

## My Project

Generated by Doxygen 1.9.2



<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Namespace Documentation</b>	<b>9</b>
5.1 LocallyStationaryModels::cd Namespace Reference	9
5.1.1 Detailed Description	9
5.2 LocallyStationaryModels::gf Namespace Reference	9
5.2.1 Detailed Description	10
5.2.2 Function Documentation	10
5.2.2.1 make_grid()	10
5.2.2.2 pizza()	10
5.3 LocallyStationaryModels::kf Namespace Reference	10
5.3.1 Detailed Description	11
5.3.2 Function Documentation	11
5.3.2.1 gaussian()	11
5.3.2.2 identity()	11
5.3.2.3 make_kernel()	11
<b>6 Class Documentation</b>	<b>13</b>
6.1 LocallyStationaryModels::Anchor Class Reference	13
6.1.1 Detailed Description	13
6.1.2 Constructor & Destructor Documentation	13
6.1.2.1 Anchor()	13
6.1.3 Member Function Documentation	14
6.1.3.1 get_origin()	14
6.1.3.2 get_tiles_dimensions()	14
6.2 LocallyStationaryModels::Exponential Class Reference	14
6.2.1 Member Function Documentation	15
6.2.1.1 operator()()	15
6.3 LocallyStationaryModels::Gaussian Class Reference	15
6.3.1 Member Function Documentation	15
6.3.1.1 operator()()	16
6.4 LocallyStationaryModels::Grid Class Reference	16
6.4.1 Detailed Description	16
6.4.2 Constructor & Destructor Documentation	17

6.4.2.1 Grid()	17
6.4.3 Member Function Documentation	17
6.4.3.1 build_grid()	17
6.4.3.2 get_grid()	17
6.4.3.3 get_normh()	18
6.4.3.4 get_x()	18
6.4.3.5 get_y()	18
6.5 LocallyStationaryModels::Kernel Class Reference	18
6.5.1 Detailed Description	19
6.5.2 Constructor & Destructor Documentation	19
6.5.2.1 Kernel()	19
6.5.3 Member Function Documentation	19
6.5.3.1 build_kernel()	19
6.5.3.2 build_simple_kernel() [1/2]	19
6.5.3.3 build_simple_kernel() [2/2]	20
6.5.3.4 get_kernel()	20
6.5.3.5 operator()()	20
6.6 LocallyStationaryModels::Matern Class Reference	21
6.6.1 Member Function Documentation	21
6.6.1.1 operator()()	21
6.7 LocallyStationaryModels::MaternNuFixed Class Reference	21
6.7.1 Member Function Documentation	22
6.7.1.1 operator()()	22
6.8 LocallyStationaryModels::Opt Class Reference	22
6.8.1 Detailed Description	23
6.8.2 Constructor & Destructor Documentation	23
6.8.2.1 Opt()	23
6.8.3 Member Function Documentation	23
6.8.3.1 get_solutions()	24
6.9 LocallyStationaryModels::Predictor Class Reference	24
6.9.1 Detailed Description	24
6.9.2 Constructor & Destructor Documentation	24
6.9.2.1 Predictor()	24
6.10 LocallyStationaryModels::SampleVar Class Reference	25
6.10.1 Detailed Description	25
6.10.2 Constructor & Destructor Documentation	25
6.10.2.1 SampleVar()	25
6.10.3 Member Function Documentation	26
6.10.3.1 build_samplevar()	26
6.10.3.2 get_denominators()	26
6.10.3.3 get_grid()	26
6.10.3.4 get_kernel()	27

6.10.3.5 get_normh()	27
6.10.3.6 get_squaredweights()	27
6.10.3.7 get_variogram()	27
6.10.3.8 get_x()	27
6.10.3.9 get_y()	28
6.11 LocallyStationaryModels::Smt Class Reference	28
6.11.1 Detailed Description	28
6.11.2 Constructor & Destructor Documentation	28
6.11.2.1 Smt() [1/2]	28
6.11.2.2 Smt() [2/2]	29
6.11.3 Member Function Documentation	29
6.11.3.1 get_anchorpos()	29
6.11.3.2 get_optimal_delta()	29
6.11.3.3 get_solutions()	30
6.11.3.4 smooth_vector()	30
6.12 LocallyStationaryModels::TargetFunction Struct Reference	30
6.12.1 Detailed Description	31
6.12.2 Constructor & Destructor Documentation	31
6.12.2.1 TargetFunction()	31
6.12.3 Member Function Documentation	31
6.12.3.1 operator>() [1/2]	32
6.12.3.2 operator>() [2/2]	32
6.13 LocallyStationaryModels::Tolerances Struct Reference	32
6.13.1 Detailed Description	33
6.14 LocallyStationaryModels::VariogramFunction Class Reference	33
6.14.1 Member Function Documentation	33
6.14.1.1 operator>()	34
<b>7 File Documentation</b>	<b>35</b>
7.1 anchor.hpp	35
7.2 grid.hpp	36
7.3 gridfunctions.hpp	36
7.4 kernel.hpp	37
7.5 kernelfunctions.hpp	37
7.6 kriging.hpp	37
7.7 samplevar.hpp	38
7.8 smooth.hpp	39
7.9 tolerances.hpp	39
7.10 traits.hpp	39
7.11 variogramfit.hpp	40
7.12 variogramfunctions.hpp	41
<b>Index</b>	<b>43</b>



# Chapter 1

## Namespace Index

### 1.1 Namespace List

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

<a href="#">LocallyStationaryModels::cd</a>	Collects all the typedef used inside the code . . . . .	9
<a href="#">LocallyStationaryModels::gf</a>	Collect the grid functions and the corresponding factory function . . . . .	9
<a href="#">LocallyStationaryModels::kf</a>	Collect the kernel functions and the corresponding factory function . . . . .	10





## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

LocallyStationaryModels::Anchor . . . . .	13
LocallyStationaryModels::Grid . . . . .	16
LocallyStationaryModels::Kernel . . . . .	18
LocallyStationaryModels::Opt . . . . .	22
LocallyStationaryModels::Predictor . . . . .	24
LocallyStationaryModels::SampleVar . . . . .	25
LocallyStationaryModels::Smt . . . . .	28
LocallyStationaryModels::TargetFunction . . . . .	30
LocallyStationaryModels::Tolerances . . . . .	32
LocallyStationaryModels::VariogramFunction . . . . .	33
LocallyStationaryModels::Exponential . . . . .	14
LocallyStationaryModels::Gaussian . . . . .	15
LocallyStationaryModels::Matern . . . . .	21
LocallyStationaryModels::MaternNuFixed . . . . .	21



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">LocallyStationaryModels::Anchor</a>	
Simple class to find the anchor points given the data . . . . .	13
<a href="#">LocallyStationaryModels::Exponential</a>	14
<a href="#">LocallyStationaryModels::Gaussian</a>	15
<a href="#">LocallyStationaryModels::Grid</a>	
Class to build the grid . . . . .	16
<a href="#">LocallyStationaryModels::Kernel</a>	
Class to compute the kernel matrix . . . . .	18
<a href="#">LocallyStationaryModels::Matern</a>	21
<a href="#">LocallyStationaryModels::MaternNuFixed</a>	21
<a href="#">LocallyStationaryModels::Opt</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 . . . . .	22
<a href="#">LocallyStationaryModels::Predictor</a>	
Class to perform kriging on the data . . . . .	24
<a href="#">LocallyStationaryModels::SampleVar</a>	
Class to build and store the empiric variogram in all the anchor points . . . . .	25
<a href="#">LocallyStationaryModels::Smt</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 . . . . .	28
<a href="#">LocallyStationaryModels::TargetFunction</a>	
Functor to pass to the optimizer that contains the wls to be minimized . . . . .	30
<a href="#">LocallyStationaryModels::Tolerances</a>	
Collects all the tolerances and the constants used inside the code . . . . .	32
<a href="#">LocallyStationaryModels::VariogramFunction</a>	33



## Chapter 4

# File Index

### 4.1 File List

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

<a href="#">anchor.hpp</a>	??
<a href="#">grid.hpp</a>	??
<a href="#">gridfunctions.hpp</a>	??
<a href="#">kernel.hpp</a>	??
<a href="#">kernelfunctions.hpp</a>	??
<a href="#">kriging.hpp</a>	??
<a href="#">samplevar.hpp</a>	??
<a href="#">smooth.hpp</a>	??
<a href="#">tolerances.hpp</a>	??
<a href="#">traits.hpp</a>	??
<a href="#">variogramfit.hpp</a>	??
<a href="#">variogramfunctions.hpp</a>	??



## 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 **matrixl** = Eigen::MatrixXi
- using **vectorptr** = std::shared\_ptr< vector >
- using **matrixptr** = std::shared\_ptr< matrix >
- using **matrixlptr** = std::shared\_ptr< matrixl >
- using **vectorind** = std::vector< size\_t >
- using **kernelfunction** = std::function< double(const vector &, const vector &, const double &)>
- using **gridfunction** = std::function< matrixlptr(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

- matrixlptr [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

<i>id</i>	name of the function of choice
-----------	--------------------------------

#### 5.2.2.2 pizza()

```
cd::matrixIpPtr LocallyStationaryModels::gf::pizza (
    const cd::matrixpPtr & 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

<i>data</i>	a shared pointer to the matrix of the coordinates
<i>n_angles</i>	number of slices of the pizza
<i>n_intervals</i>	number of the pieces for each slice of the pizza
<i>epsilon</i>	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

$e^{(-\text{norm}(x-y)^2/\epsilon^2)}$

#### 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**

<i>id</i>	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

<i>data</i>	shared pointer to the matrix with the coordinates of the dataset points
<i>n_pieces</i>	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

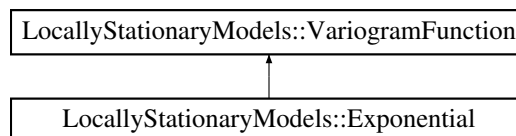
the dimensions (height and width) of each cell of the grid

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

- `anchor.hpp`

## 6.2 LocallyStationaryModels::Exponential Class Reference

Inheritance diagram for LocallyStationaryModels::Exponential:



### 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

<i>params</i>	a vector with lambda1, lambda2, phi and sigma in this exact order
---------------	---

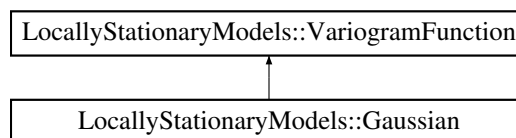
Implements [LocallyStationaryModels::VariogramFunction](#).

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:



### 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

<i>params</i>	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::matrixlptr [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

<i>id</i>	name of the grid function
<i>epsilon</i>	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

<i>data</i>	a shared pointer to the matrix of the coordinates
<i>n_angles</i>	number of slices of the pizza
<i>n_intervals</i>	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 constructor 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

<i>id</i>	name of the kernel function
<i>epsilon</i>	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

<i>data</i>	a shared pointer to the matrix with the coordinates of the original dataset
<i>anchorpoints</i>	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**

<i>coordinates</i>	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**

<i>coordinates</i>	a shared pointer to the matrix with the coordinates
<i>epsilon</i>	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**

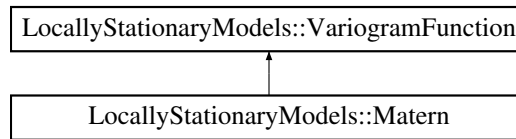
*m\_f*(*x*,*y*) where *m\_f* is the kernel function

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:



### 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

$\sigma * \sigma * (1 - \text{std::pow}(\text{std::sqrt}(2 * \nu) * h, \nu) * \text{std::cyl\_bessel\_k}(\nu, \text{std::sqrt}(2 * \nu) * h) / (\text{std::tgamma}(\nu) * \text{std::pow}(2, \nu - 1)))$

#### Parameters

<i>params</i>	a vector with lambda1, lambda2, phi, sigma and nu in this exact order
---------------	---

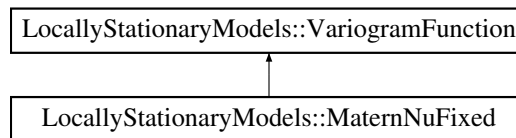
Implements [LocallyStationaryModels::VariogramFunction](#).

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:



## 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

$\sigma * \sigma * (1 - \text{std::pow}(\text{std::sqrt}(2 * \nu) * h, \nu) * \text{std::cyl\_bessel\_k}(\nu, \text{std::sqrt}(2 * \nu) * h) / (\text{std::gamma}(\nu) * \text{std::pow}(2, \nu - 1)))$

#### Parameters

<i>params</i>	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 functioned aottimizzare 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

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

<i>id</i>	name of the variogram function associated with the problem
<i>z</i>	the vector with the value of the function Y in the known points
<i>mysmt</i>	the one used to previously smooth the variogram
<i>b</i>	the radius of the neighbourhood of the point where to perform kriging
<i>data</i>	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::matrixlptr [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

<i>kernel_id</i>	the name of the function you want to use for the kernel
<i>n_angles</i>	the number of angles to be passed to the grid
<i>n_intervals</i>	the number of intervals to be passed to the grid
<i>epsilon</i>	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

<i>data</i>	a shared pointer to the matrix of the coordinates of the original dataset
<i>anchorpoints</i>	a shared pointer to the matrix of the coordinates of the anchor poitns
<i>z</i>	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 squaredweights 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

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

<i>solutions</i>	a shared pointer to the solutions of the optimization
<i>anchorpos</i>	a vector containing the indeces of the anchor position obtained by clustering
<i>d</i>	a shared pointer to the matrix of the coordinates
<i>delta</i>	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

<i>pos</i>	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 gammaisopt*
- std::shared\_ptr< [VariogramFunction](#) > **m\_gammaisopt**  
*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

<i>empiricvariogram</i>	a shared pointer to the empiric variogram
<i>squaredweights</i>	a shared pointer to the squared weights
<i>mean_x</i>	a shared pointer to the vector of the abscissas of the centers
<i>mean_y</i>	a shared pointer to the vector of the ordinates of the centers
<i>x0</i>	the index of the position x0
<i>id</i>	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

<i>params</i>	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

<i>params</i>	a vector containing the previous value of the parameters of the function (lambda1, lambda2, phi, sigma, etc.)
<i>grad</i>	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

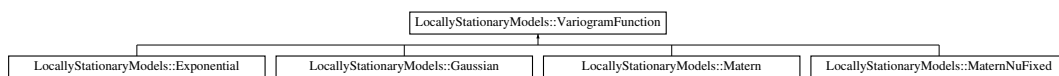
collects all the tolerances and the constants used inside the code

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

- tolerances.hpp

## 6.14 LocallyStationaryModels::VariogramFunction Class Reference

Inheritance diagram for LocallyStationaryModels::VariogramFunction:



### 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

```
1 // Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
2 // Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>
3
4 #ifndef LOCALLY_STATIONARY_MODELS_ANCHOR
5 #define LOCALLY_STATIONARY_MODELS_ANCHOR
6
7 #include "traits.hpp"
8
9 namespace LocallyStationaryModels {
10 class Anchor {
11 private:
12     cd::matrixptr m_data;
13     double m_n_pieces;
14     double m_width = 0;
15     double m_height = 0;
16     double m_piece_width = 0;
17     double m_piece_height = 0;
18     double m_origin_x = 0;
19     double m_origin_y = 0;
20
21     Eigen::VectorXi find_indeces()
22     {
23         size_t n = m_data->rows();
24
25         m_origin_x = (m_data->col(0)).minCoeff() * (1 - Tolerances::anchor_tolerance);
26         m_origin_y = (m_data->col(1)).minCoeff() * (1 - Tolerances::anchor_tolerance);
27
28         m_width = (m_data->col(0)).maxCoeff() * (1 + Tolerances::anchor_tolerance) - m_origin_x;
29         m_height = (m_data->col(1)).maxCoeff() * (1 + Tolerances::anchor_tolerance) - m_origin_y;
30         m_piece_width = m_width / m_n_pieces;
31         m_piece_height = m_height / m_n_pieces;
32
33         // fill a vector with the position of each point
34         Eigen::VectorXi result(n);
35         for (size_t i = 0; i < n; ++i) {
36             cd::vector coordinates = m_data->row(i);
37             result(i) = ceil((coordinates(0) - m_origin_x) / m_piece_width)
38                 + m_n_pieces * floor((coordinates(1) - m_origin_y) / m_piece_height);
39         }
40         return result;
41     }
42
43 public:
44     Anchor(const cd::matrixptr& data, const double& n_pieces)
45         : m_data(data)
46         , m_n_pieces(n_pieces) {}
47
48     const cd::matrix find_anchorpoints()
49     {
50         size_t n = m_data->rows();
51         Eigen::VectorXi indeces = find_indeces();
52
53         // build a new vector without duplicates
54         std::vector<size_t> positions;
55         for (size_t i = 0; i < n; ++i) {
56             size_t pos = indeces(i);
57             if (std::find(positions.begin(), positions.end(), pos) == positions.end())
58                 positions.push_back(pos);
59         }
60         return cd::matrix(m_data->rows(), positions.size(), m_data->col(0).data(), m_data->col(1).data());
61     }
62 }
```

```

74     }
75
76     // fill a new matrix with the coordinates of each anchorpoints
77     cd::matrix anchorpos(positions.size(), m_data->cols());
78     for (size_t i = 0; i < anchorpos.rows(); ++i) {
79         size_t I = positions[i];
80         anchorpos(i, 0) = m_origin_x
81             + (I - floor((I * (1 - Tolerances::anchor_tolerance)) / m_n_pieces) * m_n_pieces) *
82             m_piece_width
83             - m_piece_width / 2;
84         anchorpos(i, 1) = m_origin_y + ceil((I * (1 - Tolerances::anchor_tolerance)) / m_n_pieces) *
85             m_piece_height
86             - m_piece_height / 2;
87     }
88     return anchorpos;
89 }
90
91 std::pair<double, double> get_origin() const { return std::make_pair(m_origin_x, m_origin_y); }
92 std::pair<double, double> get_tiles_dimensions() const { return std::make_pair(m_piece_width,
93     m_piece_height); }
94 }; // class Anchor
95 } // namespace LocallyStationaryModels
96 #endif // LOCALLY_STATIONARY_MODELS_ANCHOR

```

## 7.2 grid.hpp

```

1 // Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
2 // Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>
3
4 #ifndef LOCALLY_STATIONARY_MODELS_GRID
5 #define LOCALLY_STATIONARY_MODELS_GRID
6
7 #include "traits.hpp"
8
9 namespace LocallyStationaryModels {
10 class Grid {
11 private:
12     cd::gridfunction m_f;
13     cd::matrixIptr m_g = std::make_shared<cd::matrixI>(0, 0);
14     cd::vectorptr m_normh
15         = nullptr;
16     cd::vectorptr m_mean_x
17         = nullptr;
18     cd::vectorptr m_mean_y
19         = nullptr;
20     double m_epsilon;
21
22     void build_normh(const cd::matrixptr& data);
23
24 public:
25     Grid(const std::string& id, const double& epsilon);
26
27     Grid();
28
29     void build_grid(const cd::matrixptr& data, const size_t& n_angles, const size_t& n_intervals);
30
31     const cd::matrixIptr get_grid() const;
32     const cd::vectorptr get_normh() const;
33     const cd::vectorptr get_x() const;
34     const cd::vectorptr get_y() const;
35 }; // class Grid
36 } // namespace LocallyStationaryModels
37 #endif // LOCALLY_STATIONARY_MODELS_GRID

```

## 7.3 gridfunctions.hpp

```

1 // Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
2 // Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>
3
4 #ifndef LOCALLY_STATIONARY_MODELS_GRID_FUNCTIONS
5 #define LOCALLY_STATIONARY_MODELS_GRID_FUNCTIONS
6
7 #include "traits.hpp"
8
9 namespace LocallyStationaryModels {
10 namespace gf {
11     cd::matrixIptr pizza(

```

```

24         const cd::matrixptr& data, const size_t& n_angles, const size_t& n_intervals, const double&
           epsilon);
25
30     cd::gridfunction make_grid(const std::string& id);
31 } // namespace gf
32 } // namespace LocallyStationaryModels
33
34 #endif // LOCALLY_STATIONARY_MODELS_GRID_FUNCTIONS

```

## 7.4 kernel.hpp

```

1 // Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
2 // Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>
3
4 #ifndef LOCALLY_STATIONARY_MODELS_KERNEL
5 #define LOCALLY_STATIONARY_MODELS_KERNEL
6
7 #include "traits.hpp"
8
9 namespace LocallyStationaryModels {
10 class Kernel {
11 private:
12     double m_epsilon;
13     cd::kernelfunction m_f;
14     cd::matrixptr m_k = std::make_shared<cd::matrix>(0, 0);
15
16 public:
17     Kernel(const std::string& id, const double& epsilon);
18
19     Kernel();
20
21     double operator()(const cd::vector& x, const cd::vector& y) const;
22
23     void build_kernel(const cd::matrixptr& data, const cd::matrixptr& anchorpoints);
24
25     void build_simple_kernel(const cd::matrixptr& coordinates);
26
27     void build_simple_kernel(const cd::matrixptr& coordinates, const double& epsilon);
28
29     const cd::matrixptr get_kernel() const;
30 }; // class Kernel
31 } // namespace LocallyStationaryModels
32
33 #endif // LOCALLY_STATIONARY_MODELS_KERNEL

```

## 7.5 kernelfunctions.hpp

```

1 // Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
2 // Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>
3
4 #ifndef LOCALLY_STATIONARY_MODELS_KERNEL_FUNCTIONS
5 #define LOCALLY_STATIONARY_MODELS_KERNEL_FUNCTIONS
6
7 #include "traits.hpp"
8
9 namespace LocallyStationaryModels {
10 namespace kf {
11     double gaussian(const cd::vector& x, const cd::vector& y, const double& epsilon);
12
13     double identity(const cd::vector& x, const cd::vector& y, const double& epsilon);
14
15     cd::kernelfunction make_kernel(const std::string& id);
16 } // namespace kf
17 } // namespace LocallyStationaryModels
18
19 #endif // LOCALLY_STATIONARY_MODELS_KERNEL_FUNCTION

```

## 7.6 kriging.hpp

```

1 // Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
2 // Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>
3
4 #ifndef LOCALLY_STATIONARY_MODELS_KRIGING
5 #define LOCALLY_STATIONARY_MODELS_KRIGING
6

```

```

7 #include "smooth.hpp"
8 #include "traits.hpp"
9 #include "variogramfit.hpp"
10
11 namespace LocallyStationaryModels {
12     class Predictor {
13     private:
14         std::shared_ptr<VariogramFunction> m_gammaisoptr;
15         cd::vectorptr m_z = nullptr;
16         Smt m_smt;
17         double m_b;
18         cd::vectorptr m_means = nullptr;
19         cd::matrixptr m_data = nullptr;
20
21         cd::vectorind build_neighbourhood(const cd::vector& pos) const;
22         cd::vectorind build_neighbourhood(const size_t& pos) const;
23
24         cd::vector build_eta(cd::vector& params, cd::vectorind& neighbourhood) const;
25
26         std::pair<cd::vector, double> build_etakriging(const cd::vector& params, const cd::vector& pos)
27             const;
28
29     public:
30         Predictor(
31             const std::string& id, const cd::vectorptr& z, const Smt& mysmt, const double& b, const
32             cd::matrixptr& data);
33         Predictor();
34
35         template <typename Input, typename Output> Output predict_mean(const Input& pos) const;
36
37         template <typename Input, typename Output> Output predict_z(const Input& pos) const;
38     }; // class Predictor
39 } // namespace LocallyStationaryModels
40
41 #endif // LOCALLY_STATIONARY_MODELS_KRIGING

```

## 7.7 samplevar.hpp

```

1 // Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
2 // Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>
3
4 #ifndef LOCALLY_STATIONARY_MODELS_SAMPLEVAR
5 #define LOCALLY_STATIONARY_MODELS_SAMPLEVAR
6
7 #include "grid.hpp"
8 #include "kernel.hpp"
9 #include "traits.hpp"
10
11 namespace LocallyStationaryModels {
12     class SampleVar {
13     private:
14         cd::matrixptr m_variogram = nullptr;
15         cd::matrixptr m_denominators = nullptr;
16         cd::matrixptr m_squaredweights = nullptr;
17         Kernel m_kernel;
18         Grid m_grid;
19         size_t m_n_angles;
20         size_t m_n_intervals;
21
22         void build_squaredweights();
23
24     public:
25         SampleVar(const std::string& kernel_id, const size_t& n_angles, const size_t& n_intervals, const
26             double& epsilon);
27
28         SampleVar();
29
30         void build_samplevar(const cd::matrixptr& data, const cd::matrixptr& anchorpoints, const
31             cd::vectorptr& z);
32
33         const cd::matrixptr get_variogram() const;
34         const cd::matrixptr get_denominators() const;
35         const cd::matrixptr get_squaredweights() const;
36         const cd::vectorptr get_x() const;
37         const cd::vectorptr get_y() const;
38         const cd::matrixptr get_kernel() const;
39         const cd::matrixptr get_grid() const;
40         const cd::vectorptr get_normh() const;
41     }; // class SampleVar
42 } // namespace LocallyStationaryModels
43
44 #endif // LOCALLY_STATIONARY_MODELS_SAMPLEVAR

```

## 7.8 smooth.hpp

```

1 // Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
2 // Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>
3
4 #ifndef LOCALLY_STATIONARY_MODELS_SMOOTH
5 #define LOCALLY_STATIONARY_MODELS_SMOOTH
6
7 #include "kernel.hpp"
8 #include "traits.hpp"
9
10 namespace LocallyStationaryModels {
11 class Smt {
12 private:
13     cd::matrixptr m_solutions = nullptr;
14     cd::matrixptr m_anchorpos = nullptr;
15
16     Kernel m_kernel;
17
18     double m_optimal_delta = 0;
19
20     double smooth_value(const size_t& pos, const size_t& n) const;
21
22     double smooth_value(const cd::vector& pos, const size_t& n) const;
23
24 public:
25     Smt(const cd::matrixptr& solutions, const cd::matrixptr& anchorpos, const double& min_delta,
26         const double& max_delta, const std::string& kernel_id);
27     Smt(const cd::matrixptr& solutions, const cd::matrixptr& anchorpos, const double delta,
28         const std::string& kernel_id);
29     Smt();
30
31     template <class Input> cd::vector smooth_vector(const Input& pos) const
32     {
33         cd::vector result(m_solutions->cols());
34         for (size_t i = 0; i < m_solutions->cols(); ++i) {
35             result(i) = smooth_value(pos, i);
36         }
37         return result;
38     };
39
40     const cd::matrixptr get_solutions() const;
41     double get_optimal_delta() const;
42     const cd::matrixptr get_anchorpos() const;
43 }; // class Smt
44 } // namespace LocallyStationaryModels
45
46 #endif // LOCALLY_STATIONARY_MODELS_SMOOTH

```

## 7.9 tolerances.hpp

```

1 // Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
2 // Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>
3
4 #ifndef LOCALLY_STATIONARY_MODELS_TOLERANCES
5 #define LOCALLY_STATIONARY_MODELS_TOLERANCES
6
7 namespace LocallyStationaryModels {
8 struct Tolerances {
9     static constexpr double anchor_tolerance = 1e-6;
10     inline static double pi = 4 * std::atan(1.);
11     static constexpr double min_determinant = 1e-12;
12     static constexpr double param_epsilon = 1e-6;
13     static constexpr double param_max_iterations = 1000000;
14     static constexpr double min_norm = 1e-12;
15     static constexpr double infinity = 1e12;
16     static constexpr double n_deltas = 1000;
17     static constexpr double gradient_step = 10e-8;
18 }; // struct Tolerances
19 } // namespace LocallyStationaryModels
20
21 #endif // LOCALLY_STATIONARY_MODELS_TOLERANCES

```

## 7.10 traits.hpp

```

1 // Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
2 // Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>
3
4 #ifndef LOCALLY_STATIONARY_MODELS_TRAITS

```

```

5 #define LOCALLY_STATIONARY_MODELS_TRAITS
6
7 #include <algorithm>
8 #include <cfloat>
9 #include <cmath>
10 #include <functional>
11 #include <iostream>
12 #include <memory>
13 #include <omp.h>
14 #include <string>
15 #include <vector>
16
17 #include "Eigen/Dense"
18 #include "tolerances.hpp"
19
20 namespace LocallyStationaryModels {
21 namespace cd {
22     // defining basic types
23     using vector = Eigen::VectorXd;
24     using matrix = Eigen::MatrixXd;
25     using matrixI = Eigen::MatrixXi;
26     using vectorptr = std::shared_ptr<vector>;
27     using matrixptr = std::shared_ptr<matrix>;
28     using matrixIptr = std::shared_ptr<matrixI>;
29     using vectorind = std::vector<size_t>;
30
31     // defining function types
32     using kernelfunction = std::function<double(const vector&, const vector&, const double&)>;
33     using gridfunction = std::function<matrixIptr(const matrixptr&, const size_t&, const size_t&, const
double&)>;
34
35 } // namespace cd
36 } // namespace LocallyStationaryModels
37
38 #endif // LOCALLY_STATIONARY_MODELS_TRAITS

```

## 7.11 variogramfit.hpp

```

1 // Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
2 // Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>
3
4 #ifndef LOCALLY_STATIONARY_MODELS_GRADIENT
5 #define LOCALLY_STATIONARY_MODELS_GRADIENT
6
7 #include "LBFGS/LBFGSB.h"
8 #include "traits.hpp"
9 #include "variogramfunctions.hpp"
10
11 namespace LocallyStationaryModels {
12 struct TargetFunction {
13     const cd::matrixptr m_empiricvariogram;
14     const cd::matrixptr m_squaredweights;
15     const cd::vectorptr
        m_mean_x;
16     const cd::vectorptr
        m_mean_y;
17     size_t m_x0;
18     std::shared_ptr<VariogramFunction> m_gammaisoptr;
19
20     TargetFunction(const cd::matrixptr& empiricvariogram, const cd::matrixptr& squaredweights,
21         const cd::vectorptr& mean_x, const cd::vectorptr& mean_y, const size_t& x0, const std::string&
id);
22
23     double operator()(const cd::vector& params, const cd::vector& grad);
24
25     double operator()(const cd::vector& params);
26 }; // struct TargetFunction
27
28 class Opt {
29 private:
30     cd::matrixptr m_empiricvariogram;
31     cd::matrixptr m_squaredweights;
32     cd::vectorptr m_mean_x;
33     cd::vectorptr m_mean_y;
34     std::string m_id;
35     cd::vector m_initialparameters;
36     cd::vector m_lowerbound;
37     cd::vector m_upperbound;
38     cd::matrixptr m_solutions = nullptr;
39
40     cd::vector findonesolution(const size_t& pos) const;
41
42 public:

```

```

86     Opt(const cd::matrixptr& empiricvariogram, const cd::matrixptr& squaredweights, const cd::vectorptr&
      mean_x,
87         const cd::vectorptr& mean_y, const std::string& id, const cd::vector& initialparameters,
88         const cd::vector& lowerbound, const cd::vector& upperbound);
89
90     void findallsolutions();
91
92     cd::matrixptr get_solutions() const;
93 }; // class Opt
94 } // namespace LocallyStationaryModels
95
96 #endif // LOCALLY_STATIONARY_MODELS_GRADIENT

```

## 7.12 variogramfunctions.hpp

```

1 // Copyright (C) Luca Crippa <luca7.crippa@mail.polimi.it>
2 // Copyright (C) Giacomo De Carlo <giacomo.decarlo@mail.polimi.it>
3
4 #ifndef LOCALLY_STATIONARY_MODES_VARIOGRAMFUNCTIONS
5 #define LOCALLY_STATIONARY_MODES_VARIOGRAMFUNCTIONS
6
7 #include "traits.hpp"
8
9 namespace LocallyStationaryModels {
10 class VariogramFunction {
11 protected:
12     double compute_anisotropic_h(
13         const double& lambda1, const double& lambda2, const double& phi, const double& x, const double&
14         y);
15
16 public:
17     VariogramFunction() = default;
18     virtual double operator()(const cd::vector& params, const double& x, const double& y) = 0;
19 }; // class VariogramFunction
20
21 class Exponential : public VariogramFunction {
22 public:
23     Exponential() = default;
24     double operator()(const cd::vector& params, const double& x, const double& y) override;
25 }; // class Exponential
26
27 class Matern : public VariogramFunction {
28 public:
29     Matern() = default;
30     double operator()(const cd::vector& params, const double& x, const double& y) override;
31 }; // class Matern
32
33 class MaternNuFixed : public VariogramFunction {
34 private:
35     double m_nu = 0.5;
36 public:
37     MaternNuFixed(const double& nu)
38         : m_nu(nu) {};
39     double operator()(const cd::vector& params, const double& x, const double& y) override;
40 }; // class MaternNuFixed
41
42 class Gaussian : public VariogramFunction {
43 public:
44     Gaussian() = default;
45     double operator()(const cd::vector& params, const double& x, const double& y) override;
46 }; // class Gaussian
47
48 std::shared_ptr<VariogramFunction> make_variogramiso(const std::string& id);
49 } // namespace LocallyStationaryModels
50
51 #endif // LOCALLY_STATIONARY_MODES_VARIOGRAM_FUNCTIONS

```





# Index

- Anchor
  - LocallyStationaryModels::Anchor, [13](#)
- build\_grid
  - LocallyStationaryModels::Grid, [17](#)
- build\_kernel
  - LocallyStationaryModels::Kernel, [19](#)
- build\_samplevar
  - LocallyStationaryModels::SampleVar, [26](#)
- build\_simple\_kernel
  - LocallyStationaryModels::Kernel, [19](#), [20](#)
- gaussian
  - LocallyStationaryModels::kf, [11](#)
- get\_anchorpos
  - LocallyStationaryModels::Smt, [29](#)
- get\_denominators
  - LocallyStationaryModels::SampleVar, [26](#)
- get\_grid
  - LocallyStationaryModels::Grid, [17](#)
  - LocallyStationaryModels::SampleVar, [26](#)
- get\_kernel
  - LocallyStationaryModels::Kernel, [20](#)
  - LocallyStationaryModels::SampleVar, [26](#)
- get\_normh
  - LocallyStationaryModels::Grid, [17](#)
  - LocallyStationaryModels::SampleVar, [27](#)
- get\_optimal\_delta
  - LocallyStationaryModels::Smt, [29](#)
- get\_origin
  - LocallyStationaryModels::Anchor, [14](#)
- get\_solutions
  - LocallyStationaryModels::Opt, [23](#)
  - LocallyStationaryModels::Smt, [29](#)
- get\_squaredweights
  - LocallyStationaryModels::SampleVar, [27](#)
- get\_tiles\_dimensions
  - LocallyStationaryModels::Anchor, [14](#)
- get\_variogram
  - LocallyStationaryModels::SampleVar, [27](#)
- get\_x
  - LocallyStationaryModels::Grid, [18](#)
  - LocallyStationaryModels::SampleVar, [27](#)
- get\_y
  - LocallyStationaryModels::Grid, [18](#)
  - LocallyStationaryModels::SampleVar, [27](#)
- Grid
  - LocallyStationaryModels::Grid, [17](#)
- identity
  - LocallyStationaryModels::kf, [11](#)
- Kernel
  - LocallyStationaryModels::Kernel, [19](#)
- LocallyStationaryModels::Anchor, [13](#)
  - Anchor, [13](#)
  - get\_origin, [14](#)
  - get\_tiles\_dimensions, [14](#)
- LocallyStationaryModels::cd, [9](#)
- LocallyStationaryModels::Exponential, [14](#)
  - operator(), [15](#)
- LocallyStationaryModels::Gaussian, [15](#)
  - operator(), [15](#)
- LocallyStationaryModels::gf, [9](#)
  - make\_grid, [10](#)
  - pizza, [10](#)
- LocallyStationaryModels::Grid, [16](#)
  - build\_grid, [17](#)
  - get\_grid, [17](#)
  - get\_normh, [17](#)
  - get\_x, [18](#)
  - get\_y, [18](#)
  - Grid, [17](#)
- LocallyStationaryModels::Kernel, [18](#)
  - build\_kernel, [19](#)
  - build\_simple\_kernel, [19](#), [20](#)
  - get\_kernel, [20](#)
  - Kernel, [19](#)
  - operator(), [20](#)
- LocallyStationaryModels::kf, [10](#)
  - gaussian, [11](#)
  - identity, [11](#)
  - make\_kernel, [11](#)
- LocallyStationaryModels::Matern, [21](#)
  - operator(), [21](#)
- LocallyStationaryModels::MaternNuFixed, [21](#)
  - operator(), [22](#)
- LocallyStationaryModels::Opt, [22](#)
  - get\_solutions, [23](#)
  - Opt, [23](#)
- LocallyStationaryModels::Predictor, [24](#)
  - Predictor, [24](#)
- LocallyStationaryModels::SampleVar, [25](#)
  - build\_samplevar, [26](#)
  - get\_denominators, [26](#)
  - get\_grid, [26](#)
  - get\_kernel, [26](#)
  - get\_normh, [27](#)
  - get\_squaredweights, [27](#)

- get\_variogram, [27](#)
- get\_x, [27](#)
- get\_y, [27](#)
- SampleVar, [25](#)
- LocallyStationaryModels::Smt, [28](#)
  - get\_anchorpos, [29](#)
  - get\_optimal\_delta, [29](#)
  - get\_solutions, [29](#)
  - smooth\_vector, [30](#)
  - Smt, [28](#), [29](#)
- LocallyStationaryModels::TargetFunction, [30](#)
  - operator(), [31](#), [32](#)
  - TargetFunction, [31](#)
- LocallyStationaryModels::Tolerances, [32](#)
- LocallyStationaryModels::VariogramFunction, [33](#)
  - operator(), [33](#)
- make\_grid
  - LocallyStationaryModels::gf, [10](#)
- make\_kernel
  - LocallyStationaryModels::kf, [11](#)
- operator()
  - LocallyStationaryModels::Exponential, [15](#)
  - LocallyStationaryModels::Gaussian, [15](#)
  - LocallyStationaryModels::Kernel, [20](#)
  - LocallyStationaryModels::Matern, [21](#)
  - LocallyStationaryModels::MaternNuFixed, [22](#)
  - LocallyStationaryModels::TargetFunction, [31](#), [32](#)
  - LocallyStationaryModels::VariogramFunction, [33](#)
- Opt
  - LocallyStationaryModels::Opt, [23](#)
- pizza
  - LocallyStationaryModels::gf, [10](#)
- Predictor
  - LocallyStationaryModels::Predictor, [24](#)
- SampleVar
  - LocallyStationaryModels::SampleVar, [25](#)
- smooth\_vector
  - LocallyStationaryModels::Smt, [30](#)
- Smt
  - LocallyStationaryModels::Smt, [28](#), [29](#)
- TargetFunction
  - LocallyStationaryModels::TargetFunction, [31](#)