



Proposal/Contract no.: 033811  
Project start: September 1, 2006  
Project end: August 31, 2009



# INTAMAP

*Interoperability and Automated Mapping*

SIXTH FRAMEWORK PROGRAMME

PRIORITY IST-2005-2.5.12

ICT for Environmental Risk Management

## Deliverable 2.6

Technical report documenting the developed code in R

Title of Deliverable	Sequential methods using robust likelihood models
Deliverable reference number	INTAMAP D2.6
Related WP and Tasks	WP2, Task 2.6
Type of Document	Public
Authors	Dionisis Hristopoulos and Giannis Spiliopoulos
Date	31 August 2009
Version	2.0

**Project coordinator**

Dr. Edzer J. Pebesma

Utrecht University, The Netherlands

E-mail: [e.pebesma@geo.uu.nl](mailto:e.pebesma@geo.uu.nl)

<http://www.intamap.org/>

## Revision History

Version	Date	Changes	Authors
1.0	20-05-2009	First draft - submission to Edzer Pebesma and Gerard Heuvelink	Dionisis Hristopoulos and Giannis Spiliopoulos
1.1	9-7-2009	Revisions, minor edits	Olivier Baume
2.0	31-8-2009	Edits to make into INTAMAP deliverable	Dionisis Hristopoulos

## Related task

### Task 2.6 Development of Computer Code (R) (month 13-36)

This task is crucial for the operational deployment of the developed methodology in an automated mapping system. Therefore, work on this task will begin relatively early in the project (M 12) and will continue until the completion of the WP (M 36), in order to incorporate all the methodological advances, allow sufficient time for testing the code, and verifying that methodological updates are seamlessly incorporated in existing code.

Active partners: TUC, AST, UKLU, UU.

### **Legal Notices**

The information in this document is subject to change without notice. The Members of the INTAMAP Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the INTAMAP Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

# Contents

1	Introduction . . . . .	2
1.1	Intamap Package . . . . .	2
	estimateAnisotropy . . . . .	4
	rotateAnisotropicData . . . . .	7
1.2	IntamapInteractive package . . . . .	9
	doSegmentation . . . . .	10
	anisotropyChoice . . . . .	12
2	Conclusions . . . . .	14
3	Appendix . . . . .	15
3.1	EstimateAnisotropy - rotateAnisotropicData . . . . .	15

## Executive Summary

This deliverable describes the code developed in R which implements the non-parametric identification of geometric anisotropy in spatially distributed scalar variables (e.g., scattered environmental data). The code is an implementation of the method of the covariance Tensor identity (CTI), which has been described in the Deliverable 2.3. We remind the user that in the case of scattered data in two dimensions, on which this project focuses, the anisotropy parameters involve the anisotropic ratio and the orientation angle of the principal anisotropy axes. This report describes the two main functions developed for anisotropy estimation and coordinate system transformation.

Within **INTAMAP** the code described here will be used in different ways by the spatial interpolation modules. (I) It will be used to provide initial conditions and prior probability density function models for the Bayesian methods of spatial analysis and interpolation, i.e., the trans-Gaussian kriging and the copula-based kriging developed by the University of Klagenfurt, as well as the sparse sequential approach developed by Aston University. (II) At the same time, the CTI method will provide estimates of anisotropy parameters that can be used to quantify anisotropy in the ordinary kriging algorithms developed by the University of Utrecht. The code implements a statistical test for isotropy, which measures if the estimated anisotropy is statistically significant. If this is the case, it will implement the necessary coordinate transformations for incorporating the anisotropy in the spatial interpolation. If the anisotropy is not statistically significant, it will inform the ordinary kriging module that an isotropic spatial variogram can be used.

# 1 Introduction

The R code developed for the **INTAMAP** project is divided into two sets of functions. The first set includes functions that are used by the core **INTAMAP** package. The two main functions used in this package are called **estimateAnisotropy** and **rotateAnisotropicData**. Both functions involve a number of internal routines. The function **estimateAnisotropy** calculates and returns estimates of the anisotropy parameters treating all the data as a single cluster. It also performs the isotropy test to decide if a coordinate transformation is necessary. The function **rotateAnisotropicData** performs a transformation of the coordinate system into a rotated and rescaled isotropic system, provided that the isotropy test is negative. Following this step, the omnidirectional variogram is estimated in the new system and fitted to a theoretical variogram model selected from the Gaussian, exponential, Matérn and spherical models. This is accomplished using the **automap** package developed by Paul Hiemstra (UU).

The second set of functions is integrated in the **INTAMAP INTERACTIVE** package. This is the development package of Intamap, which includes routines that were developed during the **INTAMAP** project but are yet not integrated into the public **INTAMAP** package. The routines that are included in the **INTAMAP INTERACTIVE** package require at this stage significant user involvement. TUC developed for the **INTAMAP INTERACTIVE** package the functions **doSegmentation** and **anisotropyChoice**. The function **doSegmentation** first removes isolated (without neighbors) distant stations from the input data, and then performs a segmentation into different clusters, based on the local sampling density function, and the distance of the station location from the centers of the clusters. The function **anisotropyChoice** uses the clusters defined by **doSegmentation** to estimate anisotropy parameters. The function provides two types of output: anisotropy parameters within each cluster, and global anisotropy parameters for the entire study area.

## 1.1 Intamap Package

The **INTAMAP** Package includes the main body of routines used by the **INTAMAP** project for anisotropy estimation from scattered 2D data. TUC has implemented two R functions for detection of anisotropy using the CTI method. These functions have been integrated with the spatial interpolation modules developed by ASTON and UKLU in the framework of the **INTAMAP** project. The function **estimateAnisotropy** implements the CTI method for the estimation of anisotropy parameters from scattered data. the function **rotateAnisotropicData** implements an isotropic transformation of the coordinate system. The isotropic transformation involves a rotation of the axes, aiming to align the coordinate system with the principal directions of anisotropy, and a rescaling of the distances that equalizes the major and minor axes. As a result, the variogram should be isotropic in the transformed coordinate system. Table 1 given below specifies the internal functions called by the main functions **estimateAnisotropy** and **rotateAnisotropicData**. Information on the operation and arguments of the functions **estimateAnisotropy** and **rotateAnisotropicData** are given in the help files below.

Function Name	Description
<b>estimateAnisotropy</b>	<b>public INTAMAP</b> function for anisotropy estimation using the CTI method
<code>estimateAnisotropySc</code>	Performs bilinear interpolation of scattered 2-D data over a square lattice (bicubic and thin-plate spline interpolation are also possible, if specified explicitly in the call). This function calls <code>estimateAnisotropyGrid</code> .
<code>estimateAnisotropyGrid</code>	Estimates sample derivatives and anisotropy parameters estimates based on 2D grid data obtained from ( <code>estimateAnisotropySc</code> ).
<code>jpdf</code>	Tests if estimated anisotropy ratio is inside the 95% confidence interval for statistical isotropy based on the non-parametric approximation of the joint pdf of the anisotropy parameters.
<code>biharmonicsSplineInterpolation</code>	Performs biharmonic spline interpolation; it is accessible only if <code>estimateAnisotropySc</code> is used interactively.
<b>rotateAnisotropicData</b>	<b>public INTAMAP</b> function that performs isotropic transformation of the data based on anisotropy estimates

Table 1: List of internal functions used by the two main functions of the `Intamap` package, `estimateAnisotropy` and `rotateAnisotropicData`.

### 1.1.1 Function `estimateAnisotropy`

---

`estimateAnisotropy`    *estimateAnisotropy*

---

#### Description

This function estimates geometric anisotropy parameters for 2-D scattered data using the CTI method.

#### Usage

```
estimateAnisotropy(object, depVar)
```

#### Arguments

`object`            (i) An `Intamap` type object (see `intamap`-package) containing one `SpatialPointsDataFrame` data frame named `observations` which includes the observed values (ii) or a `SpatialPointsDataFrame` which includes both coordinates and observations.

`depVar`            name of the dependent variable; this is used only in case (ii).

#### Details

Given the input object that defines  $N$  coordinate pairs  $(x,y)$  and observed values  $(z)$ , this method estimates of the geometric anisotropy parameters. Geometric anisotropy is a statistical property, which implies that the iso-level contours of the covariance function are elliptical. In this case the anisotropy is determined from the anisotropic ratio ( $R$ ) and the orientation angle ( $\theta$ ) of the ellipse.

Assuming a Cartesian coordinate system of axes  $x$  and  $y$ ,  $\theta$  represents the angle between the horizontal axis and PA1, where PA1 is one of the principal axes of the ellipse, arbitrarily selected (PA2 will denote the other axis).  $R$  represents the ratio of the correlation along PA1 divided by the correlation length PA2. Note that the returned value of  $R$  is always greater than one (see `value` below.)

The estimation is based on the Covariance Tensor Identity (CTI) method. In CTI, the Hessian matrix of the covariance function is estimated from sample derivatives. The anisotropy parameters are estimated by explicit solutions of nonlinear equations that link  $(R,\theta)$  with ratios of the covariance Hessian matrix elements.

To estimate the sample derivatives from scattered data, a background square lattice is used. The lattice extends in the horizontal direction from `x.min` to `x.max` where `x.min` (`x.max`) is equal to the minimum (maximum)  $x$ -coordinate of the data, and similarly in the vertical direction. The cell step in each direction is equal to the length of the lattice to the respective direction divided by the square root of  $N$ .

BiLinear interpolation, as implemented in `akima` package, is used to interpolate the field's z values at the nodes of the lattice.

The CTI method is described in detail in (Chorti and Hristopulos, 2008).

Note that to be compatible with `gstat` the returned estimate of the anisotropy ratio is always greater than 1.

## Value

(i) If the input is an `Intamap` object, the value is a modification of the input object, containing a list element `anisPar` with the estimated anisotropy parameters. (ii) if the input is a `SpatialPointsDataFrame`, then only the list `anisPar` is returned. The list `anisPar` contains the following elements:

<code>ratio</code>	The estimate of the anisotropy ratio parameter. Using the degeneracy of the anisotropy under simultaneous ratio inversion and axis rotation transformations, the returned value of the ratio is always greater than 1.
<code>direction</code>	The estimate of the anisotropy orientation angle. It returns the angle between the major anisotropy axis and the horizontal axis, and its value is in the interval (-90,90) degrees.
<code>Q</code>	A 3x1 array containing the sample estimates of the diagonal and off-diagonal elements (Q11,Q22,Q12) of the covariance Hessian matrix evaluated at zero lag.
<code>doRotation</code>	Boolean value indicating if the estimated anisotropy is statistically significant. This value is based on a statistical test of the isotropic (R= 1) hypothesis using a non-parametric approximation for the 95 percent confidence interval for R. This approximation leads to conservative (wider than the true) estimates of the confidence interval. If <code>doRotation==TRUE</code> then an isotropy restoring transformation (rotation and rescaling) is performed on the coordinates. If <code>doRotation==FALSE</code> no action is taken.

## Note

This function uses `akima` package to perform "bilinear" interpolation. The source code also allows other interpolation methods, but this option is not available when the function is called from within `INTAMAP`.

In the `gstat` package, the anisotropy ratio is defined in the interval (0,1) and the orientation angle is the angle between the vertical axis and the major anisotropy axis, measured in the clockwise direction. If one wants to use ordinary kriging inside `INTAMAP` the necessary transformations are performed in the function `estimateParameters.automap`. If one wants to use ordinary kriging in the `gstat` package (but outside `INTAMAP`) the required transformations can be found in the source code of the `estimateParameters.automap` function.

## Author(s)

A.Chorti, D.T.Hristopulos,G. Spiliopoulos

## References

- [1] <http://www.intamap.org>,
- [2] A. Chorti and D. T. Hristopulos (2008). Non-parametric Identification of Anisotropic (Elliptic) Correlations in Spatially Distributed Data Sets, IEEE Transactions on Signal Processing, 56(10), 4738-4751 (2008).
- [3] Em.Petrakis and D. T. Hristopulos (2009). A non-parametric test of statistical isotropy for Differentiable Spatial Random Fields in Two Dimensions. Work in progress. email: dionisi@mred.tuc.gr

## Examples

```
library(intamap)
data(sic2004)
coordinates(sic.val)~x+y
sic.val$value=sic.val$dayx

params=NULL

estimateAnisotropy(sic.val,depVar = "joker")
```

## Keywords

spatial, nonparametric, htest

### 1.1.2 Function `rotateAnisotropicData`

---

`rotateAnisotropicData`

*rotateAnisotropicData*

---

#### Description

This function applies an isotropic transformation of the coordinates specified in `object`.

#### Usage

```
rotateAnisotropicData(object, anisPar)
```

#### Arguments

- |                      |   |
|----------------------|---|
| <code>object</code>  | (i) An <code>Intamap</code> type object (see <code>00Introduction</code> ) containing one <code>SpatialPointsDataFrame</code> data frame named <code>observations</code> which includes the observed values (ii) or a <code>SpatialPointsDataFrame</code> which includes both coordinates and observations or (iii) <code>SpatialPoints</code> which includes only coordinates to be rotated.   |
| <code>anisPar</code> | An array containing the anisotropy parameters (anisotropy ratio and axes orientation) (see <code>estimateAnisotropy</code> ) for the rotation. If <code>object</code> is the output of <code>estimateAnisotropy</code> function, these parameters are part of <code>object</code> . In cases (ii) and (iii) <code>anisPar</code> defines the two anisotropy parameters. For the definition of the anisotropy parameters see <code>estimateAnisotropy</code> . |

#### Details

This function performs a rotation and rescaling of the coordinate axes in order to obtain a new coordinate system, in which the observations become statistically isotropic. This assumes that the estimates of the anisotropy ratio and the orientation angle provided in `anisPar` are accurate.

#### Value

- (i) A modified object with transformed coordinates if `rotateAnisotropicData` is called with an `Intamap` object as input (see `00Introduction`) or (ii) the transformed coordinates if a `SpatialPointsDataFrame` is used as input or (iii) the transformed coordinates if a `link[sp]{SpatialPoints}` object is the input.

#### Author(s)

Hristopulos Dionisis, Spiliopoulos Giannis

## References

- [1] <http://www.intamap.org>
- [2] A. Chorti and D. T. Hristopulos (2008). Non-parametric Identification of Anisotropic (Elliptic) Correlations in Spatially Distributed Data Sets, IEEE Transactions on Signal Processing, 56(10), 4738-4751 (2008).

## See Also

`estimateAnisotropy`

## Examples

```
library(gstat)
data(sic2004)
coordinates(sic.val)=~x+y
sic.val$value=sic.val$dayx

params=NULL

obj<-list(
  observations=sic.val
)
obj<-estimateAnisotropy(obj)
print(obj$anisPar)

obj$observations<-rotateAnisotropicData(obj$observations,obj$anisPar)

obj<-estimateAnisotropy(obj)
print(obj$anisPar)
```

## Keywords

spatial

## 1.2 IntamapInteractive package

The **INTAMAP INTERACTIVE** package includes functions for the estimation of anisotropy and automated interpolation that are not included in the core **INTAMAP** package. The functions described below still use the CTI for anisotropy estimation, but they provide more flexibility than the **estimateAnisotropy** function. In particular, using the functions **doSegmentation** and **anisotropyChoice** allow to estimate anisotropy parameters within different clusters. The clusters are defined using segmentation techniques from image analysis. The criteria for the segregation of the data locations are based on the sampling density and the geographical location (proximity) between stations. In addition, the methods in this package also allow for the calculation of a coarse-grained estimate of anisotropy parameters  $(R, \theta)$  over the entire study area. The coarse-grained estimates are not simply averages of the cluster anisotropy parameters. Instead they are derived from the CTI equations, using coarse-grained estimates of the covariance Hessian tensor; these estimates are derived by area-based averages of the estimates per cluster. Unlike **estimateAnisotropy**, the methods **doSegmentation** and **anisotropyChoice** require the user to choose parameters that control the segmentation of the data into clusters.

Table 2 given below specifies the internal functions called by the main functions **doSegmentation** and **anisotropyChoice**.

Function Name	Description
<b>doSegmentation</b>	<b>public INTAMAP</b> function for the segmentation of scattered data into clusters
<b>segmentData</b>	Main internal segmentation function; it calls sequentially the internal functions below; it can be used as a main function, if the data coordinates are defined in the input.
<b>rmDist</b>	Removes isolated and distant points (a point is considered isolated if it has no neighbours within a specified range and distant if its distance from the centroid of the data points exceeds a specified threshold)
<b>calculateDensity</b>	Estimates sampling density matrix and assigns sampling density values to each observation point.
<b>detectEdges</b>	Performs edge detection on the sampling density grid using a logarithmic [5x5] and 3x3 averaging filters.
<b>findPerimeters</b>	Identifies and labels closed edge perimeters
<b>fillPerimeters</b>	Assigns sensor points to the already identified perimeters and rejects clusters containing fewer than 50 points.
<b>assingRestToClusters</b>	Assigns unassigned sensor points to a neighbouring cluster, using a <i>soft</i> parameter to determine the relative weights of the sampling density and distance from neighbouring clusters.
<b>anisotropyChoice</b>	<b>public INTAMAP</b> function for multicluster anisotropy estimation; it also returns coarse-grained global estimates of anisotropy.

Table 2: List of internal functions used by the two main functions of the IntamapInteractive package, **doSegmentation** and **anisotropyChoice**.

### 1.2.1 Function doSegmentation

---

doSegmentation

*Spatial Segmentation - Clustering for Scattered Observations*

---

#### Description

This function performs segmentation of scattered 2D data based on sampling density and location.

#### Usage

```
doSegmentation(object)
```

#### Arguments

**object** An Intamap type object containing the element (list) **observations**, which includes the coordinates of the observation locations

#### Details

This function performs segmentation of scattered 2D data based on sampling density and location. Let us assume that  $N_o$  is the number of observation locations. If  $N_o < 200$ , then a single cluster is returned. (1) The segmentation algorithm first removes isolated distant points, if there are any, from the observation locations. Points  $(x_i, y_i)$  are characterized as ‘isolated’ and ‘distant’ if they satisfy the following conditions :  $|x_i - \text{mean}(x)| > 4 * \text{std}(x)$  or  $|y_i - \text{mean}(y)| > 4 * \text{std}(y)$  and distance from closest neighbor  $> \sqrt{(\text{std}(x)/2)^2 + (\text{std}(y)/2)^2}$ . After the first step the size of the original dataset is reduced to  $N$  ( $N = N_o - \text{isolated points}$ ) points. (2) A sampling density matrix (lattice) consisting of  $N$  cells that cover the study area is constructed. Each cell is assigned a density value equal to the number of observation points inside the cell. In addition, each observation point is assigned the sampling density value of the containing cell. (3) Unsupervised clustering edge detection is used to determine potential cluster perimeters. (4) Each closed region’s perimeter is labeled with a different cluster (segment) number. (5) All observation points internal to a cluster perimeter are assigned to the specific cluster. (6) Each cluster that contains fewer than 50 observation points is rejected. (7) The observation points that have not initially been assigned to a cluster and those belonging to rejected (small) clusters are assigned at this stage. The assignment takes into account both the distance of the points from the centroids of the accepted clusters as well as the mean sampling density of the clusters.

Note: The  $N_o < 200$  empirical constraint is used to avoid extreme situations in which the sampling density is concentrated inside a few cells of the background lattice, thereby inhibiting the edge detection algorithm.

## Value

A modified Intamap object which additionally includes the list element `clusters`. This element is a list that contains (i) the indices of removed points from `observations`; (ii) the indices of the clusters to which the remaining observation points are assigned and (iii) the number of clusters detected.

`clusters` list element added to the original object containing the segmentation results.

`rmdist` Indices of removed points.

`index` Index array identifying the cluster in which each observation point belongs.

`clusterNumber` Number of clusters detected.

## Author(s)

A. Chorti, Spiliopoulos Giannis, Hristopoulos Dionisis

## References

[1] D. T. Hristopoulos, M. P. Petrakis, G. Spiliopoulos, A. Chorti (2009). Non-parametric estimation of geometric anisotropy from environmental sensor network measurements, StatGIS 2009: Geoinformatics for Environmental Surveillance Proceedings (ed. G. Dubois).

## Examples

```
library(intamapInteractive)

data(walker)
coordinates(walker)=~X+Y
object=createIntamapObject(observations=walker)
object=doSegmentation(object)

print(summary(object$clusters$index))
```

## Keywords

spatial,cluster

## 1.2.2 Function `anisotropyChoice`

---

<code>anisotropyChoice</code>	<i>anisotropyChoice</i>
-------------------------------	-------------------------

---

### Description

This function combines segmentation of scattered 2D data and estimation of anisotropy parameters using the CTI method.

### Usage

```
anisotropyChoice(object)
```

### Arguments

`object` An Intamap type object containing one `SpatialPointsDataFrame` with observations.

### Details

The function `AnisotropyChoice` function employs the `doSegmentation` function to automatically separate the original dataset into clusters based on the sampling density and the spatial locations of the data (see `doSegmentation` for details). The results of the segmentation procedure and the anisotropy analysis per cluster are returned in a matrix of dimension `[cl]x5`, where `[cl]` is the number of clusters. Each row of the matrix contains the cluster index, the anisotropy ratio, the anisotropy direction, the number of cluster points and the area inside the convex hull of the cluster. In addition, a single set of anisotropy parameters is returned in the element `anisPar`. These parameters are calculated using weighted averages of the covariance Hessian matrix estimates in each cluster. The weights are based on the area enclosed by the convex hull of each cluster.

### Value

`object`: A modified Intamap type object is returned, which contains the results of the anisotropy parameter estimation. The anisotropy parameters are returned in the element `anisPar` as described below.

`anisPar` List element in `object` that contains a list with the following elements:

`ratio` A coarse-grained anisotropy ratio for all the data

`direction` A coarse-grained anisotropy orientation for all the data

`clusters` A matrix of dimension `[cl]x5` which determines the anisotropy per cluster. Each row of `clusters` gives the (cluster id, anisotropy ratio, anisotropy direction, number of points, area) for each cluster detected.

**clusters** list element added to the original object containing the segmentation results.

**index** Index array identifying the cluster in which each observation point belongs. Zero value means that the observations has been removed.

**clusterNumber** Number of clusters detected.

## Note

This function uses the `akima` package to perform "bilinear" and "bicubic" interpolation for the estimation of spatial derivatives

## Author(s)

D.T. Hristopulos, G.Spiliopoulos, A.Chorti

## References

- [1] <http://www.intamap.org>
- [2] A. Chorti and D. T. Hristopulos (2008). Non-parametric Identification of Anisotropic (Elliptic) Correlations in Spatially Distributed Data Sets, *IEEE Transactions on Signal Processing*, 56(10), 4738-4751 (2008).
- [3] D. T. Hristopulos, M. P. Petrakis, G. Spiliopoulos, A. Chorti (2009). Non-parametric estimation of geometric anisotropy from environmental sensor network measurements, *StatGIS 2009: Geoinformatics for Environmental Surveillance Proceedings* (ed. G. Dubois).

## Examples

```
library(intamapInteractive)

data(walker)
coordinates(walker)=~X+Y
object=createIntamapObject(observations=walker)
object=anisotropyChoice(object)

print(summary(object$clusters$index))
print(object$anisPar)
```

## Keywords

spatial, cluster, nonparametric, htest

## 2 Conclusions

This deliverable contains the R code generated for the implementation of anisotropy estimation by means of the Covariance Tensor Identity (CTI) method, as well as comments for the usage of this code in connection with the spatial interpolation methods developed for Intamap. These functions implement the advances in the anisotropy estimation achieved in the frame of WP2. Four main functions are fully described and commented. Each main function calls a number of internal functions that implement specific tasks of the anisotropy estimation procedure. The first two of the main functions, **estimateAnisotropy** and **rotateAnisotropicData**, are a part of the core **INTAMAP** package. These functions permit the estimation of anisotropy parameters from scattered 2D data and a transformation into the estimated isotropic coordinate system. These CTI anisotropy estimates are used as initial guesses for the *sparse sequential* (AST) and for the *spatial copula* interpolation methods (UKLU), thereby reducing the time needed for the estimation of anisotropy parameters by these methods. The same functions, **estimateAnisotropy** and **rotateAnisotropicData**, are used to provide anisotropy estimates for the *ordinary kriging* interpolation method implemented by the University of Utrecht.

As a general comment, the anisotropy estimates are reliable for smooth (differentiable) distribution of the observed process, dense and spatially ordered sampling of the study area, and statistical stationarity. The anisotropy ratio estimate tends to be reduced by lower sampling density, increased spatial disorder of the sampling network, and small number of sampling stations. In such cases the estimates tend to become more isotropic. Non-stationarity can lead to significant changes in the anisotropy estimates with respect to the routine (stationary) case. In the case of non-stationarity, we recommend to study the potential use of a trend and apply our methods to the residuals. Cross validation analysis of Intamap gamma dose rate distribution scenarios shows that accounting for anisotropy can increase the accuracy of interpolation in certain cases (depending on the accuracy of the estimated anisotropy parameters). In all the studied cases, the anisotropy-based cross validation measures have not been less accurate than the ones obtained using the isotropic assumption.

Two other main functions, **doSegmentation** and **anisotropyChoice** have been developed and implemented for users who may want more flexibility. These functions implement the CTI method using a multi-cluster segmentation of the study area. The functions **doSegmentation** and **anisotropyChoice** include a number of parameters that control the segmentation procedure. The default values for these parameters have been empirically determined using the data sets provided by the Intamap scenarios (described in the Intamap Deliverable 5.4). These functions return both anisotropy estimates per cluster, as well as a coarse-grained anisotropy estimate for the entire study area. The latter estimate is obtained with an average covariance Hessian matrix, derived by weighing the cluster matrices proportionally to the area enclosed by each cluster. The functions **doSegmentation** and **anisotropyChoice** are more suitable for cases of strong non-stationarity in the data, and for handling an heterogeneous sampling pattern.

### 3 Appendix

#### 3.1 EstimateAnisotropy - rotateAnisotropicData

```
#####  
#Function :EstimateAnisotropy function  
#Description Receives a Intamap Object or calls estimateAnisotropySc  
# function and returns a new KrigingObject which  
# includes anisotropy parameters estimation.  
#  
#Input object & The Intamap object Object or Spatial points data  
# &frame  
# anisPar & Only used if object is Spatial and includes the  
# & dependant variable  
#Output  
# :object & (i) The modified intamap object in case  
# &of Intamap object input with a list element anisPar  
# &anisPar$(list(ratio=R,direction=theta.deg))  
# &(ii) Only the anisPar list element in case of Spatial  
# &input.  
#####  
estimateAnisotropy<-function(object,depVar){  
  
  if (is(object,"Spatial")) {  
    observations = object  
  } else {  
    observations = object$observations  
  }  
  
  if (missing(depVar)) {  
    if ("formulaString" %in% names(object)) {  
      depVar=as.character(object$formulaString[[2]])  
    } else depVar = "value"  
  }  
  
  # params = object$params  
  xy<-as.matrix(coordinates(observations))  
  
  anisPar<-estimateAnisotropySc(xy[,1],xy[,2],observations[[depVar]],  
  method="linear",pl=FALSE)  
  
  if (is(object,"Spatial")) {  
    anisPar  
  } else {  
    object$anisPar = anisPar  
    object  
  }  
}  
  
#####  
#Function : rotateAnisotropicData function  
#Description Receives an Intamap Object containing
```

```

# an element observations containing the data coordinates
# and a anisPar element containing the anisotropy parameters
# that will be used for the data rotation.
#Input
# :object & The original Intamap or Spatial Object.
# :anisPar & The anisotropy Parameters to be used
# & for transformation of the data.
#Output :object & with the transformed coordinates (intamap
# &input) or the transformed
# &coordinates(spatial input)
#####
rotateAnisotropicData<-function(object,anisPar){
  #if (inherits(object,"Spatial")) {
  #EJP: changed inherits into is
  if (is(object,"Spatial")) {
    locations = object
  } else {
    if (missing(anisPar)) anisPar = object$anisPar
    locations = object$observations
  }
  if (missing(anisPar) || is.null(anisPar)) stop("Argument anisPar missing or
equal
to NULL")

xy<-as.matrix(coordinates(locations))
dataNames=colnames(xy)
x<-as.matrix(xy[,1])
y<-as.matrix(xy[,2])

theta=(anisPar$direction)*pi/180
R=anisPar$ratio

#rotate data
x_new=(x)*cos(theta)+y*sin(theta)
y_new=R*(-x*sin(theta)+y*cos(theta))

coords=as.data.frame(cbind(x_new,y_new))
colnames(coords)<-dataNames

coordinates(coords) = as.formula(paste("~",dataNames[1],"+",dataNames[2]))

if ("data" %in% names(getSlots(class(locations)))) {
  coords = SpatialDataFrame(coords, data = locations@data)
}
if (!is(locations,"Spatial")) {
  object$observations = coords
  object
} else
  coords
}

```

```

#old version that supports only Intamap Object.
rotateAnisotropicDataOld<-function(object){
if ("formulaString" %in% names(object)) {
  formulaString = object$formulaString
} else formulaString = as.formula("value ~ 1")
depVar=as.character(formulaString[[2]])

xy<-as.matrix(coordinates(object$observations))
dataNames=colnames(xy)
x<-as.matrix(xy[,1])
y<-as.matrix(xy[,2])

theta=(object$anisPar$direction)*pi/180
R=object$anisPar$ratio

#rotate data
x_new=(x)*cos(theta)+y*sin(theta)
y_new=R*(-x*sin(theta)+y*cos(theta))

coords=cbind(x_new,y_new)
colnames(coords)<-dataNames
object$observations =
SpatialPointsDataFrame(coords,data=object$observations@data)

return(object)
}

```

```

##### Internal functions #####
#
#####
#Function estimateAnisotropySc
#Description : This is the main fuction used in order to calculate
# anisotropy parameters from scattered data
# Given a Cartesian coordinates system of axes x and y and an ellipsoid
# of correlation isolevels has principal directions a and b. The
# anisotropy rotation angle(theta.deg) express the angle between
# principal direction a and axes x.
#
#Inputs &x &The x-axis coordinate of field
# &y &The y-axis coordinate of field
# &r &The field value in (xi,yi) point
# &len &The length of the interpolated field
# &method &The method that we will use for interpolation
# & &"cubic","linear","v4"
# &min.x &minimum x value
# &max.x &maximum x value
# &min.y &minimum y value

```

```

# &max.y &maximum x value
# &deb &toggles debugging mode Only Used
# Interactively, outside intamap
# &pl &toggles plot Only Used
# Interactively, outside intamap
# &br &borders coordinates Only
# Used Interactively, outside intamap
#Outputs &R &Anisotropy ratio
# &theta.deg &Anisotropy rotation angle in degrees
#Packages : akima package
#####
estimateAnisotropySc<-function(x, y, r, len=length(x), method="linear",
min.x=min(x),
max.x=max(x), min.y=min(y), max.y=max(y),deb=FALSE,pl=FALSE,br){

plot.borders=TRUE
if(missing(br)) {plot.borders=FALSE}

if(len<50) return(list(ratio=1,
direction=0,Q=c(1,1,1),doRotation=FALSE))

#mesh creation
step=min(c(abs((max.x-min.x)/sqrt(len)),abs((max.y-min.y)/sqrt(len))))

xn<-seq(min.x,max.x,by=step)
yn<-seq(min.y,max.y,by=step)

mesh<-meshgrid(xn,yn)

#selection of interpolation method
if (method=="cubic") {
ri<-interp(x,y,r,xn,yn,linear=FALSE,extrap=TRUE,duplicate="mean")
} #akima package
if (method=="linear") {
ri<-interp(x,y,r,xn,yn,linear=TRUE,duplicate="mean")
ri$z<-t(ri$z) }
#akima
package
if (method=="v4") {
ri<-biharmonicSplineAnisotropy(x,y,r,mesh$x,mesh$y)}

#calculate anisotropy parameters over regular grid
res=estimateAnisotropyGrid(mesh$x,mesh$y,ri$z)

if((pl==TRUE & plot.borders==TRUE)){
image(mesh$x[1,],mesh$y[,1],t(ri$z),col=rainbow(20),
#color.palette=rainbow,

```

```

plot.title=title(main=method),xlim=range(br[,1]),ylim=range(br[,2]))
points(br,pch=".",xlim=(br[,1]),ylim=(br[,2]))
}else if ((pl==T & plot.borders==FALSE)){
  image(mesh$x[1,],mesh$y[,1],t(ri$z),col=rainbow(20),
  #color.palette=rainbow,
plot.title=title(main=method))
}

return(list(ratio=res$R,
direction=res$theta.deg,Q=res$Q,doRotation=res$dump$doRotation))
}
#####

# function [R , theta_deg ]= estimateAnisotropyGrid(xi ,yi , ri)
#   theta_deg , R :variables to b calculated
#   xi ,yi : Coordinates on Grid
#   ri : Random field at (xi,yi)
#####

estimateAnisotropyGrid<-function(xi,yi,ri){
  dump=NULL
  #Gradient function
  mgradient<-function(k,stx,sty){

  gg<-function(m,st){

  m<-as.matrix(m)
  n=dim(m)[1] #rows
  p=dim(m)[2] #columns

  g<-matrix(0,n,p)
  g[1,]<-(m[2,]- m[1,])/st
  g[n,]<-(m[n,]-m[n-1,])/st
  g[2:(n-1),]=(m[3:n,]- m[1:(n-2),])/(2*st)

  return (g)
  }

  y=gg(k,sty)
  x=t(gg(t(k),stx))
  return(list(x=x,y=y))
}

stepx<-xi[1,2]-xi[1,1]
stepy<-yi[2,1]-yi[1,1]
divZ<-mgradient(ri,stepx,stepy)

```

```

divX2<-as.vector(divZ$x*divZ$x)
divY2<-as.vector(divZ$y*divZ$y)
divXY<-as.vector(divZ$x*divZ$y)

#get rid of Na
Q11<-mean(divX2[is.na(divX2)==FALSE])
Q22<-mean(divY2[is.na(divY2)==FALSE])
Q12<-mean(divXY[is.na(divXY)==FALSE])

if(Q11<10^-31 || is.na(Q11) || is.na(Q12) || is.na(Q22)){
dump=NULL
dump$doRotation=FALSE
return (list(R=1,theta.deg=0,Q=cbind(Q11,Q22,Q12),dump=dump))
}else{

Zdiag<-Q22/Q11
Zoff<-Q12/Q11
theta<-0.5*atan(2*Zoff/(1-Zdiag))

R<-sqrt(1+((1-Zdiag)/(Zdiag-(1+Zdiag)*(cos(theta))^2)))

#Test for isotropy
#dump<-jpdf(seq(0,3,by=0.01),seq(-90,90,by=1),R,theta*180/pi,length(xi))
dump=jpdf(length(xi),R)

if(is.na(R)){
R=1
theta=0
dump$doRotation=FALSE
}

#Make sure that all results come in the same interval
if (R<1){
R=1/R

if ((theta+pi/2>-pi/2) & (theta+pi/2)<(pi/2)){
theta=theta+pi/2
}else{
theta=theta-pi/2
}

}

theta.deg<-theta*180/pi
}

return (list(R=R,theta.deg=theta.deg,Q=cbind(Q11,Q22,Q12),dump=dump))
}

```

```

# Description: function that tests if the predicted anisotropy ratio
# is inside the 95%
confidence
interval of the jpdf function
# for statistical
isotropy. Replaced jpdf_old for speed reasons
jpdf=function(N,R){

doRotation=TRUE
r=6 #comes from 95%interval

floor=sqrt((N-2*sqrt((N-r)*r))/(N-2*r))
ceil =sqrt((N+2*sqrt((N-r)*r))/(N-2*r))

if(R>floor && R <ceil) doRotation=FALSE

return(list(doRotation=doRotation))
}

#####
#Function : jpdf_old function
#
#Description: calculates the joint pdf function for R
# and theta over a given region
#Arguments :
# R : a mesh type matrix of the R region
# theta : a mesh type matrix of the theta region
# Rest : the given ratio value
# thetaEst: the given theta value
#
#Longer Description:
# The joint pdf gives the probability of R AND theta
# to lie within the infinitesimal region dR*dtheta. It
# is normalized to 1 if integrated to R belonging to [0, inf) and
# theta belonging to (-45, 45) degrees. At the moment it is
# not dependent to a specific covariance function model since it
# is a first approximation. But this is useful since it accounts
# for the worst-case scenario. In other words, the confidence
# region it provides is conservative. A version which will
# account for a specific correlation model is under development.
#
# A few words about the confidence interval (region). The idea is to
# define the parametric equation of the curve in the (R,theta)-plane
# which "encloses" 95% of the probability. The equation has the form
# c(R,theta)=0 so the points (R, theta) for which the equation holds,
# define the confidence region boundary. Any point (R,, theta,) for
# which c(r,,theta,)>0 lies within the confidence region. The percentile
# (here 95%) determined by the parameter r and is the inverse of
# the chi-square distribution with 2 degrees of freedom
# and the percentile requested (e.g. 0.95).

```

```
#####
jpdf_old<-function(Rf,thetaf,Rest,thetaEst,N){

  erfc <- function(x){ 2 * pnorm(x * sqrt(2), lower = FALSE)}

  RRest=Rest
  tthetaEst=thetaEst

  #so far the jpdf is not numerically stable for large values of N due to
  #the exponential terms ~exp(N). But this is not a problem for the
  #evaluation of the confidence interval. So I moved the...
  #if(N>1200){N=1200}
  #...where it is necessary.

  Rest=1
  thetaEst=0

  #create mesh grids
  mesh<-meshgrid(Rf,thetaf)
  R<-mesh$x
  theta<-mesh$y

  #threshold value to decide if the given set is in 95% interval of Jpdf
  threshold=5.99416

  #first convert degrees to Radians
  theta=theta*pi/180
  thetaEst=thetaEst*pi/180

  #Matrix elements of H -- constants
  s1=2*(cos(thetaEst)^2 +Rest^2 * sin(thetaEst)^2 )
  s2=2*(sin(thetaEst)^2 +Rest^2 * cos(thetaEst)^2 )
  s12=2*sin(thetaEst)* cos(thetaEst)*(1-Rest^2)

  #Determinant of H -- constant
  d=s1*s2-s12^2

  qd = (R^2 +tan(theta)^2) / (1 + R^2 *tan(theta)^2)
  qo = tan(theta) * (1-R^2) / (1+R^2 *tan(theta)^2)

  #Jacobian for the transformation (qd,qo) -> (R,theta)
  J1= ((2*R*abs(R^2 -1)*(1/cos(theta))^6))
  J2=(1+R^2 *tan(theta)^2)^3
  J=J1/J2
}
```

```

a = N/(2*d^2) * (qd^2 * s1^2 + 2 * qd * s12^2 + s2^2 - 4 * qo * s12 * (qd *
s1 +
s2) - 2 * qo^2 * (d - 2 * s1*s2))
b = -(N/d) * (qd * s1 - 2 * qo * s12 + s2);
k = (N/d)^(3/2) / (4*sqrt(2)* pi^(3/2));

#contour plot
lhs = 2 * (R^4 * s1^2 + 2 * R^2 * s12^2 + s2^2) * threshold + tan(theta)^2 *
(-2 *
N * (2 * s12^2 + s1 * s2 + R^4 * (2 * s12^2 + s1 * s2) + R^2 * (s1^2 - 4 *
s1 *
s2 + s2^2)) + 4 * (2 * s12^2 + R^2 * (-2 * s12^2 + (s1 - s2)^2) + s1 * s2 +
R^4 *
(2 * s12^2 + s1 * s2))* threshold + (-N * (4 * R^2 * s12^2 + (s1 - R^2 *
s2)^2)
+ 2 * (s1^2 + 2 * R^2 * s12^2 + R^4 * s2^2) * threshold) * tan(theta)^2)
rhs = N *(R^4 * s1^2 + s2^2 + R^2 * (4 * s12^2 - 2 * s1 * s2)) + 4 * (-1 +
R^2)
* s12 * (N - 2 * threshold) * tan(theta) * (R^2 * s1 + s2 + (s1 + R^2 * s2)
*
tan(theta)^2)
cc= lhs - rhs

#(see comment in the beginning of the function)
if(N>1200){N=1200}

#jpdf
p = (pi/180)*k*J* (exp(-N/2) * (-4 * sqrt(a)* b + (4*a + b^2)*
exp(b^2/(8*a))*sqrt(2*pi)*erfc(b/(2*sqrt(2)*sqrt(a)))))/(8*a^(5/2))

#The parametric equation of the contour interval is cc=0. If cc>=0 we are in
the
#interior of the confidence interval
booleanC<-(cc>=0)

maxV<-max(mesh$x[which(booleanC)])
minV<-min(mesh$x[which(booleanC)])
print(paste("jpdf 95% R maximum ratio for statistical isotropy: ",maxV))
doRotation=(RRest>maxV || RRest<minV)
# doRotation=(max(mesh$x[which(booleanC)])<RRest)

# contour(as.matrix(Rf),as.matrix(thetaf),t(p))
# contour(Rf,thetaf,t(p))

return(list(c=cc,p=p,bc=booleanC,doRotation=doRotation,R=maxV))

```

```
}
```

```
#####  
#biharmonics Spline interpolation  
# scattered data coordinates -> x, y  
# data value @(x,y) -> z  
# grid ->xi, yi  
#####  
biharmonicSplineAnisotropy<-function(x,y,z,xi,yi){  
#sortrows function  
SortMat <- function(Mat, Sort)  
{  
  m <- do.call("order", as.data.frame(Mat[, Sort]))  
  Mat[m, ]  
}  
  
#sorting input  
Mat<-cbind(x,y,z)  
Mat<-SortMat(Mat,c(2,1))  
x<-Mat[,1]  
y<-Mat[,2]  
z<-Mat[,3]  
  
#Move to Complex field  
  
xy<-as.matrix(x+y*1i)  
d<-matrix(xy,length(xy),length(xy))  
  
#Calculate distances  
d<-abs(d-t(d))  
  
#remove zeros so we can use log  
ind=which(d==0)  
if(length(ind)>0){  
  d[ind]=1  
}  
d=d^2*(log(d)-1)  
  
#return replace zeros to diag elements  
d[seq(1,length(d),by=(dim(d)[1]+1))]=0  
  
ind=which(is.na(d)==TRUE)  
if(length(ind)>0){  
  d[ind]=1  
}  
}
```

```

#calculate weights
wweights=solve(d)%*%z
rm(d)
m=dim(xi)[1]
n=dim(xi)[2]
zi<-matrix(0,m,n)
xy=t(xy)

for(i in 1:m){
  for(j in 1:n){
    d = abs(xi[i,j]+(1i*yi[i,j])-xy)
    mask=which(d==0)

    if (length(mask)>0) {d[mask]=1}
    g<-(d)^2*(log(d)-1)
    if(length(mask)>0) { g[mask]=0}
    zi[i,j]=g%% wweights
  }
}

return(list(x=xi,y=yi,z=zi))

}

#Meshgrid function
meshgrid <- function(a,b) {
  list(
    x=outer(b*0,a,FUN="+"),
    y=outer(b,a*0,FUN="+")
  )
}

#####End of internal functions#####

doSegmentation<-function(object){
# ptm<-proc.time()

print("Clustering - Segmentation")

xy<-coordinates(object$observations)
dd<-cbind(xy[,1:2],object$observations$value)

if(dim(xy)[1]<200){
object$clusters=list(

index=matrix(1,dim(xy)[1],1),

```

```

clusterNumber=1,

rmdist=NULL

)
return(object)
}

rmdist=TRUE
#Step 0 remove distant points
if(rmdist==TRUE){
ind<-rmDist(dd)
if(length(ind)>0) dd<-dd[-ind,]
}

object$clusters<-segmentData(ddd=dd,pl=FALSE,dev=FALSE,
br=object$predictionLocations)

index<-object$clusters$index

#create the index that corresponds to the original dataset.
final_index<-matrix(0,length(dd[,2]),1)

if(length(ind)>0){final_index[-ind]=index
}else{final_index=index}

object$clusters$index<-final_index

object$clusters$rmdist<-ind

#return single a value for all Europe.

# print("Finished in : ")
# print(proc.time()-ptm)
return(object)
}
#####
#Description : This is the main segmentation function if used outside
# Intamap. Calls sequentially the steps described in doSegmentation
# help file.
#
#Input:
# ddd: A nx2 or nx3 matrix with the observations. First column should
# the horizontal and the second one the vertical coordinate.
# pl : boolean that toggles plotting. If TRUE several plots during the
# segmentation procedure are shown. (Only used interactively)
# dev: boolean that toggles saving the plots. (Only used interactively)
#

```

```

# soft: parameter that controls the weight of sampling density and
# distance from neighbouring clusters.
# br: mx2 matrix with boorders coordinates in case of plotting. This
# this parameter is actually deactivated in line 79.
#
#Output: A list with the following elements
# index: a nx1 array with the indices.
# clusterNumber: The total number of clusters detected
#
#####
#
segmentData<-function(ddd,pl=FALSE,dev=FALSE,soft=0.2,br){
plot.borders=TRUE

# if(pl==TRUE) {
# par(ask=FALSE)
# }else {dev=FALSE}
if(missing(br)==TRUE){
plot.borders=FALSE
# br<-borders()
# br<-long2INSP(br)
}

#Only for bench
ptm<-proc.time()
tim=0 #toggle print times
cl_min_size=50

#####
#Step 1 density matrix
#####
dat<-list(den=calculateDensity(ddd[,1],ddd[,2],1,4,pl=FALSE,dev),
dat=ddd)

#####
# if(tim==1){
# print("Step1")
# tmp<-(proc.time() - ptm)
# print(tmp)
# }
#####

Ng<-dat$den$Ng
N<-dat$den$N
stepx<-dat$den$stepx
stepy<-dat$den$stepy

```

```

xx<-dat$dat[,1]
yy<-dat$dat[,2]
d.mtx<-dat$den$density_L

x<-dim(d.mtx)[1]
y<-dim(d.mtx)[2]
#####
#Step 2 Find edges
#####
      edg<-detectEdges(d.mtx)

# if(pl==TRUE){
# image(edg,col=terrain.colors(12))
# if(dev==TRUE){
# dev.copy2eps(file="edges.eps")
# }
# }

#####
#   if(tim==1){
#     print("Step2")
#     print(proc.time() - tmp -ptm)
#     tmp<-proc.time() - tmp -ptm
#   }
#####

#Step 3   Separate the closed perimeters and label them
# inputs:  edg      :a (0,1) matrix indicating which points are edges
#   Ng      :Ng number
# output:  cl$ll    :a list with cluster indentified
#   cl$number :the number of cluster indentified
cl<-findPerimeters(edg,Ng,pl=FALSE,dev)
#if no or just one cluster was identified index all clusters as 1.
if(cl$number==0 | cl$number==1){
return(list(index=matrix(1,length(ddd[,1]),1),clusterNumber=1))
}

#####
#   if(tim==1){
#     print("Step3")
#     print(proc.time() - tmp -ptm)
#     tmp<-proc.time() - tmp -ptm
#   }
#####

#####
#STEP 4: find internal to the perimeters points and reject clusters that are too
small
#####

st4<-fillPerimeters(xx,yy,Ng,N,cl,cl_min_size,
```

```

dat$den$mav_dens_xy,stepx,stepy,pl=FALSE,dev)      #
return(list(clnum=clnum,clust_dens=clust_dens,Mx=Mx,My=My,id=id))

#####
#   if(tim==1){
#       print("Step4")
#       print(proc.time() - tmp -ptm)
#       tmp<-proc.time() - tmp -ptm
#   }
#####

#####
#STEP 5 :: Label points that have not yet been labeled
#####

out<-assignRestToClusters(ddd[,1], ddd[,2], st4$id, st4$clnum, st4$Mx,
st4$My,dat$den$mav_dens_xy, st4$clust_dens,soft)
cc<-cbind(xx,yy)

#   if(tim==1){
#       print("Step5")
#       print(proc.time() - tmp -ptm)
#       tmp<-proc.time() - tmp -ptm
#   }

if(tim==1){
print("Total")
print(proc.time() - ptm)
}

#plot the result
if(pl==TRUE){
if (plot.borders){
plot(br,pch=".")

for (temp in 1:st4$clnum){
Cluster=ddd[which(out==temp),1:2]
points(Cluster[,1:2],col=temp,pch="*")

mtext(line=(temp-1)%%4,col=temp+1,paste("cluster",temp),adj=round(temp/8))
#mtext(". =borders ")
}
}else{

plot(ddd[,1:2],pch="*")

for (temp in 1:st4$clnum){
Cluster=ddd[which(out==temp),1:2]
points(Cluster[,1:2],col=temp,pch="*")
}
}
}

```

```

mtext(line=(temp-1)%4,col=temp,paste("cluster",temp),adj=round(temp/8))
#mtext(". =borders ")
}

}

}

#return(list(st4=st4,out=out,cc=cc,dat=dat))
return(list(index=out,clusterNumber=st4$clnum))
#return(out)
}

#####INTERNAL FUNCTIONS#####
#####
#STEP 1
calculateDensity<-function(x,y,gd,L,pl,dev){

xy<-cbind(x,y)
N<-dim(xy)[1]
d<-dim(xy)[2]

#calculating the grid step
stepx<-diff(range(x)) / (round(sqrt(gd*N))-1)*1
stepy<-diff(range(y)) / (round(sqrt(gd*N))-1)*1

#creating the grid
xg<-seq(min(x),max(x),by=stepx)
yg<-seq(min(y),max(y),by=stepy)

xg<-matrix(xg,length(xg),length(xg))
yg<-matrix(yg,length(yg),length(yg))

#expand grid by 2 with zero filling on each dimension in order to have
space
#for use image filters
xg<-rbind(matrix(0,2,dim(xg)[2]),xg,matrix(0,2,dim(xg)[2]))
xg<-t(cbind(matrix(0,dim(xg)[1],2),xg,matrix(0,dim(xg)[1],2)))

yg<-rbind(matrix(0,2,dim(yg)[2]),yg,matrix(0,2,dim(yg)[2]))
yg<-cbind(matrix(0,dim(yg)[1],2),yg,matrix(0,dim(yg)[1],2))

#initiate variables

```

```

Ng=dim(xg)[1]
denst<-matrix(0,Ng,Ng)
inn=vector("list",N)
#the radius of the square is going to be equal to the grid size, cetred
#on each node. Therefore, we are looking for points that are +/-stepx/2
#and +/-stepy/2 apart.
#density_xy=zeros(N, 1);

for( i in 1:Ng){
for (j in 1:Ng){
ind<-indfinal<-NULL
ind<-which(x<xg[i,j]+stepx/2 & x>xg[i,j]-stepx/2)
if(length(ind)==0){ denst[i,j]=0
}else{
indfinal<-which(y[ind]<yg[i,j]+stepy/2 & y[ind]>
yg[i,j]-stepy/2)
denst[i,j]=length(indfinal)

}

}

}

#attribute a density to the initial points
density_xy<-matrix(0,1,N)
dens_xy<-NULL
index<-vector("list",N)

for (st in 1:N){
ind=which(x[st]-x>0 & abs(x[st]-x)<stepx/2)
indfinal=which(y[st]-y[ind]>=0 & abs(y[st]-y[ind])<stepy/2)
dens_xy[st]=length(indfinal)+1
inn[[st]]<-c(indfinal)
}
density_L=t(denst)
mav_dens_xy=dens_xy

#moving average filter, (smoothing)
density_L<-applyFilter(density_L,"average")

#Plot the Density
if(pl==TRUE){
filled.contour(density_L,color=terrain.colors)
if(dev==TRUE){
dev.copy2eps(file="smoothed")
}
}
return(list(mav_dens_xy=mav_dens_xy,density_L=density_L,Ng=Ng,

```

```

stepx=stepx,stepy=stepy,N=N,inn=inn))
}

#####
#STEP 2
#This function applies filters on images
#####
applyFilter<-function(dat,input,TH){

#####INTERNAL FUNCTION #####
#Create Laplacian of gaussian filter
#####
create.log<-function(){
  eps=2.2204e-016
  #first we calculate gaussian
  g=(-1:1)
  x=outer(g*0,g,"+")
  y=outer(-g,g*0,"+")
  sigma=0.5

  arg = -(x*x + y*y)/(2*(sigma^2))
  hh=exp(arg)
  hh[(hh<eps*max(hh))]=0

  sumh=sum(hh)
  if(sumh!=0){
hh=hh/sumh
  }
  #now we calculate laplacian
  h1=hh*(x*x+y*y - 2*(sigma^2))/(sigma^2)^2
  h2=h1-sum(h1)/prod(5,5)

  #sum to zero

  h2=h2-sum(h2)/length(h2)
  return(h2)
}

#mask selection
mask<-switch(input,
  "laplace" =matrix(c(0,-1,0,-1,4,-1,0,-1,0),3,3),
  "average"=(matrix(c(1,1,1,1,1,1,1,1,1),3,3)/9),
  "average22"=(matrix(c(-1,-1,-1,-1,9,-1,-1,-1,-1),3,3)),
  # "rows"=matrix(c(1,sqrt(2),1,0,0,0,-1,-sqrt(2),-1),3,3),
  # "columns"=matrix( c( -1 ,0 ,1 , -sqrt(2) ,0 ,sqrt(2), -1, 0,

```

```

1),3,3)
    "rows"=matrix(c(1,1,1,0,0,0,-1,-1,-1),3,3),
    "columns"=matrix( c( -1 ,0 ,1 , -1 ,0 ,1, -1, 0, 1),3,3),
    "average5"=matrix(1/25,5,5),
    "log"=create.log()

)

#Threshold initialization
#TH<-1
x<-dim(dat)[1]
y<-dim(dat)[2]

ll<-(dim(mask)[2]-1)/2
zerosx<-matrix(0,ll,y)
zerosy<-matrix(0,(x+2*ll),ll)
dat<-rbind(zerosx,dat,zerosx)
dat<-cbind(zerosy,dat,zerosy)

rm(zerosx,zerosy)
out<-dat

for (i in (1+ll):(x+ll)){
  for(j in (1+ll):(y+ll)){

#out[i,j]<-{if(sum(mask * dat[(i-ll):(i+ll),(j-ll):(j+ll)])>TH)
1 else
0 }
out[i,j]<-sum(mask*dat[(i-ll):(i+ll),(j-ll):(j+ll)])
  }
}
  return(out[(1+ll):(x+ll),(1+ll):(y+ll)])
}

detectEdges<-function(dat){

thresh=0
m<-dim(dat)[1]
n<-dim(dat)[2]

rr<-2:(m-1)
cc<-2:(n-1)
edg<-matrix(0,m,n)

dat<-applyFilter(dat,"average",0)

out<-applyFilter(dat,"log",0)

ed=which((out[rr,cc]<0 & out[rr,cc+1]>0) &
abs(out[rr,cc]-out[rr,cc+1])>thresh,arr.ind=TRUE)

```

```

edg[ed+1]=1

ed=which((out[rr,(cc-1)]>0 & out[rr,cc]<0) &
abs(out[rr,(cc-1)]-out[rr,cc])>thresh,arr.ind=TRUE)
edg[ed+1]=1

ed=which((out[rr,cc]<0 & out[rr+1,cc]>0) &
abs(out[rr,cc]-out[rr+1,cc])>thresh,arr.ind=TRUE)
edg[ed+1]=1

ed=which((out[(rr-1),cc]>0 & out[rr,cc]<0) &
abs(out[(rr-1),cc]-out[rr,cc])>thresh,arr.ind=TRUE)
edg[ed+1]=1

ed=which(out[rr,cc]==0,arr.ind=TRUE)

if(length(ed)>0){
zeros=ed
zz=which(out[zeros-c(-1,0)]<0 & out[zeros+c(2,1)]>0 & abs
(out[zeros-c(-1,0)]-out[zeros+c(2,1)])>2*thresh ,arr.ind=TRUE)
edg[zz+c(1,1)]=1

zz=which(out[zeros-c(-1,0)]>0 & out[zeros+c(2,1)]<0 & abs
(out[zeros-c(-1,0)]-out[zeros+c(2,1)])>2*thresh ,arr.ind=TRUE)
edg[zz+c(1,1)]=1

zz=which(out[zeros-c(0,-1)]<0 & out[zeros+c(1,2)]>0 & abs
(out[zeros-c(0,-1)]-out[zeros+c(1,2)])>2*thresh ,arr.ind=TRUE)
edg[zz+c(1,1)]=1

zz=which(out[zeros-c(0,-1)]>0 & out[zeros+c(1,2)]<0 & abs
(out[zeros-c(0,-1)]-out[zeros+c(1,2)])>2*thresh ,arr.ind=TRUE)
edg[zz+c(1,1)]=1
}

return(edg)
}

#####
#Step 3
#####
findPerimeters<-function(edg,Ng,pl=FALSE,dev=FALSE){

#####INTERNAL FUNCTIONS TO findPerimeters#####
find_neighb<-function(cx,cy,jj,Ng){
nx<- (((cx[jj]-cx)==0 )| ((cx[jj]-cx)==1) | (cx[jj]-cx==-1))
ny<- (((cy[jj]-cy)==0 )| ((cy[jj]-cy)==1) | (cy[jj]-cy==-1))

same_point<-((cx[jj]-cx)==0 & (cy[jj]-cy)==0)

```

```

edgex=(( (cx[jj]==1)|(cx[jj]==2) )|((cx[jj]==Ng) | (cx[jj]==(Ng-1) ) ))
edgey=(( (cy[jj]==1)|(cy[jj]==2) )|((cy[jj]==Ng) | (cy[jj]==(Ng-1) ) ))

# if we are at an edge point we will continue searching even with a
#single neighbour
np_n<-xor((nx & ny),same_point)
np_n<-which(np_n==TRUE)
edge<-(edgex | edgey )
return(list(np_n=np_n,edge=edge))

}

#we will search and remove duplicate points
rm.duplicate<-function(input){
  L=length(input)
  for(i in 1:L){
    if((input[i]!=-1)){
      input[(which((input[(i+1):L]-input[i])==0)+i)]=-1
    }
  }
  return(input[which(input!=-1)])
}

#function that removes old points
rm.old<-function(input){
  L=length(input)
  for(i in 1:L){
    if((input[i]!=-1)){
      input[(which((input[(i+1):L]-input[i])==0)+i)]=-1
    }
  }
  return(input[which(input!=-1)])
}

#plot.point<-function(dat,jj){
# plot(dat)
# points(dat[jj,1],dat[jj,2],col=2)
#}
#####END OF INTERNAL FUNCTIONS#####

# if(missing(dev)){dev=FALSE}
# if(missing(pl)){pl=FALSE}

indices<-which(edg==1,arr.ind=TRUE)
len<-dim(indices)[1]

#here are the coords of edges
cx<-indices[,1]
cy<-indices[,2]

```

```

cl<-matrix(0,length(cx),1)

continuee=1 #flag indicating that the search for the next region is
NOT over
cl_counter=0 #temporary counter of the number of clusters

  while(continuee==1){

    jj=1 #start at the first point
    cl_counter=cl_counter+1 #increase the number of
clusters:
the number of clusters at this point equals the number
cl_ind=NULL #the indices of the perimeter
points
that belong to the cluster (the region perimeter in other words)
fn=find_neighb(cx, cy, jj, Ng) #np_n is the number of
neighbours of
the point under investigation, edgee is a flag indicating
#whether the point under
investigation
is an edge point (in an open
curve
which normally would not occur,
however just in case)

    if(pl==TRUE){
#plot.point(cbind(cx,cy),jj)      #plots the edge
points
#PLOTTING
dat=cbind(cx,cy)
plot(dat)
points(dat[jj,1],dat[jj,2],col=2)
points(cx[fn$np_n],cy[fn$np_n],col=3)
#PLOTTING
}

    if (length(fn$np_n)>1){      #then we don,t have a single
point
      #find all points of the cluster
cl_ind=c(cl_ind ,t(fn$np_n))
cl_ind_n=cl_ind #temporary
variable
cluster_end=0 #flag indicating
that
the cluster perimeter is not yet finalized
size_cl=length(cl_ind) #cluster size
tmp=0 #old cluster

```

```

length
while (cluster_end==0){
  size_cl=length(cl_ind)
  for (k in (1+tmp):length(cl_ind_n)){
    nfn=find_neighb(cx, cy, cl_ind_n[k], Ng)
#find all
    neighbouring points    returns->(np_n, edgee)
    cl_ind_n=c(cl_ind_n, t(nfn$np_n))
#add
    new neighbouring points to the cluster perimeter
    cl_ind_n=rm.duplicate(cl_ind_n)
    if(pl==TRUE){
points(cx[cl_ind_n],cy[cl_ind_n],col=3)
#visualize
data selection #PLOTTING
}
}

    #%We might at this point have introduced a point twice in
the
    perimeter because it has
    #%multiple neighboring points. So we need to remove
duplicate points
    cl_ind_n=rm.duplicate(cl_ind_n)

    #keep the old length in order to search only in new points
tmp<-length(cl_ind)

    if ((length(cl_ind)-length(cl_ind_n))==0){
    cluster_end=1    #when there
are
    no more neighbours, the perimeter is finalised
    }
    cl_ind=cl_ind_n
}

#create new variable cluster
assign(paste(cluster,,cl_counter,sep=""),cbind(cx[cl_ind],
cy[cl_ind]))

#remove points already assigned to a cluster
cx=cx[-cl_ind]
cy=cy[-cl_ind]
include_edge_points=1
np=0
if (length(cx)==0){
  continuee=0;
}

}else{

```

```

cluster_end=1;
cl_ind=jj;

#assign points to a cluster variable
assign(paste(,cluster,,cl_counter,sep=""),cbind(cx[cl_ind],
cy[cl_ind]))

#remove points already assigned to a cluster
cx=cx[-cl_ind]
cy=cy[-cl_ind]
np=0;
if (length(cx)==0){
  continuee=0;
}
}
}#end of while

#plot(indices)
ll=vector("list",cl_counter)
for (i in 1:cl_counter){
  cl<-get(paste("cluster",i,sep=""))
  if(pl==TRUE){
points(cl,col=(i+1))
#PLOTTING
}
  ll[[i]]=cl;
}

# if (dev==TRUE){dev.copy2eps(file="Edges.eps") }
  return(list(ll=ll,number=cl_counter))
}

#####
#Step 4
#####
fillPerimeters<-function(x,y, Ng, N, clust , cl_min_size, mav_dens_xy, stepx,
stepy,pl=FALSE,dev=FALSE){

#####Internal Functions to fill Perimeters#####
#####
#The cluster perimeter is defined from the points xx and yy.
#We want to identify wether the point (x,y) is inside this perimeter.
#####
isinside_cluster<-function(xx,yy,x,y,stepx,stepy){

if( ( (x>(max(xx)+stepx/2)) | (x<(min(xx)-stepx/2)) | (y>
(max(yy)+stepy/2)) |
(y<(min(yy)-stepy/2) ))==TRUE){
not_inside=1

```

```

}else{
ind=which(y<yy+stepy/2 & y>yy-stepy/2)
rx=xx[ind]
ry=yy[ind]
if(length(ind)==0){
not_inside=1
}else{
  if(length(which(x>rx))==0 | length(which(x<rx))==0){
    not_inside=1
  }else{
    not_inside=0
  }
}
}
return(not_inside)
}
#####END IF INTERNAL FUNCTIONS#####

if(pl==TRUE){
plot(x,y,pch="*")
#PLOTTING
}

ax= (max(x)-min(x)) / (Ng-1)
ay= (max(y)-min(y)) / (Ng-1)

b.x=min(x)-ax
b.y=min(y)-ay

not_inside=1
id=matrix(0,N,1)

for(b in 1:N){
for(fff in 1:clust$number){
xxx=clust$11[[fff]][,1]
yyy=clust$11[[fff]][,2]

#Decide if a point is inside a perimeter or not.

not_inside=isinside_cluster(ax*xxx+b.x,ay*yyy+b.y,x[b],y[b],stepx,stepy)
if(not_inside==0){
id[b]=fff
not_inside=1
break
}
}
}

if(pl==TRUE){
for(b in 1:clust$number){

```

```

xxx=clust$ll[[b]][,1]
yyy=clust$ll[[b]][,2]

points(x[id==b],y[id==b],col=b+3,pch="*")
#PLOTTING
points(ax*xxx+b.x,ay*yyy+b.y,pch="+",col=3)
#PLOTTING
}
}

#reject cluster with too few points
clnum=0
clust_dens=NULL
Mx=NULL
My=NULL

for(ff in 1:clust$number){
if(length(x[which(id==ff)])<cl_min_size){
id[which(id==ff)]=0
}else{
clnum=clnum+1
temp=which(id==ff)
id[temp]=clnum

xxx=clust$ll[[ff]][,1]
yyy=clust$ll[[ff]][,2]

assign(paste("final_cluster",clnum,sep=""),cbind(ax*xxx+b.x,ay*yyy+b.y))

#We evaluate the cluster centres as the centroids of the
#clusters

Mx[clnum]=sum(x[temp]*mav_dens_xy[temp])/sum(mav_dens_xy[temp]);
My[clnum]=sum(y[temp]*mav_dens_xy[temp])/sum(mav_dens_xy[temp]);

clust_dens[clnum]=mean(mav_dens_xy[temp])

if(is.na(Mx[clnum])|is.na(My[clnum])){
Mx=Mx[-clnum]
My=My[-clnum]
clust_dens=clust_dens[-clnum]
id[which(id==ff)]=0
# rm(paste("final_cluster",clnum,sep=""))
clnum=clnum-1
}
}
}

```

```

if(dev==TRUE){dev.copy2eps(file="fillPerimeters.eps") }
return(list(clnum=clnum,clust_dens=clust_dens,Mx=Mx,My=My,id=id))
}

assignRestToClusters<-function(x, y, id, clnum, Mx, My, mav_dens_xy, clust_dens,
soft){

#INTERNAL function calculates cost
fcost<-function(x,y,Mx,My,soft,mav,clust,maxden,maxdist){
cost= (sqrt((x-Mx)^2+(y-My)^2))/maxdist
+soft*(sqrt(abs(mav-clust))/maxdens)
return(cost)
}

N<-length(x)
index<-matrix(0,N,1)

#in order to avoid dividing by zero
if(length(which(clust_dens!=0)==0)){
clust_dens=clust_dens+1
}

for(i in 1:N){
if(id[i]>0){
index[i]=id[i]
}else{

#choose the 2 closest
ind2<-which(rank(sqrt((x[i]-Mx)^2+(y[i]-My)^2))<2.6)

if(length(ind2)!=0){

maxdist=max(sqrt((x[i]-Mx[ind2])^2+(y[i]-My[ind2])^2))+0.00000001

maxdens=max(sqrt(abs(mav_dens_xy[i]-clust_dens[ind2])))+0.00000001

#calculate clusters, costs and assign point to
the
minimum one

cost=fcost(x[i],y[i],Mx[ind2],My[ind2],soft,mav_dens_xy[i],
clust_dens[ind2],maxdens,maxdist)
index[i]=ind2[which.min(cost)]
}
else{

index[i]=which.min(rank(sqrt((x[i]-Mx)^2+(y[i]-My)^2)))
}
}

```

```

}
}
  return(index)

}#

#borders<-function(){
# library(maptools)
# eur<-read.shape(,countryLim.shp,)
# brd<-NULL
# iter=length(eur$Shapes)
#
# for(i in 1:iter){
# brd<-rbind(brd,eur$Shapes[[i]]$verts)
# }
# return(brd)
#}

#####
#
#function :rmDist
#
#Description :Removes distant points from the original dataset
#
# in order to increase edge detection accuracy
#
# the removed points will not be assigned in step5
#
#####
#
rmDist<-function(data){
std.x<-sd(data[,2])/2
std.y<-sd(data[,1])/2

dd<-sqrt(std.x^2+std.y^2)

#find which points are too distant
x.ind=which(data[,2]>mean(data[,2])+4*sd(data[,2]) |data[,2]<
mean(data[,2])-4*sd(data[,2]) )
y.ind=which(data[,1]>mean(data[,1])+4*sd(data[,1])|data[,1]<
mean(data[,1])-4*sd(data[,1]))
ind=union(x.ind , y.ind)

#check if these points don,t have any point near
if(length(ind)>2){
dat<-as.matrix(data[ind,])

xy=as.matrix(dat[,2]+1i*dat[,1])
xy<-matrix(xy,length(xy),length(xy))

```

```

distances=abs(xy-t(xy))
index<-which(distances < dd & distances >0,arr.ind=TRUE)
if(length(index)>0){
index<-matrix(index,,2)
index=union(index[,1],index[,2])
ind=ind[-index]
}

}

return(ind)
}

#####
####
#
#Function : AnisotropyChoice(object)
#
#Description : This method combines segmentation of a large dataset
# and anistropy parameters estimation for scattered data.
#
#Arguments : An intamap type object
#
#Details : The function AnisotropyChoice function employs the
# doSegmentation function to automatically separate the original
# dataset into clusters based on the sampling density and the spatial
# locations of the data. The results of the segmentation procedure and
# the anisotropy analysis per cluster are returned in a matrix of
# dimension [c1]x5, where [c1] is the number of clusters . Each row of
# the matrix contains the cluster index, the anisotropy ratio, the
# anisotropy direction, the number of cluster points and the area
# inside the convex hull of the cluster. In addition, a single set of
# anisotropy parameters is returned in the element \code{anisPar}.
# These parameters are calculated using weighted averages of the
# covariance Hessian matrix estimates in each cluster. The weights are
# based on the area enclosed by the convex hull of each cluster.
#####
#####

anisotropyChoice<-function(object){

params<-object$params
params$doSegmentation=TRUE
if(params$doSegmentation==FALSE){
#This is the case that no Segmentation is done

object$params$AnisPar<-estimateAnisotropy(object)
return (object)
}
}

```

```

}else if (params$doSegmentation==TRUE){
#returns list with a single pair of anisotropy parameters for all
data(singleCl)
#and a matrix (cln,5) with the values for

if ("formulaString" %in% names(object)) formulaString =
object$formulaString
else formulaString = as.formula("value ~ 1")
depVar=as.character(formulaString[[2]])

xy<-as.matrix(coordinates(object$observations))
z<-object$observations[[depVar]]

#first run segmentation
xyz<-as.matrix(cbind(xy,z))
if(dim(xy)[1]<200){
object$clusters=list(

index=matrix(1,dim(xy)[1],1),

clusterNumber=1,

rmdist=NULL

)
rmInd=NULL
clNumber=1
index=matrix(1,dim(xy)[1],1)
ind=index
}else{

rmdist=TRUE
#Step 0 remove distant points
if(rmdist==TRUE){
rmInd<-rmDist(xyz)
if(length(rmInd)>0){
xyz_d<-xyz[-rmInd,]
}else{
xyz_d=xyz
}
}

# do Segmentation
segmentResult<-segmentData(ddd=xyz_d,pl=FALSE,dev=FALSE)

```

```

#This index is
ind<-segmentResult$index
clNumber<-segmentResult$clusterNumber
#rmDist<-segmentResult$clusters$rmDist

}

xyz_new<-xyz
if (length(rmInd)>0){
xyz_new<-xyz[-rmInd,]
}

Qs<-matrix(0,clNumber,3)
area<-matrix(0,clNumber,1)

results=matrix(0,clNumber,5)

#create an index size of original data xyz 0 value means that the point
#is not assigned to a cluster and should not be included in anisotropy
estimation.
index=matrix(0,length(xyz[,1]),1)
if(length(rmInd)>0){
index[-rmInd]=ind
}else{
index=ind
}

object$clusters$index=index
object$clusters$clusterNumber=clNumber

#find anisotropy for each cluster and calculate Q,s matrices
#Qs are slope tensors - mean covariances matrices

for(i in 1:clNumber){
#select points that belong to the cluster i
temp<-xyz_new[which(i==ind),]

tempPar<-intamap:::estimateAnisotropySc(temp[,1],temp[,2],temp[,3])

Qs[i,]<-tempPar$Q

```

```

#calculate area of each cluster
#####
cdat<-chull(temp[,1],temp[,2])

cdat<-SpatialPoints(data.frame(x=temp[cdat,1],y=temp[cdat,2]))
# proj4string(cdat)=CRS("+init=epsg:3035")
# cdat<-spTransform(cdat,CRS("+init=epsg:4326"))
convHull<-cbind(coordinates(cdat))
convHull<-rbind(convHull,convHull[1,]) #close Polygon

# conv_Poly=Polygon(as.matrix(convHull))
# ps=appendPolys(NULL,convHull,1,1,FALSE)
#
# attr(ps,"projection")="LL"
# psUTM = convUL(ps)
# polygonArea=calcArea(psUTM,rollup=1)
# area[i]=polygonArea$area

area[i]=area(convHull)
#####

#store all results together

results[i,]=c(i,tempPar$ratio,tempPar$direction,length(which(ind==i)),area[i])

}

#here we use Q matrices to create a weighted mean for Qs
#with respect to the area of each clusters chull
Qs<-apply(kronecker(matrix(1,1,3),area)*Qs,2,sum)/sum(area)

#the weighted Q matrices (slope tensors)
Q11=Qs[1]
Q22=Qs[2]
Q12=Qs[3]

if(Q11==0){
return (list(R=1,theta.deg=0))
}else{

#calculate the weighted correlation lengths
Zdiag<-Q22/Q11
Zoff<-Q12/Q11
theta<-0.5*atan(2*Zoff/(1-Zdiag))

```

```

R<-sqrt(1+((1-Zdiag)/(Zdiag-(1+Zdiag)*(cos(theta))^2)))

if (R<1){
R=1/R

if ((theta+pi/2>-pi/2) & (theta+pi/2)<(pi/2)){
theta=theta+pi/2
}else{
theta=theta-pi/2
}
}

theta.deg<-theta*180/pi
}

singleCl=list(ratio=R,direction=theta.deg)

object$anisPar=singleCl
object$anisPar$clusters=results

#must return object
#return(list(clParams=results,singleCl=singleCl))
return(object)

}else{

# printf("Wrong Choise for anisotropy")
return(-1)

}

}

area <- function(x, y)
{
if(missing(y)) {
if(is.matrix(x) && ncol(x) == 2) {
y <- x[, 2]
x <- x[, 1]
}
else if(!is.null(x$x) && !is.null(x$y)) {
y <- x$y
}
}
}

```

```

x <- x$x
}
}
x <- c(x, x[1])
y <- c(y, y[1])
i <- 2:length(x)
return(0.5 * sum(x[i] * y[i - 1] - x[i - 1] * y[i]))
}

#
rmDist<-function(data){
std.x<-sd(data[,2])/2
std.y<-sd(data[,1])/2

dd<-sqrt(std.x^2+std.y^2)

#find which points are too distant
x.ind=which(data[,2]>mean(data[,2])+4*sd(data[,2]) | data[,2]<
mean(data[,2])-4*sd(data[,2]) )
y.ind=which(data[,1]>mean(data[,1])+4*sd(data[,1]) | data[,1]<
mean(data[,1])-4*sd(data[,1]))
ind=union(x.ind , y.ind)

#check if these points don,t have any point near
if(length(ind)>2){
dat<-as.matrix(data[ind,])

xy=as.matrix(dat[,2]+1i*dat[,1])
xy<-matrix(xy,length(xy),length(xy))

distances=abs(xy-t(xy))
index<-which(distances < dd & distances >0,arr.ind=TRUE)
if(length(index)>0){
index<-matrix(index,,2)
index=union(index[,1],index[,2])
ind=ind[-index]
}
}

return(ind)
}

```