Package 'smnet'

March 30, 2017

Type Package

Title Smoothing for Stream Network Data

Version 2.1.1

Date 2017-03-30

Depends R (>= 3.0.1), SSN, spam

Imports splines, RSQLite, DBI, methods, stats, graphics

Author Alastair Rushworth

Maintainer Alastair Rushworth <alastairmrushworth@gmail.com>

Description Fits flexible additive models to data on stream networks, taking account of flow-connectivity of the network. Models are fitted using penalised least squares.

License GPL-2

LazyLoad yes

ByteCompile yes

Repository CRAN

NeedsCompilation no

Date/Publication 2017-03-30 16:21:08 UTC

R topics documented:

get_adjacency *Construct an Adjacency Matrix*

Description

Builds a sparse adjacency matrix from a user specified SSN data directory, by extracting and processing the binaryID.db table. The resulting output of this function is required input for fitting spatial additive network models to SSN objects using the main smnet function.

Usage

```
get_adjacency(ssn_directory, netID)
```

Arguments

ssn_directory	Required character string indicating the path to the location of the .ssn directory which contains the binaryID.db table
netID	Integer specifying the particular stream network of interest within the SSN object. Defaults to 1.

Value

List with two elements.

adjacency	Sparse adjacency matrix of class "spam" with row and column dimension equal to the number of stream segments. If the i^th column has non-zero elements j_1 and j_2 then this indicates that j_1 and j_2 are direct upstream neighbours of i. If the i^{th} column has sum 1, then this indicates that i has only one upstream neighbour, and therefore no confluence lies between them; by default the spatial penalties treat these differently.
bid	Character vector of binary identifiers for each stream segment, used only for automatic calculation of Shreve's stream order within smnet

Author(s)

Alastair Rushworth

See Also

smnet

Examples

```
# Set up an SSN object - this part taken
# from the SSN:::SimulateOnSSN help file
example_network_1<- createSSN(
                     n          = 50,
                     obsDesign  = binomialDesign(200),
```

```
                    predDesign = binomialDesign(50),
                    importToR  = TRUE,
                    path = paste(tempdir(),"/example_network_1",sep = ""),
                    treeFunction = iterativeTreeLayout
                    )

# plot the simulated network structure with prediction locations
# plot(example_network, bty = "n", xlab = "x-coord", ylab = "y-coord")

## create distance matrices, including between predicted and observed
createDistMat(example_network_1, "preds", o.write=TRUE, amongpred = TRUE)

## extract the observed and predicted data frames
observed_data           <- getSSNdata.frame(example_network_1, "Obs")
prediction_data         <- getSSNdata.frame(example_network_1, "preds")

## associate continuous covariates with the observation locations
#  data generated from a normal distribution
obs                     <- rnorm(200)
observed_data[,"X"]     <- obs
observed_data[,"X2"]    <- obs^2

## associate continuous covariates with the prediction locations
#  data generated from a normal distribution
pred                    <- rnorm(50)
prediction_data[,"X"]   <- pred
prediction_data[,"X2"]  <- pred^2

## simulate some Gaussian data that follows a 'tail-up' spatial process
sims <- SimulateOnSSN(
              ssn.object     = example_network_1,
              ObsSimDF       = observed_data,
              PredSimDF      = prediction_data,
              PredID         = "preds",
              formula        = ~ 1 + X,
              coefficients   = c(1, 10),
              CorModels      = c("Exponential.tailup"),
              use.nugget     = TRUE,
              CorParms       = c(10, 5, 0.1),
              addfunccol     = "addfunccol")$ssn.object


## extract the observed and predicted data frames, now with simulated values
sim1DFpred         <- getSSNdata.frame(sims, "preds")
sim1preds          <- sim1DFpred[,"Sim_Values"]
sim1DFpred[,"Sim_Values"] <- NA
sims               <- putSSNdata.frame(sim1DFpred, sims, "preds")

# create the adjacency matrix for use with smnet
adjacency    <- get_adjacency(
                 paste(tempdir(), "/example_network_1", sep = ""),
                 net = 1
                 )
```

```
# not run - plot the adjacency matrix
# display(adjacency[[1]])

# sometimes it is useful to see which varaibles are valid network weights
# in the data contained within the SSN object
show_weights(sims, adjacency)

# fit a penalised spatial model to the stream network data
# Sim_Values are quadratic in the X covariate.  To highlight
# the fitting of smooth terms, this is treated as non-linear
# and unknown using m().
mod_smn       <- smnet(formula = Sim_Values ~ m(X) + m(X2) +
network(adjacency = adjacency, weight = "shreve"),
                      data.object = sims, netID = 1)

# not run - plot different summaries of the model
plot(mod_smn, type = "covariates")
plot(mod_smn, type = "segments", weight = 4, shadow = 2)
plot(mod_smn, type = "full", weight = 4, shadow = 2)

# obtain predictions at the prediction locations and plot
# against true values
preds <- predict(mod_smn, newdata = getSSNdata.frame(sims, "preds"))
plot(preds$predictions, sim1preds)

# obtain summary of the fitted model
summary(mod_smn)
```

`m` *Specify Smooth Terms in Formulae*

Description

Function used to set up univariate or bivariate smooth terms based on P-splines, for use within a call to `smnet`.

Usage

```
m(..., k = -1, cyclic = F)
```

Arguments

`...`	one or more variables for creating P-spline smooths
`k`	integer defining the number of uniformly spaced B-spline basis functions for the smooth, default is 10. For 2d (and higher) smooths, this is the marginal basis size.
`cyclic`	logical vector indicating whether the smooth should be cyclic. Based on the harmonic smoother of Eiler and Marx (2004)

Value

term	character vector of the names of the variables involved in the smooth to be set up
bs.dim	number of B-spline basis functions to be used in the smooth

Author(s)

Alastair Rushworth

References

Modified version of s originally from package mgcv, Simon Wood (2014).

See Also

smnet

Examples

```
# Set up an SSN object - this part taken
# from the SSN:::SimulateOnSSN help file
example_network_2<- createSSN(
                    n          = 50,
                    obsDesign  = binomialDesign(200),
                    predDesign = binomialDesign(50),
                    importToR  = TRUE,
                    path = paste(tempdir(),"/example_network_2",sep = ""),
                    treeFunction = iterativeTreeLayout
                    )

# plot the simulated network structure with prediction locations
# plot(example_network_2, bty = "n", xlab = "x-coord", ylab = "y-coord")

## create distance matrices, including between predicted and observed
createDistMat(example_network_2, "preds", o.write=TRUE, amongpred = TRUE)

## extract the observed and predicted data frames
observed_data           <- getSSNdata.frame(example_network_2, "Obs")
prediction_data         <- getSSNdata.frame(example_network_2, "preds")

## associate continuous covariates with the observation locations
#  data generated from a normal distribution
obs                     <- rnorm(200)
observed_data[,"X"]     <- obs
observed_data[,"X2"]    <- obs^2

## associate continuous covariates with the prediction locations
#  data generated from a normal distribution
pred                    <- rnorm(50)
prediction_data[,"X"]   <- pred
```

```
prediction_data[,"X2"]   <- pred^2

## simulate some Gaussian data that follows a 'tail-up' spatial process
sims <- SimulateOnSSN(
               ssn.object      = example_network_2,
               ObsSimDF        = observed_data,
               PredSimDF       = prediction_data,
               PredID          = "preds",
               formula         = ~ 1 + X,
               coefficients    = c(1, 10),
               CorModels       = c("Exponential.tailup"),
               use.nugget      = TRUE,
               CorParms        = c(10, 5, 0.1),
               addfunccol      = "addfunccol")$ssn.object


## extract the observed and predicted data frames, now with simulated values
sim1DFpred         <- getSSNdata.frame(sims, "preds")
sim1preds          <- sim1DFpred[,"Sim_Values"]
sim1DFpred[,"Sim_Values"] <- NA
sims               <- putSSNdata.frame(sim1DFpred, sims, "preds")

# create the adjacency matrix for use with smnet
adjacency    <- get_adjacency(
                 paste(tempdir(), "/example_network_2", sep = ""),
                 net = 1
                 )

# not run - plot the adjacency matrix
# display(adjacency[[1]])

# sometimes it is useful to see which varaibles are valid network weights
# in the data contained within the SSN object
show_weights(sims, adjacency)

# fit a penalised spatial model to the stream network data
# Sim_Values are quadratic in the X covariate.  To highlight
# the fitting of smooth terms, this is treated as non-linear
# and unknown using m().
mod_smn       <- smnet(formula = Sim_Values ~ m(X) + m(X2) +
network(adjacency = adjacency, weight = "shreve"),
                      data.object = sims, netID = 1)

# not run - plot different summaries of the model
plot(mod_smn, type = "covariates")
plot(mod_smn, type = "segments", weight = 4, shadow = 2)
plot(mod_smn, type = "full", weight = 4, shadow = 2)

# obtain predictions at the prediction locations and plot
# against true values
preds <- predict(mod_smn, newdata = getSSNdata.frame(sims, "preds"))
plot(preds$predictions, sim1preds)
```

```
# obtain summary of the fitted model
summary(mod_smn)
```

`network` *Specify Network Smoother in Formulae*

Description

This function specifies all of the information required to smooth parameters over the segments of a stream network using an adjacency matrix, and a vector of flow weights.

Usage

```
network(adjacency = NULL, weight = "autoShreve", fixed.df = NULL)
```

Arguments

`adjacency`	A sparse adjacency matrix of class `"spam"` that describes the flow connectedness of the stream network. `adjacency` is typically obtained from a call to get_adjacency.
`weight`	A character string indicating the column name of a numeric vector of flow weights contained in the `data.object` that has been passed to smnet. Defaults to `"autoShreve"` which automatically constructs a weighting based on Shreve order, useful if data does not include an appropriate weight. For more information on choosing appropriate `weight` inputs from a given data set, see show_weights.
`fixed.df`	Positive scalar indicating a fixed number of degrees of freedom to allocate to the stream network component, overriding the criterion minimisation for this component. Under the default setting, NULL, the degrees of freedom are chosen automatically.

Value

A list combining the processed input components above. For internal use within `smnet`.

`adjacency`	Sparse adjacency matrix
`weight`	Numeric vector of flow weights
`netID`	Integer identifying network of interest

Author(s)

Alastair Rushworth

See Also

smnet, get_adjacency, show_weights

Examples

```
# Set up an SSN object - this part taken
# from the SSN:::SimulateOnSSN help file
example_network_3<- createSSN(
                    n          = 50,
                    obsDesign  = binomialDesign(200),
                    predDesign = binomialDesign(50),
                    importToR  = TRUE,
                    path = paste(tempdir(),"/example_network_3",sep = ""),
                    treeFunction = iterativeTreeLayout
                    )

# plot the simulated network structure with prediction locations
# plot(example_network_3, bty = "n", xlab = "x-coord", ylab = "y-coord")

## create distance matrices, including between predicted and observed
createDistMat(example_network_3, "preds", o.write=TRUE, amongpred = TRUE)

## extract the observed and predicted data frames
observed_data           <- getSSNdata.frame(example_network_3, "Obs")
prediction_data         <- getSSNdata.frame(example_network_3, "preds")

## associate continuous covariates with the observation locations
#  data generated from a normal distribution
obs                     <- rnorm(200)
observed_data[,"X"]     <- obs
observed_data[,"X2"]    <- obs^2

## associate continuous covariates with the prediction locations
#  data generated from a normal distribution
pred                    <- rnorm(50)
prediction_data[,"X"]   <- pred
prediction_data[,"X2"]  <- pred^2

## simulate some Gaussian data that follows a 'tail-up' spatial process
sims <- SimulateOnSSN(
               ssn.object      = example_network_3,
               ObsSimDF        = observed_data,
               PredSimDF       = prediction_data,
               PredID          = "preds",
               formula         = ~ 1 + X,
               coefficients    = c(1, 10),
               CorModels       = c("Exponential.tailup"),
               use.nugget      = TRUE,
               CorParms        = c(10, 5, 0.1),
               addfunccol      = "addfunccol")$ssn.object


## extract the observed and predicted data frames, now with simulated values
sim1DFpred         <- getSSNdata.frame(sims, "preds")
sim1preds          <- sim1DFpred[,"Sim_Values"]
```

```
sim1DFpred[,"Sim_Values"] <- NA
sims               <- putSSNdata.frame(sim1DFpred, sims, "preds")

# create the adjacency matrix for use with smnet
adjacency    <- get_adjacency(
                 paste(tempdir(), "/example_network_3", sep = ""),
                 net = 1
                 )

# not run - plot the adjacency matrix
# display(adjacency[[1]])

# sometimes it is useful to see which varaibles are valid network weights
# in the data contained within the SSN object
show_weights(sims, adjacency)

# fit a penalised spatial model to the stream network data
# Sim_Values are quadratic in the X covariate.  To highlight
# the fitting of smooth terms, this is treated as non-linear
# and unknown using m().
mod_smn       <- smnet(formula = Sim_Values ~ m(X) + m(X2) +
network(adjacency = adjacency, weight = "shreve"),
                      data.object = sims, netID = 1)

# not run - plot different summaries of the model
plot(mod_smn, type = "covariates")
plot(mod_smn, type = "segments", weight = 4, shadow = 2)
plot(mod_smn, type = "full", weight = 4, shadow = 2)

# obtain predictions at the prediction locations and plot
# against true values
preds <- predict(mod_smn, newdata = getSSNdata.frame(sims, "preds"))
plot(preds$predictions, sim1preds)

# obtain summary of the fitted model
summary(mod_smn)
```

`plot.smnet` *Plot a Stream Network Model*

Description

Plot linear, univariate and bivariate smooth effects and network smooth terms that resulting from a call to `smnet`.

Usage

```
  ## S3 method for class 'smnet'
plot(x, type = "covariates", se = FALSE, res = FALSE, weight = NULL,
  sites = FALSE, sites.col = NULL, sites.cex = 1, network.col = NULL,
  shadow = 0, key = TRUE, legend.text = NULL, legend.range = NULL, ...)
```

Arguments

x	An object of class **smnet**
type	Character string identifying the type of plot to be produced. The default, "covariates", produces plots of all linear and smooth components (the latter corresponding to each appearance of m in the model formula). "full" plots the stream network fitted mean using the full set of spatial points contained in the associated SSN object. "segments" plots the stream network fitted mean using a set of connnected line segments to represent the spatial network, this can be faster for large networks than "full".
se	Logical. When TRUE and type = "covariates", standard errors are shown on plots of linear and smooth terms. When type = "segment" or "full" spatial standard errors are plotted.
res	Logical. When TRUE, partial residuals are shown on plots of linear and smooth component. Ignored when type = "full" or "segments"
weight	Positive real number that scales stream segment widths (as determined using Shreve order) to indicate relative size of stream segments. Ignored when type = "covariate". Currently only "autoShreve" is supported, defaults to NULL in which all streams segments are plotted with lines with identical widths.
sites	Logical indicating whether locations of observation sites should be added to spatial plots. Ignored when type = "covariate" and defaults to FALSE.
sites.col	Single colour to plot observation locations. If not provided, points will be coloured according to the default legend and average observation at each location.
sites.cex	Expansion factor for the size of plotted observation points
network.col	Single colour to represent all stream segments. By default, network is coloured according to fitted values from model. Ignored when type = "covariate".
shadow	Positive scalar that adds a black outline to spatial stream segment plots, useful if the colour scale has many light colours. Ignored when type = "covariate" and defaults to 0 (no shadow is drawn).
key	Logical. Plots a colour legend for network plots. Ignored when type = "covariates".
legend.text	Character annotation to add to color scale. Ignored if key = FALSE or type = "covariates".
legend.range	Range of values represented by the color scale. Ignored if key = FALSE or type = "covariates".
...	Other arguments passed to plot

Author(s)

Alastair Rushworth

See Also

predict.smnet, summary.smnet

Examples

```
# Set up an SSN object - this part taken
# from the SSN:::SimulateOnSSN help file
example_network_4<- createSSN(
                     n          = 50,
                     obsDesign  = binomialDesign(200),
                     predDesign = binomialDesign(50),
                     importToR  = TRUE,
                     path = paste(tempdir(),"/example_network_4",sep = ""),
                     treeFunction = iterativeTreeLayout
                     )

# plot the simulated network structure with prediction locations
# plot(example_network_4, bty = "n", xlab = "x-coord", ylab = "y-coord")

## create distance matrices, including between predicted and observed
createDistMat(example_network_4, "preds", o.write=TRUE, amongpred = TRUE)

## extract the observed and predicted data frames
observed_data            <- getSSNdata.frame(example_network_4, "Obs")
prediction_data          <- getSSNdata.frame(example_network_4, "preds")

## associate continuous covariates with the observation locations
#  data generated from a normal distribution
obs                      <- rnorm(200)
observed_data[,"X"]      <- obs
observed_data[,"X2"]     <- obs^2

## associate continuous covariates with the prediction locations
#  data generated from a normal distribution
pred                     <- rnorm(50)
prediction_data[,"X"]    <- pred
prediction_data[,"X2"]   <- pred^2

## simulate some Gaussian data that follows a 'tail-up' spatial process
sims <- SimulateOnSSN(
               ssn.object      = example_network_4,
               ObsSimDF        = observed_data,
               PredSimDF       = prediction_data,
               PredID          = "preds",
               formula         = ~ 1 + X,
               coefficients    = c(1, 10),
               CorModels       = c("Exponential.tailup"),
               use.nugget      = TRUE,
               CorParms        = c(10, 5, 0.1),
               addfunccol      = "addfunccol")$ssn.object


## extract the observed and predicted data frames, now with simulated values
sim1DFpred         <- getSSNdata.frame(sims, "preds")
sim1preds          <- sim1DFpred[,"Sim_Values"]
```

```
sim1DFpred[,"Sim_Values"] <- NA
sims               <- putSSNdata.frame(sim1DFpred, sims, "preds")

# create the adjacency matrix for use with smnet
adjacency    <- get_adjacency(
                 paste(tempdir(), "/example_network_4", sep = ""),
                 net = 1
                 )

# not run - plot the adjacency matrix
# display(adjacency[[1]])

# sometimes it is useful to see which varaibles are valid network weights
# in the data contained within the SSN object
show_weights(sims, adjacency)

# fit a penalised spatial model to the stream network data
# Sim_Values are quadratic in the X covariate.  To highlight
# the fitting of smooth terms, this is treated as non-linear
# and unknown using m().
mod_smn       <- smnet(formula = Sim_Values ~ m(X) + m(X2) +
network(adjacency = adjacency, weight = "shreve"),
                      data.object = sims, netID = 1)

# not run - plot different summaries of the model
plot(mod_smn, type = "covariates")
plot(mod_smn, type = "segments", weight = 4, shadow = 2)
plot(mod_smn, type = "full", weight = 4, shadow = 2)

# obtain predictions at the prediction locations and plot
# against true values
preds <- predict(mod_smn, newdata = getSSNdata.frame(sims, "preds"))
plot(preds$predictions, sim1preds)

# obtain summary of the fitted model
summary(mod_smn)
```

`predict.smnet` *Predict From a Stream Network Model.*

Description

Get predictions and standard errors for a new set of spatial locations and associated covariate values from a model fitted by `smnet`.

Usage

```
## S3 method for class 'smnet'
predict(object, newdata =  NULL, ...)
```

Arguments

object	Object of class smnet, usually the result of a call to smnet.
newdata	New design matrix at which to make predictions
...	other arguments passed to predict.smnet

Value

predictions	vector of predictions corresponding to prediction points in the original SpatialStreamNetwork input object
predictions.se	vector of prediction standard errors

Author(s)

Alastair Rushworth

See Also

smnet

Examples

```
# Set up an SSN object - this part taken
# from the SSN:::SimulateOnSSN help file
example_network_5<- createSSN(
                    n          = 50,
                    obsDesign  = binomialDesign(200),
                    predDesign = binomialDesign(50),
                    importToR  = TRUE,
                    path = paste(tempdir(),"/example_network_5",sep = ""),
                    treeFunction = iterativeTreeLayout
                    )

# plot the simulated network structure with prediction locations
# plot(example_network_5, bty = "n", xlab = "x-coord", ylab = "y-coord")

## create distance matrices, including between predicted and observed
createDistMat(example_network_5, "preds", o.write=TRUE, amongpred = TRUE)

## extract the observed and predicted data frames
observed_data           <- getSSNdata.frame(example_network_5, "Obs")
prediction_data         <- getSSNdata.frame(example_network_5, "preds")

## associate continuous covariates with the observation locations
#  data generated from a normal distribution
obs                     <- rnorm(200)
observed_data[,"X"]     <- obs
observed_data[,"X2"]    <- obs^2

## associate continuous covariates with the prediction locations
```

```
#  data generated from a normal distribution
pred                    <- rnorm(50)
prediction_data[,"X"]   <- pred
prediction_data[,"X2"]  <- pred^2

## simulate some Gaussian data that follows a 'tail-up' spatial process
sims <- SimulateOnSSN(
               ssn.object      = example_network_5,
               ObsSimDF        = observed_data,
               PredSimDF       = prediction_data,
               PredID          = "preds",
               formula         = ~ 1 + X,
               coefficients    = c(1, 10),
               CorModels       = c("Exponential.tailup"),
               use.nugget      = TRUE,
               CorParms        = c(10, 5, 0.1),
               addfunccol      = "addfunccol")$ssn.object


## extract the observed and predicted data frames, now with simulated values
sim1DFpred         <- getSSNdata.frame(sims, "preds")
sim1preds          <- sim1DFpred[,"Sim_Values"]
sim1DFpred[,"Sim_Values"] <- NA
sims               <- putSSNdata.frame(sim1DFpred, sims, "preds")

# create the adjacency matrix for use with smnet
adjacency    <- get_adjacency(
                 paste(tempdir(), "/example_network_5", sep = ""),
                 net = 1
                 )

# not run - plot the adjacency matrix
# display(adjacency[[1]])

# sometimes it is useful to see which varaibles are valid network weights
# in the data contained within the SSN object
show_weights(sims, adjacency)

# fit a penalised spatial model to the stream network data
# Sim_Values are quadratic in the X covariate.  To highlight
# the fitting of smooth terms, this is treated as non-linear
# and unknown using m().
mod_smn       <- smnet(formula = Sim_Values ~ m(X) + m(X2) +
network(adjacency = adjacency, weight = "shreve"),
                      data.object = sims, netID = 1)

# not run - plot different summaries of the model
plot(mod_smn, type = "network-covariates")
plot(mod_smn, type = "network-segments", weight = 4, shadow = 2)
plot(mod_smn, type = "network-full", weight = 4, shadow = 2)

# obtain predictions at the prediction locations and plot
# against true values
```

```
preds <- predict(mod_smn, newdata = getSSNdata.frame(sims, "preds"))
plot(preds$predictions, sim1preds)

# obtain summary of the fitted model
summary(mod_smn)
```

---

show_weights *Search for and Validate Weights in an SSN Object*

Description

Explore SSN objects for valid stream weights for use in fitting stream network models.

Usage

```
show_weights(SSNobject, adjacency, netID = 1)
```

Arguments

SSNobject	SpatialStreamNetwork containing data to be searched for valid network weights
adjacency	adjacency object corresponding to SSNobject, resulting from a call to get_adjacency
netID	Positive integer indicating the network number to investigate, if multiple networks are contained inSSNobject. Default is 1.

Value

Prints the names of valid weight variables to the console.

Author(s)

Alastair Rushworth

See Also

get_adjacency, network

Examples

```
# Set up an SSN object - this part taken
# from the SSN:::SimulateOnSSN help file
example_network_6<- createSSN(
                     n          = 50,
                     obsDesign  = binomialDesign(200),
                     predDesign = binomialDesign(50),
                     importToR  = TRUE,
                     path = paste(tempdir(),"/example_network_6",sep = ""),
                     treeFunction = iterativeTreeLayout
                     )
```

```
# plot the simulated network structure with prediction locations
# plot(example_network_6, bty = "n", xlab = "x-coord", ylab = "y-coord")

## create distance matrices, including between predicted and observed
createDistMat(example_network_6, "preds", o.write=TRUE, amongpred = TRUE)

## extract the observed and predicted data frames
observed_data            <- getSSNdata.frame(example_network_6, "Obs")
prediction_data          <- getSSNdata.frame(example_network_6, "preds")

## associate continuous covariates with the observation locations
#  data generated from a normal distribution
obs                      <- rnorm(200)
observed_data[,"X"]      <- obs
observed_data[,"X2"]     <- obs^2

## associate continuous covariates with the prediction locations
#  data generated from a normal distribution
pred                     <- rnorm(50)
prediction_data[,"X"]    <- pred
prediction_data[,"X2"]   <- pred^2

## simulate some Gaussian data that follows a 'tail-up' spatial process
sims <- SimulateOnSSN(
               ssn.object      = example_network_6,
               ObsSimDF        = observed_data,
               PredSimDF       = prediction_data,
               PredID          = "preds",
               formula         = ~ 1 + X,
               coefficients    = c(1, 10),
               CorModels       = c("Exponential.tailup"),
               use.nugget      = TRUE,
               CorParms        = c(10, 5, 0.1),
               addfunccol      = "addfunccol")$ssn.object


## extract the observed and predicted data frames, now with simulated values
sim1DFpred         <- getSSNdata.frame(sims, "preds")
sim1preds          <- sim1DFpred[,"Sim_Values"]
sim1DFpred[,"Sim_Values"] <- NA
sims               <- putSSNdata.frame(sim1DFpred, sims, "preds")

# create the adjacency matrix for use with smnet
adjacency    <- get_adjacency(
                 paste(tempdir(), "/example_network_6", sep = ""),
                 net = 1
                 )

# not run - plot the adjacency matrix
# display(adjacency[[1]])

# sometimes it is useful to see which varaibles are valid network weights
```

```
# in the data contained within the SSN object
show_weights(sims, adjacency)

# fit a penalised spatial model to the stream network data
# Sim_Values are quadratic in the X covariate.  To highlight
# the fitting of smooth terms, this is treated as non-linear
# and unknown using m().
mod_smn       <- smnet(formula = Sim_Values ~ m(X) + m(X2) +
network(adjacency = adjacency, weight = "shreve"),
                       data.object = sims, netID = 1)

# not run - plot different summaries of the model
plot(mod_smn, type = "covariates")
plot(mod_smn, type = "segments", weight = 4, shadow = 2)
plot(mod_smn, type = "full", weight = 4, shadow = 2)

# obtain predictions at the prediction locations and plot
# against true values
preds <- predict(mod_smn, newdata = getSSNdata.frame(sims, "preds"))
plot(preds$predictions, sim1preds)

# obtain summary of the fitted model
summary(mod_smn)
```

`smnet` *Additive Modelling for Stream Networks*

Description

Fits (Gaussian) additive models to river network data based on the flexible modelling framework described in O'Donnell et al. (2014). Data must be in the form of a `SpatialStreamNetwork` object as used by the `SSN` package (Ver Hoef et al., 2012). Smoothness of covariate effects is represented via a penalised B-spline basis (P-splines) and parameter estimates are obtained using penalised least-squares. Optimal smoothness is achieved by optimization of `AIC`, `GCV` or `AICC`.

The formula interpreter used for penalised additive components is modelled on the code found in the package `mgcv`.

Usage

```
smnet(formula, data.object, netID = 1, method = "AICC", control = NULL)
```

Arguments

`formula`	A formula statement similar to those used by `lm` and `mgcv:::gam`. Smooth functions are represented with `m(..., k = 20)` function, up to 2d smooth interactions are currently supported. Spatial network components are specified using `network(...)` function. Further details can be found in m and network and the examples below.

data.object	An object of class "SpatialStreamNetwork"
netID	Integer indicating the particular stream network to model, generally only user-specified when multiple networks are contained within data.object, default is 1.
method	Character string determining the performance criterion for choosing optimal smoothness, options are "AICC", "AIC" or "GCV".
control	A list of options that control smoothness selection via optimisation. See 'Details'.

Details

control is a list whose elements control smoothness selection:maxit limits the number of iterations made by the optimiser (default = 500). approx, positive integer, if specified indicates the number of Monte-Carlo samples to collect using an approximate version of performance criterion when direct evaluation is slow - this may be much faster if the network has a large number of segments or the data is large, for example approx = 100 often works well (defaults to NULL). checks, logical, specifies whether additivity checks should be performed on the input weights default = TRUE. trace, default = 0, if set to 1, optim will print progress. tol, the relative tolerance of optim. optim.method, the optimiser - default is "Nelder-Mead" see ?optim for details.

Value

Object of class smnet with components

fitted.values	vector of fitted values
residuals	vector of residuals: response minus fitted values
coefficients	vector of parameter estimates
R2	R^2 statistic
R2.adj	adjusted R^2 statistic
df.residual	residual degrees of freedom
ssn.object	unchanged SSN input data object
internals	model objects for internal use by other functions

Author(s)

Alastair Rushworth

References

Ver Hoef, J.M., Peterson, E.E., Clifford, D., Shah, R. (2012) SSN: An R Package for Spatial Statistical Modeling on Stream Networks

O' Donnell, D., Rushworth, A.M., Bowman, A.W., Scott, E.M., Hallard, M. (2014) Flexible regression models over river networks. Journal of the Royal Statistical Society: Series C (Applied Statistics). 63(1) 47–63.

Reinhard Furrer, Stephan R. Sain (2010). spam: A Sparse Matrix R Package with Emphasis on MCMC Methods for Gaussian Markov Random Fields. Journal of Statistical Software, 36(10), 1-25. URL: http://www.jstatsoft.org/v36/i10/

See Also

get_adjacency, plot.smnet

Examples

```
# Set up an SSN object - this part taken
# from the SSN:::SimulateOnSSN help file
example_network_8<- createSSN(
                    n          = 50,
                    obsDesign  = binomialDesign(200),
                    predDesign = binomialDesign(50),
                    importToR  = TRUE,
                    path = paste(tempdir(),"/example_network_8",sep = ""),
                    treeFunction = iterativeTreeLayout
                    )

# plot the simulated network structure with prediction locations
# plot(example_network_8, bty = "n", xlab = "x-coord", ylab = "y-coord")

## create distance matrices, including between predicted and observed
createDistMat(example_network_8, "preds", o.write=TRUE, amongpred = TRUE)

## extract the observed and predicted data frames
observed_data           <- getSSNdata.frame(example_network_8, "Obs")
prediction_data         <- getSSNdata.frame(example_network_8, "preds")

## associate continuous covariates with the observation locations
#  data generated from a normal distribution
obs                     <- rnorm(200)
observed_data[,"X"]     <- obs
observed_data[,"X2"]    <- obs^2

## associate continuous covariates with the prediction locations
#  data generated from a normal distribution
pred                    <- rnorm(50)
prediction_data[,"X"]   <- pred
prediction_data[,"X2"]  <- pred^2

## simulate some Gaussian data that follows a 'tail-up' spatial process
sims <- SimulateOnSSN(
               ssn.object      = example_network_8,
               ObsSimDF        = observed_data,
               PredSimDF       = prediction_data,
               PredID          = "preds",
               formula         = ~ 1 + X,
               coefficients    = c(1, 10),
               CorModels       = c("Exponential.tailup"),
               use.nugget      = TRUE,
               CorParms        = c(10, 5, 0.1),
               addfunccol      = "addfunccol")$ssn.object
```

```
## extract the observed and predicted data frames, now with simulated values
sim1DFpred         <- getSSNdata.frame(sims, "preds")
sim1preds          <- sim1DFpred[,"Sim_Values"]
sim1DFpred[,"Sim_Values"] <- NA
sims               <- putSSNdata.frame(sim1DFpred, sims, "preds")

# create the adjacency matrix for use with smnet
adjacency    <- get_adjacency(
                 paste(tempdir(), "/example_network_8", sep = ""),
                 net = 1
                 )

# not run - plot the adjacency matrix
# display(adjacency[[1]])

# sometimes it is useful to see which varaibles are valid network weights
# in the data contained within the SSN object
show_weights(sims, adjacency)

# fit a penalised spatial model to the stream network data
# Sim_Values are quadratic in the X covariate.  To highlight
# the fitting of smooth terms, this is treated as non-linear
# and unknown using m().
mod_smn       <- smnet(formula = Sim_Values ~ m(X) + m(X2) +
network(adjacency = adjacency, weight = "shreve"),
                      data.object = sims, netID = 1)

# not run - plot different summaries of the model
plot(mod_smn, type = "network-covariates")
plot(mod_smn, type = "network-segments", weight = 4, shadow = 2)
plot(mod_smn, type = "network-full", weight = 4, shadow = 2)

# obtain predictions at the prediction locations and plot
# against true values
preds <- predict(mod_smn, newdata = getSSNdata.frame(sims, "preds"))
plot(preds$predictions, sim1preds)

# obtain summary of the fitted model
summary(mod_smn)
```

`summary.smnet` *Summarise Stream Network Model*

Description

Generate summaries of linear and smooth components of an `smnet` object.

Usage

```
## S3 method for class 'smnet'
summary(object, ...)
```

Arguments

object	An object of class smnet.
...	other arguments passed to summary

Value

List object with components

1	linear.terms: the linear components of the fitted model
2	smooth.terms: the values of the smoothing parameters on the log scale, and the partial degrees of freedom associated with each smooth component. Note: Network components always have two smoothing parameters, where the second is a (usually small) ridge parameter.

Author(s)

Alastair Rushworth

See Also

smnet

Examples

```
# Set up an SSN object - this part taken
# from the SSN:::SimulateOnSSN help file
example_network_9<- createSSN(
                    n          = 50,
                    obsDesign  = binomialDesign(200),
                    predDesign = binomialDesign(50),
                    importToR  = TRUE,
                    path = paste(tempdir(),"/example_network_9",sep = ""),
                    treeFunction = iterativeTreeLayout
                    )

# plot the simulated network structure with prediction locations
# plot(example_network_9, bty = "n", xlab = "x-coord", ylab = "y-coord")

## create distance matrices, including between predicted and observed
createDistMat(example_network_9, "preds", o.write=TRUE, amongpred = TRUE)

## extract the observed and predicted data frames
observed_data           <- getSSNdata.frame(example_network_9, "Obs")
prediction_data         <- getSSNdata.frame(example_network_9, "preds")
```

```
## associate continuous covariates with the observation locations
#  data generated from a normal distribution
obs                     <- rnorm(200)
observed_data[,"X"]     <- obs
observed_data[,"X2"]    <- obs^2

## associate continuous covariates with the prediction locations
#  data generated from a normal distribution
pred                    <- rnorm(50)
prediction_data[,"X"]   <- pred
prediction_data[,"X2"]  <- pred^2

## simulate some Gaussian data that follows a 'tail-up' spatial process
sims <- SimulateOnSSN(
               ssn.object     = example_network_9,
               ObsSimDF       = observed_data,
               PredSimDF      = prediction_data,
               PredID         = "preds",
               formula        = ~ 1 + X,
               coefficients   = c(1, 10),
               CorModels      = c("Exponential.tailup"),
               use.nugget     = TRUE,
               CorParms       = c(10, 5, 0.1),
               addfunccol     = "addfunccol")$ssn.object


## extract the observed and predicted data frames, now with simulated values
sim1DFpred        <- getSSNdata.frame(sims, "preds")
sim1preds         <- sim1DFpred[,"Sim_Values"]
sim1DFpred[,"Sim_Values"] <- NA
sims              <- putSSNdata.frame(sim1DFpred, sims, "preds")

# create the adjacency matrix for use with smnet
adjacency    <- get_adjacency(
                 paste(tempdir(), "/example_network_9", sep = ""),
                 net = 1
                 )

# not run - plot the adjacency matrix
# display(adjacency[[1]])

# sometimes it is useful to see which varaibles are valid network weights
# in the data contained within the SSN object
show_weights(sims, adjacency)

# fit a penalised spatial model to the stream network data
# Sim_Values are quadratic in the X covariate.  To highlight
# the fitting of smooth terms, this is treated as non-linear
# and unknown using m().
mod_smn       <- smnet(formula = Sim_Values ~ m(X) + m(X2) +
network(adjacency = adjacency, weight = "shreve"),
                      data.object = sims, netID = 1)
```

```
# not run - plot different summaries of the model
plot(mod_smn, type = "covariates")
plot(mod_smn, type = "segments", weight = 4, shadow = 2)
plot(mod_smn, type = "full", weight = 4, shadow = 2)

# obtain predictions at the prediction locations and plot
# against true values
preds <- predict(mod_smn, newdata = getSSNdata.frame(sims, "preds"))
plot(preds$predictions, sim1preds)

# obtain summary of the fitted model
summary(mod_smn)
```

Index