My Project

Generated by Doxygen 1.9.2

Chapter 1

Namespace Index

1.1 Namespace List

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

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

Namespace Documentation

5.1 LocallyStationaryModels::cd Namespace Reference

collects all the typedef used inside the code

Typedefs

- using **vector** = Eigen::VectorXd
- using **matrix** = Eigen::MatrixXd
- using **matrixI** = Eigen::MatrixXi
- using **vectorptr** = std::shared_ptr< vector >
- using **matrixptr** = std::shared_ptr< matrix >
- using **matrixIptr** = std::shared_ptr< matrixI >
- using **vectorind** = std::vector< size_t >
- using **kernelfunction** = std::function< double(const vector &, const vector &, const double &)>
- using **gridfunction** = std::function< matrixIptr(const matrixptr &, const size_t &, const size_t &, const double &)>

5.1.1 Detailed Description

collects all the typedef used inside the code

Namespace cd

5.2 LocallyStationaryModels::gf Namespace Reference

collect the grid functions and the corresponding factory function

Functions

- matrixIptr pizza (const cd::matrixptr &data, const size_t &n_angles, const size_t &n_intervals, const double &epsilon)

  this function builds a 2D-grid using a "a fette di pizza" (slices-of-pizza like) algorithm to partition the domain
- gridfunction make_grid (const std::string &id)

  allow to select between the preferred method to build the grid

5.2.1 Detailed Description

collect the grid functions and the corresponding factory function

Namespace gf

5.2.2 Function Documentation

5.2.2.1 make_grid()

```
cd::gridfunction LocallyStationaryModels::gf::make_grid (
             const std::string & id )
```

allow to select between the preferred method to build the grid

Parameters

id	name of the function of choice

5.2.2.2 pizza()

```
cd::matrixIptr LocallyStationaryModels::gf::pizza (
             const cd::matrixptr & data,
             const size_t & n_angles,
             const size_t & n_intervals,
             const double & epsilon )
```

this function builds a 2D-grid using a "a fette di pizza" (slices-of-pizza like) algorithm to partition the domain

Parameters

data	a shared pointer to the matrix of the coordinates
n_angles	number of slices of the pizza
n_intervals	number of the pieces for each slice of the pizza
epsilon	bandwidth parameter epsilon. Same of the kernel

5.3 LocallyStationaryModels::kf Namespace Reference

collect the kernel functions and the corresponding factory function

Functions

- double gaussian (const vector &x, const vector &y, const double &epsilon)
- double identity (const vector &x, const vector &y, const double &epsilon)
- kernelfunction make_kernel (const std::string &id)

  allow to select between pre-built kernel functions

5.3.1 Detailed Description

collect the kernel functions and the corresponding factory function

Namespace kf

5.3.2 Function Documentation

5.3.2.1 gaussian()

```
double LocallyStationaryModels::kf::gaussian (
             const cd::vector & x,
             const cd::vector & y,
             const double & epsilon )
```

Returns

e^(-norm(x-y)^2/epsilon^2)

5.3.2.2 identity()

```
double LocallyStationaryModels::kf::identity (
             const cd::vector & x,
             const cd::vector & y,
             const double & epsilon )
```

Returns

1 only if the norm of the difference between x and y is less the the square of epsilon

5.3.2.3 make_kernel()

```
cd::kernelfunction LocallyStationaryModels::kf::make_kernel (
             const std::string & id )
```

allow to select between pre-built kernel functions

Parameters

id	a string with the name of the kernel function you want to use

Chapter 6

Class Documentation

6.1 LocallyStationaryModels::Anchor Class Reference

a simple class to find the anchor points given the data

```
#include <anchor.hpp>
```

Public Member Functions

- Anchor (const cd::matrixptr &data, const double &n_pieces)

  constructor
- const cd::matrix **find_anchorpoints** ()

  this function returns the coordinates of the anchor points in a way such that every anchor point has at least one point of the domain in its neighbourhood
- std::pair< double, double > get_origin () const
- std::pair< double, double > get_tiles_dimensions () const

6.1.1 Detailed Description

a simple class to find the anchor points given the data

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Anchor()

```
LocallyStationaryModels::Anchor::Anchor (
            const cd::matrixptr & data,
            const double & n_pieces )  [inline]
```

constructor

Parameters

data	shared pointer to the matrix with the coordinates of the dataset points
n_pieces	the number of tiles per row and column of the grid

6.1.3 Member Function Documentation

6.1.3.1 get_origin()

```
std::pair< double, double > LocallyStationaryModels::Anchor::get_origin ( ) const  [inline]
```

Returns

the coordinates of the origin of the grid

6.1.3.2 get_tiles_dimensions()

```
std::pair< double, double > LocallyStationaryModels::Anchor::get_tiles_dimensions ( ) const
[inline]
```

Returns

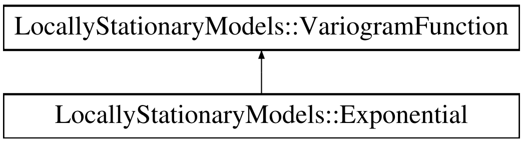
the dimensions (height and width) of each cell of the grid

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

- anchor.hpp

6.2 LocallyStationaryModels::Exponential Class Reference

Inheritance diagram for LocallyStationaryModels::Exponential:


LocallyStationaryModels::VariogramFunction
LocallyStationaryModels::Exponential

Public Member Functions

- double operator() (const cd::vector &params, const double &x, const double &y) override

Additional Inherited Members

6.2.1 Member Function Documentation

6.2.1.1 operator()()

```
double LocallyStationaryModels::Exponential::operator() (
            const cd::vector & params,
            const double & x,
            const double & y )  [override], [virtual]
```

Returns

sigma ∗ sigma ∗ (1 - exp(-h))

Parameters

params	a vector with lambda1, lambda2, phi and sigma in this exact order

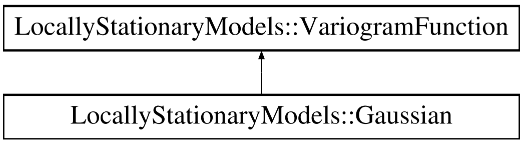
Implements LocallyStationaryModels::VariogramFunction.

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

- variogramfunctions.hpp
- variogramfunctions.cpp

6.3 LocallyStationaryModels::Gaussian Class Reference

Inheritance diagram for LocallyStationaryModels::Gaussian:


LocallyStationaryModels::VariogramFunction
LocallyStationaryModels::Gaussian

Public Member Functions

- double operator() (const cd::vector &params, const double &x, const double &y) override

Additional Inherited Members

6.3.1 Member Function Documentation

6.3.1.1 operator()()

```
double LocallyStationaryModels::Gaussian::operator() (
            const cd::vector & params,
            const double & x,
            const double & y )  [override], [virtual]
```

Returns

sigma ∗ sigma ∗ (1 - exp(-h∗h))

Parameters

params	a vector with lambda1, lambda2, phi and sigma in this exact order

Implements LocallyStationaryModels::VariogramFunction.

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

- variogramfunctions.hpp
- variogramfunctions.cpp

6.4 LocallyStationaryModels::Grid Class Reference

class to build the grid

`#include <grid.hpp>`

Public Member Functions

- Grid (const std::string &id, const double &epsilon)

  constructor
- **Grid** ()

  constructor. Use the Pizza style by default
- void build_grid (const cd::matrixptr &data, const size_t &n_angles, const size_t &n_intervals)

  build the grid
- const cd::matrixIptr get_grid () const
- const cd::vectorptr get_normh () const
- const cd::vectorptr get_x () const
- const cd::vectorptr get_y () const

6.4.1 Detailed Description

class to build the grid

6.4.2 Constructor & Destructor Documentation

6.4.2.1 Grid()

```
LocallyStationaryModels::Grid::Grid (
             const std::string & id,
             const double & epsilon )
```

constructor

Parameters

id	name of the grid function
epsilon	the same epsilon regulating the kernel

6.4.3 Member Function Documentation

6.4.3.1 build_grid()

```
void LocallyStationaryModels::Grid::build_grid (
             const cd::matrixptr & data,
             const size_t & n_angles,
             const size_t & n_intervals )
```

build the grid

Parameters

data	a shared pointer to the matrix of the coordinates
n_angles	number of slices of the pizza
n_intervals	number of the pieces for each slice of the pizza

6.4.3.2 get_grid()

```
const matrixIptr LocallyStationaryModels::Grid::get_grid ( ) const
```

Returns

a shared pointer to the grid

6.4.3.3 get_normh()

```
const vectorptr LocallyStationaryModels::Grid::get_normh ( ) const
```

Returns

a shared pointer to m_normh

6.4.3.4 get_x()

```
const vectorptr LocallyStationaryModels::Grid::get_x ( ) const
```

Returns

a pointer to the vector containing the xs of the centers of the cells of the grid

6.4.3.5 get_y()

```
const vectorptr LocallyStationaryModels::Grid::get_y ( ) const
```

Returns

a pointer to the vector containing the ys of the centers of the cells of the grid

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

- grid.hpp
- grid.cpp

6.5 LocallyStationaryModels::Kernel Class Reference

class to compute the kernel matrix

```
#include <kernel.hpp>
```

Public Member Functions

- Kernel (const std::string &id, const double &epsilon)

  constructor
- **Kernel** ()

  default constuctor with a gaussian kernel and epsilon equal to 1.
- double operator() (const cd::vector &x, const cd::vector &y) const
- void build_kernel (const cd::matrixptr &data, const cd::matrixptr &anchorpoints)

  build the "star" version of the kernel that contains the standardized kernel weights in such a way that each row sums to one
- void build_simple_kernel (const cd::matrixptr &coordinates)

  build the "standard" version of the kernel needed for smoothing
- void build_simple_kernel (const cd::matrixptr &coordinates, const double &epsilon)

  build the "standard" version of the kernel needed for smoothing
- const cd::matrixptr get_kernel () const

6.5.1 Detailed Description

class to compute the kernel matrix

6.5.2 Constructor & Destructor Documentation

6.5.2.1 Kernel()

```
LocallyStationaryModels::Kernel::Kernel (
            const std::string & id,
            const double & epsilon )
```

constructor

Parameters

id	name of the kernel function
epsilon	value of the bandwidth parameter epsilon

6.5.3 Member Function Documentation

6.5.3.1 build_kernel()

```
void LocallyStationaryModels::Kernel::build_kernel (
            const cd::matrixptr & data,
            const cd::matrixptr & anchorpoints )
```

build the "star" version of the kernel that contains the standardized kernel weights in such a way that each row sums to one

Parameters

data	a shared pointer to the matrix with the coordinates of the original dataset
anchorpoints	a shared pointer to the matrix with the coordinates of the anchor points

6.5.3.2 build_simple_kernel() [1/2]

```
void LocallyStationaryModels::Kernel::build_simple_kernel (
            const cd::matrixptr & coordinates )
```

build the "standard" version of the kernel needed for smoothing

Parameters

coordinates	a shared pointer to the matrix with the coordinates

6.5.3.3 build_simple_kernel() [2/2]

```
void LocallyStationaryModels::Kernel::build_simple_kernel (
            const cd::matrixptr & coordinates,
            const double & epsilon )
```

build the "standard" version of the kernel needed for smoothing

Parameters

coordinates	a shared pointer to the matrix with the coordinates
epsilon	replace the old epsilon with a new value

6.5.3.4 get_kernel()

```
const matrixptr LocallyStationaryModels::Kernel::get_kernel ( ) const
```

Returns

a shared pointer to the matrix pointed by m_k

6.5.3.5 operator()()

```
double LocallyStationaryModels::Kernel::operator() (
            const cd::vector & x,
            const cd::vector & y ) const
```

Returns

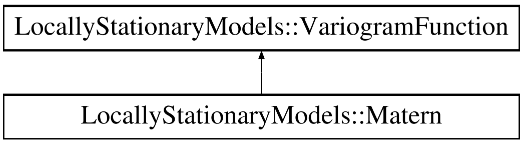
m_f(x ,y) where m_f is the kernel function

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

- kernel.hpp
- kernel.cpp

6.6 LocallyStationaryModels::Matern Class Reference

Inheritance diagram for LocallyStationaryModels::Matern:


LocallyStationaryModels::VariogramFunction
LocallyStationaryModels::Matern

Public Member Functions

- double operator() (const cd::vector &params, const double &x, const double &y) override

Additional Inherited Members

6.6.1 Member Function Documentation

6.6.1.1 operator()()

```
double LocallyStationaryModels::Matern::operator() (
            const cd::vector & params,
            const double & x,
            const double & y )  [override], [virtual]
```

Returns

sigma ∗ sigma ∗(1 - std::pow(std::sqrt(2∗nu)∗h, nu)∗std::cyl_bessel_k(nu, std::sqrt(2∗nu)∗h)/(std↩
::tgamma(nu)∗std::pow(2,nu-1)))

Parameters

params	a vector with lambda1, lambda2, phi, sigma and nu in this exact order

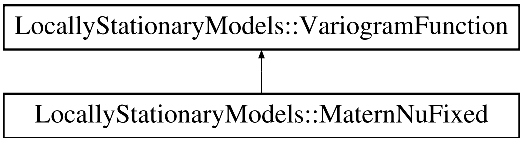
Implements LocallyStationaryModels::VariogramFunction.

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

- variogramfunctions.hpp
- variogramfunctions.cpp

6.7 LocallyStationaryModels::MaternNuFixed Class Reference

Inheritance diagram for LocallyStationaryModels::MaternNuFixed:


LocallyStationaryModels::VariogramFunction
LocallyStationaryModels::MaternNuFixed

Public Member Functions

- **MaternNuFixed** (const double &nu)
- double operator() (const cd::vector &params, const double &x, const double &y) override

Additional Inherited Members

6.7.1 Member Function Documentation

6.7.1.1 operator()()

```
double LocallyStationaryModels::MaternNuFixed::operator() (
             const cd::vector & params,
             const double & x,
             const double & y )  [override], [virtual]
```

Returns

sigma ∗ sigma ∗(1 - std::pow(std::sqrt(2∗nu)∗h, nu)∗std::cyl_bessel_k(nu, std::sqrt(2∗nu)∗h)/(std::tgamma(nu)∗std::pow(2,nu-1)))

Parameters

params	a vector with lambda1, lambda2, phi and sigma in this exact order

Implements LocallyStationaryModels::VariogramFunction.

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

- variogramfunctions.hpp
- variogramfunctions.cpp

6.8 LocallyStationaryModels::Opt Class Reference

a class to estimate the value of the parameters of the variogram in each point by optimizing the correspondent funzionedaottimizzare relying on the library LBFGSpp

```
#include <variogramfit.hpp>
```

Public Member Functions

- Opt (const cd::matrixptr &empiricvariogram, const cd::matrixptr &squaredweights, const cd::vectorptr &mean_x, const cd::vectorptr &mean_y, const std::string &id, const cd::vector &initialparameters, const cd::vector &lowerbound, const cd::vector &upperbound)

  constructor
- void **findallsolutions** ()

  find the optimal solution in all the position
- cd::matrixptr get_solutions () const

6.8.1 Detailed Description

a class to estimate the value of the parameters of the variogram in each point by optimizing the correspondent funzionedaottimizzare relying on the library LBFGSpp

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Opt()

```
LocallyStationaryModels::Opt::Opt (
             const cd::matrixptr & empiricvariogram,
             const cd::matrixptr & squaredweights,
             const cd::vectorptr & mean_x,
             const cd::vectorptr & mean_y,
             const std::string & id,
             const cd::vector & initialparameters,
             const cd::vector & lowerbound,
             const cd::vector & upperbound )
```

constructor

Parameters

empiricvariogram	a shared pointer to the empiric variogram
squaredweights	a shared pointer to the squared weights
mean_x	a shared pointer to the vector of the abscissas of the centers
mean_y	a shared pointer to the vector of the ordinates of the centers
id	the name of the variogram of your choice
initialparameters	the initial value of the parameters required from the optimizer to start the search for a minimum
lowerbound	the lower bounds for the parameters in the nonlinear optimization problem
upperbound	the upper bounds for the parameters in the nonlinear optimization problem

6.8.3 Member Function Documentation

6.8.3.1 get_solutions()

```
cd::matrixptr LocallyStationaryModels::Opt::get_solutions ( ) const
```

Returns

the solutions found by solving the problem of nonlinear optimization

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

- variogramfit.hpp
- variogramfit.cpp

6.9 LocallyStationaryModels::Predictor Class Reference

class to perform kriging on the data

`#include <kriging.hpp>`

Public Member Functions

- Predictor (const std::string &id, const cd::vectorptr &z, const Smt &mysmt, const double &b, const cd::matrixptr &data)

  constructor
- **Predictor** ()

  gammaiso set by default to exponential
- template<typename Input , typename Output >
  Output **predict_mean** (const Input &pos) const

  predict the mean
- template<typename Input , typename Output >
  Output **predict_z** (const Input &pos) const

  predict Z
- template<> double **predict_mean** (const size_t &pos) const

6.9.1 Detailed Description

class to perform kriging on the data

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Predictor()

```
LocallyStationaryModels::Predictor::Predictor (
             const std::string & id,
             const cd::vectorptr & z,
             const Smt & mysmt,
             const double & b,
             const cd::matrixptr & data )
```

constructor

Parameters

id	name of the variogram function associated with the problem
z	the vector with the value of the function Y in the known points
mysmt	the one used to previously smooth the variogram
b	the radius of the neighbourhood of the point where to perform kriging
data	a shared pointer to the matrix with the coordinates of the original dataset

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

- kriging.hpp
- kriging.cpp

6.10 LocallyStationaryModels::SampleVar Class Reference

a class to build and store the empiric variogram in all the anchor points

```
#include <samplevar.hpp>
```

Public Member Functions

- SampleVar (const std::string &kernel_id, const size_t &n_angles, const size_t &n_intervals, const double &epsilon)

  constructor
- **SampleVar** ()

  a default constructor for the class which calls the default constructors for both the kernel and the grid
- void build_samplevar (const cd::matrixptr &data, const cd::matrixptr &anchorpoints, const cd::vectorptr &z)

  build the matrix of the empiric variogram
- const cd::matrixptr get_variogram () const
- const cd::matrixptr get_denominators () const
- const cd::matrixptr get_squaredweights () const
- const cd::vectorptr get_x () const
- const cd::vectorptr get_y () const
- const cd::matrixptr get_kernel () const
- const cd::matrixIptr get_grid () const
- const cd::vectorptr get_normh () const

6.10.1 Detailed Description

a class to build and store the empiric variogram in all the anchor points

6.10.2 Constructor & Destructor Documentation

6.10.2.1 SampleVar()

```
LocallyStationaryModels::SampleVar::SampleVar (
            const std::string & kernel_id,
            const size_t & n_angles,
            const size_t & n_intervals,
            const double & epsilon )
```

constructor

Parameters

kernel_id	the name of the function you want to use for the kernel
n_angles	the number of angles to be passed to the grid
n_intervals	the number of inervals to be passed to the grid
epsilon	the bandwidth parameter regulating the kernel

6.10.3 Member Function Documentation

6.10.3.1 build_samplevar()

```
void LocallyStationaryModels::SampleVar::build_samplevar (
             const cd::matrixptr & data,
             const cd::matrixptr & anchorpoints,
             const cd::vectorptr & z )
```

build the matrix of the empiric variogram

Parameters

data	a shared pointer to the matrix of the coordinates of the original dataset
anchorpoints	a shared pointer to the matrix of the coordinates of the anchor poitns
z	a shared pointer to the vector of the value of Z

6.10.3.2 get_denominators()

```
const matrixptr LocallyStationaryModels::SampleVar::get_denominators ( ) const
```

Returns

a shared pointer to the matrix of the denominators

6.10.3.3 get_grid()

```
const matrixIptr LocallyStationaryModels::SampleVar::get_grid ( ) const
```

Returns

m_grid.m_g

6.10.3.4 get_kernel()

```
const matrixptr LocallyStationaryModels::SampleVar::get_kernel ( ) const
```

Returns

m_kernel.m_k

6.10.3.5 get_normh()

```
const vectorptr LocallyStationaryModels::SampleVar::get_normh ( ) const
```

Returns

m_grid.m_normh

6.10.3.6 get_squaredweights()

```
const matrixptr LocallyStationaryModels::SampleVar::get_squaredweights ( ) const
```

Returns

a shared pointers to the squaredweigths required to evaluate the function to be optimized

6.10.3.7 get_variogram()

```
const matrixptr LocallyStationaryModels::SampleVar::get_variogram ( ) const
```

Returns

a shared pointer to the sample variogram

6.10.3.8 get_x()

```
const vectorptr LocallyStationaryModels::SampleVar::get_x ( ) const
```

Returns

m_grid.m_mean_x

6.10.3.9 get_y()

```
const vectorptr LocallyStationaryModels::SampleVar::get_y ( ) const
```

Returns

m_grid.m_mean_y

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

- samplevar.hpp
- samplevar.cpp

6.11 LocallyStationaryModels::Smt Class Reference

a class to perform kernel smoothing of the paramters estimated in the anchor points to get the non stationary value of the parameters in any position of the domain

```
#include <smooth.hpp>
```

Public Member Functions

- Smt (const cd::matrixptr &solutions, const cd::matrixptr &anchorpos, const double &min_delta, const double &max_delta, const std::string &kernel_id)

  constructor
- Smt (const cd::matrixptr &solutions, const cd::matrixptr &anchorpos, const double delta, const std::string &kernel_id)

  constructor
- **Smt** ()

  constructor. Call the default constructor for m_kernel
- template<class Input >
  cd::vector smooth_vector (const Input &pos) const

  smooth all the parameters for a point in position pos
- const cd::matrixptr get_solutions () const
- double get_optimal_delta () const
- const cd::matrixptr get_anchorpos () const

6.11.1 Detailed Description

a class to perform kernel smoothing of the paramters estimated in the anchor points to get the non stationary value of the parameters in any position of the domain

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Smt() [1/2]

```
LocallyStationaryModels::Smt::Smt (
              const cd::matrixptr & solutions,
              const cd::matrixptr & anchorpos,
              const double & min_delta,
              const double & max_delta,
              const std::string & kernel_id )
```

constructor

Parameters

solutions	a shared pointer to the solutions of the optimization
anchorpos	a vector containing the indeces of the anchor position obtained by clustering
d	a shared pointer to the matrix of the coordinates
min_delta	the minimum exponent for the cross-validation of the delta bandwidth parameter for gaussian kernel smoothing
max_delta	the maximum exponent for the cross-validation of the delta bandwidth parameter for gaussian kernel smoothing

6.11.2.2 Smt() [2/2]

```
LocallyStationaryModels::Smt::Smt (
            const cd::matrixptr & solutions,
            const cd::matrixptr & anchorpos,
            const double delta,
            const std::string & kernel_id )
```

constructor

Parameters

solutions	a shared pointer to the solutions of the optimization
anchorpos	a vector containing the indeces of the anchor position obtained by clustering
d	a shared pointer to the matrix of the coordinates
delta	a user-chosen value for delta

6.11.3 Member Function Documentation

6.11.3.1 get_anchorpos()

```
const cd::matrixptr LocallyStationaryModels::Smt::get_anchorpos ( ) const
```

Returns

a shared pointer the coordinates of the anchorpoints

6.11.3.2 get_optimal_delta()

```
double LocallyStationaryModels::Smt::get_optimal_delta ( ) const
```

Returns

the delta found by cross-validation evaluated on sigma, the same delta is used for all the parameters

6.11.3.3 get_solutions()

```
const cd::matrixptr LocallyStationaryModels::Smt::get_solutions ( ) const
```

Returns

a shared pointer to the solutions found by the optimizer

6.11.3.4 smooth_vector()

```
template<class Input >
cd::vector LocallyStationaryModels::Smt::smooth_vector (
             const Input & pos ) const  [inline]
```

smooth all the parameters for a point in position pos

Parameters

pos	a vector of coordinates or the index of the position of the point where to find the smoothed value of the parameters

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

- smooth.hpp
- smooth.cpp

6.12 LocallyStationaryModels::TargetFunction Struct Reference

functor to pass to the optimizer that contains the wls to be minimized

```
#include <variogramfit.hpp>
```

Public Member Functions

- TargetFunction (const cd::matrixptr &empiricvariogram, const cd::matrixptr &squaredweights, const cd::vectorptr &mean_x, const cd::vectorptr &mean_y, const size_t &x0, const std::string &id)

  constructor
- double operator() (const cd::vector &params, cd::vector &grad)
- double operator() (const cd::vector &params)

Public Attributes

- const cd::matrixptr **m_empiricvariogram**

  sample variogram matrix
- const cd::matrixptr **m_squaredweights**

  matrix of the squared weights
- const cd::vectorptr **m_mean_x**

  vector with the x of each cell of the grid (mean of the x of all the pairs inside)
- const cd::vectorptr **m_mean_y**

  vector with the y of each cell of the grid (mean of the y of all the pairs inside)
- size_t **m_x0**

  index of the position where to evaluate gammaisoptr
- std::shared_ptr< VariogramFunction > **m_gammaisoptr**

  pointer to the variogram function

6.12.1 Detailed Description

functor to pass to the optimizer that contains the wls to be minimized

6.12.2 Constructor & Destructor Documentation

6.12.2.1 TargetFunction()

```
LocallyStationaryModels::TargetFunction::TargetFunction (
             const cd::matrixptr & empiricvariogram,
             const cd::matrixptr & squaredweights,
             const cd::vectorptr & mean_x,
             const cd::vectorptr & mean_y,
             const size_t & x0,
             const std::string & id )
```

constructor

Parameters

empiricvariogram	a shared pointer to the empiric variogram
squaredweights	a shared pointer to the squared weights
mean_x	a shared pointer to the vector of the abscissas of the centers
mean_y	a shared pointer to the vector of the ordinates of the centers
x0	the index of the position x0
id	the name of the variogram of your choice

6.12.3 Member Function Documentation

6.12.3.1 operator()() [1/2]

```
double LocallyStationaryModels::TargetFunction::operator() (
            const cd::vector & params )
```

Parameters

params	a vector containing the previous value of the parameters of the function (lambda1, lambda2, phi, sigma, etc.)

6.12.3.2 operator()() [2/2]

```
double LocallyStationaryModels::TargetFunction::operator() (
            const cd::vector & params,
            cd::vector & grad )
```

Parameters

params	a vector containing the previous value of the parameters of the function (lambda1, lambda2, phi, sigma, etc.)
grad	a vector containing the previous value of the gradient which is updated at each iteration

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

- variogramfit.hpp
- variogramfit.cpp

6.13 LocallyStationaryModels::Tolerances Struct Reference

collects all the tolerances and the constants used inside the code

```
#include <tolerances.hpp>
```

Static Public Attributes

- static constexpr double **anchor_tolerance** = 1e-6

  value of the noise to be added to the anchor points grid to prevent out of domain points
- static double **pi** = 4 ∗ std::atan(1.)

  default value of pi
- static constexpr double **min_determinant** = 1e-12

  minimum threshold below which the determinant of a matrix is considered to be 0
- static constexpr double **param_epsilon** = 1e-6

  optimization termination condition parameter epsilon
- static constexpr double **param_max_iterations** = 1000000

  optimization termination condition parameter max_iterations

- static constexpr double **min_norm** = 1e-12

  minimun threshold below which the norm of a vector is considered to be 0
- static constexpr double **infinity** = 1e12

  huge value to be considered as infinite when returning inf would cause troubles
- static constexpr double **n_deltas** = 1000

  number of delta between min_delta and max_delta to perform cross-validation
- static constexpr double **gradient_step** = 10e-8

  step for the numerical computation of the gradient

6.13.1 Detailed Description

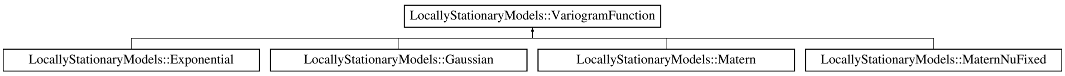
collects all the tolerances and the constants used inside the code

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

- tolerances.hpp

6.14 LocallyStationaryModels::VariogramFunction Class Reference

Inheritance diagram for LocallyStationaryModels::VariogramFunction:


LocallyStationaryModels::VariogramFunction
LocallyStationaryModels::Exponential
LocallyStationaryModels::Gaussian
LocallyStationaryModels::Matern
LocallyStationaryModels::MaternNuFixed

Public Member Functions

- virtual double operator() (const cd::vector &params, const double &x, const double &y)=0

  return f(params, x, y)

Protected Member Functions

- double **compute_anisotropic_h** (const double &lambda1, const double &lambda2, const double &phi, const double &x, const double &y)

  convert the isotropic variogram in the equivalent anisotropic one calculating the norm of the spatial lag rotated and expanded according to the eigenvalues and eigenvector of the anisotropy matrix

6.14.1 Member Function Documentation

6.14.1.1 operator()()

```
virtual double LocallyStationaryModels::VariogramFunction::operator() (
            const cd::vector & params,
            const double & x,
            const double & y ) [pure virtual]
```

return f(params, x, y)

Implemented in LocallyStationaryModels::Exponential, LocallyStationaryModels::Matern, LocallyStationaryModels::MaternNuFixed, and LocallyStationaryModels::Gaussian.

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

- variogramfunctions.hpp
- variogramfunctions.cpp

Chapter 7

File Documentation

7.1 anchor.hpp

```
// Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
// Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>

#ifndef LOCALLY_STATIONARY_MODELS_ANCHOR
#define LOCALLY_STATIONARY_MODELS_ANCHOR

#include "traits.hpp"

namespace LocallyStationaryModels {
class Anchor {
private:
    cd::matrixptr m_data;
    double m_n_pieces;
    double m_width = 0;
    double m_height = 0;
    double m_piece_width = 0;
    double m_piece_height = 0;
    double m_origin_x = 0;
    double m_origin_y = 0;

    Eigen::VectorXi find_indeces()
    {
        size_t n = m_data->rows();

        m_origin_x = (m_data->col(0)).minCoeff() * (1 - Tolerances::anchor_tolerance);
        m_origin_y = (m_data->col(1)).minCoeff() * (1 - Tolerances::anchor_tolerance);

        m_width = (m_data->col(0)).maxCoeff() * (1 + Tolerances::anchor_tolerance) - m_origin_x;
        m_height = (m_data->col(1)).maxCoeff() * (1 + Tolerances::anchor_tolerance) - m_origin_y;
        m_piece_width = m_width / m_n_pieces;
        m_piece_height = m_height / m_n_pieces;

        // fill a vector with the position of each point
        Eigen::VectorXi result(n);
        for (size_t i = 0; i < n; ++i) {
            cd::vector coordinates = m_data->row(i);
            result(i) = ceil((coordinates(0) - m_origin_x) / m_piece_width)
                + m_n_pieces * floor((coordinates(1) - m_origin_y) / m_piece_height);
        }
        return result;
    }

public:
    Anchor(const cd::matrixptr& data, const double& n_pieces)
        : m_data(data)
        , m_n_pieces(n_pieces) {};

    const cd::matrix find_anchorpoints()
    {
        size_t n = m_data->rows();
        Eigen::VectorXi indeces = find_indeces();

        // build a new vector without duplicates
        std::vector<size_t> positions;
        for (size_t i = 0; i < n; ++i) {
            size_t pos = indeces(i);
            if (std::find(positions.begin(), positions.end(), pos) == positions.end())
                positions.push_back(pos);
        }

        // fill a new matrix with the coordinates of each anchorpoins
        cd::matrix anchorpos(positions.size(), m_data->cols());
        for (size_t i = 0; i < anchorpos.rows(); ++i) {
            size_t I = positions[i];
            anchorpos(i, 0) = m_origin_x
                + (I - floor((I * (1 - Tolerances::anchor_tolerance)) / m_n_pieces) * m_n_pieces) * m_piece_width
                - m_piece_width / 2;
            anchorpos(i, 1) = m_origin_y + ceil((I * (1 - Tolerances::anchor_tolerance)) / m_n_pieces) * m_piece_height
                - m_piece_height / 2;
        }
        return anchorpos;
    }

    std::pair<double, double> get_origin() const { return std::make_pair(m_origin_x, m_origin_y); }
    std::pair<double, double> get_tiles_dimensions() const { return std::make_pair(m_piece_width, m_piece_height); }
}; // class Anchor
} // namespace LocallyStationaryModels

#endif // LOCALLY_STATIONARY_MODELS_ANCHOR
```

7.2 grid.hpp

```
// Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
// Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>

#ifndef LOCALLY_STATIONARY_MODELS_GRID
#define LOCALLY_STATIONARY_MODELS_GRID

#include "traits.hpp"

namespace LocallyStationaryModels {
class Grid {
private:
    cd::gridfunction m_f;
    cd::matrixIptr m_g = std::make_shared<cd::matrixI>(0, 0);
    cd::vectorptr m_normh
        = nullptr;
    cd::vectorptr m_mean_x
        = nullptr;
    cd::vectorptr m_mean_y
        = nullptr;
    double m_epsilon;

    void build_normh(const cd::matrixptr& data);

public:
    Grid(const std::string& id, const double& epsilon);

    Grid();

    void build_grid(const cd::matrixptr& data, const size_t& n_angles, const size_t& n_intervals);

    const cd::matrixIptr get_grid() const;
    const cd::vectorptr get_normh() const;
    const cd::vectorptr get_x() const;
    const cd::vectorptr get_y() const;
}; // class Grid
} // namespace LocallyStationaryModels

#endif // LOCALLY_STATIONARY_MODELS_GRID
```

7.3 gridfunctions.hpp

```
// Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
// Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>

#ifndef LOCALLY_STATIONARY_MODELS_GRID_FUNCTIONS
#define LOCALLY_STATIONARY_MODELS_GRID_FUNCTIONS

#include "traits.hpp"

namespace LocallyStationaryModels {
namespace gf {
    cd::matrixIptr pizza(
        const cd::matrixptr& data, const size_t& n_angles, const size_t& n_intervals, const double&
    epsilon);

    cd::gridfunction make_grid(const std::string& id);
} // namespace gf
} // namespace LocallyStationaryModels

#endif // LOCALLY_STATIONARY_MODELS_GRID_FUNCTIONS
```

7.4 kernel.hpp

```
// Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
// Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>

#ifndef LOCALLY_STATIONARY_MODELS_KERNEL
#define LOCALLY_STATIONARY_MODELS_KERNEL

#include "traits.hpp"

namespace LocallyStationaryModels {
class Kernel {
private:
    double m_epsilon;
    cd::kernelfunction m_f;
    cd::matrixptr m_k = std::make_shared<cd::matrix>(0, 0);

public:
    Kernel(const std::string& id, const double& epsilon);

    Kernel();

    double operator()(const cd::vector& x, const cd::vector& y) const;

    void build_kernel(const cd::matrixptr& data, const cd::matrixptr& anchorpoints);

    void build_simple_kernel(const cd::matrixptr& coordinates);

    void build_simple_kernel(const cd::matrixptr& coordinates, const double& epsilon);

    const cd::matrixptr get_kernel() const;
}; // class Kernel
} // namespace LocallyStationaryModels

#endif // LOCALLY_STATIONARY_MODELS_KERNEL
```

7.5 kernelfunctions.hpp

```
// Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
// Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>

#ifndef LOCALLY_STATIONARY_MODELS_KERNEL_FUNCTIONS
#define LOCALLY_STATIONARY_MODELS_KERNEL_FUNCTIONS

#include "traits.hpp"

namespace LocallyStationaryModels {
namespace kf {
    double gaussian(const cd::vector& x, const cd::vector& y, const double& epsilon);

    double identity(const cd::vector& x, const cd::vector& y, const double& epsilon);

    cd::kernelfunction make_kernel(const std::string& id);
} // namespace kf
} // namespace LocallyStationaryModels

#endif // LOCALLY_STATIONARY_MODELS_KERNEL_FUNCTION
```

7.6 kriging.hpp

```
// Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
// Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>

#ifndef LOCALLY_STATIONARY_MODELS_KRIGING
#define LOCALLY_STATIONARY_MODELS_KRIGING

```

```
#include "smooth.hpp"
#include "traits.hpp"
#include "variogramfit.hpp"

namespace LocallyStationaryModels {
class Predictor {
private:
    std::shared_ptr<VariogramFunction> m_gammaisoptr;
    cd::vectorptr m_z = nullptr;
    Smt m_smt;
    double m_b;
    cd::vectorptr m_means = nullptr;
    cd::matrixptr m_data = nullptr;

    cd::vectorind build_neighbourhood(const cd::vector& pos) const;
    cd::vectorind build_neighbourhood(const size_t& pos) const;

    cd::vector build_eta(cd::vector& params, cd::vectorind& neighbourhood) const;

    std::pair<cd::vector, double> build_etakriging(const cd::vector& params, const cd::vector& pos)
    const;

public:
    Predictor(
        const std::string& id, const cd::vectorptr& z, const Smt& mysmt, const double& b, const
    cd::matrixptr& data);
    Predictor();

    template <typename Input, typename Output> Output predict_mean(const Input& pos) const;

    template <typename Input, typename Output> Output predict_z(const Input& pos) const;
}; // class Predictor
} // namespace LocallyStationaryModels

#endif // LOCALLY_STATIONARY_MODELS_KRIGING
```

7.7 samplevar.hpp

```
// Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
// Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>

#ifndef LOCALLY_STATIONARY_MODELS_SAMPLEVAR
#define LOCALLY_STATIONARY_MODELS_SAMPLEVAR

#include "grid.hpp"
#include "kernel.hpp"
#include "traits.hpp"

namespace LocallyStationaryModels {
class SampleVar {
private:
    cd::matrixptr m_variogram = nullptr;
    cd::matrixptr m_denominators = nullptr;
    cd::matrixptr m_squaredweights = nullptr;
    Kernel m_kernel;
    Grid m_grid;
    size_t m_n_angles;
    size_t m_n_intervals;

    void build_squaredweights();

public:
    SampleVar(const std::string& kernel_id, const size_t& n_angles, const size_t& n_intervals, const
    double& epsilon);

    SampleVar();

    void build_samplevar(const cd::matrixptr& data, const cd::matrixptr& anchorpoints, const
    cd::vectorptr& z);

    const cd::matrixptr get_variogram() const;
    const cd::matrixptr get_denominators() const;
    const cd::matrixptr get_squaredweights() const;
    const cd::vectorptr get_x() const;
    const cd::vectorptr get_y() const;
    const cd::matrixptr get_kernel() const;
    const cd::matrixIptr get_grid() const;
    const cd::vectorptr get_normh() const;
}; // class SampleVar
} // namespace LocallyStationaryModels

#endif // LOCALLY_STATIONARY_MODELS_SAMPLEVAR
```

7.8 smooth.hpp

```
// Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
// Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>

#ifndef LOCALLY_STATIONARY_MODELS_SMOOTH
#define LOCALLY_STATIONARY_MODELS_SMOOTH

#include "kernel.hpp"
#include "traits.hpp"

namespace LocallyStationaryModels {
class Smt {
private:
    cd::matrixptr m_solutions = nullptr;
    cd::matrixptr m_anchorpos = nullptr;

    Kernel m_kernel;

    double m_optimal_delta = 0;

    double smooth_value(const size_t& pos, const size_t& n) const;

    double smooth_value(const cd::vector& pos, const size_t& n) const;

public:
    Smt(const cd::matrixptr& solutions, const cd::matrixptr& anchorpos, const double& min_delta,
        const double& max_delta, const std::string& kernel_id);
    Smt(const cd::matrixptr& solutions, const cd::matrixptr& anchorpos, const double delta,
        const std::string& kernel_id);
    Smt();

    template <class Input> cd::vector smooth_vector(const Input& pos) const
    {
        cd::vector result(m_solutions->cols());
        for (size_t i = 0; i < m_solutions->cols(); ++i) {
            result(i) = smooth_value(pos, i);
        }
        return result;
    };

    const cd::matrixptr get_solutions() const;
    double get_optimal_delta() const;
    const cd::matrixptr get_anchorpos() const;
}; // class Smt
} // namespace LocallyStationaryModels

#endif // LOCALLY_STATIONARY_MODELS_SMOOTH
```

7.9 tolerances.hpp

```
// Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
// Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>

#ifndef LOCALLY_STATIONARY_MODELS_TOLERANCES
#define LOCALLY_STATIONARY_MODELS_TOLERANCES

namespace LocallyStationaryModels {
struct Tolerances {
    static constexpr double anchor_tolerance = 1e-6;
    inline static double pi = 4 * std::atan(1.);
    static constexpr double min_determinant = 1e-12;
    static constexpr double param_epsilon = 1e-6;
    static constexpr double param_max_iterations = 1000000;
    static constexpr double min_norm = 1e-12;
    static constexpr double infinity = 1e12;
    static constexpr double n_deltas = 1000;
    static constexpr double gradient_step = 10e-8;
}; // struct Tolerances
} // namespace LocallyStationaryModels

#endif // LOCALLY_STATIONARY_MODELS_TOLERANCES
```

7.10 traits.hpp

```
// Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
// Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>

#ifndef LOCALLY_STATIONARY_MODELS_TRAITS
```

```
#define LOCALLY_STATIONARY_MODELS_TRAITS

#include <algorithm>
#include <cfloat>
#include <cmath>
#include <functional>
#include <iostream>
#include <memory>
#include <omp.h>
#include <string>
#include <vector>

#include "Eigen/Dense"
#include "tolerances.hpp"

namespace LocallyStationaryModels {
namespace cd {
    // defining basic types
    using vector = Eigen::VectorXd;
    using matrix = Eigen::MatrixXd;
    using matrixI = Eigen::MatrixXi;
    using vectorptr = std::shared_ptr<vector>;
    using matrixptr = std::shared_ptr<matrix>;
    using matrixIptr = std::shared_ptr<matrixI>;
    using vectorind = std::vector<size_t>;

    // defining function types
    using kernelfunction = std::function<double(const vector&, const vector&, const double&)>;
    using gridfunction = std::function<matrixIptr(const matrixptr&, const size_t&, const size_t&, const double&)>;

} // namespace cd
} // namespace LocallyStationaryModels

#endif // LOCALLY_STATIONARY_MODELS_TRAITS
```

7.11 variogramfit.hpp

```
// Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
// Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>

#ifndef LOCALLY_STATIONARY_MODELS_GRADIENT
#define LOCALLY_STATIONARY_MODELS_GRADIENT

#include "LBFGS/LBFGSB.h"
#include "traits.hpp"
#include "variogramfunctions.hpp"

namespace LocallyStationaryModels {
struct TargetFunction {
    const cd::matrixptr m_empiricvariogram;
    const cd::matrixptr m_squaredweights;
    const cd::vectorptr
        m_mean_x;
    const cd::vectorptr
        m_mean_y;
    size_t m_x0;
    std::shared_ptr<VariogramFunction> m_gammaisoptr;

    TargetFunction(const cd::matrixptr& empiricvariogram, const cd::matrixptr& squaredweights,
        const cd::vectorptr& mean_x, const cd::vectorptr& mean_y, const size_t& x0, const std::string& id);

    double operator()(const cd::vector& params, cd::vector& grad);

    double operator()(const cd::vector& params);
}; // struct TargetFunction

class Opt {
private:
    cd::matrixptr m_empiricvariogram;
    cd::matrixptr m_squaredweights;
    cd::vectorptr m_mean_x;
    cd::vectorptr m_mean_y;
    std::string m_id;
    cd::vector m_initialparameters;
    cd::vector m_lowerbound;
    cd::vector m_upperbound;
    cd::matrixptr m_solutions = nullptr;

    cd::vector findonesolution(const size_t& pos) const;

public:
    Opt(const cd::matrixptr& empiricvariogram, const cd::matrixptr& squaredweights, const cd::vectorptr& mean_x,
        const cd::vectorptr& mean_y, const std::string& id, const cd::vector& initialparameters,
        const cd::vector& lowerbound, const cd::vector& upperbound);

    void findallsolutions();

    cd::matrixptr get_solutions() const;
}; // class Opt
} // namespace LocallyStationaryModels

#endif // LOCALLY_STATIONARY_MODELS_GRADIENT
```

7.12 variogramfunctions.hpp

```
// Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
// Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>

#ifndef LOCALLY_STATIONARY_MODES_VARIOGRAMFUNCTIONS
#define LOCALLY_STATIONARY_MODES_VARIOGRAMFUNCTIONS

#include "traits.hpp"

namespace LocallyStationaryModels {
class VariogramFunction {
protected:
    double compute_anisotropic_h(
        const double& lambda1, const double& lambda2, const double& phi, const double& x, const double& y);

public:
    VariogramFunction() = default;
    virtual double operator()(const cd::vector& params, const double& x, const double& y) = 0;
}; // class VariogramFunction

class Exponential : public VariogramFunction {
public:
    Exponential() = default;
    double operator()(const cd::vector& params, const double& x, const double& y) override;
}; // class Exponential

class Matern : public VariogramFunction {
public:
    Matern() = default;
    double operator()(const cd::vector& params, const double& x, const double& y) override;
}; // class Matern

class MaternNuFixed : public VariogramFunction {
private:
    double m_nu = 0.5;
public:
    MaternNuFixed(const double& nu)
        : m_nu(nu) {};
    double operator()(const cd::vector& params, const double& x, const double& y) override;
}; // class MaternNuFixed

class Gaussian : public VariogramFunction {
public:
    Gaussian() = default;
    double operator()(const cd::vector& params, const double& x, const double& y) override;
}; // class Gaussian

std::shared_ptr<VariogramFunction> make_variogramiso(const std::string& id);
} // namespace LocallyStationaryModels

#endif // LOCALLY_STATIONARY_MODES_VARIOGRAM_FUNCTIONS
```

Index