Package 'fdaPDE'

January 15, 2016

Version 0.1-1

Date 2016-01-15

Title Regression with Partial Differential Regularizations, using the Finite Element Method

Author Eardi Lila [aut, cre],
Laura M. Sangalli [aut],
Jim Ramsay [aut],
Luca Formaggia [aut]

Maintainer Eardi Lila <eardi.lila@mail.polimi.it>

Depends R (>= 3.0.0), stats, grDevices, graphics, rgl

LinkingTo RcppEigen

Suggests MASS

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.

NeedsCompilation yes

RoxygenNote 5.0.0

Repository CRAN

Date/Publication 2016-01-15 12:42:58

R topics documented:

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

Description

Sets up a Finite Element basis. It requires a triangular mesh, a `MESH2D` object, as input. The basis' functions are globally continuos surfaces, that are polynomials once restricted to a triangle in the mesh. Linear (`order = 1`) and quadratic (`order = 2`) Finite Element are currently implemented.

Usage

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

Arguments

`mesh`	A `MESH2D` object representing the domain triangulation. See create.MESH.2D.
`order`	Either "1" or "2". Order of the Finite Element basis. When `order = 1` the basis system is piecewise linear. When `order = 2` the basis system is piecewise quadratic. This parameter must be less or equal to the `order` specified in the `MESH2D` object. See create.MESH.2D.

Value

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

`order`	Either "1" or "2". Order of the Finite Element basis.
`nbasis`	Scalar. The number of basis.

detJ — The determinant of the transformation from the nodes of the reference triangle to the nodes of the i-th triangle; this coincides with the double of the area of the i-th triangle.

transf — A three-dimensional array such that `transf[i,,]` is the 2-by-2 matrix that transforms the nodes of the reference triangle to the nodes of the i-th triangle.

metric — A three-dimensional array such that `metric[i,,]` is the 2-by-2 matrix `transf[i,,]^{-1}*transf[i,,]^{-T}`. This matrix is used for the computation of the integrals over the elements of the mesh.

See Also

create.MESH.2D

Examples

```
## Creates a simple triangulated domain with a concavity; this is a MESH2D object
mesh<-create.MESH.2D(nodes=rbind(c(0, 0), c(0, 1), c(0.5, 0.5), c(1, 1), c(1, 0)),
segments=rbind(c(1, 2), c(2, 3), c(3, 4), c(4, 5), c(5, 1)), order=1)
## Plot it
plot(mesh)
## Creates the basis
FEMbasis = create.FEM.basis(mesh, order = 1)
```

---

create.MESH.2D *Create a 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.

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 MESH2D object.

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

`verbosity` This can be '0', '1' or '2'. It indicates the level of verbosity in the triangulation process. When `verbosity` = 0 no message is returned during the triangulation. When `verbosity` = 2 the triangulation process is described step by step by displayed messages. Default is `verbosity` = 0.

Value

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.

See Also

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

Examples

```
## Upload the Meuse data
data(MeuseData)
## Create a triangulation on the convex hull of these data,
## where each data location is a triangle vertex
mesh <- create.MESH.2D(nodes = MeuseData[,c(2,3)], order = 1)
## Plot the mesh
plot(mesh)
## Upload a domain boundary for these data
data(MeuseBorder)
## Create a constrained Delaunay triangulation with the provided boundary
## where each datalocation is a triangle vertex
mesh <- create.MESH.2D(nodes = MeuseData[,c(2,3)], segments = MeuseBorder, order = 1)
## Plot the mesh
plot(mesh)
```

---

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

Description

It evaluates a FEM object the specified set of locations.

Usage

```
eval.FEM(FEM, locations, CPP_CODE = TRUE)
```

Arguments

FEM	A FEM object to be evaluated.
locations	A 2-colums matrix with the spatial locations where the FEM object should be evaluated.
CPP_CODE	Boolean. If TRUE the computation relies on the C++ implementation of a Visibility Walk Algorithm (Devillers et al. 2001). This usually ensures a fast computation.

Value

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.

References

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

---

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

Description

This function defines a FEM object. This is not usualled called directly by users.

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 functional replicates.
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

```
## Upload a triangular mesh and plot it
data("mesh.2D.rectangular")
plot(mesh.2D.rectangular)
## Create a linear Finite Element basis
FEMbasis = create.FEM.basis(mesh.2D.rectangular, 1)
## Define a sinusoidal function as expansion of this basis and plot it
coeff <- sin(mesh.2D.rectangular$nodes[,1])*cos(mesh.2D.rectangular$nodes[,2])
FEM_object<- FEM(coeff, FEMbasis)
plot(FEM_object)
```

`image.FEM` *Image Plot of a FEM object*

Description

Image plot of a `FEM` object, generated by the function `FEM` or returned by `smooth.FEM.basis`, `smooth.FEM.PDE.basis` or `smooth.FEM.PDE.sv.basis` can be visualized through an image plot.

Usage

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

Arguments

`x`	A `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

`plot.FEM`

Examples

```
## Upload a triangular mesh and plot it
data("mesh.2D.rectangular")
plot(mesh.2D.rectangular)
## Create a linear Finite Element basis
FEMbasis = create.FEM.basis(mesh.2D.rectangular, 1)
## Define a sinusoidal function as expansion of this basis and plot it
coeff <- sin(mesh.2D.rectangular$nodes[,1])*cos(mesh.2D.rectangular$nodes[,2])
FEM_object<- FEM(coeff, FEMbasis)
image(FEM_object)
```

mesh.2D.rectangular *Simple Rectangular mesh*

Description

A simple rectangular mesh. This is a MESH2D object created with `create.MESH.2D`.

mesh.2D.simple *Simple mesh*

Description

A simple mesh. This is a MESH2D object created with `create.MESH.2D`.

MeuseBorder *Boundary of the Meuse River data set*

Description

This file provides the boundary of the domain of the MeuseData.

Format

A data frame with 52 rows and 2 variables.

Details

- vertex_start. A vector having as entries the row's indices of the location in MeuseData where a boundary segment starts from.
- vertex_end. A vector having as entries the row's indices of the location in MeuseData where a boundary segment ends to.

MeuseData *Meuse river data set*

Description

This data set gives locations, top soil heavy metal concentrations (ppm) and other variables, collected in a flood plain of the river Meuse. For details on this dataset, see `meuse.all` in https://cran.r-project.org/web/packages/gstat/gstat.pdf. This version of the dataset includes 155 observations. Moreover, a definition of the domain's boundary is provided through the file MeuseBorder, as it is used in many examples to illustrate fdaPDE features.

- sample: original sample number. In this version of the dataset some observations has been left out because not indicative, or outliers.
- x: numeric vector indicating the x-coordinate (m) in RDM (Dutch topographical map coordinates).
- y: numeric vector indicating the y-coordinate (m) in RDM (Dutch topographical map coordinates).
- cadmium: topsoil cadmium concentration (ppm).
- copper: topsoil copper concentration (ppm).
- lead: topsoil lead concentration (ppm).
- zinc: topsoil zinc concentration (ppm).
- elev: relative elevation.
- dist.log(m): logarithm of the distance to river Meuse (metres), as obtained during the field survey.
- om: organic matter, as percentage.
- ffreq: flooding frequency class.
- soil: soil type.

Format

A data frame with 155 rows and 12 variables.

Source

https://cran.r-project.org/web/packages/gstat/gstat.pdf http://spatial-analyst.net/book/meusegrids

plot.FEM *Plot a* FEM *object*

Description

Three-dimensional plot of a FEM object, generated by FEM or returned by smooth.FEM.basis, smooth.FEM.PDE.basis or smooth.FEM.PDE.sv.basis.

Usage

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

Arguments

x	A FEM object.
num_refinements	A natural number specifying how many bisections should 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

image.FEM

Examples

```
## Upload a triangular mesh and plot it
data("mesh.2D.rectangular")
plot(mesh.2D.rectangular)
## Create a linear Finite Element basis
FEMbasis = create.FEM.basis(mesh.2D.rectangular, 1)
## Define a sinusoidal function as expansion of this basis and plot it
coeff <- sin(mesh.2D.rectangular$nodes[,1])*cos(mesh.2D.rectangular$nodes[,2])
FEM_object<- FEM(coeff, FEMbasis)
plot(FEM_object)
```

plot.MESH2D *Plot a MESH2D object*

Description

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

Usage

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

Arguments

`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.

Examples

```
## Upload the Meuse data and a domain boundary
data(MeuseData)
data(MeuseBorder)
## Create a triangular mesh with the provided boundary
mesh <- create.MESH.2D(nodes = MeuseData[,c(2,3)], segments = MeuseBorder, order = 1)
## Plot it
plot(mesh)
```

refine.MESH.2D *Refine a 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 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.

Usage

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

Arguments

`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.
`verbosity`	This can be '0', '1' or '2'. It indicates the level of verbosity in the triangulation process.

Value

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

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

Examples

```
## Upload the Meuse data and a domain boundary for these data
data(MeuseData)
data(MeuseBorder)
## Create a Constrained Delaunay triangulation
mesh <- create.MESH.2D(nodes = MeuseData[,c(2,3)], segments = MeuseBorder, order = 1)
## Plot the mesh
plot(mesh)
## Refine the triangulation
mesh_refine <- refine.MESH.2D(mesh, minimum_angle = 30, maximum_area = 10000)
plot(mesh_refine)
```

R_elementProperties *Compute some properties for each triangular element of the mesh*

Description

Only executed when the function `create.FEM.basis` is run with the option `CPP_CODE = FALSE`. It computes some quantities associated to the linear map that transforms the ith triangle in the reference triangular element. These are used for the computation of the integrals necessary to build the mass and stiffness matrix.

Usage

```
R_elementProperties(mesh)
```

Arguments

`mesh`	A `MESH2D` mesh object representing the triangular mesh. This can be created with `create.MESH.2D`.

Value

A list with the following variables:

`detJ`	A vector of length #triangles. The ith element contains the determinant of the transformation from the reference triangle to the nodes of the i-th triangle. It's values is also the double of the area of each triangle of the basis.
`transf`	A matrix #triangles-by-2-by-2. `transf[i,,]` is the 2-by-2 tranformation matrix that transforms the nodes of the reference triangle to the nodes of the i-th triangle.
`metric`	A matrix #triangles-by-2-by-2. `metric[i,,]` is the 2-by-2 matrix `transf[i,,]^{-1}*transf[i,,]^{-T}`. This matrix is usuful for the computation of the integrals over the elements of the mesh.

R_eval.FEM *Evaluate a FEM object at a set of locations*

Description

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.

Usage

```
R_eval.FEM(FEM, locations)
```

Arguments

`FEM`	A `FEM` object to be evaluated
`locations`	A 2-columns matrix with the spatial locations where the FEM object should be evaluated

Value

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.

See Also

`R_eval.FEM.basis`

R_eval.FEM.basis *Evaluate Finite Element bases and their Derivatives at a set of locations*

Description

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.

Usage

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

Arguments

FEMbasis	An FEMbasis object representing the Finite Element basis; See create.FEM.basis.
locations	A 2-columns matrix with the spatial locations where the bases should be evaluated.
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.

Value

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

See Also

R_eval.FEM

R_mass *Compute the mass matrix*

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.

Usage

```
R_mass(FEMbasis)
```

Arguments

FEMbasis	A FEM object representing the Finite Element basis. See create.FEM.basis.

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.

See Also

R_stiff

R_smooth.FEM.basis *Spatial regression with differential regularization (fully implemented in R code)*

Description

Spatial regression with differential regularization (fully implemented in R code)

Usage

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

Arguments

locations	A #observations-by-2 matrix where each row specifies the spatial coordinates of the corresponding observations in the vector observations.
observations	A #observations vector with the observed data values over the domain. The locations of the observations can be specified with the locations argument. Otherwise if only the vector of observations is given, these are consider to be located in the corresponding node in the table nodes of the mesh. In this last case, an NA value in the observations vector indicates that there is no observation associated to the corresponding node.
FEMbasis	A FEMbasis object describing the Finite Element basis, as created by create.FEM.basis.
lambda	A scalar or vector of smoothing parameters.
covariates	A #observations-by-#covariates matrix where each row represents the covariates associated with the corresponding observed data value in observations.
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.

Value

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.

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), pp.681.703.

See Also

smooth.FEM.basis, smooth.FEM.PDE.basis, smooth.FEM.PDE.sv.basis

---

R_stiff *Compute the stiffness matrix*

Description

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.

Usage

```
R_stiff(FEMbasis)
```

Arguments

FEMbasis	A FEMbasis object representing the basis; See create.FEM.basis.

Value

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.

See Also

R_mass

`smooth.FEM.basis` *Spatial regression with differential regularization: stationary and isotropic case (Laplacian)*

Description

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.

Usage

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

Arguments

`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.
`FEMbasis`	A `FEMbasis` object describing the Finite Element basis, as created by `create.FEM.basis`.
`lambda`	A scalar or vector of smoothing parameters.
`covariates`	A #observations-by-#covariates matrix where each row represents the covariates associated with the corresponding observed data value in `observations`.
`BC`	A list with two vectors: `BC_indices`, a vector with the indices in `nodes` of boundary nodes where a Dirichlet Boundary Condition should be applied; `BC_values`, a vector with the values that the spatial field must take at the nodes indicated in `BC_indices`.
`GCV`	Boolean. If `TRUE` the following quantities are computed: the trace of the smoothing matrix, the estimated error standard deviation, and the Generalized Cross Validation criterion, for each value of the smoothing parameter specified in `lambda`.
`CPP_CODE`	Boolean. If `TRUE` the computation relies on the C++ implementation of the algorithm. This usually ensures a much faster computation.

Value

A list with the following variables:

`fit.FEM`	A `FEM` object that represents the fitted spatial field.
`PDEmisfit.FEM`	A `FEM` object that represents the Laplacian of the estimated spatial field.
`beta`	If covariates is not `NULL`, a 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`.

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), pp. 681-703.

See Also

`smooth.FEM.PDE.basis`, `smooth.FEM.PDE.sv.basis`

Examples

```
library(fdaPDE)
## Load the Meuse data and a domain boundary for these data
data(MeuseData)
data(MeuseBorder)
## Create a triangular mesh for these data with the provided boundary and plot it
order=1
mesh <- create.MESH.2D(nodes = MeuseData[,c(2,3)], segments = MeuseBorder, order = order)
plot(mesh)
## Create the Finite Element basis
FEMbasis = create.FEM.basis(mesh, order)
## Estimate zync field without using covariates, setting the smoothing parameter to 10^3.5
data = log(MeuseData[,"zinc"])
lambda = 10^3.5
ZincMeuse = smooth.FEM.basis(observations = data,
                             FEMbasis = FEMbasis, lambda = lambda)
## Plot the estimated spatial field
plot(ZincMeuse$fit.FEM)
# Now repeat the analysis using as covariates the square root of the log-distance
# from river \code{sqrt(dist.log(m))} and the altitude \code{elev}
desmat = matrix(1,nrow=nrow(MeuseData),ncol=2)
desmat[,1] = sqrt(MeuseData[,"dist.log(m)"])
desmat[,2] = MeuseData[,"elev"]
ZincMeuseCovar = smooth.FEM.basis(observations = data, covariates = desmat,
                                  FEMbasis = FEMbasis, lambda = lambda)
```

```
# Plot of the non parametric part (f) of the regression model y_i = beta_1 x_i1 + beta_2 x_i2 + f
plot(ZincMeuseCovar$fit.FEM)
# Print covariates' regression coefficients
print(ZincMeuseCovar$beta)
```

`smooth.FEM.PDE.basis` *Spatial regression with differential regularization: anysotropic case (elliptic PDE)*

Description

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.

Usage

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

Arguments

`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. `NA` values are admissible to indicate that the node is not associated with any observed data value.
`FEMbasis`	A `FEMbasis` object describing the Finite Element basis, as created by `create.FEM.basis`.
`lambda`	A scalar or vector of smoothing parameters.
`PDE_parameters`	A list specifying the parameters of the elliptic PDE in the regularizing term: `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
`covariates`	A #observations-by-#covariates matrix where each row represents the covariates associated with the corresponding observed data value in `observations`.

BC	A list with two vectors: BC_indices, a vector with the indices in nodes of boundary nodes where a Dirichlet Boundary Condition should be applied; BC_values, a vector with the values that the spatial field must take at the nodes indicated in BC_indices.
GCV	Boolean. If TRUE the following quantities are computed: the trace of the smoothing matrix, the estimated error standard deviation, and the Generalized Cross Validation criterion, for each value of the smoothing parameter specified in lambda.
CPP_CODE	Boolean. If TRUE the computation relies on the C++ implementation of the algorithm. This usually ensures a much faster computation.

Value

A list with the following variables:

fit.FEM	A FEM object that represents the fitted spatial field.
PDEmisfit.FEM	A FEM object that represents the PDE misfit for 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.

References

Azzimonti, L., Sangalli, L.M., Secchi, P., Domanin, M., and Nobile, F., 2014. Blood flow velocity field estimation via spatial regression with PDE penalization Blood flow velocity field estimation via spatial regression with PDE penalization. DOI. 10.1080/01621459.2014.946036.
Azzimonti, L., Nobile, F., Sangalli, L.M., and Secchi, P., 2014. Mixed Finite Elements for Spatial Regression with PDE Penalization. SIAM/ASA Journal on Uncertainty Quantification, 2(1), pp.305-335.

See Also

smooth.FEM.basis, smooth.FEM.PDE.sv.basis

Examples

```
# Load the mesh and plot it
data(mesh.2D.simple)
plot(mesh.2D.simple)
# Create a vector of noisy samples of an underlying spatial field,
# located over the nodes of the mesh
observations = sin(pi*mesh.2D.simple$nodes[,1]) + rnorm(n = nrow(mesh.2D.simple$nodes), sd = 0.1)
# Create the FEM basis object
```

```
FEMbasis = create.FEM.basis(mesh.2D.simple, 2)

# Set a vector of smoothing coefficients
lambda = c(10^-4, 1, 10^4)

# Set the anysotropic smoothing matrix K
PDE_parameters_anys = list(K = matrix(c(0.01,0,0,1), nrow = 2), b = c(0,0), c = 0)
# Estimate one field for each smoothing parameter and plot these
FEM_CPP_PDE = smooth.FEM.PDE.basis(observations = observations,
                                   FEMbasis = FEMbasis, lambda = lambda,
                                   PDE_parameters = PDE_parameters_anys)
plot(FEM_CPP_PDE$fit.FEM)

# Evaluate solution in three points
eval.FEM(FEM_CPP_PDE$fit.FEM, locations = rbind(c(0,0),c(0.5,0),c(-2,-2)))
```

`smooth.FEM.PDE.sv.basis`

Spatial regression with differential regularization: anysotropic and non-stationary case (elliptic PDE with space-varying coefficients)

Description

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.

Usage

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

Arguments

`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.
`FEMbasis`	A `FEMbasis` object describing the Finite Element basis, as created by `create.FEM.basis`.

lambda	A scalar or vector of smoothing parameters.
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; 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; 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
covariates	A #observations-by-#covariates matrix where each row represents the covariates associated with the corresponding observed data value in observations.
BC	A list with two vectors: BC_indices, a vector with the indices in nodes of boundary nodes where a Dirichlet Boundary Condition should be applied; BC_values, a vector with the values that the spatial field must take at the nodes indicated in BC_indices.
GCV	Boolean. If TRUE the following quantities are computed: the trace of the smoothing matrix, the estimated error standard deviation, and the Generalized Cross Validation criterion, for each value of the smoothing parameter specified in lambda.
CPP_CODE	Boolean. If TRUE the computation relies on the C++ implementation of the algorithm. This usually ensures a much faster computation.

Value

A list with the following variables:

fit.FEM	A FEM object that represents the fitted spatial field.
PDEmisfit.FEM	A FEM object that represents the PDE misfit for the estimated spatial field.
beta	If covariates is not NULL, a vector of lenght #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.

References

Azzimonti, L., Sangalli, L.M., Secchi, P., Domanin, M., and Nobile, F., 2014. Blood flow velocity field estimation via spatial regression with PDE penalization Blood flow velocity field estimation via spatial regression with PDE penalization. DOI. 10.1080/01621459.2014.946036.

Azzimonti, L., Nobile, F., Sangalli, L.M., and Secchi, P., 2014. Mixed Finite Elements for Spatial Regression with PDE Penalization. SIAM/ASA Journal on Uncertainty Quantification, 2(1), pp.305-335.

See Also

smooth.FEM.basis, smooth.FEM.PDE.basis

Examples

```
# Loading the mesh
data(mesh.2D.rectangular)
# Create the FEM basis object
FEMbasis = create.FEM.basis(mesh.2D.rectangular, 2)
# Create a vector of noisy samples of an underlying spatial field,
# located over the nodes of the mesh
observations = sin(0.2*pi*mesh.2D.rectangular$nodes[,1]) +
rnorm(n = nrow(mesh.2D.rectangular$nodes), sd = 0.1)
# Set the smoothing coefficient
lambda = c(10^-2)
#Set the space vriant coefficients of the penalizying PDE
K_func<-function(points)
{
mat<-c(0.01,0,0,1)
as.vector(0.5*mat %*% t(points[,1]^2))
}
b_func<-function(points)
{
rep(c(0,0), nrow(points))
}

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

u_func<-function(points)
{
rep(c(0), nrow(points))
}
# Assemble the parameters in one object
PDE_parameters = list(K = K_func, b = b_func, c = c_func, u = u_func)
# Estimate the underlying spatial field and plot these
FEM_CPP_PDE = smooth.FEM.PDE.sv.basis(observations = observations,
             FEMbasis = FEMbasis, lambda = lambda, PDE_parameters = PDE_parameters)
plot(FEM_CPP_PDE$fit.FEM)
```

Index