Package ‘fdaPDE’

May 15, 2020

Version 1.0-9

Date 2020-04-27

Title Statistical Analysis of Functional and Spatial Data, Based on Regression with PDE Regularization

Maintainer Eleonora Arnone `<eleonora.arnone@polimi.it>`

Depends R (>= 3.5.0), stats, grDevices, graphics, geometry, rgl, Matrix, plot3D, plot3Drgl

LinkingTo RcppEigen

Suggests MASS, testthat

Description An implementation of regression models with partial differential regularizations, making use of the Finite Element Method. The models efficiently handle data distributed over irregularly shaped domains and can comply with various conditions at the boundaries of the domain. A priori information about the spatial structure of the phenomenon under study can be incorporated in the model via the differential regularization. See Sangalli, L.M., Ramsay, J.O., Ramsay, T.O. (2013), Spatial spline regression models for an overview.

License CC BY-NC-SA 4.0

Copyright See the individual source files for copyrights information

NeedsCompilation yes

SystemRequirements C++11

RoxygenNote 7.0.2

Encoding UTF-8

Author Eardi Lila [aut],
Laura M. Sangalli [aut],
Eleonora Arnone [aut, cre],
Jim Ramsay [aut],
Luca Formaggia [aut],
Alessandra Colli [ctb],
Luca Colombo [ctb],
Carlo de Falco [ctb]

Repository CRAN

Date/Publication 2020-05-15 15:10:02 UTC

R topics documented:

`covs.test` *Covariate test function for the horseshoe domain*

Description

Implements a finite area test function the horseshoe domain.

Usage

```
covs.test(x, y)
```

Arguments

`x, y`	Points at which to evaluate the test function.

Value

Returns function evaluations.

`create.FEM.basis` *Create a FEM basis*

Description

Sets up a Finite Element basis. It requires a `mesh.2D` or `mesh.2.5D` object, as input. The basis' functions are globally continuos functions, that are polynomials once restricted to a triangle in the mesh. The current implementation includes linear finite elements (when `order = 1` in the input `mesh`) and quadratic finite elements (when `order = 2` in the input `mesh`).

Usage

```
create.FEM.basis(mesh)
```

Arguments

`mesh`	A `mesh.2D` or `mesh.2.5D` object representing the domain triangulation. See create.mesh.2D, create.mesh.2.5D.

Value

A `FEMbasis` object. This contains the `mesh`, along with some additional quantities:

- `order`Either "1" or "2". Order of the Finite Element basis.
- `nbasis`Scalar. The number of basis.
- `transf_coord`It takes value only in the 2D case. It is a list of 4 vectors: diff1x, diff1y, diff2x and diff2y. Each vector has length #triangles and encodes the information for the tranformation matrix that transforms the nodes of the reference triangle to the nodes of the i-th triangle. The tranformation matrix for the i-th triangle has the form [diff1x[i] diff2x[i]; diff1y[i] diff2y[i]].
- `detJ`It takes value only in the 2D case. A vector of length #triangles. The ith element contains the determinant of the transformation from the reference triangle to the nodes of the i-th triangle. Its value is also the double of the area of each triangle of the basis.

See Also

`create.mesh.2D`, `create.mesh.2.5D`

Examples

```
## Upload the quasicircle2D data
data(quasicircle2D)
boundary_nodes = quasicircle2D$boundary_nodes
boundary_segments = quasicircle2D$boundary_segments
locations = quasicircle2D$locations
data = quasicircle2D$data

## Create the 2D mesh
```

```
mesh = create.mesh.2D(nodes = rbind(boundary_nodes, locations), segments = boundary_segments)
## Plot it
plot(mesh)
## Create the basis
FEMbasis = create.FEM.basis(mesh)
## Upload the hub2.5D data
data(hub2.5D)
hub2.5D.nodes = hub2.5D$hub2.5D.nodes
hub2.5D.triangles = hub2.5D$hub2.5D.triangles

## Create the 2.5D mesh
mesh = create.mesh.2.5D(nodes = hub2.5D.nodes, triangles = hub2.5D.triangles)
## Plot it
plot(mesh)
## Create the basis
FEMbasis = create.FEM.basis(mesh)
```

create.mesh.2.5D — *Create a* `mesh.2.5D` *object from the nodes locations and the connectivty matrix*

Description

Create a `mesh.2.5D` object from the nodes locations and the connectivty matrix

Usage

```
create.mesh.2.5D(nodes, triangles, order = 1)
```

Arguments

`nodes`	A #nodes-by-3 matrix specifying the locations of each node.
`triangles`	A #triangles-by-3 (when `order` = 1) or #triangles-by-6 (when `order` = 2) matrix, specifying the indices of the nodes in each triangle.
`order`	Either '1' or '2'. It specifies wether each mesh triangle should be represented by 3 nodes (the triangle' vertices) or by 6 nodes (the triangle's vertices and midpoints). These are respectively used for linear (order = 1) and quadratic (order = 2) Finite Elements. Default is `order` = 1.

Value

An object of the class `mesh.2.5D` with the following output:

- `nnodes`The #nodes in the mesh.
- `ntriangles`The #triangles in the mesh.
- `nodes`A #nodes-by-3 matrix containing the x,y and z coordinate for each point of the mesh.
- `triangles`A #triangles-by-3 (when `order` = 1) or #triangles-by-6 (when `order` = 2) matrix, specifying the indices of the nodes in each triangle.

- `order`Either ’1’ or ’2’. It specifies wether each mesh triangle should be represented by 3 nodes (the triangle’ vertices) or by 6 nodes (the triangle’s vertices and midpoints). It is passed unchanged from the input.

Examples

```
library(fdaPDE)

## Upload the hub2.5D the data
data(hub2.5D)
hub2.5D.nodes = hub2.5D$hub2.5D.nodes
hub2.5D.triangles = hub2.5D$hub2.5D.triangles

## Create mesh from nodes and connectivity matrix:
mesh = create.mesh.2.5D(nodes = hub2.5D.nodes, triangles = hub2.5D.triangles)
plot(mesh)
```

`create.mesh.2D` *Create a 2D triangular mesh*

Description

This function is a wrapper of the Triangle library (http://www.cs.cmu.edu/~quake/triangle.html). It can be used to create a triangulation of the domain of interest starting from a list of points, to be used as triangles’ vertices, and a list of segments, that define the domain boundary. The resulting mesh is a Constrained Delaunay triangulation. This is constructed in a way to preserve segments provided in the input `segments` without splitting them. This imput can be used to define the boundaries of the domain. If this imput is NULL, it generates a triangulation over the convex hull of the points. It is also possible to create a mesh.2D from the nodes locations and the connectivity matrix.

Usage

```
create.mesh.2D(nodes, nodesattributes = NA, segments = NA, holes = NA,
                       triangles = NA, order = 1, verbosity = 0)
```

Arguments

`nodes` A #nodes-by-2 matrix containing the x and y coordinates of the mesh nodes.

`nodesattributes`
A matrix with #nodes rows containing nodes’ attributes. These are passed unchanged to the output. If a node is added during the triangulation process or mesh refinement, its attributes are computed by linear interpolation using the attributes of neighboring nodes. This functionality is for instance used to compute the value of a Dirichlet boundary condition at boundary nodes added during the triangulation process.

`segments`	A #segments-by-2 matrix. Each row contains the row's indices in `nodes` of the vertices where the segment starts from and ends to. Segments are edges that are not splitted during the triangulation process. These are for instance used to define the boundaries of the domain. If this is input is NULL, it generates a triangulation over the convex hull of the points specified in `nodes`.
`holes`	A #holes-by-2 matrix containing the x and y coordinates of a point internal to each hole of the mesh. These points are used to carve holes in the triangulation, when the domain has holes.
`triangles`	A #triangles-by-3 (when `order` = 1) or #triangles-by-6 (when `order` = 2) matrix. This option is used when a triangulation is already available. It specifies the triangles giving the row's indices in `nodes` of the triangles' vertices and (when `nodes` = 2) also if the triangles' edges midpoints. The triangles' vertices and midpoints are ordered as described at https://www.cs.cmu.edu/~quake/triangle.highorder.html. In this case the function `create.mesh.2D` is used to produce a complete mesh.2D object.
`order`	Either '1' or '2'. It specifies wether each mesh triangle should be represented by 3 nodes (the triangle' vertices) or by 6 nodes (the triangle's vertices and midpoints). These are respectively used for linear (order = 1) and quadratic (order = 2) Finite Elements. Default is `order` = 1.
`verbosity`	This can be '0', '1' or '2'. It indicates the level of verbosity in the triangulation process. When `verbosity` = 0 no message is returned during the triangulation. When `verbosity` = 2 the triangulation process is described step by step by displayed messages. Default is `verbosity` = 0.

Value

An object of the class mesh.2D with the following output:

- `nodes`A #nodes-by-2 matrix containing the x and y coordinates of the mesh nodes.
- `nodesmarkers`A vector of length #nodes, with entries either '1' or '0'. An entry '1' indicates that the corresponding node is a boundary node; an entry '0' indicates that the corresponding node is not a boundary node.
- `nodesattributes`A matrix with #nodes rows containing nodes' attributes. These are passed unchanged from the input.
- `triangles`A #triangles-by-3 (when `order` = 1) or #triangles-by-6 (when `order` = 2) matrix. This option is used when a triangulation is already available. It specifies the triangles giving the indices in `nodes` of the triangles' vertices and (when `nodes` = 2) also if the triangles' edges midpoints. The triangles' vertices and midpoints are ordered as described at https://www.cs.cmu.edu/~quake/triangle.highorder.html.
- `segmentsmarker`A vector of length #segments with entries either '1' or '0'. An entry '1' indicates that the corresponding element in `segments` is a boundary segment; an entry '0' indicates that the corresponding segment is not a boundary segment.
- `edges`A #edges-by-2 matrix containing all the edges of the triangles in the output triangulation. Each row contains the row's indices in `nodes`, indicating the nodes where the edge starts from and ends to.

- `edgesmarkers`A vector of lenght #edges with entries either '1' or '0'. An entry '1' indicates that the corresponding element in `edge` is a boundary edge; an entry '0' indicates that the corresponding edge is not a boundary edge.
- `neighbors`A #triangles-by-3 matrix. Each row contains the indices of the three neighbouring triangles. An entry '-1' indicates that one edge of the triangle is a boundary edge.
- `holes`A #holes-by-2 matrix containing the x and y coordinates of a point internal to each hole of the mesh. These points are used to carve holes in the triangulation, when the domain has holes.
- `order`Either '1' or '2'. It specifies wether each mesh triangle should be represented by 3 nodes (the triangle' vertices) or by 6 nodes (the triangle's vertices and midpoints). These are respectively used for linear (order = 1) and quadratic (order = 2) Finite Elements.

See Also

`refine.mesh.2D`, `create.FEM.basis`

Examples

```
library(fdaPDE)

## Upload the quasicirle2D data
data(quasicircle2D)
boundary_nodes = quasicircle2D$boundary_nodes
boundary_segments = quasicircle2D$boundary_segments
locations = quasicircle2D$locations
data = quasicircle2D$data

## Create mesh from boundary
## if the domain is convex it is sufficient to call:
mesh = create.mesh.2D(nodes = rbind(boundary_nodes, locations))
plot(mesh)

## if the domain is not convex, pass in addition the segments the compose the boundary:
mesh = create.mesh.2D(nodes = rbind(boundary_nodes, locations), segments = boundary_segments)

## Create mesh from data locations (without knowing the boundary)
mesh = create.mesh.2D(nodes = locations)
plot(mesh)
## In this case the domain is the convex hull of the data locations.
## Do this only if you do not have any information about the shape of the domain of interest.
```

`eval.FEM` *Evaluate a FEM object at a set of point locations*

Description

It evaluates a FEM object at the specified set of locations or areal regions. The locations are used for pointwise evaluations and incidence matrix for areal evaluations. The locations and the incidence matrix cannot be both NULL or both provided.

Usage

```
eval.FEM(FEM, locations = NULL, incidence_matrix = NULL)
```

Arguments

`FEM`	A `FEM` object to be evaluated.
`locations`	A 2-columns (in 2D) or 3-columns (in 2.5D) matrix with the spatial locations where the FEM object should be evaluated.
`incidence_matrix`	In case of areal evaluations, the #regions-by-#elements incidence matrix defining the regions where the FEM object should be evaluated.

Value

A vector or a matrix of numeric evaluations of the `FEM` object. If the `FEM` object contains multiple finite element functions the output is a matrix, and each row corresponds to the location (or areal region) where the evaluation has been taken, while each column corresponds to the function evaluated.

References

- Sangalli, L. M., Ramsay, J. O., & Ramsay, T. O. (2013). Spatial spline regression models. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 75(4), 681-703.
- Azzimonti, L., Sangalli, L. M., Secchi, P., Domanin, M., & Nobile, F. (2015). Blood flow velocity field estimation via spatial regression with PDE penalization. Journal of the American Statistical Association, 110(511), 1057-1071.

Examples

```
library(fdaPDE)
## Upload the horseshoe2D data
data(horseshoe2D)
boundary_nodes = horseshoe2D$boundary_nodes
boundary_segments = horseshoe2D$boundary_segments
locations = horseshoe2D$locations

## Create the 2D mesh
mesh = create.mesh.2D(nodes = rbind(boundary_nodes, locations), segments = boundary_segments)
## Create the FEM basis
FEMbasis = create.FEM.basis(mesh)
## Compute the coeff vector evaluating the desired function at the mesh nodes
## In this case we consider the fs.test() function introduced by Wood et al. 2008
coeff = fs.test(mesh$nodes[,1], mesh$nodes[,2])
## Create the FEM object
FEMfunction = FEM(coeff, FEMbasis)

## Evaluate the finite element function in the location (1,0.5)
eval.FEM(FEMfunction, locations = matrix(c(1, 0.5), ncol = 2))

## Evaluate the mean of the finite element function over the fifth triangle of the mesh
```

```
incidence_matrix = matrix(0, ncol = nrow(mesh$triangles))
incidence_matrix[1,5] = 1
eval.FEM(FEMfunction, incidence_matrix = incidence_matrix)
```

`fdaPDE-deprecated` *Deprecated Functions*

Description

These functions are Deprecated in this release of fdaPDE, they will be marked as Defunct and removed in a future version.

Usage

```
R_mass(FEMbasis)

R_stiff(FEMbasis)

R_smooth.FEM.basis(
  locations,
  observations,
  FEMbasis,
  lambda,
  covariates = NULL,
  GCV
)

R_eval.FEM.basis(FEMbasis, locations, nderivs = matrix(0, 1, 2))

R_eval.FEM(FEM, locations)

smooth.FEM.basis(
  locations = NULL,
  observations,
  FEMbasis,
  lambda,
  covariates = NULL,
  BC = NULL,
  GCV = FALSE,
  CPP_CODE = TRUE
)

smooth.FEM.PDE.basis(
  locations = NULL,
  observations,
  FEMbasis,
  lambda,
  PDE_parameters,
  covariates = NULL,
  BC = NULL,
  GCV = FALSE,
  CPP_CODE = TRUE
)

smooth.FEM.PDE.sv.basis(
  locations = NULL,
  observations,
  FEMbasis,
  lambda,
  PDE_parameters,
  covariates = NULL,
  BC = NULL,
  GCV = FALSE,
  CPP_CODE = TRUE
)

create.MESH.2D(nodes, nodesattributes = NA, segments = NA, holes = NA,
                     triangles = NA, order = 1, verbosity = 0)

refine.MESH.2D(mesh, minimum_angle, maximum_area, delaunay, verbosity)

## S3 method for class 'MESH2D'
plot(x, ...)
```

Arguments

`FEMbasis`	A `FEMbasis` object describing the Finite Element basis, as created by `create.FEM.basis`.
`locations`	A #observations-by-2 matrix where each row specifies the spatial coordinates of the corresponding observations in the vector `observations`.
`observations`	A #observations vector with the observed data values over the domain. The locations of the observations can be specified with the `locations` argument. Otherwise if only the vector of observations is given, these are consider to be located in the corresponding node in the table nodes of the mesh. In this last case, an `NA` value in the observations vector indicates that there is no observation associated to the corresponding node.
`lambda`	A scalar or vector of smoothing parameters.
`covariates`	A #observations-by-#covariates matrix where each row represents the covariates associated with the corresponding observed data value in `observations`.
`GCV`	Boolean. If `TRUE` the following quantities are computed: the trace of the smoothing matrix, the estimated error standard deviation, and the Generalized Cross Validation criterion, for each value of the smoothing parameter specified in `lambda`.
`nderivs`	A vector of lenght 2 specifying the order of the partial derivatives of the bases to be evaluated. The vectors' entries can be 0,1 or 2, where 0 indicates that only the basis functions, and not their derivatives, should be evaluated.

`FEM`	A `FEM` object to be evaluated
`BC`	vector with the Dirichlet boundary conditions to be applied.
`CPP_CODE`	Boolean, indicates whether C++ implementation ha sto be used or not.
`PDE_parameters`	A list specifying the parameters of the elliptic PDE in the regularizing term.
`nodes`	A #nodes-by-2 matrix containing the x and y coordinates of the mesh nodes.
`nodesattributes`	A matrix with #nodes rows containing nodes' attributes. These are passed unchanged to the output. If a node is added during the triangulation process or mesh refinement, its attributes are computed by linear interpolation using the attributes of neighboring nodes. This functionality is for instance used to compute the value of a Dirichlet boundary condition at boundary nodes added during the triangulation process.
`segments`	A #segments-by-2 matrix. Each row contains the row's indices in `nodes` of the vertices where the segment starts from and ends to. Segments are edges that are not splitted during the triangulation process. These are for instance used to define the boundaries of the domain. If this is input is NULL, it generates a triangulation over the convex hull of the points specified in `nodes`.
`holes`	A #holes-by-2 matrix containing the x and y coordinates of a point internal to each hole of the mesh. These points are used to carve holes in the triangulation, when the domain has holes.
`triangles`	A #triangles-by-3 (when `order` = 1) or #triangles-by-6 (when `order` = 2) matrix. This option is used when a triangulation is already available. It specifies the triangles giving the row's indices in `nodes` of the triangles' vertices and (when `nodes` = 2) also if the triangles' edges midpoints. The triangles' vertices and midpoints are ordered as described at https://www.cs.cmu.edu/~quake/triangle.highorder.html. In this case the function `create.MESH.2D` is used to produce a complete MESH2D object.
`order`	Either '1' or '2'. It specifies wether each mesh triangle should be represented by 3 nodes (the triangle' vertices) or by 6 nodes (the triangle's vertices and midpoints). These are respectively used for linear (order = 1) and quadratic (order = 2) Finite Elements. Default is `order` = 1.
`verbosity`	This can be '0', '1' or '2'. It indicates the level of verbosity in the triangulation process. When `verbosity` = 0 no message is returned during the triangulation. When `verbosity` = 2 the triangulation process is described step by step by displayed messages. Default is `verbosity` = 0.
`mesh`	A MESH2D object representing the triangular mesh, created by create.MESH.2D.
`minimum_angle`	A scalar specifying a minimun value for the triangles angles.
`maximum_area`	A scalar specifying a maximum value for the triangles areas.
`delaunay`	A boolean parameter indicating whether or not the output mesh should satisfy the Delaunay condition.
`x`	A MESH2D object defining the triangular mesh, as generated by `create.Mesh.2D` or `refine.Mesh.2D`.
`...`	Arguments representing graphical options to be passed to par.

Value

A square matrix with the integrals of all the basis' functions pairwise products. The dimension of the matrix is equal to the number of the nodes of the mesh.

A square matrix with the integrals of all the basis functions' gradients pairwise dot products. The dimension of the matrix is equal to the number of the nodes of the mesh.

A list with the following quantities:

`fit.FEM`	A `FEM` object that represents the fitted spatial field.
`PDEmisfit.FEM`	A `FEM` object that represents the Laplacian of the estimated spatial field.
`beta`	If covariates is not `NULL`, a vector of length #covariates with the regression coefficients associated with each covariate.
`edf`	If GCV is `TRUE`, a scalar or vector with the trace of the smoothing matrix for each value of the smoothing parameter specified in `lambda`.
`stderr`	If GCV is `TRUE`, a scalar or vector with the estimate of the standard deviation of the error for each value of the smoothing parameter specified in `lambda`.
`GCV`	If GCV is `TRUE`, a scalar or vector with the value of the GCV criterion for each value of the smoothing parameter specified in `lambda`.

A matrix of basis function values. Each row indicates the location where the evaluation has been taken, the column indicates the basis function evaluated

A matrix of numeric evaluations of the `FEM` object. Each row indicates the location where the evaluation has been taken, the column indicates the function evaluated.

An object of the class MESH2D with the following output:

`nodes`	A #nodes-by-2 matrix containing the x and y coordinates of the mesh nodes.
`nodesmarkers`	A vector of length #nodes, with entries either '1' or '0'. An entry '1' indicates that the corresponding node is a boundary node; an entry '0' indicates that the corresponding node is not a boundary node.
`nodesattributes`	nodesattributes A matrix with #nodes rows containing nodes' attributes. These are passed unchanged to the output. If a node is added during the triangulation process or mesh refinement, its attributes are computed by linear interpolation using the attributes of neighboring nodes. This functionality is for instance used to compute the value of a Dirichlet boundary condition at boundary nodes added during the triangulation process.
`triangles`	A #triangles-by-3 (when `order` = 1) or #triangles-by-6 (when `order` = 2) matrix. This option is used when a triangulation is already available. It specifies the triangles giving the indices in `nodes` of the triangles' vertices and (when `nodes` = 2) also if the triangles' edges midpoints. The triangles' vertices and midpoints are ordered as described at https://www.cs.cmu.edu/~quake/triangle.highorder.html.
`segmentsmarker`	A vector of length #segments with entries either '1' or '0'. An entry '1' indicates that the corresponding element in `segments` is a boundary segment; an entry '0' indicates that the corresponding segment is not a boundary segment.

`edges`	A #edges-by-2 matrix containing all the edges of the triangles in the output triangulation. Each row contains the row's indices in `nodes`, indicating the nodes where the edge starts from and ends to.
`edgesmarkers`	A vector of lenght #edges with entries either '1' or '0'. An entry '1' indicates that the corresponding element in `edge` is a boundary edge; an entry '0' indicates that the corresponding edge is not a boundary edge.
`neighbors`	A #triangles-by-3 matrix. Each row contains the indices of the three neighbouring triangles. An entry '-1' indicates that one edge of the triangle is a boundary edge.
`holes`	A #holes-by-2 matrix containing the `x` and `y` coordinates of a point internal to each hole of the mesh. These points are used to carve holes in the triangulation, when the domain has holes.
`order`	Either '1' or '2'. It specifies wether each mesh triangle should be represented by 3 nodes (the triangle' vertices) or by 6 nodes (the triangle's vertices and midpoints). These are respectively used for linear (order = 1) and quadratic (order = 2) Finite Elements. Default is `order` = 1.

A MESH2D object representing the refined triangular mesh, with the following output:

`nodes`	A #nodes-by-2 matrix containing the `x` and `y` coordinates of the mesh nodes.
`nodesmarkers`	A vector of length #nodes, with entries either '1' or '0'. An entry '1' indicates that the corresponding node is a boundary node; an entry '0' indicates that the corresponding node is not a boundary node.
`nodesattributes`	nodesattributes A matrix with #nodes rows containing nodes' attributes. These are passed unchanged to the output. If a node is added during the triangulation process or mesh refinement, its attributes are computed by linear interpolation using the attributes of neighboring nodes. This functionality is for instance used to compute the value of a Dirichlet boundary condition at boundary nodes added during the triangulation process.
`triangles`	A #triangles-by-3 (when `order` = 1) or #triangles-by-6 (when `order` = 2) matrix. This option is used when a triangulation is already available. It specifies the triangles giving the row's indices in `nodes` of the triangles' vertices and (when `nodes` = 2) also if the triangles' edges midpoints. The triangles' vertices and midpoints are ordered as described at https://www.cs.cmu.edu/~quake/triangle.highorder.html.
`edges`	A #edges-by-2 matrix. Each row contains the row's indices of the nodes where the edge starts from and ends to.
`edgesmarkers`	A vector of lenght #edges with entries either '1' or '0'. An entry '1' indicates that the corresponding element in `edge` is a boundary edge; an entry '0' indicates that the corresponding edge is not a boundary edge.
`neighbors`	A #triangles-by-3 matrix. Each row contains the indices of the three neighbouring triangles. An entry '-1' indicates that one edge of the triangle is a boundary edge.
`holes`	A #holes-by-2 matrix containing the `x` and `y` coordinates of a point internal to each hole of the mesh. These points are used to carve holes in the triangulation, when the domain has holes.

`order`	Either '1' or '2'. It specifies wether each mesh triangle should be represented by 3 nodes (the triangle' vertices) or by 6 nodes (the triangle's vertices and midpoints). These are respectively used for linear (order = 1) and quadratic (order = 2) Finite Elements. Default is `order` = 1.

`FEM` *Define a surface or spatial field by a Finite Element basis expansion*

Description

This function defines a FEM object.

Usage

```
FEM(coeff,FEMbasis)
```

Arguments

`coeff`	A vector or a matrix containing the coefficients for the Finite Element basis expansion. The number of rows (or the vector's length) corresponds to the number of basis in `FEMbasis`. The number of columns corresponds to the number of functions.
`FEMbasis`	A `FEMbasis` object defining the Finite Element basis, created by create.FEM.basis.

Value

An `FEM` object. This contains a list with components `coeff` and `FEMbasis`.

Examples

```
library(fdaPDE)
## Upload the horseshoe2D data
data(horseshoe2D)
boundary_nodes = horseshoe2D$boundary_nodes
boundary_segments = horseshoe2D$boundary_segments
locations = horseshoe2D$locations

## Create the 2D mesh
mesh = create.mesh.2D(nodes = rbind(boundary_nodes, locations), segments = boundary_segments)
## Create the FEM basis
FEMbasis = create.FEM.basis(mesh)
## Compute the coeff vector evaluating the desired function at the mesh nodes
## In this case we consider the fs.test() function introduced by Wood et al. 2008
coeff = fs.test(mesh$nodes[,1], mesh$nodes[,2])
## Create the FEM object
FEMfunction = FEM(coeff, FEMbasis)
## Plot it
plot(FEMfunction)
```

`FPCA.FEM`	*Smooth Functional Principal Component Analysis*

Description

This function implements a smooth functional principal component analysis for data defined over a planar mesh, or a smooth manifold. For details on the model see Lila et al. 2016.

Usage

```
FPCA.FEM(locations = NULL, datamatrix, FEMbasis, lambda, nPC=1,
      validation = NULL, NFolds = 5, GCVmethod = "Stochastic", nrealizations = 100)
```

Arguments

`locations`	A #observations-by-2 matrix in the 2D case and #observations-by-3 matrix in the 2.5D case, where each row specifies the spatial coordinates `x` and `y` (and `z` in 2.5D) of the corresponding observation in the `datamatrix`. If the locations of the observations coincide with (or are a subset of) the nodes of the mesh in the `FEMbasis`, leave the parameter `locations = NULL` for a faster implementation.
`datamatrix`	A matrix of dimensions #samples-by-#locations with the observed data values over the domain for each sample. The datamatrix needs to have zero mean. If the `locations` argument is left `NULL` the datamatrix has to be dimensions #samples-by-#nodes where #nodes is the number of nodes of the mesh in the FEMbasis. In this case, each observation is associated to the corresponding node in the mesh. If the data are observed only on a subset of the mesh nodes, fill with `NA` the values of the `datamatrix` in correspondence of unobserved data.
`FEMbasis`	A `FEMbasis` object describing the Finite Element basis, as created by `create.FEM.basis`.
`lambda`	A scalar or vector of smoothing parameters.
`nPC`	An integer specifying the number of Principal Components to compute.
`validation`	A string specifying the type of validation to perform. If `lambda` is a vector, it has to be specified as `"GCV"` or `"KFold"`. This parameter specify which method of cross-validation is used to select the best parameter `lambda` among those values of the smoothing parameter specified in `lambda` for each Principal Component.
`NFolds`	This parameter is used only in case `validation = "KFold"`. It is an integer specifying the number of folds to use if the KFold cross-validation method for the selection of the best parameter `lambda` is chosen. Default value is 5.
`GCVmethod`	This parameter is considered only when `validation = "GCV"`. It can be either "Exact" or "Stochastic". If set to "Exact" the algoritm performs an exact (but possibly slow) computation of the GCV index. If set to "Stochastic" the GCV is approximated by a stochastic algorithm.
`nrealizations`	The number of realizations to be used in the stochastic algorithm for the estimation of GCV.

Value

A list with the following variables:

- `loadings.FEM`A `FEM` object that represents the L^2-normalized functional loadings for each Principal Component computed.
- `scores`A #samples-by-#PrincipalComponents matrix that represents the unnormalized scores or PC vectors.
- `lambda`A vector of length #PrincipalComponents with the values of the smoothing parameter `lambda` chosen for that Principal Component.
- `variance_explained`A vector of length #PrincipalComponents where each value represent the variance explained by that component.
- `cumsum_percentage`A vector of length #PrincipalComponents containing the cumulative percentage of the variance explained by the first components.

References

Lila, E., Aston, J.A.D., Sangalli, L.M., 2016a. Smooth Principal Component Analysis over two-dimensional manifolds with an application to neuroimaging. Ann. Appl. Stat., 10(4), pp. 1854-1879.

Examples

```
library(fdaPDE)

## Load the hub data
data(hub2.5D)
hub2.5D.nodes = hub2.5D$hub2.5D.nodes
hub2.5D.triangles = hub2.5D$hub2.5D.triangles

mesh = create.mesh.2.5D(nodes = hub2.5D.nodes, triangles = hub2.5D.triangles)
## Create the Finite Element basis
FEMbasis = create.FEM.basis(mesh)
## Create a datamatrix
datamatrix = NULL
for(ii in 1:50){
  a1 = rnorm(1, mean = 1, sd = 1)
  a2 = rnorm(1, mean = 1, sd = 1)
  a3 = rnorm(1, mean = 1, sd = 1)

  func_evaluation = numeric(mesh$nnodes)
  for (i in 0:(mesh$nnodes-1)){
    func_evaluation[i+1] = a1* sin(2*pi*mesh$nodes[i+1,1]) +
                           a2* sin(2*pi*mesh$nodes[i+1,2]) +
                           a3* sin(2*pi*mesh$nodes[i+1,3]) + 1
  }
  data = func_evaluation + rnorm(mesh$nnodes, mean = 0, sd = 0.5)
  datamatrix = rbind(datamatrix, data)
}
## Compute the mean of the datamatrix and subtract it to the data
data_bar = colMeans(datamatrix)
```

```
data_demean = matrix(rep(data_bar,50), nrow=50, byrow=TRUE)

datamatrix_demeaned = datamatrix - data_demean
## Set the smoothing parameter lambda
lambda = 0.00375
## Estimate the first 2 Principal Components
FPCA_solution = FPCA.FEM(datamatrix = datamatrix_demeaned,
                      FEMbasis = FEMbasis, lambda = lambda, nPC = 2)

## Plot the functional loadings of the estimated Principal Components
plot(FPCA_solution$loadings.FEM)
```

`fs.test` *FELSPLINE test function*

Description

Implements a finite area test function based on one proposed by Tim Ramsay (2002) proposed by Simon Wood (2008).

Usage

```
fs.test(x, y, r0 = 0.1, r = 0.5, l = 3, b = 1)
```

Arguments

`x, y`	Points at which to evaluate the test function.
`r0`	The test domain is a sort of bent sausage. This is the radius of the inner bend.
`r`	The radius of the curve at the centre of the sausage.
`l`	The length of an arm of the sausage.
`b`	The rate at which the function increases per unit increase in distance along the centre line of the sausage.

Value

Returns function evaluations, or NAs for points outside the horseshoe domain.

References

- Ramsay, T. 2002. Spline smoothing over difficult regions. J.R.Statist. Soc. B 64(2):307-319
- Wood, S. N., Bravington, M. V., & Hedley, S. L. (2008). Soap film smoothing. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 70(5), 931-955.

Examples

```
library(fdaPDE)

## Upload the horseshoe2D data
data(horseshoe2D)
boundary_nodes = horseshoe2D$boundary_nodes
boundary_segments = horseshoe2D$boundary_segments
locations = horseshoe2D$locations

## Create the 2D mesh
mesh = create.mesh.2D(nodes = rbind(boundary_nodes, locations), segments = boundary_segments)
## Create the FEM basis
FEMbasis = create.FEM.basis(mesh)
## Compute the coeff vector evaluating the desired function at the mesh nodes
## In this case we consider the fs.test() function introduced by Wood et al. 2008
coeff = fs.test(mesh$nodes[,1], mesh$nodes[,2])
## Create the FEM object
FEMfunction = FEM(coeff, FEMbasis)
## Plot it
plot(FEMfunction)
```

`horseshoe2D` *Horseshoe domain*

Description

The boundary and interior nodes and connectivity matrix of a triangular mesh of the horseshoe domain. This dataset can be used to create a `MESH.2D` object with the function `create.MESH.2D`. The variables are:

- boundary_nodes. The nodes in the boundary.
- boundary_segments. The ssegments of the boundary.
- locations. The interior nodes of the mesh.

Usage

```
data(horseshoe2D)
```

`hub2.5D` *Hub domain*

Description

The nodes and connectivity matrix of a triangular mesh of a manifold representing a hub geometry. This dataset can be used to create a `MESH.2.5D` object with the function `create.MESH.2.5D`. The variables are:

- hub2.5D.nodes. The nodes of the mesh.
- hub2.5D.triangles. The triangles of the mesh.

Usage

```
data(hub2.5D)
```

---

image.FEM *Image Plot of a 2D FEM object*

Description

Image plot of a `FEM` object, generated by the function `FEM` or returned by `smooth.FEM` and `FPCA.FEM`. Only FEM objects defined over a 2D mesh can be plotted with this method.

Usage

```
## S3 method for class 'FEM'
image(x, num_refinements, ...)
```

Arguments

`x`	A 2D-mesh `FEM` object.
`num_refinements`	A natural number specifying how many bisections should by applied to each triangular element for plotting purposes. This functionality is useful where a discretization with 2nd order Finite Element is applied.
`...`	Arguments representing graphical options to be passed to plot3d.

See Also

`FEM` `plot.FEM`

Examples

```
library(fdaPDE)
## Upload the horseshoe2D data
data(horseshoe2D)
boundary_nodes = horseshoe2D$boundary_nodes
boundary_segments = horseshoe2D$boundary_segments
locations = horseshoe2D$locations

## Create the 2D mesh
mesh = create.mesh.2D(nodes = rbind(boundary_nodes, locations), segments = boundary_segments)
## Create the FEM basis
FEMbasis = create.FEM.basis(mesh)
## Compute the coeff vector evaluating the desired function at the mesh nodes
## In this case we consider the fs.test() function introduced by Wood et al. 2008
coeff = fs.test(mesh$nodes[,1], mesh$nodes[,2])
## Create the FEM object
FEMfunction = FEM(coeff, FEMbasis)
```

```
## Plot the FEM function
image(FEMfunction)
```

---

plot.FEM *Plot a* FEM *object*

Description

Three-dimensional plot of a FEM object, generated by FEM or returned by smooth.FEM or FPCA.FEM. If the mesh of the FEMbasis component is of class mesh.2D both the 3rd axis and the color represent the value of the coefficients for the Finite Element basis expansion (coeff component of the FEM object).

Usage

```
## S3 method for class 'FEM'
plot(x, num_refinements, ...)
```

Arguments

x	A FEM object.
num_refinements	A natural number specifying how many bisections should be applied to each triangular element for plotting purposes. This functionality is useful where a discretization with 2nd order Finite Element is applied. This parameter can be specified only when a FEM object defined over a 2D mesh is plotted.
...	Arguments representing graphical options to be passed to plot3d.

See Also

FEM, image.FEM

Examples

```
library(fdaPDE)
## Upload the horseshoe2D data
data(horseshoe2D)
boundary_nodes = horseshoe2D$boundary_nodes
boundary_segments = horseshoe2D$boundary_segments
locations = horseshoe2D$locations

## Create the 2D mesh
mesh = create.mesh.2D(nodes = rbind(boundary_nodes, locations), segments = boundary_segments)
## Create the FEM basis
FEMbasis = create.FEM.basis(mesh)
## Compute the coeff vector evaluating the desired function at the mesh nodes
## In this case we consider the fs.test() function introduced by Wood et al. 2008
coeff = fs.test(mesh$nodes[,1], mesh$nodes[,2])
```

```
## Create the FEM object
FEMfunction = FEM(coeff, FEMbasis)

## Plot the FEM function
plot(FEMfunction)
```

`plot.mesh.2.5D` *Plot a mesh.2.5D object*

Description

Plot the triangulation of a `mesh.2.5D` object, generated by `create.mesh.2.5D`

Usage

```
## S3 method for class 'mesh.2.5D'
plot(x, ...)
```

Arguments

`x`	A `mesh.2.5D` object generated by `create.mesh.2.5D`.
`...`	Arguments representing graphical options to be passed to par.

Examples

```
library(fdaPDE)

## Upload the hub2.5D the data
data(hub2.5D)
hub2.5D.nodes = hub2.5D$hub2.5D.nodes
hub2.5D.triangles = hub2.5D$hub2.5D.triangles

## Create mesh
mesh = create.mesh.2.5D(nodes = hub2.5D.nodes, triangles = hub2.5D.triangles)
plot(mesh)
```

`plot.mesh.2D` *Plot a mesh.2D object*

Description

Plot a mesh.2D object, generated by `create.mesh.2D` or `refine.mesh.2D`.

Usage

```
## S3 method for class 'mesh.2D'
plot(x, ...)
```

Arguments

`x`	A `mesh.2D` object defining the triangular mesh, as generated by `create.mesh.2D` or `refine.mesh.2D`.
`...`	Arguments representing graphical options to be passed to par.

Examples

```
library(fdaPDE)

## Upload the quasicirle2D data
data(quasicircle2D)
boundary_nodes = quasicircle2D$boundary_nodes
boundary_segments = quasicircle2D$boundary_segments
locations = quasicircle2D$locations
data = quasicircle2D$data

## Create mesh
mesh = create.mesh.2D(nodes = rbind(boundary_nodes, locations), segments = boundary_segments)

## Plot the mesh
plot(mesh)
```

`quasicircle2D` *Quasicircle2D domain*

Description

The boundary and interior nodes and connectivity matrix of a triangular mesh of a quasicircular domain, together with a non-stationary field observed over the nodes of the mesh. This dataset can be used to create a `MESH.2D` object with the function `create.MESH.2D` and to test the smooth.FEM function. The variables are:

- boundary_nodes. The nodes in the boundary.
- boundary_segments. The ssegments of the boundary.
- locations. The interior nodes of the mesh.
- data. The vector of observations.

Usage

```
data(quasicircle2D)
```

`quasicircle2Dareal` *Quasicircle2Dareal domain*

Description

The mesh of a quasicircular domain, together with a non-stationary field observed over seven circular subdomains and the incindence matrix defining the subdomains used by Azzimonti et. al 2015. This dataset can be used to test the smooth.FEM function for areal data. The variables are:

- incidence_matrix. The 7-by-630 incidence matrix.
- data. The vector of observations.
- mesh. The mesh for areal data.

Usage

```
data(quasicircle2Dareal)
```

References

Azzimonti, L., Sangalli, L. M., Secchi, P., Domanin, M., & Nobile, F. (2015). Blood flow velocity field estimation via spatial regression with PDE penalization. Journal of the American Statistical Association, 110(511), 1057-1071.

`refine.mesh.2D` *Refine a 2D triangular mesh*

Description

This function refines a Constrained Delaunay triangulation into a Conforming Delaunay triangulation. This is a wrapper of the Triangle library (http://www.cs.cmu.edu/~quake/triangle.html). It can be used to refine a mesh previously created with create.mesh.2D. The algorithm can add Steiner points (points through which the `segments` are splitted) in order to meet the imposed refinement conditions.

Usage

```
refine.mesh.2D(mesh, minimum_angle, maximum_area, delaunay, verbosity)
```

Arguments

`mesh`	A mesh.2D object representing the triangular mesh, created by create.mesh.2D.
`minimum_angle`	A scalar specifying a minimun value for the triangles angles.
`maximum_area`	A scalar specifying a maximum value for the triangles areas.
`delaunay`	A boolean parameter indicating whether or not the output mesh should satisfy the Delaunay condition.
`verbosity`	This can be '0', '1' or '2'. It indicates the level of verbosity in the triangulation process.

Value

A mesh.2D object representing the refined triangular mesh, with the following output:

- `nodes`A #nodes-by-2 matrix containing the x and y coordinates of the mesh nodes.
- `nodesmarkers`A vector of length #nodes, with entries either '1' or '0'. An entry '1' indicates that the corresponding node is a boundary node; an entry '0' indicates that the corresponding node is not a boundary node.
- `nodesattributes`nodesattributes A matrix with #nodes rows containing nodes' attributes. These are passed unchanged to the output. If a node is added during the triangulation process or mesh refinement, its attributes are computed by linear interpolation using the attributes of neighboring nodes. This functionality is for instance used to compute the value of a Dirichlet boundary condition at boundary nodes added during the triangulation process.
- `triangles`A #triangles-by-3 (when `order` = 1) or #triangles-by-6 (when `order` = 2) matrix.
- `edges`A #edges-by-2 matrix. Each row contains the row's indices of the nodes where the edge starts from and ends to.
- `edgesmarkers`A vector of lenght #edges with entries either '1' or '0'. An entry '1' indicates that the corresponding element in `edge` is a boundary edge; an entry '0' indicates that the corresponding edge is not a boundary edge.
- `neighbors`A #triangles-by-3 matrix. Each row contains the indices of the three neighbouring triangles. An entry '-1' indicates that one edge of the triangle is a boundary edge.
- `holes`A #holes-by-2 matrix containing the x and y coordinates of a point internal to each hole of the mesh. These points are used to carve holes in the triangulation, when the domain has holes.
- `order`Either '1' or '2'. It specifies wether each mesh triangle should be represented by 3 nodes (the triangle' vertices) or by 6 nodes (the triangle's vertices and midpoints). These are respectively used for linear (order = 1) and quadratic (order = 2) Finite Elements.

See Also

`create.mesh.2D`, `create.FEM.basis`

Examples

```
library(fdaPDE)

## Upload the quasicircle2D data
data(quasicircle2D)
boundary_nodes = quasicircle2D$boundary_nodes
boundary_segments = quasicircle2D$boundary_segments
locations = quasicircle2D$locations
data = quasicircle2D$data

## Create mesh from boundary:
mesh = create.mesh.2D(nodes = boundary_nodes, segments = boundary_segments)
plot(mesh)
## Refine the mesh with the maximum area criterion:
finemesh = refine.mesh.2D(mesh = mesh, maximum_area = 0.1)
plot(finemesh)
```

```
## Refine the mesh with the minimum angle criterion:
finemesh2 = refine.mesh.2D(mesh = mesh, minimum_angle = 30)
plot(finemesh2)
```

smooth.FEM — *Spatial regression with differential regularization*

Description

This function implements a spatial regression model with differential regularization. The regularizing term involves a Partial Differential Equation (PDE). In the simplest case the PDE involves only the Laplacian of the spatial field, that induces an isotropic smoothing. When prior information about the anisotropy or non-stationarity is available the PDE involves a general second order linear differential operator with possibly space-varying coefficients. The technique accurately handle data distributed over irregularly shaped domains. Moreover, various conditions can be imposed at the domain boundaries.

Usage

```
smooth.FEM(locations = NULL, observations, FEMbasis, lambda,
     covariates = NULL, PDE_parameters=NULL, incidence_matrix = NULL,
     BC = NULL, GCV = FALSE, GCVmethod = "Stochastic", nrealizations = 100)
```

Arguments

`locations`	A #observations-by-2 matrix in the 2D case and #observations-by-3 matrix in the 2.5D case, where each row specifies the spatial coordinates `x` and `y` (and `z` in 2.5D) of the corresponding observation in the vector `observations`. If the locations of the observations coincide with (or are a subset of) the nodes of the mesh in the `FEMbasis`, leave the parameter `locations = NULL` for a faster implementation.
`observations`	A vector of length #observations with the observed data values over the domain. If the `locations` argument is left NULL the vector of the observations have to be of length #nodes of the mesh in the FEMbasis. In this case, each observation is associated to the corresponding node in the mesh. If the observations are observed only on a subset of the mesh nodes, fill with `NA` the values of the vector `observations` in correspondence of unobserved data.
`FEMbasis`	A `FEMbasis` object describing the Finite Element basis, as created by `create.FEM.basis`.
`lambda`	A scalar or vector of smoothing parameters.
`covariates`	A #observations-by-#covariates matrix where each row represents the covariates associated with the corresponding observed data value in `observations` and each column is a different covariate.
`PDE_parameters`	A list specifying the parameters of the PDE in the regularizing term. Default is NULL, i.e. regularization is by means of the Laplacian (stationary, isotropic case). If the coefficients of the PDE are constant over the domain `PDE_parameters` must contain:

- `K`, a 2-by-2 matrix of diffusion coefficients. This induces an anisotropic smoothing with a preferential direction that corresponds to the first eigenvector of the diffusion matrix K;
- `b`, a vector of length 2 of advection coefficients. This induces a smoothing only in the direction specified by the vector `b`;
- `c`, a scalar reaction coefficient. `c` induces a shrinkage of the surface to zero.

If the coefficients of the PDE are space-varying `PDE_parameters` must contain:

- `K`, a function that for each spatial location in the spatial domain (indicated by the vector of the 2 spatial coordinates) returns a 2-by-2 matrix of diffusion coefficients. The function must support recycling for efficiency reasons, thus if the input parameter is a #point-by-2 matrix, the output should be an array with dimensions 2-by-2-by-#points.
- `b`, a function that for each spatial location in the spatial domain returns a vector of length 2 of transport coefficients. The function must support recycling for efficiency reasons, thus if the input parameter is a #point-by-2 matrix, the output should be a matrix with dimensions 2-by-#points;
- `c`, a function that for each spatial location in the spatial domain returns a scalar reaction coefficient. The function must support recycling for efficiency reasons, thus if the input parameter is a #point-by-2 matrix, the output should be a vector with length #points;
- `u`, a function that for each spatial location in the spatial domain returns a scalar reaction coefficient. `u` induces a reaction effect. The function must support recycling for efficiency reasons, thus if the input parameter is a #point-by-2 matrix, the output should be a vector with length #points.

For 2.5D, only the Laplacian is available (`PDE_parameters=NULL`).

`incidence_matrix`
A #regions-by-#triangles matrix where the element (i,j) equals 1 if the j-th triangle is in the i-th region and 0 otherwise. This is needed only for areal data. In case of pointwise data, this parameter is set to `NULL`.

`BC` A list with two vectors: `BC_indices`, a vector with the indices in `nodes` of boundary nodes where a Dirichlet Boundary Condition should be applied; `BC_values`, a vector with the values that the spatial field must take at the nodes indicated in `BC_indices`.

`GCV` Boolean. If `TRUE` the following quantities are computed: the trace of the smoothing matrix, the estimated error standard deviation, and the Generalized Cross Validation criterion, for each value of the smoothing parameter specified in `lambda`.

`GCVmethod` This parameter is considered only when `GCV=TRUE`. It can be either "Exact" or "Stochastic". If set to "Exact" the algoritm performs an exact (but possibly slow) computation of the GCV index. If set to "Stochastic" the GCV is approximated by a stochastic algorithm.

`nrealizations` This parameter is considered only when `GCV=TRUE` and `GCVmethod = "Stochastic"`. It is a positive integer that represents the number of uniform random variables used in stochastic GCV computation.

Value

A list with the following variables:

- `fit.FEM`A `FEM` object that represents the fitted spatial field.
- `PDEmisfit.FEM`A `FEM` object that represents the Laplacian of the estimated spatial field.
- `beta`If covariates is not `NULL`, a matrix with number of rows equal to the number of covariates and numer of columns equal to length of lambda. The `j`th column represents the vector of regression coefficients when the smoothing parameter is equal to `lambda[j]`.
- `edf`If GCV is `TRUE`, a scalar or vector with the trace of the smoothing matrix for each value of the smoothing parameter specified in `lambda`.
- `stderr`If GCV is `TRUE`, a scalar or vector with the estimate of the standard deviation of the error for each value of the smoothing parameter specified in `lambda`.
- `GCV`If GCV is `TRUE`, a scalar or vector with the value of the GCV criterion for each value of the smoothing parameter specified in `lambda`.

References

- Sangalli, L. M., Ramsay, J. O., Ramsay, T. O. (2013). Spatial spline regression models. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 75(4), 681-703.
- Azzimonti, L., Sangalli, L. M., Secchi, P., Domanin, M., Nobile, F. (2015). Blood flow velocity field estimation via spatial regression with PDE penalization. Journal of the American Statistical Association, 110(511), 1057-1071.

Examples

```
library(fdaPDE)

#### No prior information about anysotropy/non-stationarity (laplacian smoothing) ####
data(horseshoe2D)
boundary_nodes = horseshoe2D$boundary_nodes
boundary_segments = horseshoe2D$boundary_segments
locations = horseshoe2D$locations

mesh = create.mesh.2D(nodes = rbind(boundary_nodes, locations), segments = boundary_segments)
FEMbasis = create.FEM.basis(mesh)
lambda = 10^-1
# no covariate
data = fs.test(mesh$nodes[,1], mesh$nodes[,2]) + rnorm(nrow(mesh$nodes), sd = 0.5)

solution = smooth.FEM(observations = data, FEMbasis = FEMbasis, lambda = lambda)
plot(solution$fit.FEM)

# with covariates
covariate = covs.test(mesh$nodes[,1], mesh$nodes[,2])
data = fs.test(mesh$nodes[,1], mesh$nodes[,2]) +
             2*covariate + rnorm(nrow(mesh$nodes), sd = 0.5)

solution = smooth.FEM(observations = data, covariates = covariate,
                      FEMbasis = FEMbasis, lambda = lambda)
```

```
# beta estimate:
solution$beta
# non-parametric estimate:
plot(solution$fit.FEM)

# Choose lambda with GCV:
lambda = 10^(-2:2)
solution = smooth.FEM(observations = data,
                            covariates = covariate,
                            FEMbasis = FEMbasis,
                            lambda = lambda,
                            GCV = TRUE)
bestLambda = lambda[which.min(solution$GCV)]


#### Smoothing with prior information about anysotropy/non-stationarity and boundary conditions ####
# See Azzimonti et al. for reference to the current exemple
data(quasicircle2D)
boundary_nodes = quasicircle2D$boundary_nodes
boundary_segments = quasicircle2D$boundary_segments
locations = quasicircle2D$locations
data = quasicircle2D$data

mesh = create.mesh.2D(nodes = rbind(boundary_nodes, locations), segments = boundary_segments)
FEMbasis = create.FEM.basis(mesh)
lambda = 10^-2

# Set the PDE parameters
R = 2.8
K1 = 0.1
K2 = 0.2
beta = 0.5
K_func<-function(points)
{
  output = array(0, c(2, 2, nrow(points)))
  for (i in 1:nrow(points))
    output[,,i] = 10*rbind(c(points[i,2]^2 + K1*points[i,1]^2 +
                             K2*(R^2 - points[i,1]^2 - points[i,2]^2),
                             (K1-1)*points[i,1]*points[i,2]),
                           c((K1-1)*points[i,1]*points[i,2],
                             points[i,1]^2 + K1*points[i,2]^2 +
                             K2*(R^2 - points[i,1]^2 - points[i,2]^2)))
  output
}

b_func<-function(points)
{
  output = array(0, c(2, nrow(points)))
  for (i in 1:nrow(points))
    output[,i] = 10*beta*c(points[i,1],points[i,2])
  output
}
```

```
c_func<-function(points)
{
  rep(c(0), nrow(points))
}

u_func<-function(points)
{
  rep(c(0), nrow(points))
}
PDE_parameters = list(K = K_func, b = b_func, c = c_func, u = u_func)

# Set the boundary conditions
BC = NULL
BC$BC_indices = which(mesh$nodesmarkers == 1) # b.c. on the complete boundary
BC$BC_values = rep(0,length(BC$BC_indices)) # homogeneus b.c.

# Since the data locations are a subset of the mesh nodes for a faster solution use:
dataNA = rep(NA, FEMbasis$nbasis)
dataNA[mesh$nodesmarkers == 0] = data

solution = smooth.FEM(observations = dataNA,
                            FEMbasis = FEMbasis,
                            lambda = lambda,
                            PDE_parameters = PDE_parameters,
                            BC = BC)
plot(solution$fit.FEM)
image(solution$fit.FEM)

#### Smoothing with areal data ####
# See Azzimonti et al. for reference to the current exemple
data(quasicircle2Dareal)
incidence_matrix = quasicircle2Dareal$incidence_matrix
data = quasicircle2Dareal$data
mesh = quasicircle2Dareal$mesh

FEMbasis = create.FEM.basis(mesh)
lambda = 10^-4

# Set the PDE parameters
R = 2.8
K1 = 0.1
K2 = 0.2
beta = 0.5
K_func<-function(points)
{
  output = array(0, c(2, 2, nrow(points)))
  for (i in 1:nrow(points))
    output[,,i] = 10*rbind(c(points[i,2]^2 + K1*points[i,1]^2 +
                             K2*(R^2 - points[i,1]^2 - points[i,2]^2),
                             (K1-1)*points[i,1]*points[i,2]),
                           c((K1-1)*points[i,1]*points[i,2],
                             points[i,1]^2 + K1*points[i,2]^2 +
                             K2*(R^2 - points[i,1]^2 - points[i,2]^2)))
  output
}

b_func<-function(points)
{
  output = array(0, c(2, nrow(points)))
  for (i in 1:nrow(points))
    output[,i] = 10*beta*c(points[i,1],points[i,2])
  output
}

c_func<-function(points)
{
  rep(c(0), nrow(points))
}

u_func<-function(points)
{
  rep(c(0), nrow(points))
}
PDE_parameters = list(K = K_func, b = b_func, c = c_func, u = u_func)

# Set the boundary conditions
BC = NULL
BC$BC_indices = which(mesh$nodesmarkers == 1) # b.c. on the complete boundary
BC$BC_values = rep(0,length(BC$BC_indices)) # homogeneus b.c.

solution = smooth.FEM(observations = data,
                            incidence_matrix = incidence_matrix,
                            FEMbasis = FEMbasis,
                            lambda = lambda,
                            PDE_parameters = PDE_parameters,
                            BC = BC)
plot(solution$fit.FEM)
image(solution$fit.FEM)
```

Index