StreamNetgstat

1.0.0

Generated by Doxygen 1.8.15

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

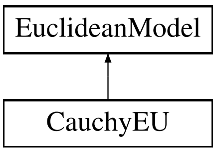
Chapter 4

Class Documentation

4.1 CauchyEU Class Reference

```
#include <EuclideanModel.hpp>
```

Inheritance diagram for CauchyEU:



Public Member Functions

- ∼CauchyEU ()=default
- double computeCov (double d) override

Additional Inherited Members

4.1.1 Detailed Description

Child class for the Euclidean covariance model, implementing the Cauchy covariance function.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 ∼CauchyEU()

```
CauchyEU::∼CauchyEU ( )  [default]
```

Default destructor.

4.1.3 Member Function Documentation

4.1.3.1 computeCov()

```
double CauchyEU::computeCov (
            double d )  [override], [virtual]
```

Compute the covariance between two points

Parameters

d	Euclidean distance between the points

Implements EuclideanModel.

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

- EuclideanModel.hpp
- EuclideanModel.cpp

4.2 Dataframe Class Reference

Public Member Functions

- Dataframe ()=default
- Dataframe (const std::vector< std::string > &colnames, const Eigen::MatrixXd &data)
- unsigned int rows () const
- unsigned int cols () const
- const Eigen::VectorXd & operator[] (const std::string &key) const
- Eigen::MatrixXd computeWeightMat (const std::string &weightVar)
- Eigen::MatrixXd computeWeightMat (const std::string &weightVar, const Eigen::VectorXd &otherPoints)
- void print () const

4.2.1 Constructor & Destructor Documentation

4.2.1.1 Dataframe() [1/2]

```
Dataframe::Dataframe ( )  [default]
```

Default constructor.

4.2.1.2 Dataframe() [2/2]

```
Dataframe::Dataframe (
            const std::vector< std::string > & colnames,
            const Eigen::MatrixXd & data )
```

Constructor.

Parameters

colnames	vector containing the names of the variables
data	matrix, whose columns correspond to the values of the statistical units per each variable

4.2.2 Member Function Documentation

4.2.2.1 cols()

```
unsigned int Dataframe::cols ( ) const
```

Returns

The data frame number of columns

4.2.2.2 computeWeightMat() [1/2]

```
Eigen::MatrixXd Dataframe::computeWeightMat (
            const std::string & weightVar )
```

Compute the weight matrix for the tail-up model corresponding to a single group of points

Parameters

weightVar	constant string indicating the name of the variable to be used to compute the weights

Returns

A matrix whose elements are the weights for the tail-up model

4.2.2.3 computeWeightMat() [2/2]

```
Eigen::MatrixXd Dataframe::computeWeightMat (
            const std::string & weightVar,
            const Eigen::VectorXd & otherPoints )
```

Compute the weight matrix for the tail-up model corresponding to two groups of points

Parameters

weightVar	constant string indicating the name of the variable to be used to compute the weights
otherPoints	vector representing the column of the weight variable of another data frame, for instance the one of the prediction points

Returns

A matrix whose elements are the weights for the tail-up model, where rows correspond to the points of the class (in general, the observed points) and columns correspond to the other group of points (in general, the prediction points)

4.2.2.4 operator[]()

```
const Eigen::VectorXd & Dataframe::operator[] (
            const std::string & key ) const
```

Operator to extract a column of the data frame

Parameters

key	string indicating the name of the variable whose column is to be extracted

Returns

The column of the data frame corresponding to the variable selected

4.2.2.5 print()

```
void Dataframe::print ( ) const
```

Printing function to visualize the data frame as a matrix with labeled columns

4.2.2.6 rows()

```
unsigned int Dataframe::rows ( ) const
```

Returns

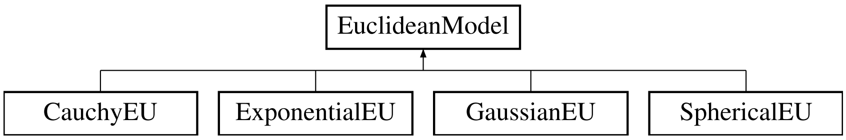
The data frame number of rows

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

- Dataframe.hpp
- Dataframe.cpp

4.3 EuclideanModel Class Reference

Inheritance diagram for EuclideanModel:



Public Member Functions

- EuclideanModel ()=default
- virtual ∼EuclideanModel ()=default
- EuclideanModel (double s, double a)
- virtual double computeCov (double d)=0
- Eigen::MatrixXd computeMatCov (const Eigen::MatrixXd &distGeo)
- void setSigma2 (double s)
- void setAlpha (double a)
- double getSigma2 () const
- double getAlpha () const

Protected Attributes

- double sigma2

    parsill of the Euclidean model
- double alpha

    range of the Euclidean model

4.3.1 Constructor & Destructor Documentation

4.3.1.1 EuclideanModel() [1/2]

`EuclideanModel::EuclideanModel ( ) [default]`

Default constructor.

4.3.1.2 ∼EuclideanModel()

`virtual EuclideanModel::∼EuclideanModel ( ) [virtual], [default]`

Default destructor.

4.3.1.3 EuclideanModel() [2/2]

```
EuclideanModel::EuclideanModel (
              double s,
              double a ) [inline]
```

Constructor.

Parameters

s	initial value of the parsill
a	initial value of the range

4.3.2 Member Function Documentation

4.3.2.1 computeCov()

```
virtual double EuclideanModel::computeCov (
             double d ) [pure virtual]
```

A pure virtual member function to compute the covariance between two points

Parameters

d	Euclidean distance between the points

Implemented in GaussianEU, ExponentialEU, SphericalEU, and CauchyEU.

4.3.2.2 computeMatCov()

```
Eigen::MatrixXd EuclideanModel::computeMatCov (
             const Eigen::MatrixXd & distGeo )
```

Computes the covariance matrix from a distance matrix

Parameters

distGeo	distance matrix

Returns

The covariance matrix

4.3.2.3 getAlpha()

```
double EuclideanModel::getAlpha ( ) const [inline]
```

Returns

the range

4.3.2.4 getSigma2()

```
double EuclideanModel::getSigma2 ( ) const  [inline]
```

Returns

the parsill

4.3.2.5 setAlpha()

```
void EuclideanModel::setAlpha (
             double a )  [inline]
```

Parameters

a	new value of the range

4.3.2.6 setSigma2()

```
void EuclideanModel::setSigma2 (
             double s )  [inline]
```

Parameters

s	new value of the parsill

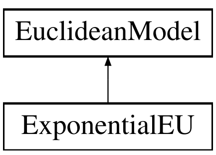
The documentation for this class was generated from the following files:

- EuclideanModel.hpp
- EuclideanModel.cpp

4.4 ExponentialEU Class Reference

```
#include <EuclideanModel.hpp>
```

Inheritance diagram for ExponentialEU:



Public Member Functions

- ∼ExponentialEU ()=default
- double computeCov (double d) override

Additional Inherited Members

4.4.1 Detailed Description

Child class for the Euclidean covariance model, implementing the exponential covariance function.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 ∼ExponentialEU()

```
ExponentialEU::∼ExponentialEU ( )  [default]
```

Default destructor.

4.4.3 Member Function Documentation

4.4.3.1 computeCov()

```
double ExponentialEU::computeCov (
            double d )  [override], [virtual]
```

Compute the covariance between two points

Parameters

d	Euclidean distance between the points

Implements EuclideanModel.

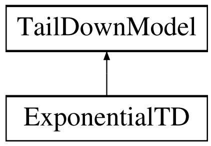
The documentation for this class was generated from the following files:

- EuclideanModel.hpp
- EuclideanModel.cpp

4.5 ExponentialTD Class Reference

```
#include <TailDownModel.hpp>
```

Inheritance diagram for ExponentialTD:



Public Member Functions

- ∼ExponentialTD ()=default
- double computeCov (double, double) override
- double computeCov (double) override

Additional Inherited Members

4.5.1 Detailed Description

Child class for the tail-down covariance model, implementing the exponential covariance function.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 ∼ExponentialTD()

```
ExponentialTD::~ExponentialTD ( )  [default]
```

Default destructor.

4.5.3 Member Function Documentation

4.5.3.1 computeCov() [1/2]

```
double ExponentialTD::computeCov (
            double a,
            double b )  [override], [virtual]
```

Compute the covariance between two flow-unconnected points

Parameters

a	shorter hydrological distance between the points and the common downstream juction
b	greater hydrological distance between the points and the common downstream juction

Implements TailDownModel.

4.5.3.2 computeCov() [2/2]

```
double ExponentialTD::computeCov (
            double h ) [override], [virtual]
```

Compute the covariance between two flow-connected points

Parameters

h	hydrological distance between the points

Implements TailDownModel.

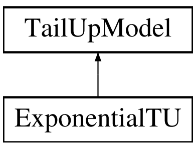
The documentation for this class was generated from the following files:

- TailDownModel.hpp
- TailDownModel.cpp

4.6 ExponentialTU Class Reference

```
#include <TailUpModel.hpp>
```

Inheritance diagram for ExponentialTU:



Public Member Functions

- virtual ∼ExponentialTU ()=default
- double computeCov (double h) override

Additional Inherited Members

4.6.1 Detailed Description

Child class for the tail-up covariance model, implementing the exponential covariance function.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 ∼ExponentialTU()

```
virtual ExponentialTU::∼ExponentialTU ( ) [virtual], [default]
```

Default destructor.

4.6.3 Member Function Documentation

4.6.3.1 computeCov()

```
double ExponentialTU::computeCov (
            double h ) [override], [virtual]
```

Compute the covariance between two flow-connected points

Parameters

h	hydrological distance between the points

Implements TailUpModel.

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

- TailUpModel.hpp
- TailUpModel.cpp

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

```
#include <Factory.hpp>
```

Public Types

- using **AbstractProduct_type** = AbstractProduct
- using **Identifier_type** = Identifier
- using **Builder_type** = Builder

Public Member Functions

- std::unique_ptr< AbstractProduct > **create** (Identifier const &name) const
- void **add** (Identifier const &, Builder_type const &)
- std::vector< Identifier > **registered** () const
- void **unregister** (Identifier const &name)

Static Public Member Functions

- static Factory & **Instance** ()

4.7.1 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(...)
```

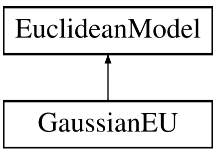
The documentation for this class was generated from the following file:

- Factory.hpp

4.8 GaussianEU Class Reference

```
#include <EuclideanModel.hpp>
```

Inheritance diagram for GaussianEU:



Public Member Functions

- ∼GaussianEU ()=default
- double computeCov (double d) override

Additional Inherited Members

4.8.1 Detailed Description

Child class for the Euclidean covariance model, implementing the Gaussian covariance function.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 ∼GaussianEU()

```
GaussianEU::~GaussianEU ( )  [default]
```

Default destructor.

4.8.3 Member Function Documentation

4.8.3.1 computeCov()

```
double GaussianEU::computeCov (
            double d )  [override], [virtual]
```

Compute the covariance between two points

Parameters

d	Euclidean distance between the points

Implements EuclideanModel.

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

- EuclideanModel.hpp
- EuclideanModel.cpp

4.9 Kriging Class Reference

Public Member Functions

- Kriging ()=default
- Kriging (const Eigen::MatrixXd &dMatPred, const Eigen::MatrixXd &dMatObs, const Eigen::MatrixXd &V, const Eigen::MatrixXd &Nop, const Eigen::MatrixXd &Npo, const Eigen::MatrixXd &D, const Eigen::MatrixXd &wMat, const Eigen::MatrixXi &connMat, const Eigen::VectorXd &param, const Eigen::VectorXd &y, std::unique_ptr< TailUpModel > &tailup_ptr, std::unique_ptr< TailDownModel > &taildown_ptr, std::unique_ptr< EuclideanModel > &euclid_ptr, int nMod, bool useNugg)
- const Eigen::MatrixXd & getPredictions () const
- void predict ()

4.9.1 Constructor & Destructor Documentation

4.9.1.1 Kriging() [1/2]

```
Kriging::Kriging ( ) [default]
```

Default constructor.

4.9.1.2 Kriging() [2/2]

```
Kriging::Kriging (
            const Eigen::MatrixXd & dMatPred,
            const Eigen::MatrixXd & dMatObs,
            const Eigen::MatrixXd & V,
            const Eigen::MatrixXd & Nop,
            const Eigen::MatrixXd & Npo,
            const Eigen::MatrixXd & D,
            const Eigen::MatrixXd & wMat,
            const Eigen::MatrixXi & connMat,
            const Eigen::VectorXd & param,
            const Eigen::VectorXd & y,
            std::unique_ptr< TailUpModel > & tailup_ptr,
            std::unique_ptr< TailDownModel > & taildown_ptr,
            std::unique_ptr< EuclideanModel > & euclid_ptr,
            int nMod,
            bool useNugg )
```

Constructor.

Parameters

dMatPred	design matrix of the prediction points
dMatObs	design matrix of the observed points
V	covariance matrix between observed points
Nop	hydrological distance matrix observed-prediction points
Npo	hydrological distance matrix prediction-observed points
D	Euclidean distance matrix
wMat	weight matrix for tail-up model
connMat	flow-connection/unconnection binary matrix
param	vector with the covariance parameters values
y	vector of the response variable values
tailup_ptr	pointer to the tail-up model selected
taildown_ptr	pointer to the tail-down model selected
euclid_ptr	pointer to the Euclidean model selected
nMod	number of covariance models
useNugg	boolean, indicating if the nuggect effects is to be considered in the mixed model

4.9.2 Member Function Documentation

4.9.2.1 getPredictions()

```
const Eigen::MatrixXd& Kriging::getPredictions ( ) const [inline]
```

Returns

the matrix containing the predicted values and the kriging variance

4.9.2.2 predict()

```
void Kriging::predict ( )
```

Perform Universal kriging

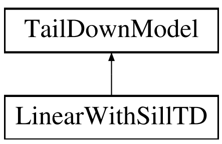
The documentation for this class was generated from the following files:

- Kriging.hpp
- Kriging.cpp

4.10 LinearWithSillTD Class Reference

```
#include <TailDownModel.hpp>
```

Inheritance diagram for LinearWithSillTD:



Public Member Functions

- ∼LinearWithSillTD ()=default
- double computeCov (double a, double b) override
- double computeCov (double) override

Additional Inherited Members

4.10.1 Detailed Description

Child class for the tail-down covariance model, implementing the linear-with-sill covariance function.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 ∼LinearWithSillTD()

```
LinearWithSillTD::~LinearWithSillTD ( ) [default]
```

Default destructor

4.10.3 Member Function Documentation

4.10.3.1 computeCov() [1/2]

```
double LinearWithSillTD::computeCov (
            double a,
            double b ) [override], [virtual]
```

Compute the covariance between two flow-unconnected points

Parameters

a	shorter hydrological distance between the points and the common downstream juction
b	greater hydrological distance between the points and the common downstream juction

Implements TailDownModel.

4.10.3.2 computeCov() [2/2]

```
double LinearWithSillTD::computeCov (
            double h ) [override], [virtual]
```

Compute the covariance between two flow-connected points

Parameters

h	hydrological distance between the points

Implements TailDownModel.

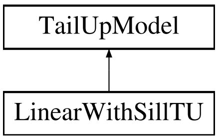
The documentation for this class was generated from the following files:

- TailDownModel.hpp
- TailDownModel.cpp

4.11 LinearWithSillTU Class Reference

```
#include <TailUpModel.hpp>
```

Inheritance diagram for LinearWithSillTU:



Public Member Functions

- virtual ~LinearWithSillTU ()=default
- double computeCov (double h) override

Additional Inherited Members

4.11.1 Detailed Description

Child class for the tail-up covariance model, implementing the linear-with-sill covariance function.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 ~LinearWithSillTU()

```
virtual LinearWithSillTU::~LinearWithSillTU ( )  [virtual], [default]
```

Default destructor.

4.11.3 Member Function Documentation

4.11.3.1 computeCov()

```
double LinearWithSillTU::computeCov (
            double h )  [override], [virtual]
```

Compute the covariance between two flow-connected points

Parameters

h	hydrological distance between the points

Implements TailUpModel.

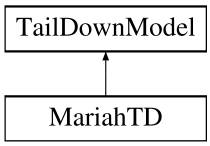
The documentation for this class was generated from the following files:

- TailUpModel.hpp
- TailUpModel.cpp

4.12 MariahTD Class Reference

```
#include <TailDownModel.hpp>
```

Inheritance diagram for MariahTD:



Public Member Functions

- ∼MariahTD ()=default
- double computeCov (double, double) override
- double computeCov (double) override

Additional Inherited Members

4.12.1 Detailed Description

Child class for the tail-down covariance model, implementing the Mariah covariance function.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 ∼MariahTD()

```
MariahTD::∼MariahTD ( )  [default]
```

Default destructor.

4.12.3 Member Function Documentation

4.12.3.1 computeCov() [1/2]

```
double MariahTD::computeCov (
            double a,
            double b )  [override], [virtual]
```

Compute the covariance between two flow-unconnected points

Parameters

a	shorter hydrological distance between the points and the common downstream juction
b	greater hydrological distance between the points and the common downstream juction

Implements TailDownModel.

4.12.3.2 computeCov() [2/2]

```
double MariahTD::computeCov (
             double h )  [override], [virtual]
```

Compute the covariance between two flow-connected points

Parameters

h	hydrological distance between the points

Implements TailDownModel.

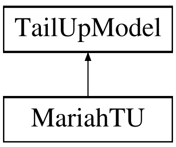
The documentation for this class was generated from the following files:

- TailDownModel.hpp
- TailDownModel.cpp

4.13 MariahTU Class Reference

```
#include <TailUpModel.hpp>
```

Inheritance diagram for MariahTU:



Public Member Functions

- virtual ~MariahTU ()=default
- double computeCov (double h) override

Additional Inherited Members

4.13.1 Detailed Description

Child class for the tail-up covariance model, implementing the Mariah covariance function.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 ∼MariahTU()

```
virtual MariahTU::~MariahTU ( )  [virtual], [default]
```

Default destructor.

4.13.3 Member Function Documentation

4.13.3.1 computeCov()

```
double MariahTU::computeCov (
            double h )  [override], [virtual]
```

Compute the covariance between two flow-connected points

Parameters

h	hydrological distance between the points

Implements TailUpModel.

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

- TailUpModel.hpp
- TailUpModel.cpp

4.14 Network Class Reference

Public Member Functions

- Network ()=default
- Network (const unsigned int id, const std::vector< Point > &obs, const std::vector< Point > &pred, const std::vector< StreamSegment > &seg)
- Network (const unsigned int id, const std::vector< Point > &obs, const std::vector< StreamSegment > &seg)
- void setID (const unsigned int id)
- const unsigned int getID () const
- const unsigned int getNObs () const
- const unsigned int getNPred () const
- const Eigen::MatrixXi & getFlowMatOO () const
- const Eigen::MatrixXi & getFlowMatOP () const

- const Eigen::MatrixXi & getFlowMatPP () const
- const Eigen::MatrixXd & getDistGeoOO () const
- const Eigen::MatrixXd & getDistGeoOP () const
- const Eigen::MatrixXd & getDistGeoPP () const
- const Eigen::MatrixXd & getDistHydroOO () const
- const Eigen::MatrixXd & getDistHydroOP () const
- const Eigen::MatrixXd & getDistHydroPO () const
- const Eigen::MatrixXd & getDistHydroPP () const
- const Points & getObsPoints () const
- const Points & getPredPoints () const
- void computeDistances (bool geo)
- void setDistPoints (bool geo, const std::vector< Eigen::MatrixXd > &matrices)
- void print () const

4.14.1 Constructor & Destructor Documentation

4.14.1.1 Network() [1/3]

```
Network::Network ( )  [default]
```

Default constructor.

4.14.1.2 Network() [2/3]

```
Network::Network (
            const unsigned int id,
            const std::vector< Point > & obs,
            const std::vector< Point > & pred,
            const std::vector< StreamSegment > & seg )
```

Constructor.

Parameters

id	networkID
obs	vector of Point objects, representing the observed points
pred	vector of Point objects, representing the prediction points
seg	vector of StreamSegment objects

4.14.1.3 Network() [3/3]

```
Network::Network (
            const unsigned int id,
```

```
const std::vector< Point > & obs,
const std::vector< StreamSegment > & seg )
```

Constructor.

Parameters

id	networkID
obs	vector of Point objects, representing the observed points
seg	vector of StreamSegment objects

4.14.2 Member Function Documentation

4.14.2.1 computeDistances()

```
void Network::computeDistances (
            bool geo )
```

Compute the distance matrices whose rows represent the observed points and whose columns representing the predicion points

Parameters

geo	boolean, indicating if the Euclidean distances have to be computed

4.14.2.2 getDistGeoOO()

```
const Eigen::MatrixXd& Network::getDistGeoOO ( ) const [inline]
```

Returns

the Euclidean distance matrix between observed points

4.14.2.3 getDistGeoOP()

```
const Eigen::MatrixXd& Network::getDistGeoOP ( ) const [inline]
```

Returns

the Euclidean distance matrix observed-prediction points

4.14.2.4 getDistGeoPP()

```
const Eigen::MatrixXd& Network::getDistGeoPP ( ) const  [inline]
```

Returns

the Euclidean distance matrix between prediction points

4.14.2.5 getDistHydroOO()

```
const Eigen::MatrixXd& Network::getDistHydroOO ( ) const  [inline]
```

Returns

the hydrological distance matrix between observed points

4.14.2.6 getDistHydroOP()

```
const Eigen::MatrixXd& Network::getDistHydroOP ( ) const  [inline]
```

Returns

the hydrological distance matrix between observed-prediction points

4.14.2.7 getDistHydroPO()

```
const Eigen::MatrixXd& Network::getDistHydroPO ( ) const  [inline]
```

Returns

the hydrological distance matrix between prediction-observed points

4.14.2.8 getDistHydroPP()

```
const Eigen::MatrixXd& Network::getDistHydroPP ( ) const  [inline]
```

Returns

the hydrological distance matrix between prediction points

4.14.2.9 getFlowMatOO()

```
const Eigen::MatrixXi& Network::getFlowMatOO ( ) const  [inline]
```

Returns

the flow-connection/unconnection binary matrix between observed points

4.14.2.10 getFlowMatOP()

```
const Eigen::MatrixXi& Network::getFlowMatOP ( ) const  [inline]
```

Returns

the flow-connection/unconnection binary matrix observed-prediction points

4.14.2.11 getFlowMatPP()

```
const Eigen::MatrixXi& Network::getFlowMatPP ( ) const  [inline]
```

Returns

the flow-connection/unconnection binary matrix between prediction points

4.14.2.12 getID()

```
const unsigned int Network::getID ( ) const  [inline]
```

Returns

the networkID

4.14.2.13 getNObs()

```
const unsigned int Network::getNObs ( ) const  [inline]
```

Returns

the number of observed points

4.14.2.14 getNPred()

```
const unsigned int Network::getNPred ( ) const  [inline]
```

Returns

the number of prediction points

4.14.2.15 getObsPoints()

```
const Points& Network::getObsPoints ( ) const  [inline]
```

Returns

the group of observed points

4.14.2.16 getPredPoints()

```
const Points& Network::getPredPoints ( ) const  [inline]
```

Returns

the group of prediction points

4.14.2.17 print()

```
void Network::print ( ) const
```

Printing function to visualize the attributes of the stream segment

4.14.2.18 setDistPoints()

```
void Network::setDistPoints (
            bool geo,
            const std::vector< Eigen::MatrixXd > & matrices )
```

Set the distance matrices regarding the relationships between the observed points and then compute the distance matrices whose rows represent the observed points and whose columns represent the predicion points

Parameters

geo	boolean, indicating if the Euclidean distances have to be computed
matrices	vector of matrices regarding the group of observed points

4.14.2.19 setID()

```
void Network::setID (
            const unsigned int id )  [inline]
```

Parameters

id	networkID

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

- Network.hpp
- Network.cpp

4.15 Optimizer Class Reference

Public Member Functions

- Optimizer (std::unique_ptr< TailUpModel > &tailup_ptr, std::unique_ptr< TailDownModel > &taildown_↩ptr, std::unique_ptr< EuclideanModel > &euclid_ptr, bool useNugg, int n_models, const Eigen::VectorXd &y, const Eigen::MatrixXd &designMat, const Eigen::MatrixXd &N, const Eigen::MatrixXd &D, const Eigen::↩MatrixXd &wMat, const Eigen::MatrixXi &connMat, bool cholesky)
- Optimizer (std::unique_ptr< TailUpModel > &tailup_ptr, std::unique_ptr< TailDownModel > &taildown_ptr, std::unique_ptr< EuclideanModel > &euclid_ptr, bool useNugg, const std::vector< double > &bounds, int n_models, const Eigen::VectorXd &y, const Eigen::MatrixXd &designMat, const Eigen::MatrixXd &N, const Eigen::MatrixXd &D, const Eigen::MatrixXd &wMat, const Eigen::MatrixXi &connMat, bool cholesky)
- const Eigen::VectorXd & getOptimTheta () const
- const Eigen::VectorXd & getBeta () const
- const Eigen::MatrixXd & getCovMat () const
- std::unique_ptr< TailUpModel > & getTailUp ()
- std::unique_ptr< TailDownModel > & getTailDown ()
- std::unique_ptr< EuclideanModel > & getEuclid ()
- void setBounds (const std::vector< double > &bounds)
- bool updateParam (const Eigen::VectorXd &theta)
- Eigen::VectorXd thetaInit ()
- double computeLogL (const Eigen::VectorXd &theta)
- std::vector< std::pair< double, Eigen::VectorXd > > simplexInit (const Eigen::VectorXd &theta0, const double tau)
- void computeTheta ()
- void glmssn ()

4.15.1 Constructor & Destructor Documentation

4.15.1.1 Optimizer() [1/2]

```
Optimizer::Optimizer (
              std::unique_ptr< TailUpModel > & tailup_ptr,
              std::unique_ptr< TailDownModel > & taildown_ptr,
              std::unique_ptr< EuclideanModel > & euclid_ptr,
              bool useNugg,
              int n_models,
              const Eigen::VectorXd & y,
              const Eigen::MatrixXd & designMat,
              const Eigen::MatrixXd & N,
              const Eigen::MatrixXd & D,
              const Eigen::MatrixXd & wMat,
              const Eigen::MatrixXi & connMat,
              bool cholesky )
```

Constructor.

Parameters

tailup_ptr	pointer to the tail-up model selected
taildown_ptr	pointer to the tail-down model selected
euclid_ptr	pointer to the Euclidean model selected
useNugg	boolean, indicating if the nuggect effects is to be considered in the mixed model
n_models	number of covariance models
y	vector of the response variable values
designMat	design matrix of the linear model
N	hydrological distance matrix
D	Euclidean distance matrix
wMat	weight matrix for tail-up model
connMat	flow-connection/unconnection binary matrix
cholesky	boolean, indicating if the user wants to always use the Cholesky decomposition when inverting positive definite matrices

4.15.1.2 Optimizer() [2/2]

```
Optimizer::Optimizer (
              std::unique_ptr< TailUpModel > & tailup_ptr,
              std::unique_ptr< TailDownModel > & taildown_ptr,
              std::unique_ptr< EuclideanModel > & euclid_ptr,
              bool useNugg,
              const std::vector< double > & bounds,
              int n_models,
              const Eigen::VectorXd & y,
              const Eigen::MatrixXd & designMat,
              const Eigen::MatrixXd & N,
              const Eigen::MatrixXd & D,
              const Eigen::MatrixXd & wMat,
              const Eigen::MatrixXi & connMat,
              bool cholesky )
```

Constructor.

Parameters

tailup_ptr	pointer to the tail-up model selected
taildown_ptr	pointer to the tail-down model selected
euclid_ptr	pointer to the Euclidean model selected
useNugg	boolean, indicating if the nuggect effects is to be considered in the mixed model
bounds	vector of upper bounds for the parsills of the covariance models
n_models	umber of covariance models
y	vector of the response variable values
designMat	designMat design matrix of the linear model
N	hydrological distance matrix
D	Euclidean distance matrix
wMat	weight matrix for tail-up model
connMat	flow-connection/unconnection binary matrix
cholesky	boolean, indicating if the user wants to always use the Cholesky decomposition when inverting positive definite matrices

4.15.2 Member Function Documentation

4.15.2.1 computeLogL()

```
double Optimizer::computeLogL (
            const Eigen::VectorXd & theta )
```

Compute the log-likelihood given the current values of the covariance parameters

Parameters

theta	vector with the current values of the covariance parameters

Returns

the value of the log-likelihood

4.15.2.2 computeTheta()

```
void Optimizer::computeTheta ( )
```

Implementation of the Nelder Mead algorithm

4.15.2.3 getBeta()

```
const Eigen::VectorXd& Optimizer::getBeta ( ) const  [inline]
```

Returns

the vector with the coefficients values for the linear model

4.15.2.4 getCovMat()

```
const Eigen::MatrixXd& Optimizer::getCovMat ( ) const  [inline]
```

Returns

the covariance matrix corresponding to the values of the parameters found

4.15.2.5 getEuclid()

```
std::unique_ptr<EuclideanModel>& Optimizer::getEuclid ( )  [inline]
```

Returns

a smart pointer to the Euclidean model selected

4.15.2.6 getOptimTheta()

```
const Eigen::VectorXd& Optimizer::getOptimTheta ( ) const  [inline]
```

Returns

the vector with the optimal values found for the covariance models

4.15.2.7 getTailDown()

```
std::unique_ptr<TailDownModel>& Optimizer::getTailDown ( )  [inline]
```

Returns

a smart pointer to the tail-down model selected

4.15.2.8 getTailUp()

```
std::unique_ptr<TailUpModel>& Optimizer::getTailUp ( ) [inline]
```

Returns

a smart pointer to the tail-up model selected

4.15.2.9 glmssn()

```
void Optimizer::glmssn ( )
```

Function that calls the Nelder Mead algorithm and stores the optimal values of the covariance parameters, the coefficients and the covariance matrix

See also

computeTheta

4.15.2.10 setBounds()

```
void Optimizer::setBounds (
            const std::vector< double > & bounds )
```

Parameters

bounds	vector of upper bounds for the parsills of the covariance models

4.15.2.11 simplexInit()

```
std::vector< std::pair< double, Eigen::VectorXd > > Optimizer::simplexInit (
            const Eigen::VectorXd & theta0,
            const double tau )
```

Compute the initial simplex for the Nelder Mead algorithm

Parameters

theta0	vector with the current values of the covariance parameters
tau	double added to component i of theta0 to obtain the i-th initial point of the simplex

Returns

TRUE if the current values of the covariance paramteres are within the bounds, FALSE otherwise

4.15.2.12 thetaInit()

```
Eigen::VectorXd Optimizer::thetaInit ( )
```

Initialize the values of the covariance parameters to find the first point of the simplex

Returns

the vector with the initial values of the covariance parameters

4.15.2.13 updateParam()

```
bool Optimizer::updateParam (
            const Eigen::VectorXd & theta )
```

Update the values of the covariance paramters during the steps of the Nelder Mead algorithm

Parameters

theta	vector with the current values of the covariance parameters

Returns

TRUE if the current values of the covariance paramteres are within the bounds, FALSE otherwise

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

- Optimizer.hpp
- Optimizer.cpp

4.16 Point Class Reference

Public Member Functions

- Point ()=default
- Point (const unsigned int r, const std::string &binID, const double dist, const unsigned int p, const double coord1, const double coord2)
- Point (const unsigned int r, const std::string &binID, const double dist, const unsigned int p, const double coord1, const double coord2, const std::string &id)
- const double getDistUpstream () const

- const unsigned int getRid () const
- const unsigned int getPid () const
- const std::vector< char > & getBinaryID () const
- const std::string getID () const
- const double getX1 () const
- const double getX2 () const
- void setRid (const unsigned int r)
- void setPid (const unsigned int p)
- void setDistUpstream (const double distUp)
- void setBinaryID (const std::string &binID)
- void setID (const std::string id)
- void setCoordinates (const double coord1, const double coord2)
- void print () const

Protected Attributes

- double distUpstream

  distance upstream
- unsigned int rid

  ID of the segment the point lies on.
- unsigned int pid

  ID of the point.
- std::vector< char > binaryID

  binaryID of the segment the point lies on, divided into characters
- std::string **ID** = "obs"
- double x1

  first coordinate
- double x2

  second coordinate

4.16.1 Constructor & Destructor Documentation

4.16.1.1 Point() [1/3]

```
Point::Point ( )  [default]
```

Default constructor

4.16.1.2 Point() [2/3]

```
Point::Point (
            const unsigned int r,
            const std::string & binID,
            const double dist,
            const unsigned int p,
            const double coord1,
            const double coord2 )
```

Constructor for an observed point

Parameters

r	ID of the segment
binID	binaryID of the segment
dist	distance upstream
p	ID of the point
coord1	first coordinate
coord2	second coordinate

4.16.1.3 Point() [3/3]

```
Point::Point (
            const unsigned int r,
            const std::string & binID,
            const double dist,
            const unsigned int p,
            const double coord1,
            const double coord2,
            const std::string & id )
```

Constructor for a prediction point

Parameters

r	ID of the segment
binID	binaryID of the segment
dist	distance upstream
p	ID of the point
coord1	first coordinate
coord2	second coordinate
id	string indicating the name of the prediction points group the point belongs to

4.16.2 Member Function Documentation

4.16.2.1 getBinaryID()

```
const std::vector<char>& Point::getBinaryID ( ) const  [inline]
```

Returns

the vector of characters representing the binaryID of the segment

4.16.2.2 getDistUpstream()

```
const double Point::getDistUpstream ( ) const  [inline]
```

Returns

the distance upstream

4.16.2.3 getID()

```
const std::string Point::getID ( ) const  [inline]
```

Returns

the string ID of the point ("obs" for observed points)

4.16.2.4 getPid()

```
const unsigned int Point::getPid ( ) const  [inline]
```

Returns

the ID of the point

4.16.2.5 getRid()

```
const unsigned int Point::getRid ( ) const  [inline]
```

Returns

the ID of the segment

4.16.2.6 getX1()

```
const double Point::getX1 ( ) const  [inline]
```

Returns

the first coordinate

4.16.2.7 getX2()

```
const double Point::getX2 ( ) const  [inline]
```

Returns

the second coordinate

4.16.2.8 print()

```
void Point::print ( ) const
```

Printing function to visualize the attributes of the stream segment

4.16.2.9 setBinaryID()

```
void Point::setBinaryID (
            const std::string & binID )
```

Parameters

binID	binaryID of the segment, to be decomposed into a vector of characters

4.16.2.10 setCoordinates()

```
void Point::setCoordinates (
            const double coord1,
            const double coord2 )
```

Parameters

coord1	first coordinate
coord2	second coordinate

4.16.2.11 setDistUpstream()

```
void Point::setDistUpstream (
            const double distUp )  [inline]
```

Parameters

distUp	distance upstream

4.16.2.12 setID()

```
void Point::setID (
            const std::string id ) [inline]
```

Parameters

id	string ID of the point

4.16.2.13 setPid()

```
void Point::setPid (
            const unsigned int p ) [inline]
```

Parameters

p	ID of the point

4.16.2.14 setRid()

```
void Point::setRid (
            const unsigned int r ) [inline]
```

Parameters

r	ID of the segment

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

- Point.hpp
- Point.cpp

4.17 Points Class Reference

Public Member Functions

- Points ()=default
- Points (const std::vector< Point > &p)
- const unsigned int getN () const
- const Eigen::MatrixXi & getFlowMat () const

- const Eigen::MatrixXd & getDistHydro () const
- const Eigen::MatrixXd & getDistGeo () const
- const std::vector< Point > & getPoints () const
- void setPoints (const std::vector< Point > &p)
- void computeDistances (bool geo, const std::map< std::string, StreamSegment > &segments)
- void setDistances (const std::vector< Eigen::MatrixXd > &matrices)

4.17.1 Constructor & Destructor Documentation

4.17.1.1 Points() [1/2]

```
Points::Points ( )  [default]
```

Default constructor

4.17.1.2 Points() [2/2]

```
Points::Points (
            const std::vector< Point > & p )
```

Constructor

Parameters

p	vector containing the Point-objects

4.17.2 Member Function Documentation

4.17.2.1 computeDistances()

```
void Points::computeDistances (
            bool geo,
            const std::map< std::string, StreamSegment > & segments )
```

Computes the distance matrices within the group of points

Parameters

geo	boolean, if TRUE then also the Euclidean distance matrix is computed
segments	map that associates to a binaryID the corresponding stream segment, used to determine the mutual position of the points along the network

4.17.2.2 getDistGeo()

```
const Eigen::MatrixXd& Points::getDistGeo ( ) const  [inline]
```

Returns

the Euclidean distance matrix

4.17.2.3 getDistHydro()

```
const Eigen::MatrixXd& Points::getDistHydro ( ) const  [inline]
```

Returns

the hydrological distance matrix

4.17.2.4 getFlowMat()

```
const Eigen::MatrixXi& Points::getFlowMat ( ) const  [inline]
```

Returns

the flow-connection/unconnection binary matrix

4.17.2.5 getN()

```
const unsigned int Points::getN ( ) const  [inline]
```

Returns

the number of points within the group

4.17.2.6 getPoints()

```
const std::vector<Point>& Points::getPoints ( ) const  [inline]
```

Returns

the vector containing the Point-objects

4.17.2.7 setDistances()

```
void Points::setDistances (
            const std::vector< Eigen::MatrixXd > & matrices )
```

Sets the distance matrices

Parameters

matrices	vector of matrices (flow-connection/unconnection, hydrological distance and, not necessarily, Euclidean distance matrix)

4.17.2.8 setPoints()

```
void Points::setPoints (
            const std::vector< Point > & p )
```

Parameters

p	vector containing the Point-objects

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

- Points.hpp
- Points.cpp

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

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

Public Types

- typedef Factory::AbstractProduct_type AbstractProduct_type
- typedef Factory::Identifier_type Identifier_type
- typedef Factory::Builder_type Builder_type
- typedef Factory Factory_type

Public Member Functions

- Proxy (Identifier_type const &)

Static Public Member Functions

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

4.18.1 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

4.18.2 Member Typedef Documentation

4.18.2.1 AbstractProduct_type

```
template<typename Factory , typename ConcreteProduct >
typedef typename Factory::AbstractProduct_type generic_factory::Proxy< Factory, Concrete←
Product >::AbstractProduct_type
```

Container for the rules.

4.18.2.2 Builder_type

```
template<typename Factory , typename ConcreteProduct >
typedef typename Factory::Builder_type generic_factory::Proxy< Factory, ConcreteProduct >←
::Builder_type
```

Builder type.

4.18.2.3 Factory_type

```
template<typename Factory , typename ConcreteProduct >
typedef Factory generic_factory::Proxy< Factory, ConcreteProduct >::Factory_type
```

Factory type.

4.18.2.4 Identifier_type

```
template<typename Factory , typename ConcreteProduct >
typedef typename Factory::Identifier_type generic_factory::Proxy< Factory, ConcreteProduct >←
::Identifier_type
```

Identifier.

4.18.3 Constructor & Destructor Documentation

4.18.3.1 Proxy()

```
template<typename F , typename C >
generic_factory::Proxy< F, C >::Proxy (
            Identifier_type const & name )
```

Constructor for the registration.

4.18.4 Member Function Documentation

4.18.4.1 Build()

```
template<typename Factory , typename ConcreteProduct >
static std::unique_ptr<AbstractProduct_type> generic_factory::Proxy< Factory, ConcreteProduct
>::Build ( ) [inline], [static]
```

Builder.

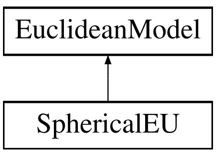
The documentation for this class was generated from the following file:

- Proxy.hpp

4.19 SphericalEU Class Reference

```
#include <EuclideanModel.hpp>
```

Inheritance diagram for SphericalEU:



Public Member Functions

- ~SphericalEU ()=default
- double computeCov (double d) override

Additional Inherited Members

4.19.1 Detailed Description

Child class for the Euclidean covariance model, implementing the spherical covariance function.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 ∼SphericalEU()

```
SphericalEU::∼SphericalEU ( ) [default]
```

Default destructor.

4.19.3 Member Function Documentation

4.19.3.1 computeCov()

```
double SphericalEU::computeCov (
            double d ) [override], [virtual]
```

Compute the covariance between two points

Parameters

d	Euclidean distance between the points

Implements EuclideanModel.

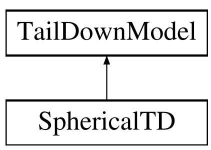
The documentation for this class was generated from the following files:

- EuclideanModel.hpp
- EuclideanModel.cpp

4.20 SphericalTD Class Reference

```
#include <TailDownModel.hpp>
```

Inheritance diagram for SphericalTD:



Public Member Functions

- ∼SphericalTD ()=default
- double computeCov (double, double) override
- double computeCov (double) override

Additional Inherited Members

4.20.1 Detailed Description

Child class for the tail-down covariance model, implementing the spherical covariance function.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 ∼SphericalTD()

```
SphericalTD::∼SphericalTD ( )  [default]
```

Default destructor

4.20.3 Member Function Documentation

4.20.3.1 computeCov() [1/2]

```
double SphericalTD::computeCov (
            double a,
            double b )  [override], [virtual]
```

Compute the covariance between two flow-unconnected points

Parameters

a	shorter hydrological distance between the points and the common downstream juction
b	greater hydrological distance between the points and the common downstream juction

Implements TailDownModel.

4.20.3.2 computeCov() [2/2]

```
double SphericalTD::computeCov (
            double h )  [override], [virtual]
```

Compute the covariance between two flow-connected points

Parameters

h	hydrological distance between the points

Implements TailDownModel.

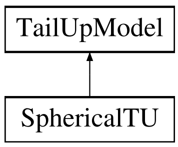
The documentation for this class was generated from the following files:

- TailDownModel.hpp
- TailDownModel.cpp

4.21 SphericalTU Class Reference

`#include <TailUpModel.hpp>`

Inheritance diagram for SphericalTU:



Public Member Functions

- virtual ∼SphericalTU ()=default
- double computeCov (double h) override

Additional Inherited Members

4.21.1 Detailed Description

Child class for the tail-up covariance model, implementing the spherical covariance function.

4.21.2 Constructor & Destructor Documentation

4.21.2.1 ∼SphericalTU()

```
virtual SphericalTU::∼SphericalTU ( )  [virtual], [default]
```

Default destructor.

4.21.3 Member Function Documentation

4.21.3.1 computeCov()

```
double SphericalTU::computeCov (
            double h )  [override], [virtual]
```

Compute the covariance between two flow-connected points

Parameters

h	hydrological distance between the points

Implements TailUpModel.

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

- TailUpModel.hpp
- TailUpModel.cpp

4.22 StreamSegment Class Reference

Public Member Functions

- StreamSegment ()=default
- StreamSegment (const unsigned int netID, const unsigned int segID, const double distUp, const std::string binID)
- const unsigned int getNetworkID () const
- const unsigned int getSegmentID () const
- const double getDistUpstream () const
- const std::string getBinaryID () const
- void setNetworkID (const unsigned int netID)
- void setSegmentID (const unsigned int segID)
- void setDistUpstream (const double distUp)
- void setBinaryID (const std::string binID)
- void print () const

4.22.1 Constructor & Destructor Documentation

4.22.1.1 StreamSegment() [1/2]

```
StreamSegment::StreamSegment ( ) [default]
```

Default constructor

4.22.1.2 StreamSegment() [2/2]

```
StreamSegment::StreamSegment (
            const unsigned int netID,
            const unsigned int segID,
            const double distUp,
            const std::string binID ) [inline]
```

Constructor.

Parameters

netID	ID of the network
segID	rid, ID of the segment
distUp	distance upstream of the most upstream point lying on the segment
binID	binaryID of the segment

4.22.2 Member Function Documentation

4.22.2.1 getBinaryID()

```
const std::string StreamSegment::getBinaryID ( ) const [inline]
```

Returns

the binaryID of the segment

4.22.2.2 getDistUpstream()

```
const double StreamSegment::getDistUpstream ( ) const [inline]
```

Returns

the upstream distance of the most upstream point lying on the segment

4.22.2.3 getNetworkID()

```
const unsigned int StreamSegment::getNetworkID ( ) const  [inline]
```

Returns

the ID of the network

4.22.2.4 getSegmentID()

```
const unsigned int StreamSegment::getSegmentID ( ) const  [inline]
```

Returns

the ID of the segment

4.22.2.5 print()

```
void StreamSegment::print ( ) const
```

Printing function to visualize the attributes of the stream segment

4.22.2.6 setBinaryID()

```
void StreamSegment::setBinaryID (
            const std::string binID )  [inline]
```

Parameters

binID	binaryID of the segment

4.22.2.7 setDistUpstream()

```
void StreamSegment::setDistUpstream (
            const double distUp )  [inline]
```

Parameters

distUp	distance upstream of the most upstream point lying on the segment

4.22.2.8 setNetworkID()

```
void StreamSegment::setNetworkID (
            const unsigned int netID )  [inline]
```

Parameters

netID	ID of the network

4.22.2.9 setSegmentID()

```
void StreamSegment::setSegmentID (
            const unsigned int segID )  [inline]
```

Parameters

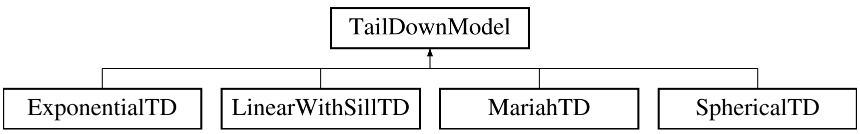
segID	ID of the segment

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

- StreamSegment.hpp
- StreamSegment.cpp

4.23 TailDownModel Class Reference

Inheritance diagram for TailDownModel:



Public Member Functions

- TailDownModel ()=default
- virtual ∼TailDownModel ()=default
- TailDownModel (double s, double a)
- virtual double computeCov (double a, double b)=0
- virtual double computeCov (double h)=0
- Eigen::MatrixXd computeMatCov (const Eigen::MatrixXi &flowMat, const Eigen::MatrixXd &distMat)
- Eigen::MatrixXd computeMatCov (const Eigen::MatrixXi &flowMat, const Eigen::MatrixXd &distMatOP, const Eigen::MatrixXd &distMatPO)
- void setSigma2 (double s)
- void setAlpha (double a)
- double getSigma2 () const
- double getAlpha () const

Protected Attributes

- double sigma2

    parsill of the tail-down model
- double alpha

    range of the tail-down model

4.23.1 Constructor & Destructor Documentation

4.23.1.1 TailDownModel() [1/2]

```
TailDownModel::TailDownModel ( ) [default]
```

Default constructor

4.23.1.2 ∼TailDownModel()

```
virtual TailDownModel::∼TailDownModel ( ) [virtual], [default]
```

Default destructor

4.23.1.3 TailDownModel() [2/2]

```
TailDownModel::TailDownModel (
            double s,
            double a ) [inline]
```

Constructor.

Parameters

s	initial value of the parsill
a	initial value of the range

4.23.2 Member Function Documentation

4.23.2.1 computeCov() [1/2]

```
virtual double TailDownModel::computeCov (
            double a,
            double b ) [pure virtual]
```

A pure virtual member function to compute the covariance between two flow-unconnected points

Parameters

a	shorter hydrological distance between the points and the common downstream juction
b	greater hydrological distance between the points and the common downstream juction

Implemented in MariahTD, ExponentialTD, SphericalTD, and LinearWithSillTD.

4.23.2.2 computeCov() [2/2]

```
virtual double TailDownModel::computeCov (
            double h )  [pure virtual]
```

A pure virtual member function to compute the covariance between two flow-connected points

Parameters

h	hydrological distance between the points

Implemented in MariahTD, ExponentialTD, SphericalTD, and LinearWithSillTD.

4.23.2.3 computeMatCov() [1/2]

```
Eigen::MatrixXd TailDownModel::computeMatCov (
            const Eigen::MatrixXi & flowMat,
            const Eigen::MatrixXd & distMat )
```

Computes the covariance matrix from a distance matrix and a flow-connection matrix

Parameters

flowMat	flow-connection/unconnection binary matrix
distMat	hydrological distance matrix

Returns

The covariance matrix

4.23.2.4 computeMatCov() [2/2]

```
Eigen::MatrixXd TailDownModel::computeMatCov (
            const Eigen::MatrixXi & flowMat,
            const Eigen::MatrixXd & distMatOP,
            const Eigen::MatrixXd & distMatPO )
```

Computes the covariance matrix from a distance matrix and a flow-connection matrix between two group of points

Parameters

flowMat	flow-connection/unconnection binary matrix
distMatOP	hydrological distance matrix, where rows represent the first group of points (in general, the observed), and columns represent the second group (in general, the prediction points)
distMatPO	hydrological distance matrix, where rows represent the second group of points (in general, the prediction points), and columns represent the first group (in general, the observed points)

Returns

The covariance matrix

4.23.2.5 getAlpha()

```
double TailDownModel::getAlpha ( ) const  [inline]
```

Returns

the range

4.23.2.6 getSigma2()

```
double TailDownModel::getSigma2 ( ) const  [inline]
```

Returns

the parsill

4.23.2.7 setAlpha()

```
void TailDownModel::setAlpha (
            double a )  [inline]
```

Parameters

a	new value of the range

4.23.2.8 setSigma2()

```
void TailDownModel::setSigma2 (
            double s ) [inline]
```

Parameters

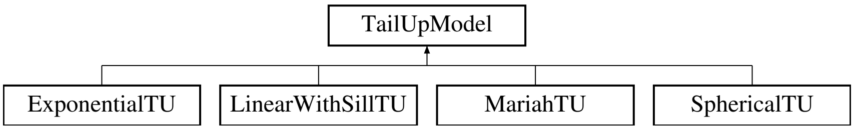
s	new value of the parsill

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

- TailDownModel.hpp
- TailDownModel.cpp

4.24 TailUpModel Class Reference

Inheritance diagram for TailUpModel:



Public Member Functions

- TailUpModel ()=default
- virtual ∼TailUpModel ()=default
- TailUpModel (double s, double a)
- virtual double computeCov (double h)=0
- Eigen::MatrixXd computeMatCov (const Eigen::MatrixXd &weightMat, const Eigen::MatrixXd &distMat)
- Eigen::MatrixXd computeMatCov (const Eigen::MatrixXd &weightMat, const Eigen::MatrixXd &distMatOP, const Eigen::MatrixXd &distMatPO)
- void setSigma2 (double s)
- void setAlpha (double a)
- double getSigma2 () const
- double getAlpha () const

Protected Attributes

- double sigma2

  parsill of the tail-down model
- double alpha

  range of the tail-down model

4.24.1 Constructor & Destructor Documentation

4.24.1.1 TailUpModel() [1/2]

```
TailUpModel::TailUpModel ( )  [default]
```

Default constructor.

4.24.1.2 ∼TailUpModel()

```
virtual TailUpModel::∼TailUpModel ( )  [virtual], [default]
```

Default destructor.

4.24.1.3 TailUpModel() [2/2]

```
TailUpModel::TailUpModel (
            double s,
            double a )  [inline]
```

Constructor.

Parameters

s	initial value of the parsill
a	initial value of the range

4.24.2 Member Function Documentation

4.24.2.1 computeCov()

```
virtual double TailUpModel::computeCov (
            double h )  [pure virtual]
```

A pure virtual member function to compute the covariance between two flow-connected points

Parameters

h	hydrological distance between the points

Implemented in MariahTU, ExponentialTU, SphericalTU, and LinearWithSillTU.

4.24.2.2 computeMatCov() [1/2]

```
Eigen::MatrixXd TailUpModel::computeMatCov (
```

```
            const Eigen::MatrixXd & weightMat,
            const Eigen::MatrixXd & distMat )
```

Computes the covariance matrix from a distance matrix and a weight matrix

Parameters

weightMat	matrix whose elements are the weights associated to pair of points
distMat	hydrological distance matrix

Returns

The covariance matrix

4.24.2.3 computeMatCov() [2/2]

```
Eigen::MatrixXd TailUpModel::computeMatCov (
            const Eigen::MatrixXd & weightMat,
            const Eigen::MatrixXd & distMatOP,
            const Eigen::MatrixXd & distMatPO )
```

Computes the covariance matrix from a distance matrix and a weight matrix between two group of points

Parameters

weightMat	matrix whose elements are the weights associated to pair of points
distMatOP	hydrological distance matrix, where rows represent the first group of points (in general, the observed), and columns represent the second group (in general, the prediction points)
distMatPO	hydrological distance matrix, where rows represent the second group of points (in general, the prediction points), and columns represent the first group (in general, the observed points)

Returns

The covariance matrix

4.24.2.4 getAlpha()

```
double TailUpModel::getAlpha ( ) const [inline]
```

Returns

the range

4.24.2.5 getSigma2()

```
double TailUpModel::getSigma2 ( ) const  [inline]
```

Returns

the parsill

4.24.2.6 setAlpha()

```
void TailUpModel::setAlpha (
            double a )  [inline]
```

Parameters

a	new value of the range

4.24.2.7 setSigma2()

```
void TailUpModel::setSigma2 (
            double s )  [inline]
```

Parameters

s	new value of the parsill

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

- TailUpModel.hpp
- TailUpModel.cpp

Chapter 5

File Documentation

5.1 Dataframe.hpp File Reference

```
#include <vector>
#include <map>
#include <string>
#include <stdexcept>
#include <Eigen/Dense>
#include <iostream>
#include <cmath>
```

Classes

- class Dataframe

5.1.1 Detailed Description

Dataframe class, representing a dataframe as a matrix with labeled columns. Labels are the names of the variables.

5.2 EuclideanModel.hpp File Reference

```
#include <Eigen/Dense>
#include <cmath>
#include <iostream>
```

Classes

- class EuclideanModel
- class CauchyEU
- class SphericalEU
- class ExponentialEU
- class GaussianEU

5.2.1 Detailed Description

Abstract class for the Euclidean covariance model.

5.3 Factory.hpp File Reference

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

Classes

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

5.3.1 Detailed Description

Factory class

5.4 FactoryHelpers.hpp File Reference

```
#include "TailUpModel.hpp"
#include "TailDownModel.hpp"
#include "EuclideanModel.hpp"
#include "Factory.hpp"
#include "Proxy.hpp"
```

Typedefs

- typedef generic_factory::Factory< TailUpModel, std::string > tailup_factory::TailUpFactory
- template<typename ConcreteProduct >
  using tailup_factory::TailUpProxy = generic_factory::Proxy< TailUpFactory, ConcreteProduct >
- typedef generic_factory::Factory< TailDownModel, std::string > taildown_factory::TailDownFactory
- template<typename ConcreteProduct >
  using taildown_factory::TailDownProxy = generic_factory::Proxy< TailDownFactory, ConcreteProduct >
- typedef generic_factory::Factory< EuclideanModel, std::string > euclidean_factory::EuclideanFactory
- template<typename ConcreteProduct >
  using euclidean_factory::EuclideanProxy = generic_factory::Proxy< EuclideanFactory, ConcreteProduct >

5.4.1 Detailed Description

Typedefs for the factory

5.4.2 Typedef Documentation

5.4.2.1 EuclideanFactory

```
typedef generic_factory::Factory< EuclideanModel, std::string > euclidean_factory::EuclideanFactory
```

Factory for the Euclidean model

5.4.2.2 EuclideanProxy

```
template<typename ConcreteProduct >
using euclidean_factory::EuclideanProxy = typedef generic_factory::Proxy<EuclideanFactory,Concrete↩
Product>
```

Proxy for the Euclidean model

5.4.2.3 TailDownFactory

```
typedef generic_factory::Factory< TailDownModel, std::string > taildown_factory::TailDownFactory
```

Factory for the tail-down model

5.4.2.4 TailDownProxy

```
template<typename ConcreteProduct >
using taildown_factory::TailDownProxy = typedef generic_factory::Proxy<TailDownFactory,Concrete↩
Product>
```

Proxy for the tail-down model

5.4.2.5 TailUpFactory

```
typedef generic_factory::Factory< TailUpModel, std::string > tailup_factory::TailUpFactory
```

Factory for the tail-up model

5.4.2.6 TailUpProxy

```
template<typename ConcreteProduct >
using tailup_factory::TailUpProxy = typedef generic_factory::Proxy<TailUpFactory,Concrete↩
Product>
```

Proxy for the tail-up model

5.5 Helpers.hpp File Reference

```
#include <iostream>
#include <vector>
#include <string>
#include <map>
#include <Eigen/Dense>
#include <cmath>
#include "Network.hpp"
```

Functions

- double helpers::geoDist (const double x11, const double x12, const double x21, const double x22)
- bool helpers::operandPair (std::pair< double, Eigen::VectorXd > p1, std::pair< double, Eigen::VectorXd > p2)
- void helpers::pointsStorage (std::vector< std::map< unsigned int, std::string >> &segmentsMaps, Eigen↩ ::MatrixXd &pointsMat, std::vector< std::vector< Point >> &storage)
- std::vector< Eigen::MatrixXd > helpers::returnBlockMatrices (const Points &p)
- Eigen::MatrixXd helpers::geoDistBetweenNets (const Points &p1, const Points &p2)
- std::vector< Eigen::MatrixXd > helpers::createDistMatrices (bool geo, const std::vector< Network > &net, unsigned int nTot)
- std::vector< Eigen::MatrixXd > helpers::createDistMatricesOP (bool geo, const std::vector< Network > &net, unsigned int nObs, unsigned int nPred)

5.5.1 Detailed Description

Helpers functions.

5.5.2 Function Documentation

5.5.2.1 createDistMatrices()

```
std::vector< Eigen::MatrixXd > helpers::createDistMatrices (
            bool geo,
            const std::vector< Network > & net,
            unsigned int nTot )
```

Build the block matrices regarding the hydrological and, if necessary, Euclidean distances and flow-connection/unconnection between observed points

Parameters

geo	boolean indicating if the Euclidean distance matrix has to be computed
net	vector of networks whose general distance matrices have to be computed
nTot	total number of observed points in the data set

Returns

vector of distance matrices

5.5.2.2 createDistMatricesOP()

```
std::vector< Eigen::MatrixXd > helpers::createDistMatricesOP (
            bool geo,
            const std::vector< Network > & net,
            unsigned int nObs,
            unsigned int nPred )
```

Build the block matrices regarding the hydrological and, if necessary, Euclidean distances and flow-connection/unconnection between observed and prediction points

Parameters

geo	boolean indicating if the Euclidean distance matrix has to be computed
net	vector of networks whose general distance matrices have to be computed
nObs	total number of observed points in the data set
nPred	total number of prediction points in the data set

Returns

vector of distance matrices

5.5.2.3 geoDist()

```
double helpers::geoDist (
            const double x11,
            const double x12,
            const double x21,
            const double x22 )
```

Compute the Euclidean distance between two points.

Parameters

x11	first coordinate of the first point
x12	second coordinate of the first point
x21	first coordinate of the second point
x22	second coordinate of the second point

Returns

the Euclidean distance between the points

5.5.2.4 geoDistBetweenNets()

```
Eigen::MatrixXd helpers::geoDistBetweenNets (
            const Points & p1,
            const Points & p2 )
```

Compute the Euclidean distance between points belonging to different networks

Parameters

p1	Points object of the first network
p2	Points object of the second network

Returns

Euclidean distance matrix

5.5.2.5 operandPair()

```
bool helpers::operandPair (
            std::pair< double, Eigen::VectorXd > p1,
            std::pair< double, Eigen::VectorXd > p2 )
```

Compare pairs of <double, Eigen::VectorXd>, looking just at the double

Parameters

p1	first pair
p2	second pair

Returns

TRUE if the first pair is "less than" the second pair

5.5.2.6 pointsStorage()

```
void helpers::pointsStorage (
            std::vector< std::map< unsigned int, std::string >> & segmentsMaps,
            Eigen::MatrixXd & pointsMat,
            std::vector< std::vector< Point >> & storage )
```

Store a group of points, with all their attributes, belonging to different networks

Parameters

segmentsMaps	vector of maps, one per each network, that associates to each segmentID the StreamSegment object
pointsMat	matrix containing the attributes of the points
storage	vector of vectors, one per each network, of Point objects, to be filled

5.5.2.7 returnBlockMatrices()

```
std::vector< Eigen::MatrixXd > helpers::returnBlockMatrices (
            const Points & p )
```

Return the distance matrices and flow-connection/unconnection binary matrix of a group of points

Parameters

p	Points object

Returns

vector of matrices

5.6 interface.cpp File Reference

```
#include <iostream>
#include <string>
#include <vector>
#include <Eigen/Dense>
#include <list>
#include <Rcpp.h>
#include <RcppEigen.h>
#include "Dataframe.hpp"
#include "Network.hpp"
#include "Optimizer.hpp"
#include "Kriging.hpp"
```

Functions

- RcppExport SEXP createHydroDistanceMatrices_MultipleNets (SEXP net_num, SEXP bin_tables, SEXP network_data, SEXP obs_points)
- RcppExport SEXP createDistanceMatrices_MultipleNets (SEXP net_num, SEXP bin_tables, SEXP network_data, SEXP obs_points)
- RcppExport SEXP getSSNModel_MultipleNets (SEXP net_num, SEXP bin_tables, SEXP network_data, S↩ EXP obs_points, SEXP obs_data, SEXP var_names, SEXP model_names, SEXP nugg, SEXP dist_matrices, SEXP model_bounds, SEXP use_cholesky)

- RcppExport SEXP doSSNKriging_MultipleNets (SEXP net_num, SEXP bin_tables, SEXP network_data, S↩ EXP obs_points, SEXP pred_points, SEXP obs_data, SEXP pred_data, SEXP var_names, SEXP model_↩ names, SEXP nugg, SEXP param, SEXP cov_mat, SEXP dist_matrices)
- RcppExport SEXP getSSNModelKriging_MultipleNets (SEXP net_num, SEXP bin_tables, SEXP network↩ _data, SEXP obs_points, SEXP pred_points, SEXP obs_data, SEXP pred_data, SEXP var_names, SEXP model_names, SEXP nugg, SEXP dist_matrices, SEXP model_bounds, SEXP use_cholesky)
- RcppExport SEXP createHydroDistanceMatrices_SingleNet (SEXP bin_table, SEXP network_data, SEXP obs_points)
- RcppExport SEXP createDistanceMatrices_SingleNet (SEXP bin_table, SEXP network_data, SEXP obs_↩ points)
- RcppExport SEXP getSSNModel_SingleNet (SEXP bin_table, SEXP network_data, SEXP obs_points, SEXP obs_data, SEXP var_names, SEXP model_names, SEXP nugg, SEXP dist_matrices, SEXP model_bounds, SEXP use_cholesky)
- RcppExport SEXP doSSNKriging_SingleNet (SEXP bin_table, SEXP network_data, SEXP obs_points, S↩ EXP pred_points, SEXP obs_data, SEXP pred_data, SEXP var_names, SEXP model_names, SEXP nugg, SEXP param, SEXP cov_mat, SEXP dist_matrices)
- RcppExport SEXP getSSNModelKriging_SingleNet (SEXP bin_table, SEXP network_data, SEXP obs_↩ points, SEXP pred_points, SEXP obs_data, SEXP pred_data, SEXP var_names, SEXP model_names, SEXP nugg, SEXP dist_matrices, SEXP model_bounds, SEXP use_cholesky)

5.6.1 Detailed Description

Functions to create distance matrices, fit a spatial linear model and perform Universal kriging on a spatial stream network object, when the data set has multiple networks or a single one.

5.6.2 Function Documentation

5.6.2.1 createDistanceMatrices_MultipleNets()

```
RcppExport SEXP createDistanceMatrices_MultipleNets (
            SEXP net_num,
            SEXP bin_tables,
            SEXP network_data,
            SEXP obs_points )
```

Compute the hydrological and Euclidean distances between observed points, as well as the flow-connection/unconnection binary matrix

Parameters

net_num	vector containing the networkID per each network of the data set
bin_tables	list of vectors of strings, one vector per each network, containing the binaryIDs of the stream segments of each network
network_data	matrix whose columns correspond to the following attributes of the stream segments: networkID, segmentID, distance upstream
obs_points	matrix whose columns correspond to the following attributes of the observed points: networkID, segmentID, distance upstream, first coordinate, second coordinate

Returns

A list with the following fields:

- 'flowMat' flow-connection/unconnection binary matrix
- 'distHydro' hydrological distance matrix
- 'distGeo' Euclidean distance matrix

5.6.2.2 createDistanceMatrices_SingleNet()

```
RcppExport SEXP createDistanceMatrices_SingleNet (
            SEXP bin_table,
            SEXP network_data,
            SEXP obs_points )
```

Compute the hydrological and Euclidean distances between observed points, as well as the flow-connection/unconnection binary matrix

Parameters

bin_table	vector of strings containing the binaryIDs of the stream segments of the network
network_data	matrix whose columns correspond to the following attributes of the stream segments: networkID, segmentID, distance upstream
obs_points	matrix whose columns correspond to the following attributes of the observed points: networkID, segmentID, distance upstream, first coordinate, second coordinate

Returns

A list with the following fields:

- 'flowMat' flow-connection/unconnection binary matrix
- 'distHydro' hydrological distance matrix
- 'distGeo' Euclidean distance matrix

5.6.2.3 createHydroDistanceMatrices_MultipleNets()

```
RcppExport SEXP createHydroDistanceMatrices_MultipleNets (
            SEXP net_num,
            SEXP bin_tables,
            SEXP network_data,
            SEXP obs_points )
```

Functions to create distance matrices, fit a spatial linear model and perform Universal kriging on a spatial stream network object, which has multiple networks.Compute the hydrological distances between observed points, as well as the flow-connection/unconnection binary matrix

Parameters

net_num	vector containing the networkID per each network of the data set

Parameters

bin_tables	list of vectors of strings, one vector per each network, containing the binaryIDs of the stream segments of each network
network_data	matrix whose columns correspond to the following attributes of the stream segments: networkID, segmentID, distance upstream
obs_points	matrix whose columns correspond to the following attributes of the observed points: networkID, segmentID, distance upstream, first coordinate, second coordinate

Returns

A list with the following fields:

- 'flowMat' flow-connection/unconnection binary matrix
- 'distHydro' hydrological distance matrix

5.6.2.4 createHydroDistanceMatrices_SingleNet()

```
RcppExport SEXP createHydroDistanceMatrices_SingleNet (
            SEXP bin_table,
            SEXP network_data,
            SEXP obs_points )
```

Functions to create distance matrices, fit a spatial linear model and perform Universal kriging on a spatial stream network object, which has one single network.Compute the hydrological distances between observed points, as well as the flow-connection/unconnection binary matrix

Parameters

bin_table	vector of strings containing the binaryIDs of the stream segments of the network
network_data	matrix whose columns correspond to the following attributes of the stream segments: networkID, segmentID, distance upstream
obs_points	matrix whose columns correspond to the following attributes of the observed points: networkID, segmentID, distance upstream, first coordinate, second coordinate

Returns

A list with the following fields:

- 'flowMat' flow-connection/unconnection binary matrix
- 'distHydro' hydrological distance matrix

5.6.2.5 doSSNKriging_MultipleNets()

```
RcppExport SEXP doSSNKriging_MultipleNets (
            SEXP net_num,
            SEXP bin_tables,
```

```
            SEXP network_data,
            SEXP obs_points,
            SEXP pred_points,
            SEXP obs_data,
            SEXP pred_data,
            SEXP var_names,
            SEXP model_names,
            SEXP nugg,
            SEXP param,
            SEXP cov_mat,
            SEXP dist_matrices )
```

Perform Universal kriging on a spatial stream network object

Parameters

net_num	vector containing the networkID per each network of the data set
bin_tables	list of vector of strings, one vector per each network, containing the binaryIDs of the stream segments of each network
network_data	matrix whose columns correspond to the following attributes of the stream segments: networkID, segmentID, distance upstream
obs_points	matrix whose columns correspond to the following attributes of the observed points: networkID, segmentID, distance upstream, first coordinate, second coordinate
pred_points	matrix whose columns correspond to the following attributes of the prediction points: networkID, segmentID, distance upstream, first coordinate, second coordinate
obs_data	matrix whose columns correspond to the values of the responde variable and model covariates of the observed points
pred_data	matrix whose columns correspond to the values of the responde variable and model covariates of the prediction points
var_names	vector of strings indicating first the name of the response variable and then the name of the covariates
model_names	vector of strings indicating the covariance models selected
nugg	boolean indicating if the nugget effect is to be considered in the mixed model
param	vector with the optimal values of the covariance parameters found through the model fitting
cov_mat	covariance matrix computed using the optimal values of the covariance parameters
dist_matrices	vector of distance matrices between observed points. It can be also NULL

Returns

A list with the following fields:

- 'predictions' 2-column matrix containing the predicted values and kriging variance of each prediction point

5.6.2.6 doSSNKriging_SingleNet()

```
RcppExport SEXP doSSNKriging_SingleNet (
            SEXP bin_table,
            SEXP network_data,
            SEXP obs_points,
            SEXP pred_points,
```

```
            SEXP obs_data,
            SEXP pred_data,
            SEXP var_names,
            SEXP model_names,
            SEXP nugg,
            SEXP param,
            SEXP cov_mat,
            SEXP dist_matrices )
```

Perform Universal kriging on a spatial stream network object

Parameters

bin_table	vector of strings containing the binaryIDs of the stream segments of the network
network_data	matrix whose columns correspond to the following attributes of the stream segments: networkID, segmentID, distance upstream
obs_points	matrix whose columns correspond to the following attributes of the observed points: networkID, segmentID, distance upstream, first coordinate, second coordinate
pred_points	matrix whose columns correspond to the following attributes of the prediction points: networkID, segmentID, distance upstream, first coordinate, second coordinate
obs_data	matrix whose columns correspond to the values of the responde variable and model covariates of the observed points
pred_data	matrix whose columns correspond to the values of the responde variable and model covariates of the prediction points
var_names	vector of strings indicating first the name of the response variable and then the name of the covariates
model_names	vector of strings indicating the covariance models selected
nugg	boolean indicating if the nugget effect is to be considered in the mixed model
param	vector with the optimal values of the covariance parameters found through the model fitting
cov_mat	covariance matrix computed using the optimal values of the covariance parameters
dist_matrices	vector of distance matrices between observed points. It can be also NULL

Returns

A list with the following fields:
- 'predictions' 2-column matrix containing the predicted values and kriging variance of each prediction point

5.6.2.7 getSSNModel_MultipleNets()

```
RcppExport SEXP getSSNModel_MultipleNets (
            SEXP net_num,
            SEXP bin_tables,
            SEXP network_data,
            SEXP obs_points,
            SEXP obs_data,
            SEXP var_names,
            SEXP model_names,
            SEXP nugg,
            SEXP dist_matrices,
            SEXP model_bounds,
            SEXP use_cholesky )
```

Fit a spatial linear model on a spatial stream network

Parameters

net_num	vector containing the networkID per each network of the data set
bin_tables	list of vector of strings, one vector per each network, containing the binaryIDs of the stream segments of each network
network_data	matrix whose columns correspond to the following attributes of the stream segments: networkID, segmentID, distance upstream
obs_points	matrix whose columns correspond to the following attributes of the observed points: networkID, segmentID, distance upstream, first coordinate, second coordinate
obs_data	matrix whose columns correspond to the values of the responde variable and model covariates of the observed points
var_names	vector of strings indicating first the name of the response variable and then the name of the covariates
model_names	vector of strings indicating the covariance models selected
nugg	boolean indicating if the nugget effect is to be considered in the mixed model
dist_matrices	vector of distance matrices between observed points. It can be also NULL
model_bounds	vector of upper bounds for the parsills of the covariance models. It can be also NULL
use_cholesky	boolean indicating if the Cholesky decomposition is to be always preferred for computing the inverse of positive definite matrices

Returns

A list with the following fields:

- 'optTheta' vector with the optimal values of the covariance parameters found through the model fitting
- 'betaValues' vector with the coefficients of the linear model
- 'covMatrix' covariance matrix computed using the optimal values of the covariance parameters

5.6.2.8 getSSNModel_SingleNet()

```
RcppExport SEXP getSSNModel_SingleNet (
             SEXP bin_table,
             SEXP network_data,
             SEXP obs_points,
             SEXP obs_data,
             SEXP var_names,
             SEXP model_names,
             SEXP nugg,
             SEXP dist_matrices,
             SEXP model_bounds,
             SEXP use_cholesky )
```

Fit a spatial linear model on a spatial stream network

Parameters

bin_table	vector of strings containing the binaryIDs of the stream segments of the network
network_data	matrix whose columns correspond to the following attributes of the stream segments: networkID, segmentID, distance upstream
obs_points	matrix whose columns correspond to the following attributes of the observed points: networkID, segmentID, distance upstream, first coordinate, second coordinate

Parameters

obs_data	matrix whose columns correspond to the values of the responde variable and model covariates of the observed points
var_names	vector of strings indicating first the name of the response variable and then the name of the covariates
model_names	vector of strings indicating the covariance models selected
nugg	boolean indicating if the nugget effect is to be considered in the mixed model
dist_matrices	vector of distance matrices between observed points. It can be also NULL
model_bounds	vector of upper bounds for the parsills of the covariance models. It can be also NULL
use_cholesky	boolean indicating if the Cholesky decomposition is to be always preferred for computing the inverse of positive definite matrices

Returns

A list with the following fields:

- 'optTheta' vector with the optimal values of the covariance parameters found through the model fitting
- 'betaValues' vector with the coefficients of the linear model
- 'covMatrix' covariance matrix computed using the optimal values of the covariance parameters

5.6.2.9 getSSNModelKriging_MultipleNets()

```
RcppExport SEXP getSSNModelKriging_MultipleNets (
            SEXP net_num,
            SEXP bin_tables,
            SEXP network_data,
            SEXP obs_points,
            SEXP pred_points,
            SEXP obs_data,
            SEXP pred_data,
            SEXP var_names,
            SEXP model_names,
            SEXP nugg,
            SEXP dist_matrices,
            SEXP model_bounds,
            SEXP use_cholesky )
```

Fit a spatial linear model on a spatial stream network and perform Universal kriging

Parameters

net_num	vector containing the networkID per each network of the data set
bin_tables	list of vector of strings, one vector per each network, containing the binaryIDs of the stream segments of each network
network_data	matrix whose columns correspond to the following attributes of the stream segments: networkID, segmentID, distance upstream
obs_points	matrix whose columns correspond to the following attributes of the observed points: networkID, segmentID, distance upstream, first coordinate, second coordinate
pred_points	matrix whose columns correspond to the following attributes of the prediction points: networkID, segmentID, distance upstream, first coordinate, second coordinate

Parameters

obs_data	matrix whose columns correspond to the values of the responde variable and model covariates of the observed points
pred_data	matrix whose columns correspond to the values of the responde variable and model covariates of the prediction points
var_names	vector of strings indicating first the name of the response variable and then the name of the covariates
model_names	vector of strings indicating the covariance models selected
nugg	boolean indicating if the nugget effect is to be considered in the mixed model
dist_matrices	vector of distance matrices between observed points. It can be also NULL
model_bounds	vector of upper bounds for the parsills of the covariance models. It can be also NULL
use_cholesky	boolean indicating if the Cholesky decomposition is to be always preferred for computing the inverse of positive definite matrices

Returns

A list with the following fields:

- 'optTheta' vector with the optimal values of the covariance parameters found through the model fitting
- 'betaValues' vector with the coefficients of the linear model
- 'covMatrix' covariance matrix computed using the optimal values of the covariance parameters
- 'predictions' 2-column matrix containing the predicted values and kriging variance of each prediction point

5.6.2.10 getSSNModelKriging_SingleNet()

```
RcppExport SEXP getSSNModelKriging_SingleNet (
             SEXP bin_table,
             SEXP network_data,
             SEXP obs_points,
             SEXP pred_points,
             SEXP obs_data,
             SEXP pred_data,
             SEXP var_names,
             SEXP model_names,
             SEXP nugg,
             SEXP dist_matrices,
             SEXP model_bounds,
             SEXP use_cholesky )
```

Fit a spatial linear model on a spatial stream network and perform Universal kriging

Parameters

bin_table	vector of strings containing the binaryIDs of the stream segments of the network
network_data	matrix whose columns correspond to the following attributes of the stream segments: networkID, segmentID, distance upstream
obs_points	matrix whose columns correspond to the following attributes of the observed points: networkID, segmentID, distance upstream, first coordinate, second coordinate
pred_points	matrix whose columns correspond to the following attributes of the prediction points: networkID, segmentID, distance upstream, first coordinate, second coordinate

Parameters

obs_data	matrix whose columns correspond to the values of the responde variable and model covariates of the observed points
pred_data	matrix whose columns correspond to the values of the responde variable and model covariates of the prediction points
var_names	vector of strings indicating first the name of the response variable and then the name of the covariates
model_names	vector of strings indicating the covariance models selected
nugg	boolean indicating if the nugget effect is to be considered in the mixed model
dist_matrices	vector of distance matrices between observed points. It can be also NULL
model_bounds	vector of upper bounds for the parsills of the covariance models. It can be also NULL
use_cholesky	boolean indicating if the Cholesky decomposition is to be always preferred for computing the inverse of positive definite matrices

Returns

A list with the following fields:

- 'optTheta' vector with the optimal values of the covariance parameters found through the model fitting
- 'betaValues' vector with the coefficients of the linear model
- 'covMatrix' covariance matrix computed using the optimal values of the covariance parameters
- 'predictions' 2-column matrix containing the predicted values and kriging variance of each prediction point

5.7 Kriging.hpp File Reference

```
#include <iostream>
#include <vector>
#include <Eigen/Dense>
#include <Eigen/SVD>
#include <Eigen/QR>
#include <Eigen/Eigenvalues>
#include <cmath>
#include "Dataframe.hpp"
#include "Network.hpp"
#include "Factory.hpp"
#include "FactoryHelpers.hpp"
#include "Helpers.hpp"
#include "Proxy.hpp"
#include "TailUpModel.hpp"
#include "TailDownModel.hpp"
#include "EuclideanModel.hpp"
```

Classes

- class Kriging

5.7.1 Detailed Description

Kriging class, for performing universal kriging. It calculates prediction values and kriging variance for prediction sites based on the results of a linear model fitting. The matrices, used for computing the covariance, describe relationships between observed and prediction points.

5.8 Network.hpp File Reference

```
#include <iostream>
#include <vector>
#include <string>
#include <Eigen/Dense>
#include "Points.hpp"
#include "Helpers.hpp"
```

Classes

- class Network

5.8.1 Detailed Description

Network class, representing a single network of the data set.

5.9 Optimizer.hpp File Reference

```
#include <iostream>
#include <vector>
#include <Eigen/Dense>
#include <Eigen/SVD>
#include <Eigen/QR>
#include <Eigen/Eigenvalues>
#include <cmath>
#include <stdexcept>
#include <Rcpp.h>
#include "Factory.hpp"
#include "FactoryHelpers.hpp"
#include "Helpers.hpp"
#include "Proxy.hpp"
#include "TailUpModel.hpp"
#include "TailDownModel.hpp"
#include "EuclideanModel.hpp"
```

Classes

- class Optimizer

5.9.1 Detailed Description

Optimizer class, for fitting a spatial linear model with spatially autocorrelated errors using normal likelihood methods. The optimization is performed via the Nelder-Mead algorithm. It uses matrices regarding relationships between observed points.

5.10 Point.hpp File Reference

```
#include <string>
#include <vector>
#include <iostream>
```

Classes

- class Point

5.10.1 Detailed Description

Point class, representing a single point, observed or for prediction. Each point has information about the binaryID and the rid of the segment it lies on, its upstream distance and its coordinates.

5.11 Points.hpp File Reference

```
#include <iostream>
#include <vector>
#include <map>
#include <Eigen/Dense>
#include <stdexcept>
#include "Point.hpp"
#include "StreamSegment.hpp"
```

Classes

- class Points

5.11.1 Detailed Description

Points class, representing a group of points, observed or prediction ones. Each group has three matrices that will be used in computing the covariance matrix: the flow-connection/unconnection binary matrix, the hydrological and Euclidean distance matrices.

5.12 Proxy.hpp File Reference

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

Classes

- class generic_factory::Proxy< Factory, ConcreteProduct >

5.12.1 Detailed Description

Proxy class

5.13 StreamSegment.hpp File Reference

```
#include <string>
#include <iostream>
```

Classes

- class StreamSegment

5.13.1 Detailed Description

StreamSegment class, representing a segment of a stream network as a continuous line. Each segment has an ID (rid) and a binaryID representing the position in the network.

5.14 TailDownModel.hpp File Reference

```
#include <cmath>
#include <Eigen/Dense>
#include <iostream>
```

Classes

- class TailDownModel
- class LinearWithSillTD
- class SphericalTD
- class ExponentialTD
- class MariahTD

5.14.1 Detailed Description

Abstract class for the tail-down covariance model.

5.15 TailUpModel.hpp File Reference

```
#include <cmath>
#include <Eigen/Dense>
#include <iostream>
```

Classes

- class TailUpModel
- class LinearWithSillTU
- class SphericalTU
- class ExponentialTU
- class MariahTU

5.15.1 Detailed Description

Abstract class for the tail-up covariance model.