Kriging prediction for manifold-valued data

Generated by Doxygen 1.8.14

Contents

Chapter 1

What is Manifoldgstat Package

R Package to make inference and prediction for manifold-valued data analysis. This package provides a C++ implementation of the functions to create a model, for spatial dependent manifold valued data, in order to perform kriging. In each location, specified by a vector of coordinates ([lat,long], [x,y] or [x,y,z]), the datum is supposed to be a symmetric and positive definite matrix (possibly a correlation matrix). The algorithm exploits a projection of these data on a tangent space, where the tangent point is either provided by the user or computed as intrinsic mean of the data in input.

References

JD. Pigoli, A. Menafoglio & P. Secchi (2016). Kriging prediction for manifold-valued random fields. Journal of Multivariate Analysis, 145, 117-131.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 Coordinates Class Reference

Class to store the coordinates of the data points.

`#include <Coordinates.hpp>`

Public Member Functions

- Coordinates (const std::shared_ptr< const MatrixXd > coords)
  Constructor.
- unsigned int get_N_station () const
  Return the number of data points in the analysis.
- const std::shared_ptr< const MatrixXd > get_coords () const
  Return the matrix of the coordinates.
- unsigned int get_n_coords () const
  Return the number of coordinates for each data point.

Private Attributes

- const std::shared_ptr< const MatrixXd > _coords

5.1.1 Detailed Description

Class to store the coordinates of the data points.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Coordinates()

```
Coordinates::Coordinates (
            const std::shared_ptr< const MatrixXd > coords )  [inline]
```

Constructor.

Parameters

coords	Matrix containg the coordinates of the data points

5.1.3 Member Data Documentation

5.1.3.1 _coords

```
const std::shared_ptr<const MatrixXd> Coordinates::_coords  [private]
```

Matrix of the coordinates

The documentation for this class was generated from the following files:

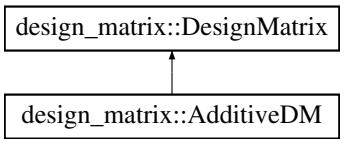
- Coordinates.hpp
- Coordinates.cpp

5.2 design_matrix::AdditiveDM Class Reference

Class for the computation of the design_matrix when `model_ts=="Additive"`

```
#include <DesignMatrix.hpp>
```

Inheritance diagram for design_matrix::AdditiveDM:



Public Member Functions

- MatrixXd compute_design_matrix (const Coordinates &coords) const override

  Compute the design matrix with no covariates besides the coordinates.
- MatrixXd compute_design_matrix (const Coordinates &coords, const MatrixXd &X) const override

  Compute the design matrix with extra covariates besides the coordinates.
- ~AdditiveDM ()=default

  Destructor.

5.2.1 Detailed Description

Class for the computation of the design_matrix when `model_ts=="Additive"`

5.2.2 Member Function Documentation

5.2.2.1 compute_design_matrix() [1/2]

```
MatrixXd AdditiveDM::compute_design_matrix (
            const Coordinates & coords ) const  [override], [virtual]
```

Compute the design matrix with no covariates besides the coordinates.

Parameters

coords	Matrix of the coordinates of the locations

Returns

Design matrix

Implements design_matrix::DesignMatrix.

5.2.2.2 compute_design_matrix() [2/2]

```
MatrixXd AdditiveDM::compute_design_matrix (
            const Coordinates & coords,
            const MatrixXd & X ) const  [override], [virtual]
```

Compute the design matrix with extra covariates besides the coordinates.

Parameters

coords	Matrix of the coordinates of the locations
X	Matrix of the additional covariates for the locations

Returns

Design matrix

Implements design_matrix::DesignMatrix.

The documentation for this class was generated from the following files:

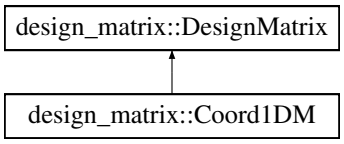
- DesignMatrix.hpp
- DesignMatrix.cpp

5.3 design_matrix::Coord1DM Class Reference

Class for the computation of the design_matrix when `model_ts=="Coord1"`

`#include <DesignMatrix.hpp>`

Inheritance diagram for design_matrix::Coord1DM:



Public Member Functions

- MatrixXd compute_design_matrix (const Coordinates &coords) const override

  Compute the design matrix with no covariates besides the coordinates.
- MatrixXd compute_design_matrix (const Coordinates &coords, const MatrixXd &X) const override

  Compute the design matrix with extra covariates besides the coordinates.
- ~Coord1DM ()=default

  Destructor.

5.3.1 Detailed Description

Class for the computation of the design_matrix when `model_ts=="Coord1"`

5.3.2 Member Function Documentation

5.3.2.1 compute_design_matrix() [1/2]

```
MatrixXd Coord1DM::compute_design_matrix (
            const Coordinates & coords ) const  [override], [virtual]
```

Compute the design matrix with no covariates besides the coordinates.

Parameters

coords	Matrix of the coordinates of the locations

Returns

Design matrix

Implements design_matrix::DesignMatrix.

5.3.2.2 compute_design_matrix() [2/2]

```
MatrixXd Coord1DM::compute_design_matrix (
            const Coordinates & coords,
            const MatrixXd & X ) const  [override], [virtual]
```

Compute the design matrix with extra covariates besides the coordinates.

Parameters

coords	Matrix of the coordinates of the locations
X	Matrix of the additional covariates for the locations

Returns

Design matrix

Implements design_matrix::DesignMatrix.

The documentation for this class was generated from the following files:

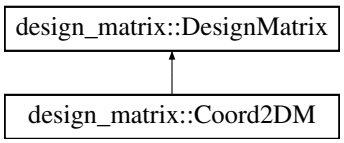
- DesignMatrix.hpp
- DesignMatrix.cpp

5.4 design_matrix::Coord2DM Class Reference

Class for the computation of the design_matrix when `model_ts=="Coord2"`

```
#include <DesignMatrix.hpp>
```

Inheritance diagram for design_matrix::Coord2DM:



Public Member Functions

- MatrixXd compute_design_matrix (const Coordinates &coords) const override

  Compute the design matrix with no covariates besides the coordinates.
- MatrixXd compute_design_matrix (const Coordinates &coords, const MatrixXd &X) const override

  Compute the design matrix with extra covariates besides the coordinates.
- ~Coord2DM ()=default

  Destructor.

5.4.1 Detailed Description

Class for the computation of the design_matrix when `model_ts=="Coord2"`

5.4.2 Member Function Documentation

5.4.2.1 compute_design_matrix() [1/2]

```
MatrixXd Coord2DM::compute_design_matrix (
            const Coordinates & coords ) const  [override], [virtual]
```

Compute the design matrix with no covariates besides the coordinates.

Parameters

coords	Matrix of the coordinates of the locations

Returns

Design matrix

Implements design_matrix::DesignMatrix.

5.4.2.2 compute_design_matrix() [2/2]

```
MatrixXd Coord2DM::compute_design_matrix (
            const Coordinates & coords,
            const MatrixXd & X ) const  [override], [virtual]
```

Compute the design matrix with extra covariates besides the coordinates.

Parameters

coords	Matrix of the coordinates of the locations
X	Matrix of the additional covariates for the locations

Returns

Design matrix

Implements design_matrix::DesignMatrix.

The documentation for this class was generated from the following files:

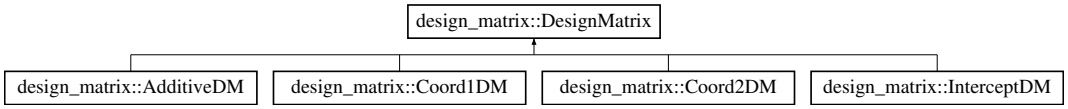
- DesignMatrix.hpp
- DesignMatrix.cpp

5.5 design_matrix::DesignMatrix Class Reference

Abstract class for the computation of the design_matrix.

`#include <DesignMatrix.hpp>`

Inheritance diagram for design_matrix::DesignMatrix:



Public Member Functions

- virtual MatrixXd compute_design_matrix (const Coordinates &coords) const =0

  Compute the design matrix with no covariates besides the coordinates.
- virtual MatrixXd compute_design_matrix (const Coordinates &coords, const MatrixXd &X) const =0

  Compute the design matrix with extra covariates besides the coordinates.
- virtual ~DesignMatrix ()=default

  Destructor.

5.5.1 Detailed Description

Abstract class for the computation of the design_matrix.

5.5.2 Member Function Documentation

5.5.2.1 compute_design_matrix() [1/2]

```
virtual MatrixXd design_matrix::DesignMatrix::compute_design_matrix (
            const Coordinates & coords ) const  [pure virtual]
```

Compute the design matrix with no covariates besides the coordinates.

Parameters

coords	Matrix of the coordinates of the locations

Returns

Design matrix

Implemented in design_matrix::AdditiveDM, design_matrix::Coord2DM, design_matrix::Coord1DM, and design_matrix::InterceptDM.

5.5.2.2 compute_design_matrix() [2/2]

```
virtual MatrixXd design_matrix::DesignMatrix::compute_design_matrix (
            const Coordinates & coords,
            const MatrixXd & X ) const  [pure virtual]
```

Compute the design matrix with extra covariates besides the coordinates.

Parameters

coords	Matrix of the coordinates of the locations
X	Matrix of the additional covariates for the locations

Returns

Design matrix

Implemented in design_matrix::AdditiveDM, design_matrix::Coord2DM, design_matrix::Coord1DM, and design_matrix::InterceptDM.

The documentation for this class was generated from the following file:

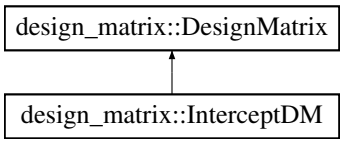
- DesignMatrix.hpp

5.6 design_matrix::InterceptDM Class Reference

Class for the computation of the design_matrix when `model_ts=="Intercept"`

`#include <DesignMatrix.hpp>`

Inheritance diagram for design_matrix::InterceptDM:



Public Member Functions

- MatrixXd compute_design_matrix (const Coordinates &coords) const override

  Compute the design matrix with no covariates besides the coordinates.
- MatrixXd compute_design_matrix (const Coordinates &coords, const MatrixXd &X) const override

  Compute the design matrix with extra covariates besides the coordinates.
- ~InterceptDM ()=default

  Destructor.

5.6.1 Detailed Description

Class for the computation of the design_matrix when `model_ts=="Intercept"`

5.6.2 Member Function Documentation

5.6.2.1 compute_design_matrix() [1/2]

```
MatrixXd InterceptDM::compute_design_matrix (
            const Coordinates & coords ) const  [override], [virtual]
```

Compute the design matrix with no covariates besides the coordinates.

Parameters

coords	Matrix of the coordinates of the locations

Returns

Design matrix

Implements design_matrix::DesignMatrix.

5.6.2.2 compute_design_matrix() [2/2]

```
MatrixXd InterceptDM::compute_design_matrix (
            const Coordinates & coords,
            const MatrixXd & X ) const  [override], [virtual]
```

Compute the design matrix with extra covariates besides the coordinates.

Parameters

coords	Matrix of the coordinates of the locations
X	Matrix of the additional covariates for the locations

Returns

Design matrix

Implements design_matrix::DesignMatrix.

The documentation for this class was generated from the following files:

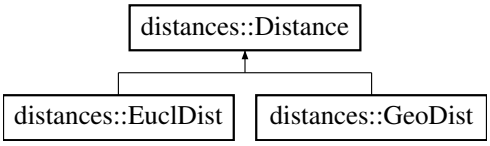
- DesignMatrix.hpp
- DesignMatrix.cpp

5.7 distances::Distance Class Reference

Abstract class for the computation of the distance between data locations.

`#include <Distance.hpp>`

Inheritance diagram for distances::Distance:



Public Member Functions

- virtual double compute_distance (const Vec &P1, const Vec &P2) const =0

  Compute the distance between two locations.
- std::shared_ptr< const MatrixXd > create_distance_matrix (const Coordinates &coordinates, unsigned int N) const

  Compute the distance matrix among a set of locations.
- std::vector< double > create_distance_vector (const Coordinates &coordinates, const Vec &new_coord) const

  Compute the vector of distances between a point and a set of locations.
- virtual ~Distance ()=default

  Destructor.

5.7.1 Detailed Description

Abstract class for the computation of the distance between data locations.

5.7.2 Member Function Documentation

5.7.2.1 compute_distance()

```
virtual double distances::Distance::compute_distance (
            const Vec & P1,
            const Vec & P2 ) const  [pure virtual]
```

Compute the distance between two locations.

Parameters

P1	Vector of coordinates for the first location
P2	Vector of coordinates for the second location

Returns

Points' distance

Implemented in distances::GeoDist, and distances::EuclDist.

5.7.2.2 create_distance_matrix()

```
std::shared_ptr< const MatrixXd > Distance::create_distance_matrix (
            const Coordinates & coordinates,
            unsigned int N ) const
```

Compute the distance matrix among a set of locations.

Parameters

coordinates	Matrix of the coordinates of the locations
N	Number of locations

Returns

Matrix of distances

5.7.2.3 create_distance_vector()

```
std::vector< double > Distance::create_distance_vector (
            const Coordinates & coordinates,
            const Vec & new_coord ) const
```

Compute the vector of distances between a point and a set of locations.

Parameters

coordinates	Matrix of the coordinates of the locations
new_coord	Vector of coordinates of the new location

Returns

Vector of distances

The documentation for this class was generated from the following files:

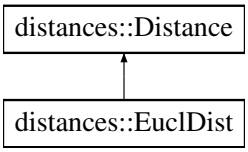
- Distance.hpp
- Distance.cpp

5.8 distances::EuclDist Class Reference

Class for the computation of the distance between data locations when `distance=="Eucldist"`

`#include <Distance.hpp>`

Inheritance diagram for distances::EuclDist:



Public Member Functions

- double compute_distance (const Vec &P1, const Vec &P2) const override

  Compute the distance between two locations.
- ~EuclDist ()=default

  Destructor.

5.8.1 Detailed Description

Class for the computation of the distance between data locations when `distance=="Eucldist"`

5.8.2 Member Function Documentation

5.8.2.1 compute_distance()

```
double EuclDist::compute_distance (
            const Vec & P1,
            const Vec & P2 ) const  [override], [virtual]
```

Compute the distance between two locations.

Parameters

P1	Vector of coordinates for the first location
P2	Vector of coordinates for the second location

Returns

Points' distance

Implements distances::Distance.

The documentation for this class was generated from the following files:

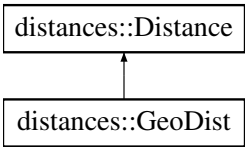
- Distance.hpp
- Distance.cpp

5.9 distances::GeoDist Class Reference

Class for the computation of the distance between data locations when `distance=="Geodist"`

`#include <Distance.hpp>`

Inheritance diagram for distances::GeoDist:



Public Member Functions

- double compute_distance (const Vec &P1, const Vec &P2) const override

  Compute the distance between two locations.
- ~GeoDist ()=default

  Destructor.

Static Private Attributes

- static double constexpr Earth_R = 6371.0
- static double constexpr eps_dbl = std::numeric_limits<double>::epsilon()

5.9.1 Detailed Description

Class for the computation of the distance between data locations when `distance=="Geodist"`

5.9.2 Member Function Documentation

5.9.2.1 compute_distance()

```
double GeoDist::compute_distance (
            const Vec & P1,
            const Vec & P2 ) const  [override], [virtual]
```

Compute the distance between two locations.

Parameters

P1	Vector of coordinates for the first location
P2	Vector of coordinates for the second location

Returns

Points' distance

Implements distances::Distance.

5.9.3 Member Data Documentation

5.9.3.1 Earth_R

```
double constexpr distances::GeoDist::Earth_R = 6371.0  [static], [private]
```

Earth radius

5.9.3.2 eps_dbl

```
double constexpr distances::GeoDist::eps_dbl = std::numeric_limits<double>::epsilon()  [static],
[private]
```

Machine epsilon

The documentation for this class was generated from the following files:

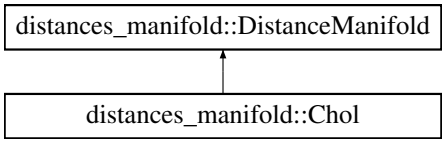
- Distance.hpp
- Distance.cpp

5.10 distances_manifold::Chol Class Reference

Class for the computation of the distance on the manifold when `manifold_metric=="Chol"`

```
#include <DistanceManifold.hpp>
```

Inheritance diagram for distances_manifold::Chol:



Public Member Functions

- double compute_distance (const MatrixXd &M1, const MatrixXd &M2) const override

  Compute the distance between two matrices on the manifold.
- ∼Chol ()=default

  Destructor.

5.10.1 Detailed Description

Class for the computation of the distance on the manifold when `manifold_metric=="Chol"`

Note

The data on the manifold must be correlation matrices

5.10.2 Member Function Documentation

5.10.2.1 compute_distance()

```
double Chol::compute_distance (
            const MatrixXd & M1,
            const MatrixXd & M2 ) const  [override], [virtual]
```

Compute the distance between two matrices on the manifold.

Parameters

M1	First matrix
M2	Second matrix

Returns

Distance

Implements distances_manifold::DistanceManifold.

The documentation for this class was generated from the following files:

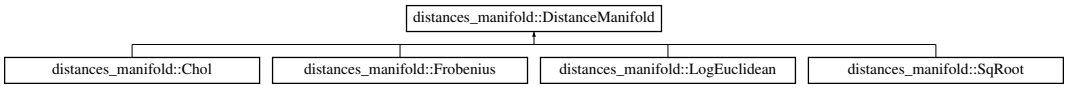
- DistanceManifold.hpp
- DistanceManifold.cpp

5.11 distances_manifold::DistanceManifold Class Reference

Abstract class for the computation of the distance on the manifold.

`#include <DistanceManifold.hpp>`

Inheritance diagram for distances_manifold::DistanceManifold:



Public Member Functions

- virtual double compute_distance (const MatrixXd &M1, const MatrixXd &M2) const =0

  Compute the distance between two matrices on the manifold.
- virtual ~DistanceManifold ()=default

  Destructor.

5.11.1 Detailed Description

Abstract class for the computation of the distance on the manifold.

5.11.2 Member Function Documentation

5.11.2.1 compute_distance()

```
virtual double distances_manifold::DistanceManifold::compute_distance (
            const MatrixXd & M1,
            const MatrixXd & M2 ) const  [pure virtual]
```

Compute the distance between two matrices on the manifold.

Parameters

M1	First matrix
M2	Second matrix

Returns

Distance

Implemented in distances_manifold::Chol, distances_manifold::SqRoot, distances_manifold::LogEuclidean, and distances_manifold::Frobenius.

The documentation for this class was generated from the following file:

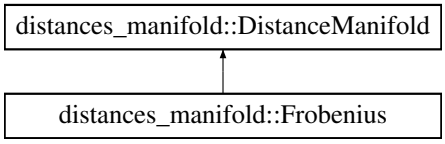
- DistanceManifold.hpp

5.12 distances_manifold::Frobenius Class Reference

Class for the computation of the distance on the manifold when `manifold_metric=="Frobenius"`

`#include <DistanceManifold.hpp>`

Inheritance diagram for distances_manifold::Frobenius:



Public Member Functions

- double compute_distance (const MatrixXd &M1, const MatrixXd &M2) const override

  Compute the distance between two matrices on the manifold.
- ~Frobenius ()=default

  Destructor.

5.12.1 Detailed Description

Class for the computation of the distance on the manifold when `manifold_metric=="Frobenius"`

5.12.2 Member Function Documentation

5.12.2.1 compute_distance()

```
double Frobenius::compute_distance (
            const MatrixXd & M1,
            const MatrixXd & M2 ) const  [override], [virtual]
```

Compute the distance between two matrices on the manifold.

Parameters

M1	First matrix
M2	Second matrix

Returns

Distance

Implements distances_manifold::DistanceManifold.

The documentation for this class was generated from the following files:

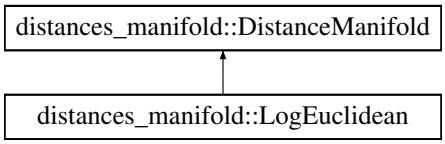
- DistanceManifold.hpp
- DistanceManifold.cpp

5.13 distances_manifold::LogEuclidean Class Reference

Class for the computation of the distance on the manifold when `manifold_metric=="LogEuclidean"`

`#include <DistanceManifold.hpp>`

Inheritance diagram for distances_manifold::LogEuclidean:



Public Member Functions

- double compute_distance (const MatrixXd &M1, const MatrixXd &M2) const override

  Compute the distance between two matrices on the manifold.
- ~LogEuclidean ()=default

  Destructor.

5.13.1 Detailed Description

Class for the computation of the distance on the manifold when `manifold_metric=="LogEuclidean"`

5.13.2 Member Function Documentation

5.13.2.1 compute_distance()

```
double LogEuclidean::compute_distance (
            const MatrixXd & M1,
            const MatrixXd & M2 ) const  [override], [virtual]
```

Compute the distance between two matrices on the manifold.

Parameters

M1	First matrix
M2	Second matrix

Returns

Distance

Implements distances_manifold::DistanceManifold.

The documentation for this class was generated from the following files:

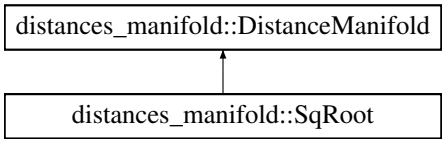
- DistanceManifold.hpp
- DistanceManifold.cpp

5.14 distances_manifold::SqRoot Class Reference

Class for the computation of the distance on the manifold when `manifold_metric=="SqRoot"`

`#include <DistanceManifold.hpp>`

Inheritance diagram for distances_manifold::SqRoot:



Public Member Functions

- double compute_distance (const MatrixXd &M1, const MatrixXd &M2) const override
  Compute the distance between two matrices on the manifold.
- ~SqRoot ()=default
  Destructor.

5.14.1 Detailed Description

Class for the computation of the distance on the manifold when `manifold_metric=="SqRoot"`

5.14.2 Member Function Documentation

5.14.2.1 compute_distance()

```
double SqRoot::compute_distance (
            const MatrixXd & M1,
            const MatrixXd & M2 ) const  [override], [virtual]
```

Compute the distance between two matrices on the manifold.

Parameters

M1	First matrix
M2	Second matrix

Returns

Distance

Implements distances_manifold::DistanceManifold.

The documentation for this class was generated from the following files:

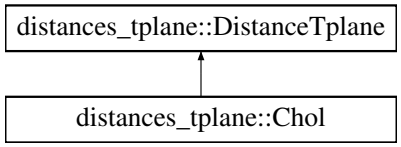
- DistanceManifold.hpp
- DistanceManifold.cpp

5.15 distances_tplane::Chol Class Reference

Class for the computation of the distance on the tangent space when `ts_metric=="Correlation"`

`#include <DistanceTplane.hpp>`

Inheritance diagram for distances_tplane::Chol:



Public Member Functions

- ~Chol ()=default

  Destructor.
- double norm (const MatrixXd &M1) const override

  Compute the norm of a matrix on the tangent space.
- void set_members (const MatrixXd &Sigma) override

  Set the members that will be used in the computation of the norms and distances, according to the metric on the tangent space.

5.15.1 Detailed Description

Class for the computation of the distance on the tangent space when `ts_metric=="Correlation"`

5.15.2 Member Function Documentation

5.15.2.1 norm()

```
double Chol::norm (
            const MatrixXd & M1 ) const  [override], [virtual]
```

Compute the norm of a matrix on the tangent space.

Parameters

M1	Matrix

Returns

Norm

Implements distances_tplane::DistanceTplane.

5.15.2.2 set_members()

```
void Chol::set_members (
            const MatrixXd & Sigma )  [override], [virtual]
```

Set the members that will be used in the computation of the norms and distances, according to the metric on the tangent space.

Parameters

Sigma	Tangent point

Implements distances_tplane::DistanceTplane.

The documentation for this class was generated from the following files:

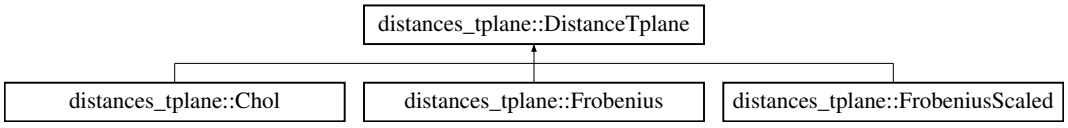
- DistanceTplane.hpp
- DistanceTplane.cpp

5.16 distances_tplane::DistanceTplane Class Reference

Abstract class for the computation of the distance on the tangent space.

```
#include <DistanceTplane.hpp>
```

Inheritance diagram for distances_tplane::DistanceTplane:



Public Member Functions

- double compute_distance (const MatrixXd &M1, const MatrixXd &M2) const

  Compute the distance between two matrices on the tangent space.
- virtual double norm (const MatrixXd &M1) const =0

  Compute the norm of a matrix on the tangent space.
- virtual void set_members (const MatrixXd &Sigma)=0

  Set the members that will be used in the computation of the norms and distances, according to the metric on the tangent space.
- virtual ~DistanceTplane ()=default

  Destructor.

5.16.1 Detailed Description

Abstract class for the computation of the distance on the tangent space.

5.16.2 Member Function Documentation

5.16.2.1 compute_distance()

```
double DistanceTplane::compute_distance (
            const MatrixXd & M1,
            const MatrixXd & M2 ) const
```

Compute the distance between two matrices on the tangent space.

Parameters

M1	First matrix
M2	Second matrix

Returns

Distance

5.16.2.2 norm()

```
virtual double distances_tplane::DistanceTplane::norm (
            const MatrixXd & M1 ) const  [pure virtual]
```

Compute the norm of a matrix on the tangent space.

Parameters

M1	Matrix

Returns

Norm

Implemented in distances_tplane::Chol, distances_tplane::FrobeniusScaled, and distances_tplane::Frobenius.

5.16.2.3 set_members()

```
virtual void distances_tplane::DistanceTplane::set_members (
            const MatrixXd & Sigma )  [pure virtual]
```

Set the members that will be used in the computation of the norms and distances, according to the metric on the tangent space.

Parameters

Sigma	Tangent point

Implemented in distances_tplane::Chol, distances_tplane::FrobeniusScaled, and distances_tplane::Frobenius.

The documentation for this class was generated from the following files:

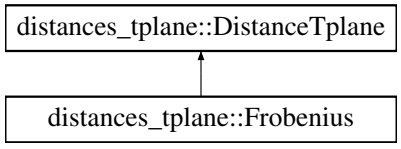
- DistanceTplane.hpp
- DistanceTplane.cpp

5.17 distances_tplane::Frobenius Class Reference

Class for the computation of the distance on the tangent space when `ts_metric=="Frobenius"`

```
#include <DistanceTplane.hpp>
```

Inheritance diagram for distances_tplane::Frobenius:



Public Member Functions

- ~Frobenius ()=default

  Destructor.
- double norm (const MatrixXd &M1) const override

  Compute the norm of a matrix on the tangent space.
- void set_members (const MatrixXd &Sigma) override

  Set the members that will be used in the computation of the norms and distances, according to the metric on the tangent space.

5.17.1 Detailed Description

Class for the computation of the distance on the tangent space when `ts_metric=="Frobenius"`

5.17.2 Member Function Documentation

5.17.2.1 norm()

```
double Frobenius::norm (
            const MatrixXd & M1 ) const  [override], [virtual]
```

Compute the norm of a matrix on the tangent space.

Parameters

M1	Matrix

Returns

Norm

Implements distances_tplane::DistanceTplane.

5.17.2.2 set_members()

```
void Frobenius::set_members (
            const MatrixXd & Sigma )  [override], [virtual]
```

Set the members that will be used in the computation of the norms and distances, according to the metric on the tangent space.

Parameters

Sigma	Tangent point

Implements distances_tplane::DistanceTplane.

The documentation for this class was generated from the following files:

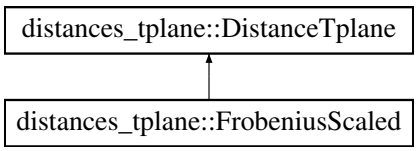
- DistanceTplane.hpp
- DistanceTplane.cpp

5.18 distances_tplane::FrobeniusScaled Class Reference

Class for the computation of the distance on the tangent space when `ts_metric=="FrobeniusScaled"`

`#include <DistanceTplane.hpp>`

Inheritance diagram for distances_tplane::FrobeniusScaled:



Public Member Functions

- ~FrobeniusScaled ()=default

  Destructor.
- double norm (const MatrixXd &M1) const override

  Compute the norm of a matrix on the tangent space.
- void set_members (const MatrixXd &Sigma) override

  Set the members that will be used in the computation of the norms and distances, according to the metric on the tangent space.

Private Attributes

- MatrixXd _SigmaInv
- unsigned int _p

5.18.1 Detailed Description

Class for the computation of the distance on the tangent space when `ts_metric=="FrobeniusScaled"`

5.18.2 Member Function Documentation

5.18.2.1 norm()

```
double FrobeniusScaled::norm (
            const MatrixXd & M1 ) const  [override], [virtual]
```

Compute the norm of a matrix on the tangent space.

Parameters

M1	Matrix

Returns

Norm

Implements distances_tplane::DistanceTplane.

5.18.2.2 set_members()

```
void FrobeniusScaled::set_members (
            const MatrixXd & Sigma )  [override], [virtual]
```

Set the members that will be used in the computation of the norms and distances, according to the metric on the tangent space.

Parameters

Sigma	Tangent point

Implements distances_tplane::DistanceTplane.

5.18.3 Member Data Documentation

5.18.3.1 _SigmaInv

```
MatrixXd distances_tplane::FrobeniusScaled::_SigmaInv  [private]
```

Inverse of the tangent point Sigma

5.18.3.2 _p

```
unsigned int distances_tplane::FrobeniusScaled::_p  [private]
```

Dimension of the matrices on the tangent space

The documentation for this class was generated from the following files:

- DistanceTplane.hpp
- DistanceTplane.cpp

5.19 generic_factory::Factory< AbstractProduct, Identifier, Builder > Class Template Reference

A generic factory.

`#include <Factory.hpp>`

Public Types

- using AbstractProduct_type = AbstractProduct
  The container for the rules.
- using Identifier_type = Identifier
  The identifier.
- using Builder_type = Builder
  The builder type.

Public Member Functions

- std::unique_ptr< AbstractProduct > create (Identifier const &name) const
  Get the rule with given name.
- void add (Identifier const &, Builder_type const &)
  Register the given rule.
- std::vector< Identifier > registered () const
  Returns a list of registered rules.
- void unregister (Identifier const &name)
  Unregister a rule.
- ~Factory ()=default
  Destructor.

Static Public Member Functions

- static Factory & Instance ()
  Method to access the only instance of the factory. We use Meyer's trick to istantiate the factory.

Private Types

- typedef std::map< Identifier, Builder_type > Container_type
  Type of the object used to store the object factory.

Private Member Functions

- Factory ()=default
  Constructor made private since it is a Singleton.
- Factory (Factory const &)=delete
  Copy constructor deleted since it is a Singleton.
- Factory & operator= (Factory const &)=delete
  Assignment operator deleted since it is a Singleton.

Private Attributes

- Container_type _storage

  It contains the actual object factory.

5.19.1 Detailed Description

template<typename AbstractProduct, typename Identifier, typename Builder = std::function<std::unique_ptr<AbstractProduct>()>>
class generic_factory::Factory< AbstractProduct, Identifier, Builder >

A generic factory.

It is implemented as a Singleton. The compulsory way to access a method is Factory::Instance().method(). Typycally to access the factory one does

```
auto&  myFactory = Factory<A,I,B>::Instance();
myFactory.add(...)
```

5.19.2 Member Function Documentation

5.19.2.1 create()

```
template<typename AbstractProduct , typename Identifier , typename Builder >
std::unique_ptr< AbstractProduct > generic_factory::Factory< AbstractProduct, Identifier,
Builder >::create (
             Identifier const & name ) const
```

Get the rule with given name.

The pointer is null if no rule is present.

The documentation for this class was generated from the following file:

- Factory.hpp

5.20 generic_factory::Proxy< Factory, ConcreteProduct > Class Template Reference

```
#include <Proxy.hpp>
```

Public Types

- typedef Factory::AbstractProduct_type AbstractProduct_type

  The container for the rules.
- typedef Factory::Identifier_type Identifier_type

  The identifier.
- typedef Factory::Builder_type Builder_type

  The builder type.
- typedef Factory Factory_type

  The factory type.

Public Member Functions

- Proxy (Identifier_type const &)

  The constructor does the registration.

Static Public Member Functions

- static std::unique_ptr< AbstractProduct_type > Build ()

  The builder.

Private Member Functions

- Proxy (Proxy const &)=delete

  Copy onstructor deleted since it is a Singleton.
- Proxy & operator= (Proxy const &)=delete

  Assignment operator deleted since it is a Singleton.

5.20.1 Detailed Description

template<typename Factory, typename ConcreteProduct>
class generic_factory::Proxy< Factory, ConcreteProduct >

A simple proxy for registering into a factory.

It provides the builder as static method and the automatic registration mechanism.

Parameters

Factory	The type of the factory.
ConcreteProduct	Is the derived (concrete) type to be registered in the factory

Note

I have to use the default builder provided by the factory. No check is made to verify it

The documentation for this class was generated from the following file:

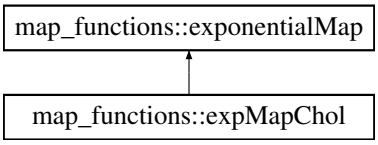
- Proxy.hpp

5.21 map_functions::expMapChol Class Reference

Class for the computation of the exponential map when `manifold_metric=="Correlation"`

`#include <MapFunctions.hpp>`

Inheritance diagram for map_functions::expMapChol:



Public Member Functions

- ~expMapChol ()=default
  Destructor.
- MatrixXd map2manifold (const MatrixXd &M) const override
  Map a tangent space matrix to the manifold.
- void set_members (const MatrixXd &Sigma) override
  Set class members.
- void set_tolerance (double tolerance_map_cor) override
  Set tolerance.

Private Attributes

- double _tolerance_map_cor
- MatrixXd _Sigma

5.21.1 Detailed Description

Class for the computation of the exponential map when `manifold_metric=="Correlation"`

5.21.2 Member Function Documentation

5.21.2.1 map2manifold()

```
MatrixXd expMapChol::map2manifold (
            const MatrixXd & M ) const  [override], [virtual]
```

Map a tangent space matrix to the manifold.

Parameters

M	Tangent space matrix to map

Returns

Manifold matrix identifying the mapped data

Implements map_functions::exponentialMap.

5.21.2.2 set_members()

```
void expMapChol::set_members (
            const MatrixXd & Sigma )  [override], [virtual]
```

Set class members.

Parameters

Sigma	Tangent point

Implements map_functions::exponentialMap.

5.21.2.3 set_tolerance()

```
void expMapChol::set_tolerance (
            double tolerance_map_cor )  [override], [virtual]
```

Set tolerance.

Parameters

tolerance_map_cor	Tolerance

Implements map_functions::exponentialMap.

5.21.3 Member Data Documentation

5.21.3.1 _tolerance_map_cor

```
double map_functions::expMapChol::_tolerance_map_cor  [private]
```

Tolerance on the norm of the columns to avoid `Nan`

5.21.3.2 _Sigma

```
MatrixXd map_functions::expMapChol::_Sigma  [private]
```

Tangent point `Sigma`

The documentation for this class was generated from the following files:

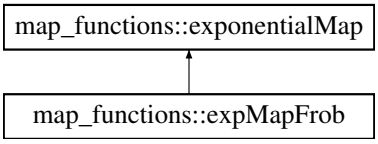
- MapFunctions.hpp
- MapFunctions.cpp

5.22 map_functions::expMapFrob Class Reference

Class for the computation of the exponential map when `manifold_metric=="Frobenius"`

```
#include <MapFunctions.hpp>
```

Inheritance diagram for map_functions::expMapFrob:



Public Member Functions

- ~expMapFrob ()=default
  Destructor.
- MatrixXd map2manifold (const MatrixXd &M) const override
  Map a tangent space matrix to the manifold.
- void set_members (const MatrixXd &Sigma) override
  Set class members.
- void set_tolerance (double tolerance_map_cor) override
  Set tolerance.

Private Attributes

- MatrixXd _sqrtSigma
- MatrixXd _sqrtSigmaInv

5.22.1 Detailed Description

Class for the computation of the exponential map when `manifold_metric=="Frobenius"`

5.22.2 Member Function Documentation

5.22.2.1 map2manifold()

```
MatrixXd expMapFrob::map2manifold (
            const MatrixXd & M ) const  [override], [virtual]
```

Map a tangent space matrix to the manifold.

Parameters

M	Tangent space matrix to map

Returns

Manifold matrix identifying the mapped data

Implements map_functions::exponentialMap.

5.22.2.2 set_members()

```
void expMapFrob::set_members (
            const MatrixXd & Sigma )  [override], [virtual]
```

Set class members.

Parameters

Sigma	Tangent point

Implements map_functions::exponentialMap.

5.22.2.3 set_tolerance()

```
void expMapFrob::set_tolerance (
            double tolerance_map_cor )  [override], [virtual]
```

Set tolerance.

Parameters

tolerance_map_cor	Tolerance

Implements map_functions::exponentialMap.

5.22.3 Member Data Documentation

5.22.3.1 _sqrtSigma

```
MatrixXd map_functions::expMapFrob::_sqrtSigma  [private]
```

Square root of the tangent point `Sigma`

5.22.3.2 _sqrtSigmaInv

```
MatrixXd map_functions::expMapFrob::_sqrtSigmaInv  [private]
```

Inverse of the square root of the tangent point `Sigma`

The documentation for this class was generated from the following files:

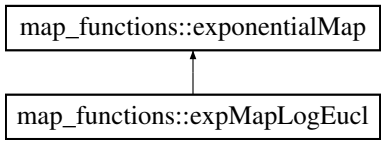
- MapFunctions.hpp
- MapFunctions.cpp

5.23 map_functions::expMapLogEucl Class Reference

Class for the computation of the exponential map when `manifold_metric=="LogEuclidean"`

```
#include <MapFunctions.hpp>
```

Inheritance diagram for map_functions::expMapLogEucl:



Public Member Functions

- ~expMapLogEucl ()=default
  Destructor.
- MatrixXd map2manifold (const MatrixXd &M) const override
  Map a tangent space matrix to the manifold.
- void set_members (const MatrixXd &Sigma) override
  Set class members.
- void set_tolerance (double tolerance_map_cor) override
  Set tolerance.

Private Attributes

- MatrixXd _Sigma

5.23.1 Detailed Description

Class for the computation of the exponential map when `manifold_metric=="LogEuclidean"`

5.23.2 Member Function Documentation

5.23.2.1 map2manifold()

```
MatrixXd expMapLogEucl::map2manifold (
            const MatrixXd & M ) const  [override], [virtual]
```

Map a tangent space matrix to the manifold.

Parameters

M	Tangent space matrix to map

Returns

Manifold matrix identifying the mapped data

Implements map_functions::exponentialMap.

5.23.2.2 set_members()

```
void expMapLogEucl::set_members (
            const MatrixXd & Sigma )  [override], [virtual]
```

Set class members.

Parameters

Sigma	Tangent point

Implements map_functions::exponentialMap.

5.23.2.3 set_tolerance()

```
void expMapLogEucl::set_tolerance (
            double tolerance_map_cor )  [override], [virtual]
```

Set tolerance.

Parameters

tolerance_map_cor	Tolerance

Implements map_functions::exponentialMap.

5.23.3 Member Data Documentation

5.23.3.1 _Sigma

```
MatrixXd map_functions::expMapLogEucl::_Sigma  [private]
```

Tangent point `Sigma`

The documentation for this class was generated from the following files:

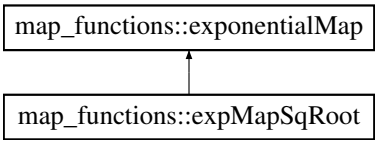
- MapFunctions.hpp
- MapFunctions.cpp

5.24 map_functions::expMapSqRoot Class Reference

Class for the computation of the exponential map when `manifold_metric=="SqRoot"`

`#include <MapFunctions.hpp>`

Inheritance diagram for map_functions::expMapSqRoot:



Public Member Functions

- ~expMapSqRoot ()=default
  Destructor.
- MatrixXd map2manifold (const MatrixXd &M) const override
  Map a tangent space matrix to the manifold.
- void set_members (const MatrixXd &Sigma) override
  Set class members.
- void set_tolerance (double tolerance_map_cor) override
  Set tolerance.

Private Attributes

- MatrixXd _Sigma

5.24.1 Detailed Description

Class for the computation of the exponential map when `manifold_metric=="SqRoot"`

5.24.2 Member Function Documentation

5.24.2.1 map2manifold()

```
MatrixXd expMapSqRoot::map2manifold (
            const MatrixXd & M ) const  [override], [virtual]
```

Map a tangent space matrix to the manifold.

Parameters

M	Tangent space matrix to map

Returns

Manifold matrix identifying the mapped data

Implements map_functions::exponentialMap.

5.24.2.2 set_members()

```
void expMapSqRoot::set_members (
            const MatrixXd & Sigma )  [override], [virtual]
```

Set class members.

Parameters

Sigma	Tangent point

Implements map_functions::exponentialMap.

5.24.2.3 set_tolerance()

```
void expMapSqRoot::set_tolerance (
            double tolerance_map_cor )  [override], [virtual]
```

Set tolerance.

Parameters

tolerance_map_cor	Tolerance

Implements map_functions::exponentialMap.

5.24.3 Member Data Documentation

5.24.3.1 _Sigma

```
MatrixXd map_functions::expMapSqRoot::_Sigma  [private]
```

Tangent point `Sigma`

The documentation for this class was generated from the following files:

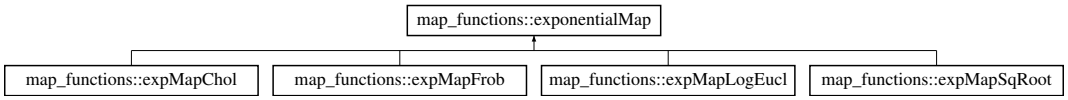
- MapFunctions.hpp
- MapFunctions.cpp

5.25 map_functions::exponentialMap Class Reference

Abstract class for the computation of the exponential map.

`#include <MapFunctions.hpp>`

Inheritance diagram for map_functions::exponentialMap:



Public Member Functions

- virtual ~exponentialMap ()=default

  Destructor.
- virtual MatrixXd map2manifold (const MatrixXd &M) const =0

  Map a tangent space matrix to the manifold.
- virtual void set_members (const MatrixXd &Sigma)=0

  Set class members.
- virtual void set_tolerance (double tolerance_map_cor)=0

  Set tolerance.

5.25.1 Detailed Description

Abstract class for the computation of the exponential map.

5.25.2 Member Function Documentation

5.25.2.1 map2manifold()

```
virtual MatrixXd map_functions::exponentialMap::map2manifold (
            const MatrixXd & M ) const  [pure virtual]
```

Map a tangent space matrix to the manifold.

Parameters

M	Tangent space matrix to map

Returns

Manifold matrix identifying the mapped data

Implemented in map_functions::expMapChol, map_functions::expMapSqRoot, map_functions::expMapLogEucl, and map_functions::expMapFrob.

5.25.2.2 set_members()

```
virtual void map_functions::exponentialMap::set_members (
            const MatrixXd & Sigma )  [pure virtual]
```

Set class members.

Parameters

Sigma	Tangent point

Implemented in map_functions::expMapChol, map_functions::expMapSqRoot, map_functions::expMapLogEucl, and map_functions::expMapFrob.

5.25.2.3 set_tolerance()

```
virtual void map_functions::exponentialMap::set_tolerance (
            double tolerance_map_cor )  [pure virtual]
```

Set tolerance.

Parameters

tolerance_map_cor	Tolerance

Implemented in map_functions::expMapChol, map_functions::expMapSqRoot, map_functions::expMapLogEucl, and map_functions::expMapFrob.

The documentation for this class was generated from the following file:

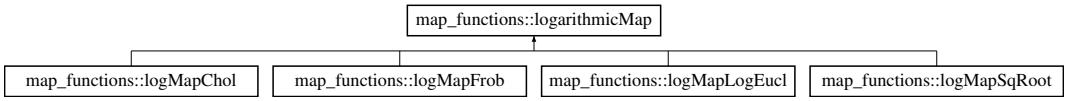
- MapFunctions.hpp

5.26 map_functions::logarithmicMap Class Reference

Abstract class for the computation of the logarithmic map.

`#include <MapFunctions.hpp>`

Inheritance diagram for map_functions::logarithmicMap:



Public Member Functions

- virtual ~logarithmicMap ()=default

  Destructor.
- virtual MatrixXd map2tplane (const MatrixXd &M) const =0

  Map a manifold matrix to the tangent space.
- virtual void set_members (const MatrixXd &Sigma)=0

  Set class members.
- virtual void set_tolerance (double tolerance_map_cor)=0

  Set tolerance.

5.26.1 Detailed Description

Abstract class for the computation of the logarithmic map.

5.26.2 Member Function Documentation

5.26.2.1 map2tplane()

```
virtual MatrixXd map_functions::logarithmicMap::map2tplane (
            const MatrixXd & M ) const  [pure virtual]
```

Map a manifold matrix to the tangent space.

Parameters

M	Manifold matrix to map

Returns

Tangent space matrix identifying the mapped data

Implemented in map_functions::logMapChol, map_functions::logMapSqRoot, map_functions::logMapLogEucl, and map_functions::logMapFrob.

5.26.2.2 set_members()

```
virtual void map_functions::logarithmicMap::set_members (
            const MatrixXd & Sigma )  [pure virtual]
```

Set class members.

Parameters

Sigma	Tangent point

Implemented in map_functions::logMapChol, map_functions::logMapSqRoot, map_functions::logMapLogEucl, and map_functions::logMapFrob.

5.26.2.3 set_tolerance()

```
virtual void map_functions::logarithmicMap::set_tolerance (
            double tolerance_map_cor )  [pure virtual]
```

Set tolerance.

Parameters

tolerance_map_cor	Tolerance

Implemented in map_functions::logMapChol, map_functions::logMapSqRoot, map_functions::logMapLogEucl, and map_functions::logMapFrob.

The documentation for this class was generated from the following file:

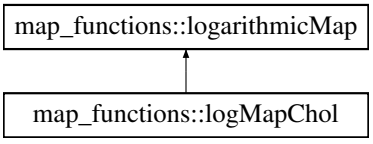
- MapFunctions.hpp

5.27 map_functions::logMapChol Class Reference

Class for the computation of the logarithmic map when `manifold_metric=="Correlation"`

`#include <MapFunctions.hpp>`

Inheritance diagram for map_functions::logMapChol:



Public Member Functions

- ~logMapChol ()=default

  Destructor.
- MatrixXd map2tplane (const MatrixXd &M) const override

  Map a manifold matrix to the tangent space.
- void set_members (const MatrixXd &Sigma) override

  Set class members.
- void set_tolerance (double tolerance_map_cor) override

  Set tolerance.

Private Member Functions

- Vec proj2tspace (const Vec &, const Vec &) const

  Project a matrix in $Chol(p)$ to the tangent space.

Private Attributes

- MatrixXd _Sigma
- double _tolerance_map_cor

5.27.1 Detailed Description

Class for the computation of the logarithmic map when `manifold_metric=="Correlation"`

5.27.2 Member Function Documentation

5.27.2.1 map2tplane()

```
MatrixXd logMapChol::map2tplane (
            const MatrixXd & M ) const  [override], [virtual]
```

Map a manifold matrix to the tangent space.

Parameters

M	Manifold matrix to map

Returns

Tangent space matrix identifying the mapped data

Implements map_functions::logarithmicMap.

5.27.2.2 set_members()

```
void logMapChol::set_members (
            const MatrixXd & Sigma )  [override], [virtual]
```

Set class members.

Parameters

Sigma	Tangent point

Implements map_functions::logarithmicMap.

5.27.2.3 set_tolerance()

```
void logMapChol::set_tolerance (
            double tolerance_map_cor )  [override], [virtual]
```

Set tolerance.

Parameters

tolerance_map_cor	Tolerance

Implements map_functions::logarithmicMap.

5.27.3 Member Data Documentation

5.27.3.1 _Sigma

```
MatrixXd map_functions::logMapChol::_Sigma  [private]
```

Tangent point `Sigma`

5.27.3.2 _tolerance_map_cor

```
double map_functions::logMapChol::_tolerance_map_cor  [private]
```

Tolerance on the norm of the columns to avoid `Nan`

The documentation for this class was generated from the following files:

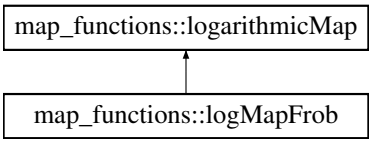
- MapFunctions.hpp
- MapFunctions.cpp

5.28 map_functions::logMapFrob Class Reference

Class for the computation of the logarithmic map when `manifold_metric=="Frobenius"`

`#include <MapFunctions.hpp>`

Inheritance diagram for map_functions::logMapFrob:



Public Member Functions

- ~logMapFrob ()=default
  Destructor.
- MatrixXd map2tplane (const MatrixXd &M) const override
  Map a manifold matrix to the tangent space.
- void set_members (const MatrixXd &Sigma) override
  Set class members.
- void set_tolerance (double tolerance_map_cor) override
  Set tolerance.

Private Attributes

- MatrixXd _sqrtSigma
- MatrixXd _sqrtSigmaInv

5.28.1 Detailed Description

Class for the computation of the logarithmic map when `manifold_metric=="Frobenius"`

5.28.2 Member Function Documentation

5.28.2.1 map2tplane()

```
MatrixXd logMapFrob::map2tplane (
            const MatrixXd & M ) const  [override], [virtual]
```

Map a manifold matrix to the tangent space.

Parameters

M	Manifold matrix to map

Returns

Tangent space matrix identifying the mapped data

Implements map_functions::logarithmicMap.

5.28.2.2 set_members()

```
void logMapFrob::set_members (
            const MatrixXd & Sigma )  [override], [virtual]
```

Set class members.

Parameters

Sigma	Tangent point

Implements map_functions::logarithmicMap.

5.28.2.3 set_tolerance()

```
void logMapFrob::set_tolerance (
            double tolerance_map_cor )  [override], [virtual]
```

Set tolerance.

Parameters

tolerance_map_cor	Tolerance

Implements map_functions::logarithmicMap.

5.28.3 Member Data Documentation

5.28.3.1 _sqrtSigma

```
MatrixXd map_functions::logMapFrob::_sqrtSigma  [private]
```

Square root of the tangent point `Sigma`

5.28.3.2 _sqrtSigmaInv

```
MatrixXd map_functions::logMapFrob::_sqrtSigmaInv  [private]
```

Inverse of the square root of the tangent point `Sigma`

The documentation for this class was generated from the following files:

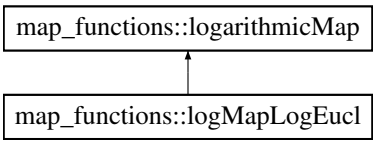
- MapFunctions.hpp
- MapFunctions.cpp

5.29 map_functions::logMapLogEucl Class Reference

Class for the computation of the logarithmic map when `manifold_metric=="LogEuclidean"`

```
#include <MapFunctions.hpp>
```

Inheritance diagram for map_functions::logMapLogEucl:



Public Member Functions

- ~logMapLogEucl ()=default
  Destructor.
- MatrixXd map2tplane (const MatrixXd &M) const override
  Map a manifold matrix to the tangent space.
- void set_members (const MatrixXd &Sigma) override
  Set class members.
- void set_tolerance (double tolerance_map_cor) override
  Set tolerance.

Private Attributes

- MatrixXd _Sigma

5.29.1 Detailed Description

Class for the computation of the logarithmic map when `manifold_metric=="LogEuclidean"`

5.29.2 Member Function Documentation

5.29.2.1 map2tplane()

```
MatrixXd logMapLogEucl::map2tplane (
            const MatrixXd & M ) const  [override], [virtual]
```

Map a manifold matrix to the tangent space.

Parameters

M	Manifold matrix to map

Returns

Tangent space matrix identifying the mapped data

Implements map_functions::logarithmicMap.

5.29.2.2 set_members()

```
void logMapLogEucl::set_members (
            const MatrixXd & Sigma )  [override], [virtual]
```

Set class members.

Parameters

Sigma	Tangent point

Implements map_functions::logarithmicMap.

5.29.2.3 set_tolerance()

```
void logMapLogEucl::set_tolerance (
            double tolerance_map_cor )  [override], [virtual]
```

Set tolerance.

Parameters

tolerance_map_cor	Tolerance

Implements map_functions::logarithmicMap.

5.29.3 Member Data Documentation

5.29.3.1 _Sigma

```
MatrixXd map_functions::logMapLogEucl::_Sigma  [private]
```

Tangent point `Sigma`

The documentation for this class was generated from the following files:

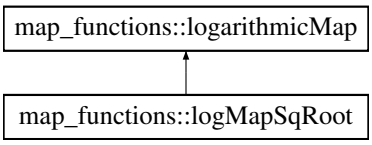
- MapFunctions.hpp
- MapFunctions.cpp

5.30 map_functions::logMapSqRoot Class Reference

Class for the computation of the logarithmic map when `manifold_metric=="SqRoot"`

`#include <MapFunctions.hpp>`

Inheritance diagram for map_functions::logMapSqRoot:



Public Member Functions

- ~logMapSqRoot ()=default
  Destructor.
- MatrixXd map2tplane (const MatrixXd &M) const override
  Map a manifold matrix to the tangent space.
- void set_members (const MatrixXd &Sigma) override
  Set class members.
- void set_tolerance (double tolerance_map_cor) override
  Set tolerance.

Private Attributes

- MatrixXd _Sigma

5.30.1 Detailed Description

Class for the computation of the logarithmic map when `manifold_metric=="SqRoot"`

5.30.2 Member Function Documentation

5.30.2.1 map2tplane()

```
MatrixXd logMapSqRoot::map2tplane (
            const MatrixXd & M ) const  [override], [virtual]
```

Map a manifold matrix to the tangent space.

Parameters

M	Manifold matrix to map

Returns

Tangent space matrix identifying the mapped data

Implements map_functions::logarithmicMap.

5.30.2.2 set_members()

```
void logMapSqRoot::set_members (
            const MatrixXd & Sigma )  [override], [virtual]
```

Set class members.

Parameters

Sigma	Tangent point

Implements map_functions::logarithmicMap.

5.30.2.3 set_tolerance()

```
void logMapSqRoot::set_tolerance (
            double tolerance_map_cor )  [override], [virtual]
```

Set tolerance.

Parameters

tolerance_map_cor	Tolerance

Implements map_functions::logarithmicMap.

5.30.3 Member Data Documentation

5.30.3.1 _Sigma

```
MatrixXd map_functions::logMapSqRoot::_Sigma  [private]
```

Tangent point `Sigma`

The documentation for this class was generated from the following files:

- MapFunctions.hpp
- MapFunctions.cpp

5.31 model_fit::Model Class Reference

Class to compute and store the linear model on the tangent space.

`#include <Model.hpp>`

Public Member Functions

- Model (const std::shared_ptr< const MatrixXd > data_tspace, const std::shared_ptr< const MatrixXd > design_matrix_model, unsigned int p, const std::string &distance_Manifold_name)

  Constructor.
- Model (const std::shared_ptr< const MatrixXd > data_tspace, const std::shared_ptr< const MatrixXd > design_matrix_model, const std::shared_ptr< const MatrixXd > design_matrix_tot, unsigned int p, const std::string &distance_Manifold_name)

  Constructor.
- void update_model (const MatrixXd &gamma_matrix)

  Update `_beta_matrix`, `_fitted_values` and `_residuals` according to the new covariogram matrix.
- MatrixXd get_beta () const

  Return `_beta_matrix`
- MatrixXd get_residuals () const

  Return `_residuals`
- MatrixXd get_fitted_values () const

  Return `_fitted_values`

Private Attributes

- const std::shared_ptr< const MatrixXd > _data_tspace
- const std::shared_ptr< const MatrixXd > _design_matrix_model
- const std::shared_ptr< const MatrixXd > _design_matrix_tot
- const std::string _distance_Manifold_name
- const unsigned int _N
- const unsigned int _p
- const unsigned int _num_cov
- const unsigned int _num_coeff
- MatrixXd _beta_matrix
- MatrixXd _fitted_values
- MatrixXd _residuals

5.31.1 Detailed Description

Class to compute and store the linear model on the tangent space.

5.31.2 Member Function Documentation

5.31.2.1 update_model()

```
void Model::update_model (
            const MatrixXd & gamma_matrix )
```

Update _beta_matrix, _fitted_values and _residuals according to the new covariogram matrix.

Parameters

gamma_matrix	Covariogram matrix

5.31.3 Member Data Documentation

5.31.3.1 _data_tspace

```
const std::shared_ptr<const MatrixXd> model_fit::Model::_data_tspace  [private]
```

BigMatrix of the data on the tangent space

5.31.3.2 _design_matrix_model

```
const std::shared_ptr<const MatrixXd> model_fit::Model::_design_matrix_model  [private]
```

Design matrix for the data in the cell (used to compute the beta)

5.31.3.3 _design_matrix_tot

```
const std::shared_ptr<const MatrixXd> model_fit::Model::_design_matrix_tot  [private]
```

Design matrix for all the data in the domain (used ti compute the residuals)

5.31.3.4 _distance_Manifold_name

```
const std::string model_fit::Model::_distance_Manifold_name  [private]
```

Name of the metric on the manifold

5.31.3.5 _N

```
const unsigned int model_fit::Model::_N  [private]
```

Number of stations in the cell

5.31.3.6 _p

```
const unsigned int model_fit::Model::_p  [private]
```

Dimension of the matrices on the manifold

5.31.3.7 _num_cov

```
const unsigned int model_fit::Model::_num_cov  [private]
```

Number of covariates in the model

5.31.3.8 _num_coeff

```
const unsigned int model_fit::Model::_num_coeff  [private]
```

Number of significant entries in a symmetric $(\_p * \_p)$ matrix. _num_coeff $= \frac{\_p*(\_p+1)}{2}$

5.31.3.9 _beta_matrix

```
MatrixXd model_fit::Model::_beta_matrix  [private]
```

$(\_num\_cov * \_num\_coeff)$ matrix where the i^{th} row contains the upper triangular part of $\beta_{..i}$, the i^{th} coefficient of the tangent space linear model

5.31.3.10 _fitted_values

```
MatrixXd model_fit::Model::_fitted_values  [private]
```

$(N\_tot * \_num\_coeff)$ matrix where the i^{th} row contains the upper triangular part of the matrix fitted in the i^{th} location on the tangent space by the linear model. _fitted_values $= (*(\_design\_matrix\_tot)) * \_beta\_matrix$

5.31.3.11 _residuals

```
MatrixXd model_fit::Model::_residuals  [private]
```

$(N\_tot * \_num\_coeff)$ matrix where the i^{th} row contains the upper triangular part of the residual matrix in the i^{th} location. _residuals $= (*(\_data\_tspace)) - \_fitted\_values$

The documentation for this class was generated from the following files:

- Model.hpp
- Model.cpp

5.32 variogram_evaluation::EmpiricalVariogram Class Reference

Class for computation and storage of the empirical variogram.

`#include <EmpiricalVariogram.hpp>`

Public Member Functions

- EmpiricalVariogram (const std::shared_ptr< const MatrixXd >, unsigned int, const Coordinates &, const distances::Distance &)

  Constructor.
- EmpiricalVariogram (const std::shared_ptr< const MatrixXd >, unsigned int, unsigned int, const Vec &, double)

  Constructor.
- void update_emp_vario (const std::vector< MatrixXd > &res, const distances_tplane::DistanceTplane &distanceTplane)

  Update `_emp_vario_values`, `_hvec` *and* `_N_hvec` *according to the new residuals* $\Delta(\boldsymbol{s_i})\, i = 1, \cdot, \_N.$
- std::vector< double > get_emp_vario_values () const

  Return `_emp_vario_values`
- std::vector< unsigned int > get_N_hvec () const

  Return `_N_hvec`
- std::vector< double > get_hvec () const

  Return `_hvec`
- unsigned int get_card_h () const

  Return `_card_h`
- unsigned int get_N () const

  Return `_N`
- double get_hmax () const

  Return `_hmax`

Private Member Functions

- void compute_hmax (const Coordinates &, const distances::Distance &)

  Compute the maximum distance to be considered.

Private Attributes

- const unsigned int _n_h
- const unsigned int _N
- const std::shared_ptr< const MatrixXd > _distanceMatrix
- std::vector< double > _emp_vario_values
- std::vector< double > _hvec
- std::vector< unsigned int > _N_hvec
- unsigned int _card_h
- Vec _d
- double _hmax
- Vec _weights

5.32.1 Detailed Description

Class for computation and storage of the empirical variogram.

5.32.2 Member Function Documentation

5.32.2.1 update_emp_vario()

```
void EmpiricalVariogram::update_emp_vario (
            const std::vector< MatrixXd > & res,
            const distances_tplane::DistanceTplane & distanceTplane )
```

Update _emp_vario_values, _hvec and _N_hvec according to the new residuals $\Delta(\boldsymbol{s_i})\, i = 1, \cdot, \_N$.

_emp_vario_values are estimated through the method of moments:

$$\widehat{\gamma}(h) = \frac{\sum\limits_{N(h)} w(\boldsymbol{s_i})\, w(\boldsymbol{s_j})\, \|\Delta(\boldsymbol{s_i}) - \Delta(\boldsymbol{s_j})\|^2}{2 \sum\limits_{N(h)} w(\boldsymbol{s_i})\, w(\boldsymbol{s_j})}$$

where $N(h) = \{(\boldsymbol{s}_i, \boldsymbol{s}_j \in D) \,:\, h - \Delta h \leq \|\boldsymbol{s}_i - \boldsymbol{s}_j\| \leq h - \Delta h\}$

Parameters

res	Vector of the N residual matrices
distanceTplane	Distance on the tangent space

5.32.3 Member Data Documentation

5.32.3.1 _n_h

```
const unsigned int variogram_evaluation::EmpiricalVariogram::_n_h  [private]
```

Number of bins

5.32.3.2 _N

```
const unsigned int variogram_evaluation::EmpiricalVariogram::_N  [private]
```

Number of data points used to compute the empirical variogram

5.32.3.3 _distanceMatrix

```
const std::shared_ptr<const MatrixXd> variogram_evaluation::EmpiricalVariogram::_distance↩
Matrix  [private]
```

Matrix of distances among the N locations

5.32.3.4 _emp_vario_values

```
std::vector<double> variogram_evaluation::EmpiricalVariogram::_emp_vario_values  [private]
```

Vector storing the variogram estimates. _emp_vario_values[i] $= \hat{\gamma}$ (_hvec[i])

5.32.3.5 _hvec

```
std::vector<double> variogram_evaluation::EmpiricalVariogram::_hvec  [private]
```

Vector storing the distances at which the variogram is estimated

5.32.3.6 _N_hvec

```
std::vector<unsigned int> variogram_evaluation::EmpiricalVariogram::_N_hvec  [private]
```

Vector storing the number of data used in the estimation of the corresponding empirical variogram value

5.32.3.7 _card_h

```
unsigned int variogram_evaluation::EmpiricalVariogram::_card_h  [private]
```

Number of distances for which the variogram has been evaluated. `_card_h = _hvec.size()`

5.32.3.8 _d

```
Vec variogram_evaluation::EmpiricalVariogram::_d  [private]
```

Vector of equispaced distances whose midpoints are the candidates to enter in `_hvec`

5.32.3.9 _hmax

```
double variogram_evaluation::EmpiricalVariogram::_hmax  [private]
```

Maximum distance considered

5.32.3.10 _weights

```
Vec variogram_evaluation::EmpiricalVariogram::_weights  [private]
```

Vector of the weights for the N data

The documentation for this class was generated from the following files:

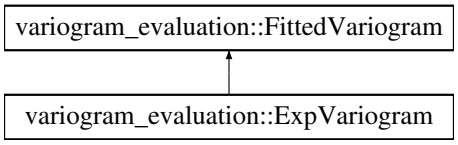
- EmpiricalVariogram.hpp
- EmpiricalVariogram.cpp

5.33 variogram_evaluation::ExpVariogram Class Reference

Class for computation and storage of the fitted variogram when `vario_model=="Exponential"`

`#include <FittedVariogram.hpp>`

Inheritance diagram for variogram_evaluation::ExpVariogram:



Public Member Functions

- double get_vario_univ (const double &h) const override

  Compute the value of the model variogram at a given distance, according to the variogram type.
- ~ExpVariogram ()=default

  Destructor.

Private Member Functions

- void get_init_par (const EmpiricalVariogram &) override

  Initialize `_parameters`
- MatrixXd compute_jacobian (const std::vector< double > &, unsigned int) const override

  Compute the jacobian of the variogram residuals (which coincides with the one of the model variogram), according to the variogram type.

Additional Inherited Members

5.33.1 Detailed Description

Class for computation and storage of the fitted variogram when `vario_model=="Exponential"`

5.33.2 Member Function Documentation

5.33.2.1 get_init_par()

```
void ExpVariogram::get_init_par (
            const EmpiricalVariogram & emp_vario )  [override], [private], [virtual]
```

Initialize _parameters

The _parameters are initialized as follows:
_parameters (0) = weighted_median (_emp_vario_values.head (2) , _N_hvec.head (2))
_parameters (1) = weighted_median (_emp_vario_values.tail (4) , _N_hvec.tail (4)) − _parameters (0)
_parameters (2) = _hvec (i^*)
where i^* is the first i s.t. |_emp_vario_values (i) − (_parameters (0) + _parameters (1))| < tol

Parameters

emp_vario	Empirical variogram

Implements variogram_evaluation::FittedVariogram.

5.33.2.2 get_vario_univ()

```
double ExpVariogram::get_vario_univ (
            const double & h ) const  [override], [virtual]
```

Compute the value of the model variogram at a given distance, according to the variogram type.

Parameters

h	The distance where to evaluate the variogram

Returns

Variogram value at distance h: $\gamma_m(h)$

Implements variogram_evaluation::FittedVariogram.

The documentation for this class was generated from the following files:

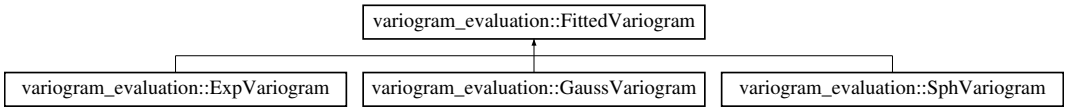
- FittedVariogram.hpp
- FittedVariogram.cpp

5.34 variogram_evaluation::FittedVariogram Class Reference

Abstract class for computation and storage of the fitted variogram.

`#include <FittedVariogram.hpp>`

Inheritance diagram for variogram_evaluation::FittedVariogram:



Public Member Functions

- double get_tau2 () const

  Return `_parameters(0)`, i.e. the nugget.
- double get_sigma2 () const

  Return `_parameters(1)`, i.e. the sill-nugget.
- double get_a () const

  Return `_parameters(2)`, i.e. the practical range.
- void evaluate_par_fitted_E (const EmpiricalVariogram &emp_vario, double max_sill, double max_a)

  Compute the parameters of the fitted variogram.
- void evaluate_par_fitted_W (const EmpiricalVariogram &emp_vario, double max_sill, double max_a)

  Compute the parameters of the fitted variogram.
- virtual double get_vario_univ (const double &h) const =0

  Compute the value of the model variogram at a given distance, according to the variogram type.
- double get_covario_univ (const double &h) const

  Compute the value of the model covariogram at a given distance, according to the variogram type.
- Vec get_vario_vec (const std::vector< double > &h_vec, unsigned int card_h) const

  Compute the values of the model variogram at a given vector of distances, according to the variogram type.
- Vec get_vario_vec (const Vec &h_vec, unsigned int card_h) const

  Compute the values of the model variogram at a given vector of distances, according to the variogram type.
- MatrixXd compute_gamma_matrix (const std::shared_ptr< const MatrixXd > distanceMatrix, unsigned int N) const

  Compute the covariogram matrix, according to the variogram type.
- Vector3d get_parameters () const

  Return `_parameters`
- void set_parameters (const Vector3d &parameters)

  Set _parameters.
- Vec get_covario_vec (const std::vector< double > &h_vec, unsigned int card_h) const

  Compute the values of the model covariogram at a given vector of distances, according to the variogram type.
- virtual ~FittedVariogram ()=default

  Destructor.

Protected Member Functions

- double weighted_median (const std::vector< double > &values, const std::vector< unsigned int > &card)

  Compute weighted median.
- virtual void get_init_par (const EmpiricalVariogram &emp_vario)=0

  Initialize `_parameters`
- void backtrack (const Vector3d &dir, Vector3d &gk, Vec &res, const std::vector< double > &h_vec, unsigned int card_h, double c, double s, const Vec &emp_vario_values, double max_sill, double max_a)

  Update `_parameters` *moving along* `dir`
- virtual MatrixXd compute_jacobian (const std::vector< double > &h_vec, unsigned int card_h) const =0

  Compute the jacobian of the variogram residuals (which coincides with the one of the model variogram), according to the variogram type.

Protected Attributes

- Vector3d _parameters

5.34.1 Detailed Description

Abstract class for computation and storage of the fitted variogram.

5.34.2 Member Function Documentation

5.34.2.1 weighted_median()

```
double FittedVariogram::weighted_median (
            const std::vector< double > & values,
            const std::vector< unsigned int > & card )  [protected]
```

Compute weighted median.

Parameters

values	Values whose median must be computed
card	Weights

Returns

Median of `values`, weighted using `card`

5.34.2.2 get_init_par()

```
virtual void variogram_evaluation::FittedVariogram::get_init_par (
            const EmpiricalVariogram & emp_vario )  [protected], [pure virtual]
```

Initialize _parameters

The _parameters are initialized as follows:
_parameters (0) = weighted_median (_emp_vario_values.head (2) , _N_hvec.head (2))
_parameters (1) = weighted_median (_emp_vario_values.tail (4) , _N_hvec.tail (4)) − _parameters (0)
_parameters (2) = _hvec (i^*)
where i^* is the first i s.t. |_emp_vario_values (i) − (_parameters (0) + _parameters (1))| < tol

Parameters

emp_vario	Empirical variogram

Implemented in variogram_evaluation::SphVariogram, variogram_evaluation::ExpVariogram, and variogram_evaluation::GaussVariogr

5.34.2.3 evaluate_par_fitted_E()

```
void FittedVariogram::evaluate_par_fitted_E (
            const EmpiricalVariogram & emp_vario,
            double max_sill,
            double max_a )
```

Compute the parameters of the fitted variogram.

Note

Like variogram_evaluation::FittedVariogram::evaluate_par_fitted_W, but different stopping criteria. This function is used when equal weights are considered

The parameters are computed using Gauss-Newton with backtrack method to solve the non-linear least square problem:

$$\begin{aligned} \underset{\text{\_parameters}}{\arg\min} \quad & \sum_{h \in \text{\_hvec}} \left(\gamma_m\left(\text{\_parameters}, h\right) - \widehat{\gamma}\left(h\right)\right)^2 \\ \text{subject to} \quad & 0 \leq \text{\_parameters}\left(0\right) \\ & 0 \leq \text{\_parameters}\left(1\right) \leq \text{max\_sill} - \text{\_parameters}\left(0\right) \\ & 0 \leq \text{\_parameters}\left(2\right) \leq \text{max\_a} \end{aligned}$$

The starting values for the _parameters are obtained through variogram_evaluation::FittedVariogram::get_init_par.

The stopping criteria is based on the decrease of the error norm.

Parameters

emp_vario	Empirical variogram
max_sill	Maximum value for the *sill*
max_a	Maximum value for *a*

5.34.2.4 evaluate_par_fitted_W()

```
void FittedVariogram::evaluate_par_fitted_W (
            const EmpiricalVariogram & emp_vario,
            double max_sill,
            double max_a )
```

Compute the parameters of the fitted variogram.

Note

Like variogram_evaluation::FittedVariogram::evaluate_par_fitted_E, but different stopping criteria. This function is used when kernel weights are considered

The parameters are computed using Gauss-Newton with backtrack method to solve the non-linear least square problem:

$$\begin{aligned} \underset{\text{\_parameters}}{\arg\min} \quad & \sum_{h \in \text{\_hvec}} \left(\gamma_m\left(\text{\_parameters}, h\right) - \widehat{\gamma}\left(h\right)\right)^2 \\ \text{subject to} \quad & 0 \leq \text{\_parameters}\left(0\right) \\ & 0 \leq \text{\_parameters}\left(1\right) \leq \text{max\_sill} - \text{\_parameters}\left(0\right) \\ & 0 \leq \text{\_parameters}\left(2\right) \leq \text{max\_a} \end{aligned}$$

The starting values for the `_parameters` are obtained through variogram_evaluation::FittedVariogram::get_init_par.

The stopping criteria is based on the difference in the decrease of the error norm between two consecutive iterations.

Parameters

emp_vario	Empirical variogram
max_sill	Maximum value for the *sill*
max_a	Maximum value for *a*

5.34.2.5 get_vario_univ()

```
virtual double variogram_evaluation::FittedVariogram::get_vario_univ (
            const double & h ) const  [pure virtual]
```

Compute the value of the model variogram at a given distance, according to the variogram type.

Parameters

h	The distance where to evaluate the variogram

Returns

Variogram value at distance h: $\gamma_m\left(h\right)$

Implemented in variogram_evaluation::SphVariogram, variogram_evaluation::ExpVariogram, and variogram_evaluation::GaussVariogr

5.34.2.6 get_covario_univ()

```
double FittedVariogram::get_covario_univ (
            const double & h ) const
```

Compute the value of the model covariogram at a given distance, according to the variogram type.

Parameters

h	The distance where to evaluate the covariogram

Returns

Covariogram value at distance h : $C_m(h) = (\text{\_parameters}(0) + \text{\_parameters}(1)) - \gamma_m(h)$

5.34.2.7 get_vario_vec() [1/2]

```
Vec FittedVariogram::get_vario_vec (
            const std::vector< double > & h_vec,
            unsigned int card_h ) const
```

Compute the values of the model variogram at a given vector of distances, according to the variogram type.

Parameters

h_vec	The distances where to evaluate the variogram
card↩ _h	Number of distances where the variogram must be computed. `card_h = h_vec.size()`

Returns

Vector of variogram values at distances $h \in$ `_h_vec`

5.34.2.8 get_vario_vec() [2/2]

```
Vec FittedVariogram::get_vario_vec (
            const Vec & h_vec,
            unsigned int card_h ) const
```

Compute the values of the model variogram at a given vector of distances, according to the variogram type.

Parameters

h_vec	The distances where to evaluate the variogram
card↩ _h	Number of distances where the variogram must be computed. `card_h = h_vec.size()`

Returns

Vector of variogram values at distances $h \in$ h_vec

5.34.2.9 compute_gamma_matrix()

```
MatrixXd FittedVariogram::compute_gamma_matrix (
            const std::shared_ptr< const MatrixXd > distanceMatrix,
            unsigned int N ) const
```

Compute the covariogram matrix, according to the variogram type.

Parameters

distanceMatrix	Matrix of distances among the N locations
N	Number of data locations

Returns

Covariogram matrix

5.34.2.10 set_parameters()

```
void FittedVariogram::set_parameters (
            const Vector3d & parameters )
```

Set _parameters.

Parameters

parameters	Vector of parameters' values

5.34.2.11 get_covario_vec()

```
Vec FittedVariogram::get_covario_vec (
            const std::vector< double > & h_vec,
            unsigned int card_h ) const
```

Compute the values of the model covariogram at a given vector of distances, according to the variogram type.

Parameters

h_vec	The distances where to evaluate the covariogram
card_h	Number of distances where the covariogram must be computed. card_h = h_vec.size()

Returns

Vector of covariogram values at distances $h \in$ `h_vec`

5.34.3 Member Data Documentation

5.34.3.1 _parameters

`Vector3d variogram_evaluation::FittedVariogram::_parameters [protected]`

Vector storing the three parameters of the fitted varriogram (nugget, *sill-nugget*, *practical range*)

The documentation for this class was generated from the following files:

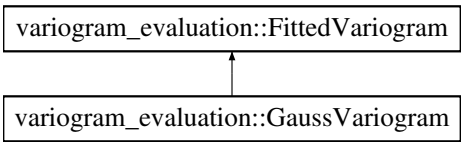
- FittedVariogram.hpp
- FittedVariogram.cpp

5.35 variogram_evaluation::GaussVariogram Class Reference

Class for computation and storage of the fitted variogram when `vario_model=="Gaussian"`

`#include <FittedVariogram.hpp>`

Inheritance diagram for variogram_evaluation::GaussVariogram:



Public Member Functions

- double get_vario_univ (const double &h) const override

  Compute the value of the model variogram at a given distance, according to the variogram type.
- ~GaussVariogram ()=default

  Destructor.

Private Member Functions

- void get_init_par (const EmpiricalVariogram &) override

  Initialize `_parameters`
- MatrixXd compute_jacobian (const std::vector< double > &, unsigned int) const override

  Compute the jacobian of the variogram residuals (which coincides with the one of the model variogram), according to the variogram type.

Additional Inherited Members

5.35.1 Detailed Description

Class for computation and storage of the fitted variogram when `vario_model=="Gaussian"`

5.35.2 Member Function Documentation

5.35.2.1 get_init_par()

```
void GaussVariogram::get_init_par (
            const EmpiricalVariogram & emp_vario )  [override], [private], [virtual]
```

Initialize `_parameters`

The `_parameters` are initialized as follows:
_parameters (0) = weighted_median (_emp_vario_values.head (2) , _N_hvec.head (2))
_parameters (1) = weighted_median (_emp_vario_values.tail (4) , _N_hvec.tail (4)) − _parameters (0)
_parameters (2) = _hvec (i^*)
where i^* is the first i s.t. |_emp_vario_values (i) − (_parameters (0) + _parameters (1))| < tol

Parameters

emp_vario	Empirical variogram

Implements variogram_evaluation::FittedVariogram.

5.35.2.2 get_vario_univ()

```
double GaussVariogram::get_vario_univ (
            const double & h ) const  [override], [virtual]
```

Compute the value of the model variogram at a given distance, according to the variogram type.

Parameters

h	The distance where to evaluate the variogram

Returns

Variogram value at distance h: $\gamma_m\,(h)$

Implements variogram_evaluation::FittedVariogram.

The documentation for this class was generated from the following files:

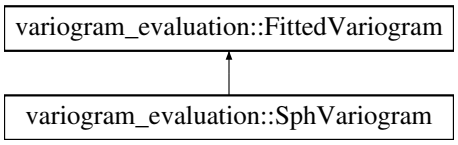
- FittedVariogram.hpp
- FittedVariogram.cpp

5.36 variogram_evaluation::SphVariogram Class Reference

Class for computation and storage of the fitted variogram when `vario_model=="Spherical"`

`#include <FittedVariogram.hpp>`

Inheritance diagram for variogram_evaluation::SphVariogram:



Public Member Functions

- double get_vario_univ (const double &h) const override

  Compute the value of the model variogram at a given distance, according to the variogram type.
- ∼SphVariogram ()=default

  Destructor.

Private Member Functions

- void get_init_par (const EmpiricalVariogram &) override

  Initialize `_parameters`
- MatrixXd compute_jacobian (const std::vector< double > &, unsigned int) const override

  Compute the jacobian of the variogram residuals (which coincides with the one of the model variogram), according to the variogram type.

Additional Inherited Members

5.36.1 Detailed Description

Class for computation and storage of the fitted variogram when `vario_model=="Spherical"`

5.36.2 Member Function Documentation

5.36.2.1 get_init_par()

```
void SphVariogram::get_init_par (
            const EmpiricalVariogram & emp_vario )  [override], [private], [virtual]
```

Initialize `_parameters`

The `_parameters` are initialized as follows:
_parameters (0) = weighted_median (_emp_vario_values.head (2) , _N_hvec.head (2))
_parameters (1) = weighted_median (_emp_vario_values.tail (4) , _N_hvec.tail (4)) − _parameters (0)
_parameters (2) = _hvec (i^*)
where i^* is the first i s.t. |_emp_vario_values (i) − (_parameters (0) + _parameters (1))| < tol

Parameters

emp_vario	Empirical variogram

Implements variogram_evaluation::FittedVariogram.

5.36.2.2 get_vario_univ()

```
double SphVariogram::get_vario_univ (
            const double & h ) const  [override], [virtual]
```

Compute the value of the model variogram at a given distance, according to the variogram type.

Parameters

h	The distance where to evaluate the variogram

Returns

Variogram value at distance h: $\gamma_m(h)$

Implements variogram_evaluation::FittedVariogram.

The documentation for this class was generated from the following files:

- FittedVariogram.hpp
- FittedVariogram.cpp

Chapter 6

File Documentation

6.1 Coordinates.hpp File Reference

Coordinates class.

```
#include <iostream>
#include <string>
#include <memory>
#include "Helpers.hpp"
```

Classes

- class Coordinates

  Class to store the coordinates of the data points.

6.1.1 Detailed Description

Coordinates class.

6.2 DesignMatrix.hpp File Reference

Classes to create the design_matrix according to the model on the tangent space.

```
#include <memory>
#include <iostream>
#include <utility>
#include <map>
#include <string>
#include <Eigen/Dense>
#include "Coordinates.hpp"
```

Classes

- class design_matrix::DesignMatrix

  Abstract class for the computation of the design_matrix.
- class design_matrix::InterceptDM

  Class for the computation of the design_matrix when `model_ts=="Intercept"`
- class design_matrix::Coord1DM

  Class for the computation of the design_matrix when `model_ts=="Coord1"`
- class design_matrix::Coord2DM

  Class for the computation of the design_matrix when `model_ts=="Coord2"`
- class design_matrix::AdditiveDM

  Class for the computation of the design_matrix when `model_ts=="Additive"`

6.2.1 Detailed Description

Classes to create the design_matrix according to the model on the tangent space.

6.3 Distance.hpp File Reference

Classes to create the compute the distances between data locations.

```
#include "Helpers.hpp"
#include "Coordinates.hpp"
#include <vector>
#include <utility>
#include <map>
#include <functional>
```

Classes

- class distances::Distance

  Abstract class for the computation of the distance between data locations.
- class distances::EuclDist

  Class for the computation of the distance between data locations when `distance=="Eucldist"`
- class distances::GeoDist

  Class for the computation of the distance between data locations when `distance=="Geodist"`

6.3.1 Detailed Description

Classes to create the compute the distances between data locations.

6.4 DistanceManifold.hpp File Reference

Classes to compute the distance on the manifold according to its metric.

```
#include "Helpers.hpp"
#include <vector>
#include <utility>
#include <map>
#include <functional>
#include <memory>
```

Classes

- class distances_manifold::DistanceManifold

  Abstract class for the computation of the distance on the manifold.
- class distances_manifold::Frobenius

  Class for the computation of the distance on the manifold when `manifold_metric=="Frobenius"`
- class distances_manifold::LogEuclidean

  Class for the computation of the distance on the manifold when `manifold_metric=="LogEuclidean"`
- class distances_manifold::SqRoot

  Class for the computation of the distance on the manifold when `manifold_metric=="SqRoot"`
- class distances_manifold::Chol

  Class for the computation of the distance on the manifold when `manifold_metric=="Chol"`

6.4.1 Detailed Description

Classes to compute the distance on the manifold according to its metric.

6.5 DistanceTplane.hpp File Reference

Classes to compute the distance on the tangent space according to its metric.

```
#include "Helpers.hpp"
#include <vector>
#include <utility>
#include <map>
#include <functional>
#include <memory>
```

Classes

- class distances_tplane::DistanceTplane

  Abstract class for the computation of the distance on the tangent space.
- class distances_tplane::Frobenius

  Class for the computation of the distance on the tangent space when `ts_metric=="Frobenius"`
- class distances_tplane::FrobeniusScaled

  Class for the computation of the distance on the tangent space when `ts_metric=="FrobeniusScaled"`
- class distances_tplane::Chol

  Class for the computation of the distance on the tangent space when `ts_metric=="Correlation"`

6.5.1 Detailed Description

Classes to compute the distance on the tangent space according to its metric.

6.6 EmpiricalVariogram.hpp File Reference

Class to compute the empirical variogram.

```
#include "Helpers.hpp"
#include "DistanceTplane.hpp"
#include "Distance.hpp"
#include "Coordinates.hpp"
```

Classes

- class variogram_evaluation::EmpiricalVariogram

  Class for computation and storage of the empirical variogram.

6.6.1 Detailed Description

Class to compute the empirical variogram.

6.7 Factory.hpp File Reference

Factory class.

```
#include <map>
#include <vector>
#include <memory>
#include <functional>
#include <stdexcept>
#include <type_traits>
#include <Rcpp.h>
```

Classes

- class generic_factory::Factory< AbstractProduct, Identifier, Builder >

  A generic factory.

6.7.1 Detailed Description

Factory class.

6.8 FittedVariogram.hpp File Reference

Classes to fit a model variogram (Gaussian, Exponential or Spherical) to an empirical one.

```
#include "EmpiricalVariogram.hpp"
#include <iostream>
#include <Rcpp.h>
```

Classes

- class variogram_evaluation::FittedVariogram

  Abstract class for computation and storage of the fitted variogram.
- class variogram_evaluation::GaussVariogram

  Class for computation and storage of the fitted variogram when `vario_model=="Gaussian"`
- class variogram_evaluation::ExpVariogram

  Class for computation and storage of the fitted variogram when `vario_model=="Exponential"`
- class variogram_evaluation::SphVariogram

  Class for computation and storage of the fitted variogram when `vario_model=="Spherical"`

6.8.1 Detailed Description

Classes to fit a model variogram (Gaussian, Exponential or Spherical) to an empirical one.

6.9 Helpers.hpp File Reference

Functions to manipulate matrices.

```
#include <Eigen/Sparse>
#include <Eigen/Dense>
#include <Eigen/Eigenvalues>
#include <Eigen/Cholesky>
#include <Eigen/SparseCholesky>
#include <cmath>
#include <string>
```

Typedefs

- typedef Eigen::VectorXd Vec

  Eigen dynamic vector of doubles.

Functions

- MatrixXd matrix_manipulation::expMat (const MatrixXd &A)

  Compute the exponential of a matrix.
- MatrixXd matrix_manipulation::logMat (const MatrixXd &A)

  Compute the logarithm of a matrix.
- MatrixXd matrix_manipulation::sqrtMat (const MatrixXd &A)

  Compute the square root of a matrix.
- std::vector< MatrixXd > matrix_manipulation::bigMatrix2VecMatrices (const MatrixXd &bigMatrix, unsigned int p, const std::string &distance_Manifold_name)

  Transform a $\left(\frac{p*(p+1)}{2}, N\right)$ *matrix in a vector of* $Np * p$ *symmetric matrices.*
- MatrixXd matrix_manipulation::VecMatrices2bigMatrix (const std::vector< MatrixXd > &vecMatrices)

  Transform a vector of $Np * p$ *symmetric matrices in a* $\left(\frac{p*(p+1)}{2}, N\right)$ *matrix.*
- MatrixXd matrix_manipulation::Chol_semidef (const MatrixXd &M1)

  Compute the Cholesky decomposition of a matrix not positive definite.
- MatrixXd matrix_manipulation::Chol_decomposition (const MatrixXd M1)

  Compute the Cholesky decomposition of a matrix.

6.9.1 Detailed Description

Functions to manipulate matrices.

6.9.2 Function Documentation

6.9.2.1 expMat()

```
MatrixXd matrix_manipulation::expMat (
            const MatrixXd & A )
```

Compute the exponential of a matrix.

Parameters

A	Matrix whose exponential must be computed

Returns

Expontial of A

6.9.2.2 logMat()

```
MatrixXd matrix_manipulation::logMat (
            const MatrixXd & A )
```

Compute the logarithm of a matrix.

Parameters

A	Matrix whose logarithm must be computed

Returns

Logarithm of A

6.9.2.3 sqrtMat()

```
MatrixXd matrix_manipulation::sqrtMat (
            const MatrixXd & A )
```

Compute the square root of a matrix.

Parameters

A	Matrix whose square root must be computed

Returns

Square root of A

6.9.2.4 bigMatrix2VecMatrices()

```
std::vector< MatrixXd > matrix_manipulation::bigMatrix2VecMatrices (
            const MatrixXd & bigMatrix,
            unsigned int p,
            const std::string & distance_Manifold_name )
```

Transform a $\left(\frac{p*(p+1)}{2}, N\right)$ matrix in a vector of $Np * p$ symmetric matrices.

Parameters

bigMatrix	Matrix whose rows store the values of the upper triangular parts of $Np * p$ symmetric matrices
p	Dimension of the matrices
distance_Manifold_name	Name of the metric on the manifold to use. If `distance_Manifold_name=="Correlation"` then only the upper triangular parts of the matrices is filled

Returns

Vector of $Np * p$ symmetricmatrices corresponding to the transformation of bigMatrix

6.9.2.5 VecMatrices2bigMatrix()

```
MatrixXd matrix_manipulation::VecMatrices2bigMatrix (
            const std::vector< MatrixXd > & vecMatrices )
```

Transform a vector of $Np * p$ symmetric matrices in a $\left(\frac{p*(p+1)}{2}, N\right)$ matrix.

Parameters

vecMatrices	Vector of symmetric matrices whose upper trinagular parts will be stored in the rows of the output matrix

Returns

Matrix $\left(\frac{p*(p+1)}{2}, N\right)$ correponsing to the transformation of vecMatrices

6.9.2.6 Chol_semidef()

```
MatrixXd matrix_manipulation::Chol_semidef (
            const MatrixXd & M1 )
```

Compute the Cholesky decomposition of a matrix not positive definite.

Parameters

M1	Positive semidefinite matrix whose Cholesky decomposition must be computed

Returns

Matrix in $Chol\,(p)$, representing the upper trianglar matrix of A's Cholesky decomposition

6.9.2.7 Chol_decomposition()

```
MatrixXd matrix_manipulation::Chol_decomposition (
            const MatrixXd M1 )
```

Compute the Cholesky decomposition of a matrix.

Parameters

M1	Positive definite or semidefine matrix whose Cholesky decomposition must be computed

Returns

Matrix in $Chol\,(p)$, representing the upper trianglar matrix of A's Cholesky decomposition

6.10 HelpersFactory.hpp File Reference

Typedefs for the factory.

```
#include "FittedVariogram.hpp"
#include "DesignMatrix.hpp"
#include "Distance.hpp"
#include "MapFunctions.hpp"
#include "DistanceManifold.hpp"
#include "DistanceTplane.hpp"
#include "Factory.hpp"
#include "Proxy.hpp"
```

Typedefs

- typedef generic_factory::Factory< variogram_evaluation::FittedVariogram, std::string > vario_factory::VariogramFactory

  Factory for the fitted variogram.
- template<typename ConcreteProduct >
  using vario_factory::VariogramProxy = generic_factory::Proxy< VariogramFactory, ConcreteProduct >

  Proxy for the fitted variogram.
- typedef generic_factory::Factory< design_matrix::DesignMatrix, std::string > design_factory::DesignFactory

  Factory for the design matrix.
- template<typename ConcreteProduct >
  using design_factory::DesignProxy = generic_factory::Proxy< DesignFactory, ConcreteProduct >

  Proxy for the design matrix.
- typedef generic_factory::Factory< distances::Distance, std::string > distance_factory::DistanceFactory

  Factory for the distance.
- template<typename ConcreteProduct >
  using distance_factory::DistanceProxy = generic_factory::Proxy< DistanceFactory, ConcreteProduct >

  Proxy for the distance.
- typedef generic_factory::Factory< map_functions::logarithmicMap, std::string > map_factory::LogMapFactory

  Factory for the logarithmic map.
- typedef generic_factory::Factory< map_functions::exponentialMap, std::string > map_factory::ExpMapFactory

  Factory for the exponential map.
- template<typename ConcreteProduct >
  using map_factory::LogMapProxy = generic_factory::Proxy< LogMapFactory, ConcreteProduct >

  Proxy for the logarithmic map.
- template<typename ConcreteProduct >
  using map_factory::ExpMapProxy = generic_factory::Proxy< ExpMapFactory, ConcreteProduct >

  Proxy for the exponential map.
- typedef generic_factory::Factory< distances_manifold::DistanceManifold, std::string > manifold_factory::ManifoldFactory

Factory for the distance on the manifold.

- template<typename ConcreteProduct >
  using manifold_factory::ManifoldProxy = generic_factory::Proxy< ManifoldFactory, ConcreteProduct >

  Proxy for the distance on the manifold.
- typedef generic_factory::Factory< distances_tplane::DistanceTplane, std::string > tplane_factory::TplaneFactory

  Factory for the distance on the tangent space.
- template<typename ConcreteProduct >
  using tplane_factory::TplaneProxy = generic_factory::Proxy< TplaneFactory, ConcreteProduct >

  Proxy for the distance on the tangent space.

6.10.1 Detailed Description

Typedefs for the factory.

6.11 interface_function.cpp File Reference

Main functions to create the model and perform kriging, along with functions to compute the distance on the manifold and the intrinsic mean.

```
#include <iostream>
#include <vector>
#include <utility>
#include <memory>
#include <Rcpp.h>
#include <RcppEigen.h>
#include "Coordinates.hpp"
#include "DesignMatrix.hpp"
#include "DistanceManifold.hpp"
#include "DistanceTplane.hpp"
#include "Distance.hpp"
#include "EmpiricalVariogram.hpp"
#include "FittedVariogram.hpp"
#include "Helpers.hpp"
#include "HelpersFactory.hpp"
#include "MapFunctions.hpp"
#include "Model.hpp"
#include "IntrinsicMean.hpp"
```

Functions

- RcppExport SEXP get_model (SEXP s_data_manifold, SEXP s_coordinates, SEXP s_X, SEXP s_Sigma, SEXP s_distance, SEXP s_manifold_metric, SEXP s_ts_metric, SEXP s_ts_model, SEXP s_vario_model, SEXP s_n_h, SEXP s_max_it, SEXP s_tolerance, SEXP s_max_sill, SEXP s_max_a, SEXP s_weight_vario, SEXP s_distance_matrix_tot, SEXP s_data_manifold_tot, SEXP s_coordinates_tot, SEXP s_X_tot, SEXP s_hmax, SEXP s_indexes_model, SEXP s_weight_intrinsic, SEXP s_tolerance_intrinsic, SEXP s_weight_↩ extrinsic, SEXP s_suppressMes, SEXP s_tolerance_map_cor)

  Given the coordinates and corresponding manifold values, this function creates a GLS model on the tangent space.
- RcppExport SEXP get_kriging (SEXP s_coordinates, SEXP s_new_coordinates, SEXP s_Sigma, SEXP s↩ _distance, SEXP s_manifold_metric, SEXP s_ts_model, SEXP s_vario_model, SEXP s_beta, SEXP s_↩ gamma_matrix, SEXP s_vario_parameters, SEXP s_residuals, SEXP s_X_new, SEXP s_tolerance_map_↩ cor)

Given the GLS model kriging prediction on new location is performed.

- RcppExport SEXP get_model_and_kriging (SEXP s_data_manifold, SEXP s_coordinates, SEXP s_X, SEXP s_Sigma, SEXP s_distance, SEXP s_manifold_metric, SEXP s_ts_metric, SEXP s_ts_model, SEXP s_vario_model, SEXP s_n_h, SEXP s_max_it, SEXP s_tolerance, SEXP s_max_sill, SEXP s_max_a, SEXP s_weight_vario, SEXP s_distance_matrix_tot, SEXP s_data_manifold_tot, SEXP s_coordinates_tot, SEXP s_X_tot, SEXP s_hmax, SEXP s_indexes_model, SEXP s_weight_intrinsic, SEXP s_tolerance_intrinsic, SEXP s_weight_extrinsic, SEXP s_new_coordinates, SEXP s_X_new, SEXP s_suppressMes, SEXP s_tolerance_map_cor)

  Given the coordinates and corresponding manifold values, this function firstly creates a GLS model on the tangent space, and then it performs kriging on the new locations.
- RcppExport SEXP intrinsic_mean (SEXP s_data, SEXP s_N, SEXP s_manifold_metric, SEXP s_ts_metric, SEXP s_tolerance, SEXP s_weight_intrinsic, SEXP s_weight_extrinsic, SEXP s_tolerance_map_cor)

  Evaluate the intrinsic mean of a given set of symmetric positive definite matrices.
- RcppExport SEXP distance_manifold (SEXP s_data1, SEXP s_data2, SEXP s_N1, SEXP s_N2, SEXP s_manifold_metric)

  Compute the manifold distance between symmetric positive definite matrices.

6.11.1 Detailed Description

Main functions to create the model and perform kriging, along with functions to compute the distance on the manifold and the intrinsic mean.

6.11.2 Function Documentation

6.11.2.1 get_model()

```
RcppExport SEXP get_model (
             SEXP s_data_manifold,
             SEXP s_coordinates,
             SEXP s_X,
             SEXP s_Sigma,
             SEXP s_distance,
             SEXP s_manifold_metric,
             SEXP s_ts_metric,
             SEXP s_ts_model,
             SEXP s_vario_model,
             SEXP s_n_h,
             SEXP s_max_it,
             SEXP s_tolerance,
             SEXP s_max_sill,
             SEXP s_max_a,
             SEXP s_weight_vario,
             SEXP s_distance_matrix_tot,
             SEXP s_data_manifold_tot,
             SEXP s_coordinates_tot,
             SEXP s_X_tot,
             SEXP s_hmax,
             SEXP s_indexes_model,
             SEXP s_weight_intrinsic,
             SEXP s_tolerance_intrinsic,
```

```
SEXP s_weight_extrinsic,
SEXP s_suppressMes,
SEXP s_tolerance_map_cor )
```

Given the coordinates and corresponding manifold values, this function creates a GLS model on the tangent space.

The manifold values are mapped on the tangent space and then a GLS model is fitted to them. A first estimate of the `beta` coefficients is obtained assuming spatially uncorrelated errors. Then, in the main the loop, new estimates of the `beta` are obtained as a result of a weighted least square problem where the weight matrix is the inverse of `gamma_matrix`. The residuals `(residuals = data_ts - fitted)` are updated accordingly. The parameters of the variogram fitted to the `residuals` (and used in the evaluation of the `gamma_matrix`) are computed using Gauss-Newton with backtrack method to solve the associated non-linear least square problem. The stopping criteria is based on the absolute value of the variogram residuals' norm if `ker.width.vario=0`, while it is based on its increment otherwise.

Note

Reference: "Kriging prediction for manifold-valued random fields."
Authors: D. Pigoli, A. Menafoglio & P. Secchi (2016)
Periodical: Journal of Multivariate Analysis, 145, 117-131.

Parameters

s_data_manifold	list of N symmetric positive definite matrices of dimension $(p * p)$
s_coordinates	$(N * 2)$ or $(N * 3)$ matrix of [lat,long], [x,y] or [x,y,z] coordinates. [lat,long] are supposed to be provided in signed decimal degrees
s_X	matrix Matrix with N rows and unrestricted number of columns of additional covariates for the tangent space model, possibly `NULL`
s_Sigma	Matrix $(p * p)$ representing the tangent point. If `NULL` the tangent point is computed as the intrinsic mean of `s_data_manifold`
s_distance	Type of distance between coordinates. It must be either "Eucldist" or "Geodist"
s_manifold_metric	Metric used on the manifold. It must be chosen among "Frobenius", "LogEuclidean", "SquareRoot", "Correlation"
s_ts_metric	Metric used on the tangent space. It must be chosen among "Frobenius", "FrobeniusScaled", "Correlation"
s_ts_model	Type of model fitted on the tangent space. It must be chosen among "Intercept", "Coord1", "Coord2", "Additive"
s_vario_model	Type of variogram fitted. It must be chosen among "Gaussian", "Spherical", "Exponential"
s_n_h	Number of bins in the emprical variogram
s_max_it	Max number of iterations for the main loop
s_tolerance	Tolerance for the main loop
s_max_sill	Maximum value allowed for *sill* in the fitted variogram. If `NULL` it is defined as $1.15 * \max($emp_vario_values$)$
s_max_a	Maximum value for *a* in the fitted variogram. If `NULL` it is defined as $1.15 *$ h_max
s_weight_vario	Vector of length N_tot to weight the locations in the computation of the empirical variogram
s_distance_matrix_tot	Matrix $(N\_tot * N\_tot)$ of distances between the locations,
s_data_manifold_tot	List of N_tot symmetric positive definite matrices of dimension $(p * p)$
s_coordinates_tot	$(N\_tot * 2)$ or $(N\_tot * 3)$ matrix of [lat,long], [x,y] or [x,y,z] coordinates. [lat,long] are supposed to be provided in signed decimal degrees),
s_X_tot	Matrix with N_tot rows and unrestricted number of columns, of additional covariates for the tangent space model. Possibly `NULL`

Parameters

s_hmax	Maximum value of distance for which the variogram is computed
s_indexes_model	Indexes corresponding to `coords` in `coords_tot`. Required only in the case `metric_manifold=="Correlation"`
s_weight_intrinsic	Vector of length N to weight the locations in the computation of the intrinsic mean. If `NULL` a vector of ones is used. Not needed if `Sigma` is provided
s_tolerance_intrinsic	Tolerance for the computation of the intrinsic mean. Not needed if `Sigma` is provided
s_weight_extrinsic	Vector of length N to weight the locations in the computation of the extrinsic mean. If `NULL` `weight_intrinsic` are used. Needed only if `Sigma` is not provided and `metric_manifold=="Correlation"`
s_suppressMes	Boolean. If `TRUE` warning messagges are not printed
s_tolerance_map_cor	Tolerance to use in the maps. Required only if `metric_manifold=="Correlation"`

Returns

A list with the following fields:

- `beta` Vector of the beta matrices of the fitted model
- `fit_vario_values` Vector of fitted variogram values in correspondence of `hh`
- `hh` Dense vector of positions at which `fit_vario_values` is computed
- `gamma_matrix` Covariogram matrix $(N * N)$
- `residuals` Vector of the N residual matrices
- `emp_vario_values` Vector of empircal variogram values in correspondence of `h_vec`
- `h_vec` Vector of positions at which the empirical variogram is computed
- `fitted_par_vario` Estimates of *nugget*, *sill-nugget* and *practical range*
- `iterations` Number of iterations of the main loop
- `Sigma` Tangent point

6.11.2.2 get_kriging()

```
RcppExport SEXP get_kriging (
             SEXP s_coordinates,
             SEXP s_new_coordinates,
             SEXP s_Sigma,
             SEXP s_distance,
             SEXP s_manifold_metric,
             SEXP s_ts_model,
             SEXP s_vario_model,
             SEXP s_beta,
             SEXP s_gamma_matrix,
             SEXP s_vario_parameters,
             SEXP s_residuals,
             SEXP s_X_new,
             SEXP s_tolerance_map_cor )
```

Given the GLS model kriging prediction on new location is performed.

The model provided is used to perform simple kriging on the tangent space in correspondence of the new locations. The estimates are then mapped to the manifold to produce the actual prediction.

Note

Reference: "Kriging prediction for manifold-valued random fields."
Authors: D. Pigoli, A. Menafoglio & P. Secchi (2016)
Periodical: Journal of Multivariate Analysis, 145, 117-131.

Parameters

s_coordinates	$(N * 2)$ or $(N * 3)$ matrix of [lat,long], [x,y] or [x,y,z] coordinates. [lat,long] are supposed to be provided in signed decimal degrees
s_new_coordinates	$(N * 2)$ or $(N * 3)$ matrix of [lat,long], [x,y] or [x,y,z] coordinates. [lat,long] are supposed to be provided in signed decimal degrees
s_Sigma	Matrix $(p * p)$ representing the tangent point. If `NULL` the tangent point is computed as the intrinsic mean of `s_data_manifold`
s_distance	Type of distance between coordinates. It must be either "Eucldist" or "Geodist"
s_manifold_metric	Metric used on the manifold. It must be chosen among "Frobenius", "LogEuclidean", "SquareRoot", "Correlation"
s_ts_model	Type of model fitted on the tangent space. It must be chosen among "Intercept", "Coord1", "Coord2", "Additive"
s_vario_model	Type of variogram fitted. It must be chosen among "Gaussian", "Spherical", "Exponential"
s_beta	Vector of the beta matrices of the fitted model
s_gamma_matrix	Covariogram matrix $(N * N)$
s_vario_parameters	Estimates of *nugget*, *sill-nugget* and *practical range*
s_residuals	Vector of the N residual matrices
s_X_new	Matrix (with the same number of rows of `s_new_coordinates`) of additional covariates for the new locations, possibly `NULL`
s_tolerance_map_cor	Tolerance to use in the maps. Required only if `metric_manifold=="Correlation"`

Returns

A list with the following field:

- `prediction` Vector of matrices predicted at the new locations

6.11.2.3 get_model_and_kriging()

```
RcppExport SEXP get_model_and_kriging (
             SEXP s_data_manifold,
             SEXP s_coordinates,
             SEXP s_X,
             SEXP s_Sigma,
             SEXP s_distance,
             SEXP s_manifold_metric,
             SEXP s_ts_metric,
             SEXP s_ts_model,
             SEXP s_vario_model,
             SEXP s_n_h,
             SEXP s_max_it,
             SEXP s_tolerance,
```

```
SEXP s_max_sill,
SEXP s_max_a,
SEXP s_weight_vario,
SEXP s_distance_matrix_tot,
SEXP s_data_manifold_tot,
SEXP s_coordinates_tot,
SEXP s_X_tot,
SEXP s_hmax,
SEXP s_indexes_model,
SEXP s_weight_intrinsic,
SEXP s_tolerance_intrinsic,
SEXP s_weight_extrinsic,
SEXP s_new_coordinates,
SEXP s_X_new,
SEXP s_suppressMes,
SEXP s_tolerance_map_cor )
```

Given the coordinates and corresponding manifold values, this function firstly creates a GLS model on the tangent space, and then it performs kriging on the new locations.

The manifold values are mapped on the tangent space and then a GLS model is fitted to them. A first estimate of the `beta` coefficients is obtained assuming spatially uncorrelated errors. Then, in the main the loop, new estimates of the `beta` are obtained as a result of a weighted least square problem where the weight matrix is the inverse of `gamma_matrix`. The residuals `(residuals = data_ts - fitted)` are updated accordingly. The parameters of the variogram fitted to the `residuals` (and used in the evaluation of the `gamma_matrix`) are computed using Gauss-Newton with backtrack method to solve the associated non-linear least square problem. The stopping criteria is based on the absolute value of the variogram residuals' norm if `ker.width.vario=0`, while it is based on its increment otherwise. Once the model is computed, simple kriging on the tangent space is performed in correspondence of the new locations and eventually the estimates are mapped to the manifold.

Note

Reference: "Kriging prediction for manifold-valued random fields."
Authors: D. Pigoli, A. Menafoglio & P. Secchi (2016)
Periodical: Journal of Multivariate Analysis, 145, 117-131.

Parameters

s_data_manifold	list of N symmetric positive definite matrices of dimension $(p * p)$
s_coordinates	$(N * 2)$ or $(N * 3)$ matrix of [lat,long], [x,y] or [x,y,z] coordinates. [lat,long] are supposed to be provided in signed decimal degrees
s_X	matrix Matrix with N rows and unrestricted number of columns of additional covariates for the tangent space model, possibly `NULL`
s_Sigma	Matrix $(p * p)$ representing the tangent point. If `NULL` the tangent point is computed as the intrinsic mean of `s_data_manifold`
s_distance	Type of distance between coordinates. It must be either "Eucldist" or "Geodist"
s_manifold_metric	Metric used on the manifold. It must be chosen among "Frobenius", "LogEuclidean", "SquareRoot", "Correlation"
s_ts_metric	Metric used on the tangent space. It must be chosen among "Frobenius", "FrobeniusScaled", "Correlation"
s_ts_model	Type of model fitted on the tangent space. It must be chosen among "Intercept", "Coord1", "Coord2", "Additive"
s_vario_model	Type of variogram fitted. It must be chosen among "Gaussian", "Spherical", "Exponential"
s_n_h	Number of bins in the emprical variogram
s_max_it	Max number of iterations for the main loop

Parameters

s_tolerance	Tolerance for the main loop
s_max_sill	Maximum value allowed for *sill* in the fitted variogram. If `NULL` it is defined as $1.15 * \max(\text{emp\_vario\_values})$
s_max_a	Maximum value for *a* in the fitted variogram. If `NULL` it is defined as $1.15 *$ h_max
s_weight_vario	Vector of length N_tot to weight the locations in the computation of the empirical variogram
s_distance_matrix_tot	Matrix $(N\_tot * N\_tot)$ of distances between the locations,
s_data_manifold_tot	List of N_tot symmetric positive definite matrices of dimension $(p * p)$
s_coordinates_tot	$(N\_tot * 2)$ or $(N\_tot * 3)$ matrix of [lat,long], [x,y] or [x,y,z] coordinates. [lat,long] are supposed to be provided in signed decimal degrees),
s_X_tot	Matrix with N_tot rows and unrestricted number of columns, of additional covariates for the tangent space model. Possibly `NULL`
s_hmax	Maximum value of distance for which the variogram is computed
s_indexes_model	Indexes corresponding to `coords` in `coords_tot`. Required only in the case `metric_manifold=="Correlation"`
s_weight_intrinsic	Vector of length N to weight the locations in the computation of the intrinsic mean. If `NULL` a vector of ones is used. Not needed if `Sigma` is provided
s_tolerance_intrinsic	Tolerance for the computation of the intrinsic mean. Not needed if `Sigma` is provided
s_weight_extrinsic	Vector of length N to weight the locations in the computation of the extrinsic mean. If `NULL` `weight_intrinsic` are used. Needed only if `Sigma` is not provided and `metric_manifold=="Correlation"`
s_new_coordinates	$(N * 2)$ or $(N * 3)$ matrix of [lat,long], [x,y] or [x,y,z] coordinates. [lat,long] are supposed to be provided in signed decimal degrees
s_X_new	Matrix (with the same number of rows of `s_new_coordinates`) of additional covariates for the new locations, possibly `NULL`
s_suppressMes	Boolean. If `TRUE` warning messagges are not printed
s_tolerance_map_cor	Tolerance to use in the maps. Required only if `metric_manifold=="Correlation"`

Returns

A list with the following fields:

- `beta` Vector of the beta matrices of the fitted model
- `fit_vario_values` Vector of fitted variogram values in correspondence of `hh`
- `hh` Dense vector of positions at which `fit_vario_values` is computed
- `gamma_matrix` Covariogram matrix $(N * N)$
- `residuals` Vector of the N residual matrices
- `emp_vario_values` Vector of empircal variogram values in correspondence of `h_vec`
- `h_vec` Vector of positions at which the empirical variogram is computed
- `fitted_par_vario` Estimates of *nugget*, *sill-nugget* and *practical range*
- `iterations` Number of iterations of the main loop
- `Sigma` Tangent point
- `prediction` Vector of matrices predicted at the new locations

6.11.2.4 intrinsic_mean()

```
RcppExport SEXP intrinsic_mean (
             SEXP s_data,
             SEXP s_N,
             SEXP s_manifold_metric,
             SEXP s_ts_metric,
             SEXP s_tolerance,
             SEXP s_weight_intrinsic,
             SEXP s_weight_extrinsic,
             SEXP s_tolerance_map_cor )
```

Evaluate the intrinsic mean of a given set of symmetric positive definite matrices.

Parameters

s_data	list of N symmetric positive definite matrices of dimension $(p * p)$
s_N	Number of data. `N = s_data.size()`
s_manifold_metric	Metric used on the manifold. It must be chosen among "Frobenius", "LogEuclidean", "SquareRoot", "Correlation"
s_ts_metric	Metric used on the tangent space. It must be chosen among "Frobenius", "FrobeniusScaled", "Correlation"
s_tolerance	Tolerance for the computation of the intrinsic_mean
s_weight_intrinsic	Vector of length N to weight the locations in the computation of the intrinsic mean. If `NULL` a vector of ones is used
s_weight_extrinsic	Vector of length N to weight the locations in the computation of the extrinsic mean. If `NULL` `weight_intrinsic` are used
s_tolerance_map_cor	Tolerance to use in the maps. Required only if `metric_manifold=="Correlation"`

Returns

A matrix representing the intrinsic mean of the `s_data`

6.11.2.5 distance_manifold()

```
RcppExport SEXP distance_manifold (
             SEXP s_data1,
             SEXP s_data2,
             SEXP s_N1,
             SEXP s_N2,
             SEXP s_manifold_metric )
```

Compute the manifold distance between symmetric positive definite matrices.

If $N1 == N2$ then the result is a vector of length $N1 = N2$ containing in position `i` the manifold distance beetween `data1[[i]]` and `data2[[i]]`. Instead if $N2 = 1$ and $N1! = 1$ the result is a vector of length $B1$ containing in position `i` the manifold distance between `data1[[i]]` and `data2[[1]]`

Parameters

s_data1	list of $N1$ symmetric positive definite matrices of dimension $(p * p)$
s_data2	list of $N2$ symmetric positive definite matrices of dimension $(p * p)$
s_N1	Number of data1. `N1 = s_data1.size()`
s_N2	Number of data2. `N2 = s_data2.size()`
s_manifold_metric	Metric used on the manifold. It must be chosen among "Frobenius", "LogEuclidean", "SquareRoot", "Correlation"

Returns

A double or a vector of distances

6.12 IntrinsicMean.hpp File Reference

Functions to compute intrinsic and extrinsic mean for manifold data.

```
#include <Eigen/Dense>
#include <memory>
#include "DistanceTplane.hpp"
#include "Helpers.hpp"
#include "MapFunctions.hpp"
```

Functions

- MatrixXd intrinsic_mean_C (const std::vector< MatrixXd > &data_manifold, std::string distance_↩ Manifold_name, map_functions::logarithmicMap &logMap, map_functions::exponentialMap &expMap, distances_tplane::DistanceTplane &distanceTplane, double tolerance, const Vec &weight_intrinsic, const Vec &weight_extrinsic)

  Compute the intrinsic mean of a set of manifold matrices.
- MatrixXd extrinsic_mean (const std::vector< MatrixXd > &data_manifold, const Vec &weight_extrinsic)

  Compute the extrinsic mean of a set of matrices in $Chol\,(p)$.

6.12.1 Detailed Description

Functions to compute intrinsic and extrinsic mean for manifold data.

6.12.2 Function Documentation

6.12.2.1 intrinsic_mean_C()

```
MatrixXd intrinsic_mean_C (
            const std::vector< MatrixXd > & data_manifold,
            std::string distance_Manifold_name,
            map_functions::logarithmicMap & logMap,
            map_functions::exponentialMap & expMap,
            distances_tplane::DistanceTplane & distanceTplane,
            double tolerance,
            const Vec & weight_intrinsic,
            const Vec & weight_extrinsic )
```

Compute the intrinsic mean of a set of manifold matrices.

Parameters

data_manifold	Vector of manifold matrices
distance_Manifold_name	Name of the metric on the manifold
logMap	Logarithmic map
expMap	Exponential map
distanceTplane	Distance on the tangent space
tolerance	Tolerance for the algorithm's loop
weight_intrinsic	Weights
weight_extrinsic	Weights for the computation of the extrinsic mean (used if `distance_Manifold_name=="Correlation"`)

Returns

Matrix identifying the intrinsic mean of `data_manifold`

6.12.2.2 extrinsic_mean()

```
MatrixXd extrinsic_mean (
            const std::vector< MatrixXd > & data_manifold,
            const Vec & weight_extrinsic )
```

Compute the extrinsic mean of a set of matrices in $Chol\,(p)$.

Parameters

data_manifold	Vector of matrices in $Chol\,(p)$
weight_extrinsic	Weights

Returns

Matrix identifying the extrinsic mean of `data_manifold`

6.13 MapFunctions.hpp File Reference

Classes to compute the exponential and logarithmic map according to the manifold metric.

```
#include <memory>
#include "DistanceManifold.hpp"
```

Classes

- class map_functions::logarithmicMap

  Abstract class for the computation of the logarithmic map.

- class map_functions::logMapFrob

  Class for the computation of the logarithmic map when `manifold_metric=="Frobenius"`
- class map_functions::logMapLogEucl

  Class for the computation of the logarithmic map when `manifold_metric=="LogEuclidean"`
- class map_functions::logMapSqRoot

  Class for the computation of the logarithmic map when `manifold_metric=="SqRoot"`
- class map_functions::logMapChol

  Class for the computation of the logarithmic map when `manifold_metric=="Correlation"`
- class map_functions::exponentialMap

  Abstract class for the computation of the exponential map.
- class map_functions::expMapFrob

  Class for the computation of the exponential map when `manifold_metric=="Frobenius"`
- class map_functions::expMapLogEucl

  Class for the computation of the exponential map when `manifold_metric=="LogEuclidean"`
- class map_functions::expMapSqRoot

  Class for the computation of the exponential map when `manifold_metric=="SqRoot"`
- class map_functions::expMapChol

  Class for the computation of the exponential map when `manifold_metric=="Correlation"`

6.13.1 Detailed Description

Classes to compute the exponential and logarithmic map according to the manifold metric.

6.14 Model.hpp File Reference

Model class.

```
#include <vector>
#include <iostream>
#include <string>
#include <utility>
#include <Eigen/IterativeLinearSolvers>
#include <memory>
#include "Helpers.hpp"
```

Classes

- class model_fit::Model

  Class to compute and store the linear model on the tangent space.

6.14.1 Detailed Description

Model class.

6.15 Proxy.hpp File Reference

Proxy class.

```
#include <string>
#include <memory>
#include <iostream>
#include <type_traits>
```

Classes

- class generic_factory::Proxy< Factory, ConcreteProduct >

6.15.1 Detailed Description

Proxy class.