Package ‘riemtan’

July 23, 2025

Title Riemannian Metrics for Symmetric Positive Definite Matrices

Version 0.1.0

Description Implements various Riemannian metrics for symmetric positive definite matrices, including AIRM (Affine Invariant Riemannian Metric, see Pennec, Fillard, and Ayache (2006) <doi:10.1007/s11263-005-3222-z>), Log-Euclidean (see Arsigny, Fillard, Pennec, and Ayache (2006) <doi:10.1002/mrm.20965>), Euclidean, Log-Cholesky (see Lin (2019) <doi:10.1137/18M1221084>), and Bures-Wasserstein metrics (see Bhatia, Jain, and Lim (2019) <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 7.3.2

Depends R (>= 4.3.0), Matrix

Imports methods, expm, R6, purrr, MASS, furrr

Suggests 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 no

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

Repository CRAN

Date/Publication 2025-04-23 10:10:02 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)
```

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.

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)
```

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.

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)
```

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.

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)
```

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.

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.

Usage

```
compute_frechet_mean(sample, tol = 0.05, max_iter = 20, lr = 0.2)
```

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.

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.

Value

The computed Frechet mean.

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)
}
```

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

Methods

Public methods:

- `CSample$new()`
- `CSample$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_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
)
```

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.

Returns: A new `CSample` object.

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

Usage:

```
CSample$compute_tangents(ref_pt = default_ref_pt(private$p))
```

Arguments:

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

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()
```

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()
```

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()`

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 = 0.05, max_iter = 20, lr = 0.2)
```

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).

Returns: None

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

Usage:

```
CSample$change_ref_pt(new_ref_pt)
```

Arguments:

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

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_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.

`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)
```

Arguments

sigma	A reference point.
v	A tangent vector to be mapped back to the manifold at `sigma`.

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)
```

Arguments

sigma	A reference point.
lambda	A point on the manifold.

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.

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`.

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)
```

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.

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)
```

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.

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)
```

Arguments

`ref_pt`	A reference point.
`v`	A tangent vector to be mapped back to the manifold at `ref_pt`.

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)
```

Arguments

sigma	A reference point.
lambda	A point on the manifold.

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.

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

Description

Changes the reference point for tangent space representations on a Riemannian manifold.

Usage

```
relocate(old_ref, new_ref, images, met)
```

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).

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)
}
```

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

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)
```

Arguments:

`new_ref_pt` A new reference point of class dppMatrix.

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

Method `compute_tangents()`**:**

Usage:

```
TangentImageHandler$compute_tangents(manifold_points)
```

Arguments:

`manifold_points` A list of connectomes

Returns: None Computes vectorizations from tangent images

Method `compute_vecs()`**:**

Usage:

```
TangentImageHandler$compute_vecs()
```

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

Method `compute_conns()`:

Usage:

`TangentImageHandler$compute_conns()`

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)`

Arguments:

`new_ref_pt` The new reference point

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_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_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)
}
```

Index