Package ‘fdaPDE’

June 17, 2022

Version 1.1-8

Date 2022-06-17

Title Functional Data Analysis and Partial Differential Equations (PDE); 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, methods

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. (2021). Spatial Regression With Partial Differential Equation Regularisation. International Statistical Review, 89(3), 505-531. for an overview.

License CC BY-NC-SA 4.0

Copyright See the individual source files for copyrights information

NeedsCompilation yes

SystemRequirements C++11, GNU make

RoxygenNote 7.2.0

Encoding UTF-8

Author Eleonora Arnone [aut, cre],
Laura M. Sangalli [aut],
Eardi Lila [aut],
Jim Ramsay [aut],
Luca Formaggia [aut],
Giovanni Ardenghi [ctb],
Aldo Clemente [ctb],
Alessandra Colli [ctb],
Alberto Colombo [ctb],

Luca Colombo [ctb],
Carlo de Falco [ctb],
Enrico Dall'Acqua [ctb],
Giulia Ferla [ctb],
Lorenzo Ghilotti [ctb],
Jiyoung Kim [ctb],
Martina Massardi [ctb],
Giorgio Meretti [ctb],
Giulio Perin [ctb],
Clara Pigolotti [ctb],
Andrea Poiatti [ctb],
Gian Matteo Rinaldi [ctb],
Stefano Spaziani [ctb],
Andrea Vicini [ctb]

Repository CRAN

Date/Publication 2022-06-17 15:00:02 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`, `mesh.2.5D` or `mesh.3D` 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`). If saveTree flag is TRUE, it saves the tree mesh information in advance inside mesh object and can be used later on to save mesh construction time.

Usage

```
create.FEM.basis(mesh, saveTree = FALSE)
```

Arguments

mesh	A mesh.2D, mesh.2.5D or mesh.3D object representing the domain triangulation. See create.mesh.2D, create.mesh.2.5D, create.mesh.3D.
saveTree	a flag to decide to save the tree mesh information in advance (default is FALSE)

Value

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

- orderEither "1" or "2" for the 2D and 2.5D case, and "1" for the 3D case. Order of the Finite Element basis.
- nbasisScalar. The number of basis.

See Also

create.mesh.2D, create.mesh.2.5D,create.mesh.3D

Examples

```
library(fdaPDE)
## Upload the quasicircle2D data
data(quasicircle2D)

## Create the 2D mesh
mesh = create.mesh.2D(nodes = rbind(quasicircle2D$boundary_nodes,
quasicircle2D$locations), segments = quasicircle2D$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.1.5D` *Create a 1.5D linear network mesh*

Description

Create a 1.5D linear network mesh

Usage

```
create.mesh.1.5D(nodes, edges = NULL, order = 1, nodesattributes = NULL)
```

Arguments

`nodes`	A #nodes-by-2 matrix containing the x and y coordinates of the mesh nodes.
`edges`	A #edges-by-2 (when `order` = 1) or #triangles-by-3 (when `order` = 2) matrix. This option is used when a triangulation is already available. It specifies the edges giving the row's indices in `nodes` of the edges' vertices and (when `nodes` = 2) also if the triangles' edges midpoints. The triangles' vertices and midpoints are ordered as 1—3—2 In this case the function `create.mesh.1.5D` is used to produce a complete mesh.1.5D object.
`order`	Either '1' or '2'. It specifies wether each mesh should be represented by 2 nodes (the edges vertices) or by 3 nodes (the edges's vertices and midpoint). These are respectively used for linear (order = 1) and quadratic (order = 2) Finite Elements. Default is `order` = 1.
`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.

Value

An object of the class mesh.1.5D 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.
- `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.

- `neighbors`A #edges-by-2 matrix of list. Each row contains the indices of the neighbouring edges. An empty entry indicates that one node of the edge is a boundary node.
- `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.

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

Description

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

Usage

```
create.mesh.2.5D(
  nodes,
  triangles = NULL,
  order = 1,
  nodesattributes = NULL,
  segments = NULL,
  holes = NULL
)
```

Arguments

`nodes`	A #nodes-by-3 matrix containing the x, y, z coordinates of the mesh nodes.
`triangles`	A #triangles-by-3 (when `order` = 1) or #triangles-by-6 (when `order` = 2) matrix. 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.
`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.
`nodesattributes`	A matrix with #nodes rows containing nodes' attributes. These are passed unchanged to the output. This has been added for consistency with the function `create.mesh.2D`.
`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. This has been added for consistency with the function `create.mesh.2D`.

holes A #holes-by-3 matrix containing the x, y, z 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. This has been added for consistency with the function `create.mesh.2D`.

Value

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

- `nodes`A #nodes-by-3 matrix containing the x, y, z 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. 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-3 matrix containing the x, y, z 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. These are passed unchanged from the input.
- `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.

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.
```

`create.mesh.3D` *Create a* `mesh.3D` *object from the connectivity matrix and nodes locations*

Description

Create a `mesh.3D` object from the connectivity matrix and nodes locations

Usage

```
create.mesh.3D(
  nodes,
  tetrahedrons,
  order = 1,
  nodesattributes = NULL,
  segments = NULL,
  holes = NULL
)
```

Arguments

`nodes`	A #nodes-by-3 matrix containing the x, y, z coordinates of the mesh nodes.

`tetrahedrons` A #tetrahedrons-by-4 (when `order` = 1) or #tetrahedrons-by-10 (when `order` = 2) matrix. It specifies the tetrahedrons giving the row's indices in `nodes` of the tetrahedrons' vertices and (when `nodes` = 2) also if the tetrahedrons' edges midpoints. The tetrahedrons' vertices and midpoints are ordered as described in "The Finite Element Method its Basis and Fundamentals" by O. C. Zienkiewicz, R. L. Taylor and J.Z. Zhu

`order` Either '1' or '2'. It specifies wether each mesh tetrahedron should be represented by 4 nodes (the tetrahedron's vertices) or by 10 nodes (the tetrahedron's vertices and edge midpoints). These are respectively used for linear (order = 1) and quadratic (order = 2) Finite Elements. Default is `order` = 1.

`nodesattributes` A matrix with #nodes rows containing nodes' attributes. These are passed unchanged to the output. This has been added for consistency with the function `create.mesh.2D`.

`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. This has been added for consistency with the function `create.mesh.2D`.

`holes` A #holes-by-3 matrix containing the x, y, z 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. This has been added for consistency with the function `create.mesh.2D`.

Value

An object of the class mesh.3D with the following output:

- `nodes`A #nodes-by-3 matrix containing the x, y, z 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.
- `tetrahedrons`A #tetrahedrons-by-4 (when `order` = 1) or #tetrahedrons-by-10 (when `order` = 2) matrix. It specifies the tetrahedrons giving the indices in `nodes` of the tetrahedrons' vertices and (when `nodes` = 2) also if the tetrahedrons' edges midpoints.
- `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.
- `faces`A #faces-by-3 matrix containing all the faces of the tetrahedrons in the output triangulation. Each row contains the row's indices in `nodes`, indicating the nodes where the face starts from and ends to.
- `facesmarkers`A vector of lenght #faces with entries either '1' or '0'. An entry '1' indicates that the corresponding element in `faces` is a boundary face; an entry '0' indicates that the corresponding edge is not a boundary face.

- neighborsA #triangles-by-4 matrix. Each row contains the indices of the four neighbouring tetrahedrons An entry '-1' indicates that one face of the tetrahedrons is a boundary face.
- holesA #holes-by-3 matrix containing the x, y, z 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. These are passed unchanged from the input.
- orderEither '1' or '2'. It specifies wether each mesh tetrahedron should be represented by 3 nodes (the tetrahedron's vertices) or by 6 nodes (the tetrahedron's vertices and midpoints). These are respectively used for linear (order = 1) and quadratic (order = 2) Finite Elements.

Examples

```
library(fdaPDE)

##Load the matrix nodes and tetrahedrons
data(sphere3Ddata)

nodes=sphere3Ddata$nodes
tetrahedrons=sphere3Ddata$tetrahedrons

##Create the triangulated mesh from the connectivity matrix and nodes locations
mesh=create.mesh.3D(nodes,tetrahedrons)
```

DE.FEM *Nonparametric density estimation with differential regularization*

Description

This function implements a nonparametric density estimation method with differential regularization (given by the square root of the L2 norm of the laplacian of the density function), when points are located over a planar mesh. The computation relies only on the C++ implementation of the algorithm.

Usage

```
DE.FEM(data, FEMbasis, lambda, fvec=NULL, heatStep=0.1, heatIter=500,
                stepProposals=NULL,tol1=1e-4, tol2=0, print=FALSE, nfolds=NULL,
             nsimulations=500, step_method="Fixed_Step", direction_method="BFGS",
                preprocess_method="NoCrossValidation", search = "tree")
```

Arguments

data	A matrix of dimensions #observations-by-ndim. Data are locations: each row corresponds to one point, the first column corresponds to the x-coordinates, the second column corresponds to the y-coordinates and, if ndim=3, the third column corresponds to the z-coordinates.
FEMbasis	A FEMbasis object describing the Finite Element basis, as created by create.FEM.basis.

lambda A scalar or vector of smoothing parameters. If it is a vector, the optimal smoothing parameter is choosen with a k-fold cross-validation procedure based on the L2 norm.

fvec A vector of length #nodes of the mesh. It corresponds to the node values of the initial density function. If this is NULL the initial density is estimated thanks to a discretized heat diffusion process that starts from the empirical density of the data. Default is NULL. N.B. This vector cannot be the constant vector of zeros since the algortihm works with the log(f).

heatStep Real specifying the time step for the discretized heat diffusionn process.

heatIter Integer specifying the number of iteriations to perform the discretized heat diffusion process.

stepProposals A scalar or a vector containing the step parameters useful for the descent algotihm. If there is a vector of parameters, the biggest one such that the functional decreases at each iteration is choosen. If it is NULL the following vector c(0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 1e-7, 1e-8, 1e-9) is proposed. Default is NULL. N.B. If the program does not receive a right parameter, it abort the R session. Try a smaller parameter.

tol1 A scalar specifying the tolerance to use for the termination criterion based on the percentage difference between two consecutive iterations of the minimization algorithm of the loss function, the log-likelihood and the penalization. Default is 1e-5.

tol2 A scalar specifying the tolerance to use for the termination criterion based on the norm of the gradient of the functional to be minimized (the true minimum is such that this norm is zero). The default does not use this criterion. Default is 0.

print A boolean that is TRUE if the user wants the value of the functional, of the log-likelihood and of the penalization term printed on console at each iteration of the descent algorithm. Default is FALSE. N.B. We suggest to let it FALSE if preprocess_method is 'RightCV' or 'SimplifiedCV'.

nfolds An integer specifying the number of folds used in cross validation techinque to find the best lambda parameter. If there is only one lambda it can be NULL. Default is NULL.

nsimulations An integer specifying the number of iterations used in the optimization algorithms. Default value is 500.

step_method String. This parameter specifies which step method use in the descent algorithm. If it is Fixed_Step, the step is constant during all the algorithm and it is choosen according to stepProposals; if it is Backtracking_Method, the step is computed at each iteration according to the backtracking method; finally if it is Wolfe_Method, the step is computed at each iteration according to the Wolfe method. Default is Fixed_Step.

direction_method String. This parameter specifies which descent direction use in the descent algorithm. If it is Gradient, the direction is the one given by the gradient descent method (the opposite to the gradient of the functional); if instead it is BFGS the direction is the one given by the BFGS method (Broyden Fletcher Goldfarb and Shanno, a Quasi-Newton method). Default is BFGS.

preprocess_method
: String. This parameter specifies the k fold cross validation technique to use, if there is more than one smoothing parameter lambda (otherwise it should be NULL). If it is RightCV the usual k fold cross validation method is performed. If it is SimplifiedCV a simplified version is performed. In the latter case the number of smoothing parameters lambda must be equal to the number of folds nfolds. Default is NULL.

search
: a flag to decide the search algorithm type (tree or naive or walking search algorithm).

Value

A list with the following variables:

FEMbasis
: Given FEMbasis with tree informations.

g
: A vector of length #nodes that represents the value of the g-function estimated for each node of the mesh. The density is the exponential of this function.

f_init
: A #nodes-by-#lambda parameters matrix. Each column contains the node values of the initial density used for the lambda given by the column.

lambda
: A scalar representing the optimal smoothing parameter selected via k fold cross validation, if in the input there is a vector of parameters; the scalar given in input otherwise.

data
: A matrix of dimensions #observations-by-ndim containing the data used in the algorithm. They are the same given in input if the domain is 2D pr 3D; they are the original data projected on the mesh if the domain is 2.5D.

CV_err
: A vector of length nfolds containing the cross validation errors obtained in each fold, if preprocess_method is either RightCV or SimplifiedCV.

References

- Ferraccioli, F., Arnone, E., Finos, L., Ramsay, J. O., Sangalli, L. M. (2021). Nonparametric density estimation over complicated domains. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 83(2), 346-368.
- Arnone, E., Ferraccioli, F., Pigolotti, C., Sangalli, L.M. (2021), A roughness penalty approach to estimate densities over two-dimensional manifolds, Computational Statistics and Data Analysis, to appear.

Examples

```
library(fdaPDE)

## Create a 2D mesh over a squared domain
Xbound <- seq(-3, 3, length.out = 10)
Ybound <- seq(-3, 3, length.out = 10)
grid_XY <- expand.grid(Xbound, Ybound)
Bounds <- grid_XY[(grid_XY$Var1 %in% c(-3, 3)) | (grid_XY$Var2 %in% c(-3, 3)), ]
mesh <- create.mesh.2D(nodes = Bounds, order = 1)
mesh <- refine.mesh.2D(mesh, maximum_area = 0.2)
```

```
FEMbasis <- create.FEM.basis(mesh)

## Generate data
n <- 50
set.seed(10)
data_x <- rnorm(n)
data_y <- rnorm(n)
data <- cbind(data_x, data_y)

plot(mesh)
points(data, col="red", pch=19, cex=0.5)

## Density Estimation
lambda = 0.1
sol <- DE.FEM(data = data, FEMbasis = FEMbasis, lambda = lambda, fvec=NULL, heatStep=0.1,
                  heatIter=500, stepProposals=NULL, tol1=1e-4, tol2=0, print=FALSE,
                  nfolds=NULL, nsimulations=300,step_method = "Fixed_Step",
                  direction_method = "BFGS",preprocess_method="NoCrossValidation")

## Visualization
n = 100
X <- seq(-3, 3, length.out = n)
Y<- seq(-3, 3, length.out = n)
grid <- expand.grid(X, Y)

evaluation <- eval.FEM(FEM(FEMbasis, coeff = sol$g), locations = grid)
evaluation <- exp(evaluation)
eval <- matrix(evaluation, n, n)

image2D(x = X, y = Y, z = eval, col = heat.colors(100), xlab = "x", ylab = "y",
        contour = list(drawlabels = FALSE), main = "Estimated density")
```

DE.heat.FEM *Density initialization*

Description

This function implements two methods for the density initialization procedure.

Usage

```
DE.heat.FEM(data, FEMbasis, lambda=NULL, heatStep=0.1, heatIter=500,
                     init="Heat", nFolds=5, search = "tree")
```

Arguments

data A matrix of dimensions #observations-by-ndim. Data are locations: each row corresponds to one point, the first column corresponds to the x-coordinates, the second column corresponds to the y-coordinates and, if ndim=3, the third column corresponds to the z-coordinates.

FEMbasis	A FEMbasis object describing the Finite Element basis, as created by create.FEM.basis.
lambda	A scalar or vector of smoothing parameters. Default is NULL. It is useful only if init='Heat'.
heatStep	Real specifying the time step for the discretized heat diffusionn process.
heatIter	Integer specifying the number of iteriations to perform the discretized heat diffusion process.
init	String. This parameter specifies the initialization procedure. It can be either 'Heat' or 'CV'.
nFolds	An integer specifying the number of folds used in cross validation techinque. It is useful only for the case init = 'CV'.
search	a flag to decide the search algorithm type (tree or naive or walking search algorithm).

Value

If init = 'Heat' it returns a matrix in which each column contains the initial vector for each lambda. If init = 'CV' it returns the initial vector associated to the lambda given.

Examples

```
library(fdaPDE)

## Create a 2D mesh over a squared domain
Xbound <- seq(-3, 3, length.out = 10)
Ybound <- seq(-3, 3, length.out = 10)
grid_XY <- expand.grid(Xbound, Ybound)
Bounds <- grid_XY[(grid_XY$Var1 %in% c(-3, 3)) | (grid_XY$Var2 %in% c(-3, 3)), ]
mesh <- create.mesh.2D(nodes = Bounds, order = 1)
mesh <- refine.mesh.2D(mesh, maximum_area = 0.2)
FEMbasis <- create.FEM.basis(mesh)

## Generate data
n <- 50
set.seed(10)
data_x <- rnorm(n)
data_y <- rnorm(n)
data <- cbind(data_x, data_y)

plot(mesh)
points(data, col="red", pch=19, cex=0.5)

## Density initialization
lambda = 0.1
sol = DE.heat.FEM(data, FEMbasis, lambda, heatStep=0.1, heatIter=500, init="Heat")

## Visualization
plot(FEM(coeff=sol$f_init, FEMbasis=FEMbasis))
```

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, search = "tree",
                bary.locations = NULL)
```

Arguments

FEM	A FEM object to be evaluated.
locations	A 2-columns (in 1.5D or 2D) or 3-columns (in 2.5D and 3D) 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.
search	a flag to decide the search algorithm type (tree or naive or walking search algorithm).
bary.locations	A list with three vectors: `locations`, location points which are same as the given locations options. (checks whether both locations are the same); `element ids`, a vector of element id of the points from the mesh where they are located; `barycenters`, a vector of barycenter of points from the located element.

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)
```

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

Description

It evaluates a `FEM.time` object at the specified set of locations or regions. If `space.time.locations` is provided `locations`, `incidence_matrix` and `time.instants` must be NULL. Otherwise `time.instants` and one of `locations` and `incidence_matrix` must be given. In this case the evaluation is perform on the tensor grid `time.instants`-by-`locations` (or `time.instants`-by-areal domains).

Usage

```
eval.FEM.time(FEM.time, locations = NULL, time.instants = NULL,
                space.time.locations = NULL, incidence_matrix = NULL, lambdaS = 1,
                      lambdaT = 1, search = "tree", bary.locations = NULL)
```

Arguments

`FEM.time`	A `FEM.time` object to be evaluated.
`locations`	A 2-columns (in case of planar mesh) or 3-columns(in case of 2D manifold in a 3D space or a 3D volume) matrix with the spatial locations where the FEM.time object should be evaluated.

`time.instants` A vector with the time instants where the FEM.time object should be evaluated.

`space.time.locations`
: A 3-columns (in case of planar mesh) or 4-columns(in case of 2D manifold in a 3D space or a 3D volume) matrix with the time instants and spatial locations where the FEM.time object should be evaluated. The first column is for the time instants. If given, `locations`, `incidence_matrix` and `time.instants` must be NULL.

`incidence_matrix`
: In case of areal data, the #regions x #elements incidence matrix defining the regions.

`lambdaS` The index of the lambdaS choosen for the evaluation.

`lambdaT` The index of the lambdaT choosen for the evaluation.

`search` a flag to decide the search algorithm type (tree or naive or walking search algorithm).

`bary.locations` A list with three vectors: `locations`, location points which are same as the given locations options. (checks whether both locations are the same); `element ids`, a vector of element id of the points from the mesh where they are located; `barycenters`, a vector of barycenter of points from the located element.

Value

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

References

Devillers, O. et al. 2001. Walking in a Triangulation, Proceedings of the Seventeenth Annual Symposium on Computational Geometry

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
time = 1:5
coeff = rep(fs.test(mesh$nodes[,1], mesh$nodes[,2]),5)*time
## Create the FEM.time object
FEM_time_function = FEM.time(coeff=coeff, time_mesh=1:5, FEMbasis=FEMbasis, FLAG_PARABOLIC=TRUE)
```

```
evaluations = eval.FEM.time(FEM_time_function, locations = matrix(c(-0.92,0), ncol=2),
                            time.instants = time)
```

---

fdaPDE-deprecated *Deprecated Functions*

Description

Only executed when smooth.FEM.basis is run with the option CPP_CODE = FALSE. It computes the mass matrix. The element (i,j) of this matrix contains the integral over the domain of the product between the ith and kth element of the Finite Element basis. As common practise in Finite Element Analysis, this quantities are computed iterating over all the mesh triangles.

Only executed when smooth.FEM.basis is run with the option CPP_CODE = FALSE. It computes the stifness matrix. The element (i,j) of this matrix contains the integral over the domain of the scalar product between the gradient of the ith and kth element of the Finite Element basis. As common practise in Finite Element Analysis, this quantities are computed iterating over all the mesh triangles.

Only executed when the function smooth.FEM.basis is run with the option CPP_CODE = FALSE. It evaluates the Finite Element basis functions and their derivatives up to order 2 at the specified set of locations. This version of the function is implemented using only R code. It is called by R_smooth.FEM.basis.

Only executed when the function smooth.FEM.basis is run with the option CPP_CODE = FALSE. It evaluates a FEM object at the specified set of locations.

This function implements a spatial regression model with differential regularization; isotropic and stationary case. In particular, the regularizing term involves the Laplacian of the spatial field. Space-varying covariates can be included in the model. The technique accurately handle data distributed over irregularly shaped domains. Moreover, various conditions can be imposed at the domain boundaries.

This function implements a spatial regression model with differential regularization; anysotropic case. In particular, the regularizing term involves a second order elliptic PDE, that models the space-variation of the phenomenon. Space-varying covariates can be included in the model. The technique accurately handle data distributed over irregularly shaped domains. Moreover, various conditions can be imposed at the domain boundaries.

This function implements a spatial regression model with differential regularization; anysotropic and non-stationary case. In particular, the regularizing term involves a second order elliptic PDE with space-varying coefficients, that models the space-variation of the phenomenon. Space-varying covariates can be included in the model. The technique accurately handle data distributed over irregularly shaped domains. Moreover, various conditions can be imposed at the domain boundaries.

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.

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 created previously 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.

Plot a mesh MESH2D object, generated by `create.MESH.2D` or `refine.MESH.2D`. Circles indicate the mesh nodes.

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 x and y of the corresponding observations in the vector observations. This parameter can be NULL. In this case the spatial coordinates of the corresponding observations are assigned as specified in observations.
observations	A vector of length #observations 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
: 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`.

CPP_CODE
: Boolean. If `TRUE` the computation relies on the C++ implementation of the algorithm. This usually ensures a much faster computation.

PDE_parameters
: A list specifying the space-varying parameters of the elliptic PDE in the regularizing term: `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. This induces an anisotropic smoothing with a local preferential direction that corresponds to the first eigenvector of the diffusion matrix K.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. This induces a local smoothing only in the direction specified by the vector `b`. 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. `c` induces a shrinkage of the surface to zero. 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.

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.
`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.

Details

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

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.

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.
- `solution`A list, note that all terms are matrices or row vectors: the `j`th column represents the vector of related to `lambda[j]` if `lambda.selection.criterion="grid"` and `lambda.selection.lossfunction="u` In all the other cases is returned just the column related to the best penalization parameter `f`Matrix, estimate of function f, first half of solution vector `g`Matrix, second half of solution vector `z_hat`Matrix, prediction of the output in the locations `beta`If `covariates` is not `NULL`, a matrix with number of rows equal to the number of covariates and number of columns equal to length of lambda. It is the regression coefficients estimate `rmse`Estimate of the root mean square error in the locations `estimated_sd`Estiimate of the standard deviation of the error
- `optimization`A detailed list of optimization related data: `lambda_solution`numerical value of best lambda acording to `lambda.selection.lossfunction`, -1 if `lambda.selection.lossfunction="unused"` `lambda_position`integer, postion in `lambda_vector` of best lambda acording to `lambda.selection.lossfunction`, -1 if `lambda.selection.lossfunction="unused"` `GCV`numeric value of GCV in correspondence of the optimum `optimization_details`list containing further information about the optimization method used and the nature of its termination, eventual number of iterations `dof`numeric vector, value of dof for all the penalizations it has been computed, empty if not computed `lambda_vector`numeric value of the penalization factors passed by the user or found in the iterations of the optimization method `GCV_vector`numeric vector, value of GCV for all the penalizations it has been computed
- `time`Duration of the entire optimization computation
- `bary.locations`A barycenter information of the given locations if the locations are not mesh nodes.

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.
- `solution`A list, note that all terms are matrices or row vectors: the `j`th column represents the vector of related to `lambda[j]` if `lambda.selection.criterion="grid"` and `lambda.selection.lossfunction="u` In all the other cases is returned just the column related to the best penalization parameter `f`Matrix, estimate of function f, first half of solution vector `g`Matrix, second half of solution vector `z_hat`Matrix, prediction of the output in the locations `beta`If `covariates` is not `NULL`, a matrix with number of rows equal to the number of covariates and number of columns equal to length of lambda. It is the regression coefficients estimate `rmse`Estimate of the root mean square error in the locations `estimated_sd`Estiimate of the standard deviation of the error
- `optimization`A detailed list of optimization related data: `lambda_solution`numerical value of best lambda acording to `lambda.selection.lossfunction`, -1 if `lambda.selection.lossfunction="unused"` `lambda_position`integer, postion in `lambda_vector` of best lambda acording to `lambda.selection.lossfunction`, -1 if `lambda.selection.lossfunction="unused"` `GCV`numeric value of GCV in correspondence of the optimum `optimization_details`list containing further information about the optimization method used and the nature of its termination, eventual number of iterations `dof`numeric vector, value of dof for all the penalizations it has been computed, empty if not computed `lambda_vector`numeric value of the penalization factors passed by the user or found in the iterations of the optimization method `GCV_vector`numeric vector, value of GCV for all the penalizations it has been computed
- `time`Duration of the entire optimization computation

- bary.locationsA barycenter information of the given locations if the locations are not mesh nodes.

A list with the following variables:

- fit.FEMA FEM object that represents the fitted spatial field.
- PDEmisfit.FEMA FEM object that represents the Laplacian of the estimated spatial field.
- solutionA list, note that all terms are matrices or row vectors: the jth column represents the vector of related to lambda[j] if lambda.selection.criterion="grid" and lambda.selection.lossfunction="u In all the other cases is returned just the column related to the best penalization parameter fMatrix, estimate of function f, first half of solution vector gMatrix, second half of solution vector z_hatMatrix, prediction of the output in the locations betaIf covariates is not NULL, a matrix with number of rows equal to the number of covariates and number of columns equal to length of lambda. It is the regression coefficients estimate rmseEstimate of the root mean square error in the locations estimated_sdEstiimate of the standard deviation of the error
- optimizationA detailed list of optimization related data: lambda_solutionnumerical value of best lambda acording to lambda.selection.lossfunction, -1 if lambda.selection.lossfunction="unused" lambda_positioninteger, postion in lambda_vector of best lambda acording to lambda.selection.lossfunction, -1 if lambda.selection.lossfunction="unused" GCVnumeric value of GCV in correspondence of the optimum optimization_detailslist containing further information about the optimization method used and the nature of its termination, eventual number of iterations dofnumeric vector, value of dof for all the penalizations it has been computed, empty if not computed lambda_vectornumeric value of the penalization factors passed by the user or found in the iterations of the optimization method GCV_vectornumeric vector, value of GCV for all the penalizations it has been computed
- timeDuration of the entire optimization computation
- bary.locationsA barycenter information of the given locations if the locations are not mesh nodes.

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.

See Also

refine.MESH.2D, create.FEM.basis

create.MESH.2D, create.FEM.basis

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)

## Create the 2D mesh
mesh = create.mesh.2D(nodes = rbind(horseshoe2D$boundary_nodes, horseshoe2D$locations),
                      segments = horseshoe2D$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)
```

FEM.time *Define a spatio-temporal field by a Finite Element basis expansion*

Description

This function defines a FEM.time object.

Usage

```
FEM.time(coeff,time_mesh,FEMbasis,FLAG_PARABOLIC=FALSE)
```

Arguments

coeff	A vector or a matrix containing the coefficients for the spatio-temporal basis expansion. The number of rows (or the vector's length) corresponds to the number of basis in FEMbasis times the number of knots in time_mesh.
time_mesh	A vector containing the b-splines knots for separable smoothing and the nodes for finite differences for parabolic smoothing
FEMbasis	A FEMbasis object defining the Finite Element basis, created by create.FEM.basis.
FLAG_PARABOLIC	Boolean. If TRUE the coefficients are from parabolic smoothing, if FALSE the separable one.

Value

A FEM.time object. This contains a list with components coeff, mesh_time, FEMbasis and FLAG_PARABOLIC.

Examples

```
library(fdaPDE)
## Upload the horseshoe2D data
data(horseshoe2D)

## Create the 2D mesh
mesh = create.mesh.2D(nodes = rbind(horseshoe2D$boundary_nodes, horseshoe2D$locations),
                      segments = horseshoe2D$boundary_segments)
## Create the FEM basis
FEMbasis = create.FEM.basis(mesh)
## Compute the coeff vector evaluating the desired function at the mesh nodes
## In this case we consider the fs.test() function introduced by Wood et al. 2008
coeff = rep(fs.test(mesh$nodes[,1], mesh$nodes[,2]),5)
```

```
time_mesh = seq(0,1,length.out = 5)
## Create the FEM object
FEMfunction = FEM.time(coeff, time_mesh, FEMbasis, FLAG_PARABOLIC = TRUE)
## Plot it at desired time
plot(FEMfunction,0.7)
```

FPCA.FEM *Smooth Functional Principal Component Analysis*

Description

This function implements a smooth functional principal component analysis over a planar mesh, a smooth manifold or a volume.

Usage

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

Arguments

- `locations` A #observations-by-2 matrix in the 2D case and #observations-by-3 matrix in the 2.5D and 3D case, where each row specifies the spatial coordinates `x` and `y` (and `z` in 2.5D and 3D) 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.

`search` a flag to decide the search algorithm type (tree or naive or walking search algorithm).

`bary.locations` A list with three vectors: `locations`, location points which are same as the given locations options. (checks whether both locations are the same); `element ids`, a vector of element id of the points from the mesh where they are located; `barycenters`, a vector of barycenter of points from the located element.

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.
- `bary.locations`A barycenter information of the given locations if the locations are not mesh nodes.

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(nrow(mesh$nodes))
  for (i in 0:(nrow(mesh$nodes)-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$nodes[i+1,3]) + 1
  }
  data = func_evaluation + rnorm(nrow(mesh$nodes), 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, exclude = FALSE)
```

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.

exclude
: Should exterior points be set to NA?

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], exclude = FALSE)
## Create the FEM object
FEMfunction = FEM(coeff, FEMbasis)
## Plot it
plot(FEMfunction)
```

---

fs.test.3D *FELSPLINE 3D test function*

Description

Implements a finite area test function based on one proposed by Tim Ramsay (2002) and by Simon Wood (2008) in 3D.

Usage

```
fs.test.3D(x, y, z, r0 = 0.25, r = 1.25, l = 5, b = 1, exclude = FALSE)
```

Arguments

x, y, z	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.
exclude	Should exterior points be set to NA?

Value

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

Examples

```
library(fdaPDE)

data(horseshoe2.5D)
mesh = horseshoe2.5D
FEMbasis=create.FEM.basis(mesh)

# Evaluation at nodes
sol_exact=fs.test.3D(mesh$nodes[,1],mesh$nodes[,3],mesh$nodes[,2])
plot(FEM(sol_exact, FEMbasis))
```

---

horseshoe2.5D *Horseshoe2.5D domain*

Description

A `mesh2.5D` object with nodes and connectivity matrix of a triangular mesh of the horseshoe 2.5D domain.

---

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`.

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`.

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)
```

`image.FEM.time` *Image plot of a 2D FEM.time object at a given time*

Description

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

Usage

```
## S3 method for class 'FEM.time'
image(x,t,lambdaS=1,lambdaT=1,num_refinements=NULL,...)
```

Arguments

`x`	A 2D-mesh `FEM.time` object.
`t`	time at which do the plot
`lambdaS`	index of the space penalization parameter to use for the plot, useful when `FEM.time` returned by `smooth.FEM.time` using GCV
`lambdaT`	index of the time penalization parameter to use for the plot, useful when `FEM.time` returned by `smooth.FEM.time` using GCV
`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.time` `image.FEM.time`

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
time = 1:5
coeff = rep(fs.test(mesh$nodes[,1], mesh$nodes[,2]),5)*time
## Create the FEM.time object
FEM_time_function = FEM.time(coeff=coeff, time_mesh=1:5,FEMbasis=FEMbasis,FLAG_PARABOLIC=TRUE)

## Plot the FEM function
t = c(1.2,1.5,3.6,2.4,4.5)
image(FEM_time_function,t)
```

`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). If the `mesh` is of class `mesh.3D`, the color of each triangle or tetrahedron represent the mean value of the coefficients for the Finite Element basis expansion (`coeff`).

Usage

```
## S3 method for class 'FEM'
plot(x, colormap = "heat.colors", num_refinements = NULL, ...)
```

Arguments

`x`	A `FEM` object.
`colormap`	A colormap exploited in the plot. The default value is the heat colormap.
`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.FEM.time *Plot a `FEM.time` object at a given time*

Description

Plot of a `FEM.time` object, generated by `FEM.time` or returned by `smooth.FEM.time`. `time_locations` and `locations` must not be both provided. If `time_locations` is provided, the spatial field is plotted for the provided temporal instnts. If `locations` is provided, the temporal evolution in the provided space locations is plotted. If both `time_locations` and `locations` are NULL a default plot is provided. 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.time` object). If the `mesh` is of class `mesh.3D`, the color of each triangle or tetrahedron represent the mean value of the coefficients for the Finite Element basis expansion (`coeff`).

Usage

```
## S3 method for class 'FEM.time'
plot(x, time_locations = NULL, locations = NULL,
              lambdaS = NULL, lambdaT = NULL, num_refinements = NULL, Nt = 100,
               add = FALSE, main = NULL, col = "red", ...)
```

Arguments

`x`	A `FEM.time` object.
`time_locations`	A vector with the instants in which plot the spatial field
`locations`	A 2-column matrix when `x$FEMbasis$mesh` is of class `mesh.2D` or a 3-column matrix otherwise with the spatial locations for which plot the temporal evolution
`lambdaS`	Index of the space penalization parameter to use for the plot, useful when `FEM.time` returned by `smooth.FEM.time` using GCV
`lambdaT`	Index of the time penalization parameter to use for the plot, useful when `FEM.time` returned by `smooth.FEM.time` using GCV
`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.
`Nt`	The number of instants to plot when `locations` is not NULL
`add`	Boolean, used only when locations is not NULL, if TURE it performs the graphic over the last plot
`main`	The title of the plot when `locations` is not NULL
`col`	The color of the plot when `locations` is not NULL. May be a single color or a vector of colors
`...`	Arguments representing graphical options to be passed to plot3d.

See Also

`FEM.time`, `image.FEM.time`

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
time = 1:5
coeff = rep(fs.test(mesh$nodes[,1], mesh$nodes[,2]),5)*time
## Create the FEM.time object
FEM_time_function = FEM.time(coeff=coeff, time_mesh=1:5, FEMbasis=FEMbasis, FLAG_PARABOLIC=TRUE)

## Plot the FEM function
```

```
plot(FEM_time_function)

## plot spatial field in some instants
t = c(1.2,1.5,3.6,2.4,4.5)
plot(FEM_time_function, t)

## plot time evolution in some locations
plot(FEM_time_function, locations = locations[1:10,])
```

---

plot.mesh.1.5D *Plot a mesh.1.5D object*

Description

Plot a mesh.1.5D object, generated by `create.mesh.1.5D` or `refine.mesh.1.5D`.

Usage

```
## S3 method for class 'mesh.1.5D'
plot(x, ...)
```

Arguments

`x`	A `mesh.1.5D` object defining the triangular mesh, as generated by `create.mesh.1.5D` or `refine.mesh.1.5D`.
`...`	Arguments representing graphical options to be passed to par.

---

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)
```

`plot.mesh.3D` *Plot a mesh.3D object*

Description

Plot a `mesh.3D` object, generated by `create.mesh.3D`.

Usage

```
## S3 method for class 'mesh.3D'
plot(x, ...)
```

Arguments

`x`	A `mesh.3D` object generated by `create.mesh.3D`.
`...`	Arguments representing graphical options to be passed to par.

Examples

```
library(fdaPDE)

##Load the matrix nodes and tetrahedrons
data(sphere3Ddata)

nodes = sphere3Ddata$nodes
tetrahedrons = sphere3Ddata$tetrahedrons

##Create the triangulated mesh from the connectivity matrix and nodes locations
mesh = create.mesh.3D(nodes,tetrahedrons)

##Plot the triangulation of the object
plot(mesh)
```

`projection.points.1.5D` *Project 2D points onto 1.5D linear network mesh*

Description

This function projects any 2D points onto 1.5D linear network mesh.

Usage

```
projection.points.1.5D(mesh, locations)
```

Arguments

`mesh`	A mesh.1.5D object representing the graph mesh, created by create.mesh.1.5D.
`locations`	2D points to be projected onto 1.5D mesh.

Value

2D points projected onto 1.5D linear network mesh.

Examples

```
library(fdaPDE)
##Create Mesh

nodes=matrix(c(0.25,0.25,0.5,0.25,0.75,0.5,0.75,0.), nrow = 4, byrow=TRUE)
edges=matrix(c(1,2,2,3,2,4),nrow = 3,byrow = TRUE)
mesh_ = create.mesh.1.5D(nodes,edges,order=1)

## Create 2D points to be projected
locations=matrix(nrow=5,ncol=2)
locations[,1] = runif(5,min=0.25,max=0.75)
locations[,2] = runif(5,min=0.25,max=0.5)

## Project the points on the mesh
loc = projection.points.1.5D(mesh_, locations)
```

`projection.points.2.5D` *Project 3D points onto 2D 2.5D triangular mesh*

Description

This function projects any 3D points onto 2.5D triangular mesh.

Usage

```
projection.points.2.5D(mesh, locations)
```

Arguments

`mesh`	A mesh.2.5D object representing the triangular mesh, created by create.mesh.2.5D.
`locations`	3D points to be projected onto 2.5D triangular mesh.

Value

3D points projected onto 2.5D triangluar mesh.

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)

## Create 3D points to be projected
x <- cos(seq(0,2*pi, length.out = 9))
y <- sin(seq(0,2*pi, length.out = 9))
z <- rep(0.5, 9)
locations = cbind(x,y,z)

## Project the points on the mesh
loc = projection.points.2.5D(mesh, locations)
```

`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.

`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.

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.by.splitting.mesh.1.5D`

Create a `mesh.1.5D` object by splitting each edge of a given mesh into two subedges.

Description

Create a `mesh.1.5D` object by splitting each edge of a given mesh into two subedges.

Usage

```
refine.by.splitting.mesh.1.5D(mesh = NULL)
```

Arguments

`mesh`	a `mesh.1.5D` object to split

Value

An object of class mesh.1.5D with splitted edges

`refine.by.splitting.mesh.2.5D`

Create a `mesh.2.5D` object by splitting each triangle of a given mesh into four subtriangles.

Description

Create a `mesh.2.5D` object by splitting each triangle of a given mesh into four subtriangles.

Usage

```
refine.by.splitting.mesh.2.5D(mesh = NULL)
```

Arguments

`mesh`	a `mesh.2.5D` object to split

Value

An object of class mesh.2.5D with splitted triangles

`refine.by.splitting.mesh.2D`
Create a `mesh.2D` object by splitting each triangle of a given mesh into four subtriangles.

Description

Create a `mesh.2D` object by splitting each triangle of a given mesh into four subtriangles.

Usage

```
refine.by.splitting.mesh.2D(mesh = NULL)
```

Arguments

`mesh`	a `mesh.2D` object to split

Value

An object of class mesh.2D with splitted triangles

`refine.by.splitting.mesh.3D`
Create a `mesh.3D` object by splitting each tetrahedron of a given mesh into eight subtetrahedrons.

Description

Create a `mesh.3D` object by splitting each tetrahedron of a given mesh into eight subtetrahedrons.

Usage

```
refine.by.splitting.mesh.3D(mesh = NULL)
```

Arguments

`mesh`	a `mesh.3D` object to split

Value

An object of class mesh.3D with splitted tetrahedrons

refine.mesh.1.5D — *Refine 1.5D mesh*

Description

Refine 1.5D mesh

Usage

```
refine.mesh.1.5D(mesh, delta)
```

Arguments

`mesh`	a `mesh.1.5D` object to refine
`delta`	the maximum allowed length

Value

An object of class mesh.1.5D with refined edges

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,
 covariates = NULL, PDE_parameters = NULL, BC = NULL,
 incidence_matrix = NULL, areal.data.avg = TRUE,
 search = "tree", bary.locations = NULL,
 family = "gaussian", mu0 = NULL, scale.param = NULL, threshold.FPIRLS = 0.0002020,
 max.steps.FPIRLS = 15, lambda.selection.criterion = "grid", DOF.evaluation = NULL,
 lambda.selection.lossfunction = NULL, lambda = NULL, DOF.stochastic.realizations = 100,
 DOF.stochastic.seed = 0, DOF.matrix = NULL, GCV.inflation.factor = 1,
 lambda.optimization.tolerance = 0.05)
```

Arguments

`locations` A #observations-by-2 matrix in the 2D case and #observations-by-3 matrix in the 2.5D and 3D case, where each row specifies the spatial coordinates `x` and `y` (and `z` in 2.5D and 3D) 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`.

`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 and 3D, only the Laplacian is available (PDE_parameters=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.

incidence_matrix
A #regions-by-#triangles/tetrahedrons matrix where the element (i,j) equals 1 if the j-th triangle/tetrahedron 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.

areal.data.avg Boolean. It involves the computation of Areal Data. If TRUE the areal data are averaged, otherwise not.

search a flag to decide the search algorithm type (tree or naive or walking search algorithm).

bary.locations A list with three vectors: locations, location points which are same as the given locations options. (checks whether both locations are the same); element ids, a vector of element id of the points from the mesh where they are located; barycenters, a vector of barycenter of points from the located element.

`family` This parameter specify the distibution within exponential family used for GLM model. The following distribution are implemented: "binomial", "exponential", "gamma", "poisson", "gaussian", "invgaussian". The default link function for binomial is `logit` if you want either `probit` or `clogloc` set `family = "probit"`, `family = "cloglog"`.

`mu0` This parameter is a vector that set the starting point for FPIRLS algorithm. It represent an initial guess of the location parameter. Default is set to observation for non binary distribution while equal to `0.5(observations + 0.5)` for binary data.

`scale.param` Dispersion parameter of the chosen distribution. This is only required for "gamma", "gaussian", "invgaussian". User may specify the parameter as a positive real number. If the parameter is not supplied, it is estimated from data according to Wilhelm Sangalli 2016.

`threshold.FPIRLS`
This parameter is used for arresting algorithm iterations. Algorithm stops when two successive iterations lead to improvement in penalized log-likelihood smaller than threshold.FPIRLS. Default value `threshold.FPIRLS = 0.0002020`.

`max.steps.FPIRLS`
This parameter is used to limit the maximum number of iteration. Default value `max.steps.FPIRLS=15`.

`lambda.selection.criterion`
This parameter is used to select the optimization method for the smoothing parameter `lambda`. The following methods are implemented: 'grid', 'newton', 'newton_fd'. The former is a pure evaluation method. A test vector of `lambda` must be provided. The remaining two are optimization methods that automatically select the best penalization according to `lambda.selection.lossfunction` criterion. They implement respectively a pure Newton method and a finite differences Newton method. Default value `lambda.selection.criterion='grid'`

`DOF.evaluation` This parameter is used to identify if and how to perform degrees of freedom computation. The following possibilities are allowed: NULL, 'exact' and 'stochastic' In the former case no degree of freedom is computed, while the other two methods enable computation. Stochastic computation of DOFs may be slightly less accurate than its deterministic counterpart, but it is fairly less time consuming. Stochastic evaluation is highly suggested for meshes with more than 5000 nodes. Default value `DOF.evaluation=NULL`

`lambda.selection.lossfunction`
This parameter is used to determine if some loss function has to be evaluated. The following possibilities are allowed: NULL and 'GCV' (generalized cross validation) If NULL is selected, `lambda.selection.criterion='grid'` is required. 'GCV' is employed for both `lambda.selection.criterion='grid'` and optimization methods. Default value `lambda.selection.lossfunction=NULL`

`lambda` a vector of spatial smoothing parameters provided if `lambda.selection.criterion='grid'`. An optional initialization otherwise.

`DOF.stochastic.realizations`
This positive integer is considered only when `DOF.evaluation = 'stochastic'`. It is the number of uniform random variables used in stochastic DOF evaluation. Default value `DOF.stochastic.realizations=100`.

DOF.stochastic.seed
: This positive integer is considered only when DOF.evaluation = 'stochastic'. It is a user defined seed employed in stochastic DOF evaluation. Default value DOF.stochastic.seed = 0 means random.

DOF.matrix
: Matrix of degrees of freedom. This parameter can be used if the DOF.matrix corresponding to lambda is available from precedent computation. This allows to save time, since the computation of the DOFs is the most time consuming part of GCV evaluation.

GCV.inflation.factor
: Tuning parameter used for the estimation of GCV. Default value GCV.inflation.factor = 1.0 or 1.8 in GAM. It is advised to set GCV.inflation.factor larger than 1 to avoid overfitting.

lambda.optimization.tolerance
: Tolerance parameter, a double between 0 and 1 that fixes how much precision is required by the optimization method: the smaller the parameter, the higher the accuracy. Used only if lambda.selection.criterion='newton' or lambda.selection.criterion='newton_fd'. Default value lambda.optimization.tolerance=0.05

Value

A list with the following variables:

- fit.FEMA FEM object that represents the fitted spatial field.
- PDEmisfit.FEMA FEM object that represents the Laplacian of the estimated spatial field.
- solutionA list, note that all terms are matrices or row vectors: the jth column represents the vector related to lambda[j] if lambda.selection.criterion="grid" and lambda.selection.lossfunction=NULL. In all the other cases, only the column related to the best smoothing parameter is returned. fMatrix, estimate of function f, first half of solution vector. gMatrix, second half of solution vector. z_hatMatrix, prediction of the output in the locations. betaIf covariates is not NULL, a matrix with number of rows equal to the number of covariates and number of columns equal to length of lambda. It is the regression coefficients estimate. rmseEstimate of the root mean square error in the locations. estimated_sdEstimate of the standard deviation of the error.
- optimizationA detailed list of optimization related data: lambda_solutionnumerical value of best lambda according to lambda.selection.lossfunction, -1 if lambda.selection.lossfunction=NULL. lambda_positioninteger, position in lambda_vector of best lambda according to lambda.selection.lossfunction, -1 if lambda.selection.lossfunction=NULL. GCVnumeric value of GCV in correspondence of the optimum. optimization_detailslist containing further information about the optimization method used and the nature of its termination, eventual number of iterations. dofvector of positive numbers, DOFs for all the lambdas in lambda_vector, empty or invalid if not computed. lambda_vectorvector of positive numbers: penalizations either passed by the user or found in the iterations of the optimization method. GCV_vectorvector of positive numbers, GCV values for all the lambdas in lambda_vector
- timeDuration of the entire optimization computation.
- bary.locationsBarycenter information of the given locations, if the locations are not mesh nodes.

- `GAM_output`A list of GAM related data: `fn_hat`A matrix with number of rows equal to number of locations and number of columns equal to length of lambda. Each column contains the evaluaton of the spatial field in the location points. `J_minima`A vector of the same length of lambda, containing the reached minima for each value of the smoothing parameter. `variance.est`A vector which returns the variance estimates for the Generative Additive Models.

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.
- Matthieu Wilhelm & Laura M. Sangalli (2016). Generalized spatial regression with differential regularization. Journal of Statistical Computation and Simulation, 86:13, 2497-2518.

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

# Choose lambda with GCV - stochastic grid evaluation:
lambda = 10^(-2:0)
solution = smooth.FEM(observations = data,
```

```
                            covariates = covariate,
                            FEMbasis = FEMbasis,
                            lambda = lambda, DOF.evaluation = 'stochastic',
                            lambda.selection.lossfunction = 'GCV')
bestLambda = solution$optimization$lambda_solution
# Choose lambda with GCV - Newton finite differences stochastic evaluation -:
solution = smooth.FEM(observations = data,
                            covariates = covariate,
                            FEMbasis = FEMbasis,
                    DOF.evaluation = 'stochastic', lambda.selection.lossfunction = 'GCV')
bestLambda = solution$optimization$lambda_solution


#### 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
#grid evaluation
solution = smooth.FEM(observations = dataNA,
                            FEMbasis = FEMbasis,
                            lambda = lambda,
                            PDE_parameters = PDE_parameters,
                            BC = BC)
plot(solution$fit.FEM)
image(solution$fit.FEM)
# Newton's method
solution = smooth.FEM(observations = dataNA,
                            FEMbasis = FEMbasis,
                            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.
#grid evaluation
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)
#Newton's method
solution = smooth.FEM(observations = data,
                            incidence_matrix = incidence_matrix,
                            FEMbasis = FEMbasis,
                            PDE_parameters = PDE_parameters,
                            BC = BC)
plot(solution$fit.FEM)
image(solution$fit.FEM)
```

smooth.FEM.time *Space-time regression with differential regularization*

Description

Space-time regression with differential regularization. Space-varying covariates can be included in the model. The technique accurately handle data distributed over irregularly shaped domains. Moreover, various conditions can be imposed at the domain boundaries.

Usage

```
smooth.FEM.time(locations = NULL, time_locations = NULL, observations, FEMbasis,
time_mesh=NULL, covariates = NULL, PDE_parameters = NULL,  BC = NULL,
incidence_matrix = NULL, areal.data.avg = TRUE,
FLAG_MASS = FALSE, FLAG_PARABOLIC = FALSE, FLAG_ITERATIVE = FALSE,
threshold = 10^(-4), max.steps = 50, IC = NULL,
search = "tree", bary.locations = NULL,
family = "gaussian", mu0 = NULL, scale.param = NULL,
threshold.FPIRLS = 0.0002020, max.steps.FPIRLS = 15,
lambda.selection.criterion = "grid", DOF.evaluation = NULL,
lambda.selection.lossfunction = NULL, lambdaS = NULL, lambdaT = NULL,
DOF.stochastic.realizations = 100, DOF.stochastic.seed = 0,
DOF.matrix = NULL, GCV.inflation.factor = 1, lambda.optimization.tolerance = 0.05)
```

Arguments

`locations`	A matrix where each row specifies the spatial coordinates `x` and `y` (and `z` if ndim=3) of the corresponding observations in the vector `observations`. This parameter can be `NULL`. In this case, if also the incidence matrix is `NULL` the spatial coordinates are assumed to coincide with the nodes of the `mesh`.
`time_locations`	A vector containing the times of the corresponding observations in the vector `observations`. This parameter can be `NULL`. In this case the temporal locations are assumed to coincide with the nodes of the `time_mesh`.
`observations`	A matrix of #locations x #time_locations with the observed data values over the spatio-temporal domain. The spatial locations of the observations can be specified with the `locations` argument.
`FEMbasis`	A `FEMbasis` object describing the Finite Element basis, as created by `create.FEM.basis`.
`time_mesh`	A vector specifying the time mesh.
`covariates`	A #observations-by-#covariates matrix where each row represents the covariates associated with the corresponding observed data value in `observations`.
`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 PDE is elliptic it 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 PDE is space-varying it 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. This induces an anisotropic smoothing with a local preferential direction

that corresponds to the first eigenvector of the diffusion matrix K.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. This induces a local smoothing only in the direction specified by the vector b. 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. c induces a shrinkage of the surface to zero. 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 and 3D only the Laplacian is available (PDE_parameters=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.

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

areal.data.avg
: Boolean. It involves the computation of Areal Data. If TRUE the areal data are averaged, otherwise not.

FLAG_MASS
: Boolean. This parameter is considerd only for separable problems i.e. when FLAG_PARABOLIC==FALSE. If TRUE the mass matrix in space and in time are used, if FALSE they are substituted with proper identity matrices.

FLAG_PARABOLIC
: Boolean. If TRUE the parabolic problem problem is selected, if FALSE the separable one.

FLAG_ITERATIVE
: Boolean. If TRUE the iterative method is selected, if FALSE the monolithic one.

threshold
: This parameter is used for arresting algorithm iterations. Algorithm stops when two successive iterations lead to improvement in penalized log-likelihood smaller than threshold. Default value threshold = 10^(-4).

max.steps
: This parameter is used to limit the maximum number of iteration. Default value max.steps=50.

IC
: Initial condition needed in case of parabolic problem i.e. when FLAG_PARABOLIC==TRUE. If FLAG_PARABOLIC==FALSE this parameter is ignored. If FLAG_PARABOLIC=TRUE and IC=NULL it is necessary to provide also data at the initial time. IC will be estimated from them.

search
: a flag to decide the search algorithm type (tree or naive or walking search algorithm).

bary.locations
: A list with three vectors: locations, location points which are same as the given locations options. (checks whether both locations are the same); element

ids, a vector of element id of the points from the mesh where they are located; barycenters, a vector of barycenter of points from the located element.

family
: This parameter specify the distibution within exponential family used for GLM model. The following distribution are implemented: "binomial", "exponential", "gamma", "poisson", "gaussian", "invgaussian". The default link function for binomial is logit if you want either probit or clogloc set family = "probit", family = "cloglog".

mu0
: This parameter is a vector that set the starting point for FPIRLS algorithm. It represent an initial guess of the location parameter. Default is set to observation for non binary distribution while equal to 0.5(observations + 0.5) for binary data.

scale.param
: Dispersion parameter of the chosen distribution. This is only required for "gamma", "gaussian", "invgaussian". User may specify the parameter as a positive real number. If the parameter is not supplied, it is estimated from data according to Wilhelm Sangalli 2016.

threshold.FPIRLS
: This parameter is used for arresting algorithm iterations. Algorithm stops when two successive iterations lead to improvement in penalized log-likelihood smaller than threshold.FPIRLS. Default value threshold.FPIRLS = 0.0002020.

max.steps.FPIRLS
: This parameter is used to limit the maximum number of iteration. Default value max.steps.FPIRLS=15.

lambda.selection.criterion
: This parameter is used to select the optimization method related to smoothing parameter lambda. The following methods are implemented: 'grid', further optimization methods are yet to come. The 'grid' is a pure evaluation method, therefore a vector of lambda testing penalizations must be provided. Default value lambda.selection.criterion='grid'

DOF.evaluation
: This parameter is used to identify if and how degrees of freedom computation has to be performed. The following possibilities are allowed: NULL, 'exact' and 'stochastic' In the former case no degree of freedom is computed, while the other two methods enable computation. Stochastic computation of DOFs may be slightly less accurate than its deterministic counterpart, but is highly suggested for meshes of more than 5000 nodes, being fairly less time consuming. Default value DOF.evaluation=NULL

lambda.selection.lossfunction
: This parameter is used to understand if some loss function has to be evaluated. The following possibilities are allowed: NULL and 'GCV' (generalized cross validation) The former case is that of lambda.selection.criterion='grid' pure evaluation, while the second can be employed for optimization methods. Default value lambda.selection.lossfunction=NULL

lambdaS
: A scalar or vector of spatial smoothing parameters.

lambdaT
: A scalar or vector of temporal smoothing parameters.

DOF.stochastic.realizations
: This parameter is considered only when DOF.evaluation = 'stochastic'. It is a positive integer that represents the number of uniform random variables used in stochastic GCV computation. Default value DOF.stochastic.realizations=100.

DOF.stochastic.seed
: This parameter is considered only when DOF.evaluation = 'stochastic'. It is a positive integer that represents user defined seed employed in stochastic GCV computation. Default value DOF.stochastic.seed=0.

DOF.matrix
: Matrix of degrees of freedom. This parameter can be used if the DOF.matrix corresponding to lambdaS and lambdaT is available from precedent computation. This allows to save time since the computation of the DOFs is the most expensive part of GCV.

GCV.inflation.factor
: Tuning parameter used for the estimation of GCV. Default value GCV.inflation.factor = 1.0. It is advised to set it grather than 1 to avoid overfitting.

lambda.optimization.tolerance
: Tolerance parameter, a double between 0 and 1 that fixes how much precision is required by the optimization method: the smaller the parameter, the higher the accuracy. Used only if lambda.selection.criterion='newton' or lambda.selection.criterion='newton_fd', thus ot implemented yet. Default value lambda.optimization.tolerance=0.05.

Value

A list with the following variables:

fit.FEM.time
: A FEM.time object that represents the fitted spatio-temporal field.

PDEmisfit.FEM.time
: A FEM.time object that represents the misfit of the penalized PDE.

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 jth column represents the vector of regression coefficients when the smoothing parameter is equal to lambda[j].

edf
: If GCV is TRUE, a scalar or matrix with the trace of the smoothing matrix for each combination of the smoothing parameters specified in lambdaS and lambdaT.

stderr
: If GCV is TRUE, a scalar or matrix with the estimate of the standard deviation of the error for each combination of the smoothing parameters specified in lambdaS and lambdaT.

GCV
: If GCV is TRUE, a scalar or matrix with the value of the GCV criterion for each combination of the smoothing parameters specified in lambdaS and lambdaT.

bestlambda
: If GCV is TRUE, a 2-elements vector with the indices of smoothing parameters returnig the lowest GCV

ICestimated
: If FLAG_PARABOLIC is TRUE and IC is NULL, a list containing a FEM object with the initial conditions, the value of the smoothing parameter lambda returning the lowest GCV and, in presence of covariates, the estimated beta coefficients

bary.locations
: A barycenter information of the given locations if the locations are not mesh nodes.

References

#' @references Arnone, E., Azzimonti, L., Nobile, F., & Sangalli, L. M. (2019). Modeling spatially dependent functional data via regression with differential regularization. Journal of Multivariate Analysis, 170, 275-295. Bernardi, M. S., Sangalli, L. M., Mazza, G., & Ramsay, J. O. (2017). A penalized regression model for spatial functional data with application to the analysis of the production of waste in Venice province. Stochastic Environmental Research and Risk Assessment, 31(1), 23-38.

Examples

```
library(fdaPDE)

data(horseshoe2D)
boundary_nodes = horseshoe2D$boundary_nodes
boundary_segments = horseshoe2D$boundary_segments
locations = horseshoe2D$locations
time_locations = seq(0,1,length.out = 5)

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

space_time_locations = cbind(rep(time_locations,each=nrow(mesh$nodes)),
                             rep(mesh$nodes[,1],5),rep(mesh$nodes[,2],5))

FEMbasis = create.FEM.basis(mesh)
lambdaS = 10^-1
lambdaT = 10^-1
data = fs.test(space_time_locations[,2],
               space_time_locations[,3])*cos(pi*space_time_locations[,1]) +
       rnorm(nrow(space_time_locations), sd = 0.5)
data = matrix(data, nrow = nrow(mesh$nodes), ncol = length(time_locations), byrow = TRUE)

solution = smooth.FEM.time(observations = data, time_locations = time_locations,
                           FEMbasis = FEMbasis, lambdaS = lambdaS, lambdaT = lambdaT)
plot(solution$fit.FEM)
```

---

sphere3Ddata *Sphere3Ddata*

Description

A dataset with information about the connectivity matrix and the nodes locations of a sphere geometry. It containes:

- nodes. A #nodes-by-3 matrix specifying the locations of each node.
- tetrahedrons. A #tetrahedrons-by-4 matrix specifying the indices of the nodes in each tetrahedron.

This dataset can be used to create a `MESH.3D` object with the function `create.MESH.3D`

Index