
```

Arguments

sigma	A symmetric matrix.
w	A numeric vector, representing the vectorized tangent image.

Value

A symmetric matrix, representing the tangent vector.

log_euclidean_vec *Vectorize at Identity Matrix (Euclidean)*

Description

Converts a symmetric matrix into a vector representation.

Usage

```
log_euclidean_vec(sigma, v)
```

Arguments

sigma	A symmetric matrix.
v	A vector.

Value

A numeric vector, representing the vectorized tangent image.

metric *Metric Object Constructor*

Description

Constructs a metric object that contains the necessary functions for Riemannian operations.

Usage

```
metric(log, exp, vec, unvec)
```

Arguments

log	A function representing the Riemannian logarithmic map. This function should accept a dppMatrix (the reference point) and another dppMatrix (the matrix whose logarithm is to be computed), and it outputs a dspMatrix (the tangent image).
exp	A function representing the Riemannian exponential map. This function should accept a dppMatrix (the reference point) and a dspMatrix (the matrix whose exponential is to be computed) and return a dppMatrix (the image on the manifold).
vec	A function representing the vectorization operation for tangent spaces. This function should accept a dppMatrix (the reference point) and a dspMatrix (the tangent image) and return a vector (the vectorized image).
unvec	A function representing the inverse of the vectorization operation. This function should accept a dppMatrix (the reference point) and a vector (the vectorized image), and it returns a dspMatrix (the tangent image).

Value

An object of class `rmetric` containing the specified functions.

`metrics` *Pre-configured Riemannian metrics for SPD matrices*

Description

Ready-to-use metric objects for various Riemannian geometries on the manifold of symmetric positive definite matrices.

Usage

```
airm

log_euclidean

euclidean

log_cholesky

bures_wasserstein
```

Format

Objects of class `rmetric` containing four functions:

log Computes the Riemannian logarithm

exp Computes the Riemannian exponential

vec Performs vectorization

unvec Performs inverse vectorization

An object of class `rmetric` of length 4.

An object of class `rmetric` of length 4.

An object of class `rmetric` of length 4.

An object of class `rmetric` of length 4.

An object of class `rmetric` of length 4.

`parallel_config` *Parallel Processing Configuration for riemtan*

Description

This module provides functions to configure and manage parallel processing using the futureverse framework (future + furrr packages).

ParquetBackend *ParquetBackend Class*

Description

Backend implementation using Apache Arrow Parquet files with lazy loading. Matrices are stored as individual Parquet files and loaded on-demand with LRU caching.

Super class

riemtan::DataBackend -> ParquetBackend

Methods

Public methods:

- ParquetBackend$new()
- ParquetBackend$get_matrix()
- ParquetBackend$get_all_matrices()
- ParquetBackend$get_matrices_parallel()
- ParquetBackend$length()
- ParquetBackend$get_dimensions()
- ParquetBackend$get_metadata()
- ParquetBackend$clear_cache()
- ParquetBackend$get_cache_size()
- ParquetBackend$clone()

Method `new()`**:** Load metadata from JSON file
Load a matrix from Parquet file
Update LRU cache with a new matrix
Initialize a ParquetBackend

Usage:

```
ParquetBackend$new(data_dir, cache_size = 10)
```

Arguments:

`data_dir` Path to directory containing Parquet files and metadata.json
`cache_size` Maximum number of matrices to cache (default 10)
`i` Integer index
`i` Integer index
`mat` A dppMatrix object

Returns: A dppMatrix object

Method `get_matrix()`**:** Get a specific matrix by index

Usage:

```
ParquetBackend$get_matrix(i)
```

Arguments:

`i` Integer index

Returns: A dppMatrix object

Method `get_all_matrices()`**:** Get all matrices (loads all from disk if necessary)

Usage:

```
ParquetBackend$get_all_matrices(parallel = NULL, progress = FALSE)
```

Arguments:

`parallel` Logical indicating whether to use parallel loading (default: NULL, auto-detect)
`progress` Logical indicating whether to show progress (default: FALSE)

Returns: A list of dppMatrix objects

Method `get_matrices_parallel()`**:** Load multiple matrices in parallel (batch loading)

Usage:

```
ParquetBackend$get_matrices_parallel(indices, progress = FALSE)
```

Arguments:

`indices` Vector of integer indices to load
`progress` Logical indicating whether to show progress (default: FALSE)

Returns: A list of dppMatrix objects

Method `length()`**:** Get the number of matrices

Usage:

```
ParquetBackend$length()
```

Returns: Integer count

Method `get_dimensions()`**:** Get matrix dimensions

Usage:

```
ParquetBackend$get_dimensions()
```

Returns: Integer p (matrices are p x p)

Method `get_metadata()`**:** Get metadata

Usage:

```
ParquetBackend$get_metadata()
```

Returns: List containing metadata information

Method `clear_cache()`**:** Clear the cache

Usage:

```
ParquetBackend$clear_cache()
```

Method `get_cache_size()`**:** Get current cache size

Usage:

```
ParquetBackend$get_cache_size()
```

Returns: Integer number of cached matrices

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ParquetBackend$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`progress_utils` *Progress Reporting Utilities for riemtan*

Description

This module provides utilities for progress reporting during computationally intensive operations, using the progressr package.

`relocate` *Relocate Tangent Representations to a New Reference Point*

Description

Changes the reference point for tangent space representations on a Riemannian manifold. Supports parallel processing via the futureverse framework for improved performance on large datasets.

Usage

```
relocate(old_ref, new_ref, images, met, progress = FALSE)
```

Arguments

`old_ref`	A reference point on the manifold to be replaced. Must be an object of class `dppMatrix` from the Matrix package.
`new_ref`	The new reference point on the manifold. Must be an object of class `dppMatrix` from the Matrix package.
`images`	A list of tangent representations relative to the old reference point. Each element in the list must be an object of class `dspMatrix`.
`met`	A metric object of class `rmetric`, containing functions for Riemannian operations (logarithmic map, exponential map, vectorization, and inverse vectorization).
`progress`	Logical indicating whether to show progress during computation (default: FALSE). Requires progressr package.

Details

This function uses parallel processing when the number of images exceeds a threshold and parallel processing is enabled via `set_parallel_plan()`. For small datasets, sequential processing is used automatically to avoid parallelization overhead.

Value

A list of tangent representations relative to the new reference point. Each element in the returned list will be an object of class `dspMatrix`.

Examples

```
if (requireNamespace("Matrix", quietly = TRUE)) {
  library(Matrix)
  data(airm)
  old_ref <- diag(2) |>
    Matrix::nearPD() |>
    _$mat |>
    Matrix::pack()
  new_ref <- diag(c(2, 3)) |>
    Matrix::nearPD() |>
    _$mat |>
    Matrix::pack()
  images <- list(
    diag(2) |> Matrix::symmpart() |> Matrix::pack(),
    diag(c(1, 0.5)) |> Matrix::symmpart() |> Matrix::pack()
  )
  relocate(old_ref, new_ref, images, airm)
}
```

`reset_parallel_plan` *Reset Parallel Plan to Sequential*

Description

Convenience function to reset parallel processing to sequential mode. Equivalent to `set_parallel_plan("sequential")`.

Usage

```
reset_parallel_plan()
```

Value

Invisibly returns the future plan object.

See Also

`set_parallel_plan()`

Examples

```
## Not run:
# Enable parallel processing
set_parallel_plan("multisession", workers = 4)

# Reset to sequential
reset_parallel_plan()

## End(Not run)
```

`rspdnorm` *Generate Random Samples from a Riemannian Normal Distribution*

Description

Simulates random samples from a Riemannian normal distribution on symmetric positive definite matrices.

Usage

```
rspdnorm(n, refpt, disp, met)
```

Arguments

`n`	Number of samples to generate.
`refpt`	Reference point on the manifold, represented as a symmetric positive definite matrix. Must be an object of class `dppMatrix` from the Matrix package.
`disp`	Dispersion matrix defining the spread of the distribution. Must be an object of class `dppMatrix` from the Matrix package.
`met`	A metric object of class `rmetric`.

Value

An object of class `CSample` containing the generated samples.

Examples

```
if (requireNamespace("Matrix", quietly = TRUE)) {
  library(Matrix)
  data(airm)
  refpt <- diag(2) |>
    Matrix::nearPD() |>
    _$mat |>
    Matrix::pack()
  disp <- diag(3) |>
    Matrix::nearPD() |>
    _$mat |>
```

```
    Matrix::pack()
  rspdnorm(10, refpt, disp, airm)
}
```

`safe_logm` *Wrapper for the matrix logarithm*

Description

Wrapper for the matrix logarithm

Usage

```
safe_logm(x)
```

Arguments

`x`	A matrix

Value

Its matrix logarithm

`set_parallel_plan` *Set Parallel Processing Plan*

Description

Configure the parallel processing strategy for riemtan operations. Uses the future package to manage parallel backends.

Usage

```
set_parallel_plan(strategy = "sequential", workers = NULL)
```

Arguments

`strategy`	Character string specifying the parallel strategy: • `"sequential"`: No parallelization (default for safety) • `"multisession"`: Parallel processing using multiple R sessions (works on all platforms including Windows) • `"multicore"`: Parallel processing using forked R processes (Unix-like systems only, faster but not available on Windows) • `"cluster"`: Parallel processing on a cluster of machines
`workers`	Integer specifying the number of parallel workers. If `NULL` (default), uses `parallel::detectCores() - 1` to leave one core free. Ignored when `strategy = "sequential"`.

Details

The `multisession` strategy is recommended for most users as it works on all platforms. The `multicore` strategy is faster on Unix-like systems but is not available on Windows.

To disable parallel processing, use `set_parallel_plan("sequential")`.

Value

Invisibly returns the future plan object.

See Also

`future::plan()`, `is_parallel_enabled()`, `should_parallelize()`

Examples

```
## Not run:
# Enable parallel processing with automatic worker detection
set_parallel_plan("multisession")

# Use 4 parallel workers
set_parallel_plan("multisession", workers = 4)

# Disable parallel processing
set_parallel_plan("sequential")

## End(Not run)
```

---

should_parallelize *Decide Whether to Use Parallel Processing*

Description

Internal function to determine if an operation should use parallel processing based on the number of items to process and current configuration.

Usage

```
should_parallelize(n, threshold = 10)
```

Arguments

`n`	Integer specifying the number of items to process.
`threshold`	Integer specifying the minimum number of items required for parallel processing to be beneficial (default: 10). Below this threshold, sequential processing is used even if parallelization is enabled.

Details

This function returns TRUE only if:

1. Parallel processing is enabled (via set_parallel_plan())
2. The number of items n is at least threshold

For small numbers of items, the overhead of parallelization typically outweighs the benefits, so sequential processing is used.

Value

Logical value: TRUE if parallel processing should be used, FALSE otherwise.

See Also

set_parallel_plan(), is_parallel_enabled()

Examples

```
## Not run:
# With parallel processing enabled
set_parallel_plan("multisession")
should_parallelize(5)    # FALSE (below threshold)
should_parallelize(20)   # TRUE (above threshold)

# With parallel processing disabled
set_parallel_plan("sequential")
should_parallelize(100)  # FALSE (sequential plan)

## End(Not run)
```

---

spd_isometry_from_identity

Reverse isometry from tangent space at identity to tangent space at P

Description

Reverse isometry from tangent space at identity to tangent space at P

Usage

```
spd_isometry_from_identity(sigma, v)
```

Arguments

sigma	A symmetric positive-definite matrix of class dppMatrix
v	A symmetric matrix of class dspMatrix

Value

A symmetric matrix of class `dspMatrix`

---

spd_isometry_to_identity *Isometry from tangent space at P to tangent space at identity*

Description

Isometry from tangent space at P to tangent space at identity

Usage

```
spd_isometry_to_identity(sigma, v)
```

Arguments

`sigma`	A symmetric positive-definite matrix of class `dppMatrix`
`v`	A symmetric matrix of class `dspMatrix`

Value

A symmetric matrix of class `dspMatrix`

TangentImageHandler *TangentImageHandler Class*

Description

TangentImageHandler Class

TangentImageHandler Class

Details

This class handles tangent images on a manifold. It provides methods to set a reference point, compute tangents, and perform various operations using a provided metric. Initialize the TangentImageHandler

Error if the tangent images have not been specified

Error if the reference point is not an object of class dppMatrix

Error if the matrix is not of type dspMatrix Tangent images getter

Active bindings

`ref_point` A matrix of type dppMatrix

`tangent_images` A list of dspMatrix objects

Methods

Public methods:

- TangentImageHandler$new()
- TangentImageHandler$set_reference_point()
- TangentImageHandler$compute_tangents()
- TangentImageHandler$compute_vecs()
- TangentImageHandler$compute_conns()
- TangentImageHandler$set_tangent_images()
- TangentImageHandler$add_tangent_image()
- TangentImageHandler$get_tangent_images()
- TangentImageHandler$relocate_tangents()
- TangentImageHandler$clone()

Method `new()`**:**

Usage:

```
TangentImageHandler$new(metric_obj, reference_point = NULL)
```

Arguments:

`metric_obj` An rmetric object for operations.

`reference_point` An optional reference point on the manifold.

Returns: A new instance of TangentImageHandler. Set a new reference point.

Method `set_reference_point()`**:** If tangent images have been created, it recomputes them by mapping to the manifold and then to the new tangent space.

Usage:

```
TangentImageHandler$set_reference_point(new_ref_pt, progress = FALSE)
```

Arguments:

`new_ref_pt` A new reference point of class dppMatrix.

`progress` Logical indicating whether to show progress (default: FALSE)

Returns: None. Computes the tangent images from the points in the manifold

Method `compute_tangents()`**:**

Usage:

```
TangentImageHandler$compute_tangents(manifold_points, progress = FALSE)
```

Arguments:

`manifold_points` A list of connectomes

`progress` Logical indicating whether to show progress (default: FALSE)

Returns: None Computes vectorizations from tangent images

Method `compute_vecs()`**:**

Usage:

```
TangentImageHandler$compute_vecs(progress = FALSE)
```

Arguments:

`progress` Logical indicating whether to show progress (default: FALSE)

Returns: A matrix, each row of which is a vectorization Computes connectomes from tangent images

Method `compute_conns()`**:**

Usage:

```
TangentImageHandler$compute_conns(progress = FALSE)
```

Arguments:

`progress` Logical indicating whether to show progress (default: FALSE)

Returns: A list of connectomes Setter for the tangent images

Method `set_tangent_images()`**:**

Usage:

```
TangentImageHandler$set_tangent_images(reference_point, tangent_images)
```

Arguments:

`reference_point` A connectome

`tangent_images` A list of tangent images

Returns: None Appends a matrix to the list of tangent images

Method `add_tangent_image()`**:**

Usage:

```
TangentImageHandler$add_tangent_image(image)
```

Arguments:

`image` Matrix to be added

Method `get_tangent_images()`**:**

Usage:

```
TangentImageHandler$get_tangent_images()
```

Returns: list of tangent matrices Wrapper for set_reference_point

Method `relocate_tangents()`**:**

Usage:

```
TangentImageHandler$relocate_tangents(new_ref_pt, progress = FALSE)
```

Arguments:

`new_ref_pt` The new reference point

`progress` Logical indicating whether to show progress (default: FALSE)

Returns: None

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TangentImageHandler$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

validate_backend *Validate Backend Object*

Description

Validates that a backend object inherits from DataBackend and implements required methods.

Usage

```
validate_backend(backend)
```

Arguments

backend	A backend object to validate

Value

None. Throws an error if validation fails.

validate_conns *Validate Connections*

Description

Validates the connections input.

Usage

```
validate_conns(conns, tan_imgs, vec_imgs, centered)
```

Arguments

conns	List of connection matrices.
tan_imgs	List of tangent images.
vec_imgs	Matrix of vector images.
centered	Logical indicating if the data is centered.

Value

None. Throws an error if the validation fails.

validate_exp_args *Validate arguments for Riemannian logarithms*

Description

Validate arguments for Riemannian logarithms

Usage

```
validate_exp_args(sigma, v)
```

Arguments

sigma	A dppMatrix object
v	A dspMatrix object

Details

Error if sigma and lambda are not of the same dimensions

Value

None

validate_log_args *Validate arguments for Riemannian logarithms*

Description

Validate arguments for Riemannian logarithms

Usage

```
validate_log_args(sigma, lambda)
```

Arguments

sigma	A dppMatrix object
lambda	A dppMatrix object

Details

Error if sigma and lambda are not of the same dimensions

Value

None

validate_metric *Validate Metric*

Description

Validates that the metric is not NULL.

Usage

```
validate_metric(metric)
```

Arguments

metric	The metric to validate.

Value

None. Throws an error if the metric is NULL.

validate_parquet_dir *Validate Parquet Directory Structure*

Description

Validate Parquet Directory Structure

Usage

```
validate_parquet_dir(data_dir, check_files = TRUE)
```

Arguments

data_dir	Path to directory to validate
check_files	If TRUE, validates that Parquet files exist (default: TRUE)

Value

None. Throws an error if validation fails.

validate_parquet_directory *Validate Parquet Directory*

Description

Validates that a directory contains properly formatted Parquet files and metadata for use with ParquetBackend.

Usage

```
validate_parquet_directory(data_dir, verbose = TRUE)
```

Arguments

data_dir	Path to directory to validate
verbose	If TRUE, prints validation details (default: TRUE)

Details

Checks:

- Directory exists
- metadata.json exists and is valid
- All expected Parquet files exist
- Parquet files have correct dimensions

Value

Logical indicating whether the directory is valid

Examples

```
## Not run:
validate_parquet_directory("my_connectomes")

## End(Not run)
```

validate_tan_imgs *Validate Tangent Images*

Description

Validates the tangent images input.

Usage

```
validate_tan_imgs(tan_imgs, vec_imgs, centered)
```

Arguments

tan_imgs	List of tangent images.
vec_imgs	List of vector images.
centered	Logical indicating if the data is centered.

Value

None. Throws an error if the validation fails.

validate_unvec_args *Validate arguments for inverse vectorization*

Description

Validate arguments for inverse vectorization

Usage

```
validate_unvec_args(sigma, w)
```

Arguments

sigma	A dppMatrix object
w	A numeric vector

Details

Error if the dimensionalities don't match

Value

None

validate_vec_args *Validate arguments for vectorization*

Description

Validate arguments for vectorization

Usage

```
validate_vec_args(sigma, v)
```

Arguments

sigma	A dppMatrix object
v	A dspMatrix object

Details

Error if sigma and v are not of the same dimensions

Value

None

validate_vec_imgs *Validate Vector Images*

Description

Validates the vector images input.

Usage

```
validate_vec_imgs(vec_imgs, centered)
```

Arguments

vec_imgs	List of vector images.
centered	Logical indicating if the data is centered.

Value

None. Throws an error if the validation fails.

vec_at_id *Vectorize at Identity Matrix*

Description

Converts a symmetric matrix into a vector representation specific to operations at the identity matrix.

Usage

```
vec_at_id(v)
```

Arguments

v	A symmetric matrix of class `dspMatrix`.

Value

A numeric vector, representing the vectorized tangent image.

Examples

```
if (requireNamespace("Matrix", quietly = TRUE)) {
  library(Matrix)
  v <- diag(c(1, sqrt(2))) |>
    Matrix::symmpart() |>
    Matrix::pack()
  vec_at_id(v)
}
```

write_connectomes_to_parquet *Write Connectomes to Parquet Files*

Description

Exports a list of SPD matrices (connectomes) to individual Parquet files with accompanying metadata.

Usage

```
write_connectomes_to_parquet(
  connectomes,
  output_dir,
  file_pattern = "matrix_%04d.parquet",
  subject_ids = NULL,
  provenance = NULL,
  overwrite = FALSE
)
```

Arguments

connectomes	A list of dppMatrix objects representing SPD matrices
output_dir	Path to output directory (will be created if it doesn't exist)
file_pattern	File naming pattern with %d placeholder for index (default: "matrix_%04d.parquet")
subject_ids	Optional vector of subject/sample identifiers (default: NULL)
provenance	Optional list containing data provenance information (default: NULL)
overwrite	If TRUE, overwrites existing directory (default: FALSE)

Details

Creates a directory structure:

- Individual Parquet files (one per matrix)
- metadata.json with dimensions, file pattern, and optional metadata

The metadata.json file contains:

- n_matrices: Number of matrices
- matrix_dim: Dimension p (matrices are p x p)
- file_pattern: Naming pattern for Parquet files
- subject_ids: Optional subject identifiers
- provenance: Optional provenance information

Value

Invisibly returns the path to the output directory

Examples

```
## Not run:
# Create sample data
mats <- replicate(10, Matrix::pack(Matrix::Matrix(diag(5), sparse = FALSE)), simplify = FALSE)

# Write to Parquet
write_connectomes_to_parquet(
  mats,
  output_dir = "my_connectomes",
  subject_ids = paste0("subj_", 1:10),
  provenance = list(study = "Example Study", date = Sys.Date())
)

## End(Not run)
```

Index