Package 'SSN'

January 27, 2015

Type Package

Title Spatial Modeling on Stream Networks

Version 1.1.4

Date 2014-6-2

Depends R (>= 3.0.2), RSQLite, sp

Imports MASS, igraph (>= 0.6), maptools, lattice, methods

Author Jay Ver Hoef and Erin Peterson

Maintainer Jay Ver Hoef <ver.hoef@SpatialStreamNetworks.com>

Description

Geostatistical modeling for data on stream networks, including models based on in-stream distance. Models are created using moving average constructions. Spatial linear models, including covariates, can be fit with ML or REML. Mapping and other graphical functions are included.

License GPL-2

LazyLoad yes

LinkingTo BH

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-11-04 01:50:30

R topics documented:

SSN-package 3
additive.function 4
AIC 5
as.SpatialLines 6
binSp 8
BlockPredict 8
BLUP 10
boxplot.SpatialStreamNetwork 11
covparms 13
createDistMat 14

createSSN . . . 17
CrossValidationSSN . . . 19
CrossValidationStatsSSN . . . 21
Design functions . . . 22
EmpiricalSemivariogram . . . 24
fitNS . . . 26
fitRE . . . 27
fitSimBin . . . 27
fitSimGau . . . 28
fitSimPoi . . . 28
fitSp . . . 29
fitSpBk . . . 29
fitSpRE1 . . . 30
fitSpRE2 . . . 30
getPreds . . . 31
getSSNdata.frame . . . 32
getStreamDistMat . . . 34
glmssn . . . 36
glmssn-class . . . 41
GR2 . . . 43
importPredpts . . . 44
importSSN . . . 46
influenceSSN-class . . . 48
InfoCritCompare . . . 49
mf04 . . . 51
mf04p . . . 51
MiddleFork04.ssn . . . 52
plot.glmssn.predict . . . 55
plot.influenceSSN . . . 57
plot.SpatialStreamNetwork . . . 59
plot.Torgegram . . . 61
poiSp . . . 63
predict.glmssn . . . 63
print.summary.glmssn . . . 64
putSSNdata.frame . . . 66
residuals.glmssn . . . 67
SimulateOnSSN . . . 68
SpatialStreamNetwork-class . . . 76
splitPredictions . . . 77
subsetSSN . . . 79
summary.glmssn . . . 80
Torgegram . . . 82
updatePath . . . 84
varcomp . . . 85
writeSSN . . . 86

Index **87**

SSN-package *Spatial Modeling on Stream Networks*

Description

Creates spatial stream network representations in R and fits spatial models.

Details

Package:	SSN
Type:	Package
Version:	1.0
Date:	2011-02-09
License:	GPL-2
LazyLoad:	yes

The SSN package provides tools to fit generalized linear models with spatial autocorrelation using normal likelihood methods (including REML) and quasi-likelihood for Poisson and Binomial families. The spatial formulation is described in Ver Hoef and Peterson (2010) and Peterson and Ver Hoef (2010). The spatial data must be formatted in a geographic information system (GIS) prior to importing it into R. Two custom ArcGIS toolboxes (v 9.3.1) have been provided to format the data: the Functional Linkage of Water Basins and Streams(FLoWS; Theobald et al. 2006) and the Spatial Tools for the Analysis of River Systems (STARS) tooset (Peterson 2011).

Author(s)

Jay Ver Hoef and Erin Peterson <support@SpatialStreamNetworks.com>

References

Ver Hoef, J. M. and Peterson, E. E. (2010) A moving average approach for spatial statistical models of stream networks (with discussion). Journal of the American Statistical Association ***105***, *6–18. DOI: 10.1198/jasa.2009.ap08248. Rejoinder pgs. 22 - 24.*

Peterson, E. E. and Ver Hoef, J. M. (2010) A mixed-model moving-average approach to geostatistical modeling in stream networks. Ecology ***91(3)***, *644–651.*

Peterson E.E.(2011)STARS: Spatial Tools for the Analysis of River Systems: A tutorial. CSIRO Technical Report EP111313. 42p.

Theobald D.M., Norman J.B., Peterson E.E., Ferraz S., Wade A., and Sherburne M.R. (2006) Functional Linkage of Water Basins and Streams (FLoWS) v1 User's Guide: ArcGIS tools for Network-based analysis of freshwater ecosystems. Natural Resource Ecology Lab, Colorado State University, Fort Collins, USA.

`additive.function` *Generate an Additive Function Value*

Description

Generate an additive function value based on a proportional influence variable into an additive function value

Usage

```
additive.function(ssn, VarName, afvName)
```

Arguments

`ssn`	a SpatialStreamNetwork-class object
`VarName`	The name of the the variable that will be used to calculate the additive function value. The data.frame `ssn@data` must contain a column with this name.
`afvName`	The name assigned to the column of additive function values, which are added to the ssn@data data.frame object, as well as the data.frames for the observed and prediction sites.

Details

Calculating the additive function values (AFVs) is a two step process; first the `VarName` values are used to calculate the segment proportional influences (PIs). Then the segment PI's are used to calculate the AFVs for each line segment, observed site, and prediction site in the SpatialStreamNetwork-class object. A detailed description of the segment PIs and the steps used to calculate AFVs are provided in Peterson and Ver Hoef (2010; Appendix A). The AFVs can also be calculated using the Spatial Tools for the Analysis of River Systems (STARS), which is a custom ArcGIS (version 9.3.1) toolbox.

Value

The SpatialStreamNetwork object, `ssn`, with a new column named `VarName` included in the data.frames for the lines, observed sites, and prediction sites to hold the AFVs.

Author(s)

Rohan Shah <support@SpatialStreamNetworks.com>

References

Peterson, E. E. and Ver Hoef, J. M. (2010) A mixed-model moving-average approach to geostatistical modeling in stream networks. Ecology ***91(3)****, 644–651.*

Peterson E.E.(2011)STARS: Spatial Tools for the Analysis of River Systems: A tutorial. CSIRO Technical Report EP111313. 42p.

Examples

```
library(SSN)
#use mf04p SpatialStreamNetwork object, already created
data(mf04p)
#make sure mf04 has the correct path, will vary for each users installation
mf04p <- updatePath(mf04p, system.file("lsndata/MiddleFork04.ssn", package = "SSN"))

#Calculate an additive function value based on an existing column.
names(mf04p@data)
mf04p <- additive.function(mf04p, "h2oAreaKm2", "areaAFV")
#notice that a column called afvArea was already included, and "areaAFV" replicates it
# in the lines data
head(mf04p@data)
# and in the observed points data
head(getSSNdata.frame(mf04p))
# and in the prediction points data
head(getSSNdata.frame(mf04p, "pred1km"))
```

`AIC` *AIC for glmssn objects*

Description

`AIC.glmssn` is a method that calculates AIC for fitted `glmssn` objects.

Usage

```
  ## S3 method for class 'glmssn'
AIC(object, ..., k = 2)
```

Arguments

`object`	an object of class glmssn
`...`	optionally more fitted model objects
`k`	numeric, the penalty per parameter to be used; the default k = 2 is the classical AIC.

Details

AIC is a generic function and this implements a method for `glmssn` objects

Value

a numeric AIC value for the specified `glmssn` object

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

glmssn

Examples

```
library(SSN)
# NOT RUN
#mf04p <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#  package = "SSN"), predpts = "pred1km", o.write = TRUE)
#  use SpatialStreamNetwork object mf04p that was already created
data(mf04p)
#make sure mf04p has the correct path, will vary for each users installation
mf04p@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")

# get some model fits stored as data objects
data(modelFits)
#NOT RUN use this one
#fitSp <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#    ssn.object = mf04p, EstMeth = "REML", family = "Gaussian",
#    CorModels = c("Exponential.tailup","Exponential.taildown",
#    "Exponential.Euclid"), addfunccol = "afvArea")
#make sure fitSP has the correct path, will vary for each users installation
fitSp$ssn@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")

#note the model was fitted using REML, so fixed effects have
# been integrated out
summary(fitSp)
AIC(fitSp)
```

---

as.SpatialLines *Methods to convert SpatialStreamNetwork objects classes to sp classes*

Description

Converts SpatialStreamNetwork objects to sp objects.

Usage

```
## S3 method for class 'SpatialStreamNetwork'
as.SpatialLines(x, ...)
## S3 method for class 'SpatialStreamNetwork'
as.SpatialPoints(x, data = "Obs", ...)
## S3 method for class 'SpatialStreamNetwork'
as.SpatialPointsDataFrame(x, data = "Obs", ...)
```

Arguments

x	an SpatialStreamNetwork object to be converted to class SpatialLines, SpatialPoints, or SpatialPointsDataFrame from the sp package.
data	the data set in the SpatialStreamNetwork object to convert. The SpatialStreamNetwork object can hold multiple spatial point data sets, including the observed data and multiple prediction data sets. See SpatialStreamNetwork-class.
...	optional arguments for specific methods written for these generics

Value

as.SpatialLines.SpatialStreamNetwork converts an object of class SpatialStreamNetwork to an object of class SpatialLines from the sp package, as.SpatialPoints.SpatialStreamNetwork converts an object of class SpatialStreamNetwork to an object of class SpatialPoints from the sp package, and as.SpatialPointsDataFrame.SpatialStreamNetwork converts an object of class SpatialStreamNetwork to an object of class SpatialPointsDataFrame from the sp package,

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

spplot

Examples

```
library(SSN)
# NOT RUN
#mf04 <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#  package = "SSN"), o.write = TRUE)
#  use SpatialStreamNetwork object mf04p that was already created
data(mf04p)
#Update path in mf04, will vary for each users installation
mf04p <- updatePath(mf04p, system.file("lsndata/MiddleFork04.ssn", package = "SSN"))

names(mf04p)

#---------
# make plots using sp methods
#---------
#plot the stream lines
plot(as.SpatialLines(mf04p), col = "blue")
# add the observed locations with size proportional
# to mean summer temperature
plot(as.SpatialPoints(mf04p), pch = 19,
  cex = as.SpatialPointsDataFrame(mf04p)$Summer_mn/9 , add = TRUE)
# add the prediction locations on the 1 km spacing
plot(as.SpatialPoints(mf04p, data = "pred1km"), cex = 1.5, add = TRUE)
# add the dense set of points for block prediction on Knapp segment
```

```
plot(as.SpatialPoints(mf04p, data = "Knapp"), pch = 19, cex = 0.3,
  col = "red", add = TRUE)
```

`binSp` *Fitted glmssn object for example data set MiddleFork.ssn*

Description

The MiddleFork04.ssn data folder contains the spatial, attribute, and topological information needed to construct a spatial stream network object using the SSN package. This is a fitted model using the `glmssn` function.

Details

See the help for `glmssn` for how the model was created.

Examples

```
library(SSN)
data(modelFits)
ls()
```

`BlockPredict` *Block Predicton for Streams Data*

Description

Block prediction for objects of class glmssn-class

Usage

```
BlockPredict(object, predpointsID)
```

Arguments

`object`	an object of class glmssn
`predpointsID`	a valid prediction points ID

Details

This function operates on `glmssn` objects in much the same way as the `predict` function. `BlockPredict` uses the locations in the predpointsID data set to compute the average prediction value in the area defined by the prediction locations. These prediction locations are used to approximate the integral over that area, so they should be evenly spaced and dense in the area where block prediction is desired. The user needs to create these prediction locations and include them in the SSN object prior to fitting the model with glmssn.

Value

A data.frame with one row and two columns. The first column, BlockPredEst, is the average prediction value, and the second column, BlockPredSE, is the standard error of the block prediction.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

References

Ver Hoef, J. M.. Peterson, E. E. and Theobald, D. (2006) Spatial statistical models that use flow and stream distance. *Environmental and Ecological Statistics* **13**, 449-464. DOI: 10.1007/s10651-006-0022-8.

Examples

```
library(SSN)
# NOT RUN
# mf04 <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#         package = "SSN"), o.write = TRUE)
# use SpatialStreamNetwork object mf04 that was already created
data(mf04p)
#make sure mf04p has the correct path, will vary for each users installation
mf04p <- updatePath(mf04p, system.file("lsndata/MiddleFork04.ssn", package = "SSN"))

## NOT RUN Distance Matrix has already been created
## createDistMat(mf04)
# mf04p <- importPredpts(mf04p, "Knapp", "ssn")
# mf04p <- importPredpts(mf04p, "CapeHorn", "ssn")
names(mf04p)
## NOTE: need the amongpreds distance matrices for block prediction
#createDistMat(mf04p, predpts = "Knapp", o.write = TRUE, amongpreds = TRUE)
# just do CapeHorn Example
createDistMat(mf04p, predpts = "CapeHorn", o.write = TRUE, amongpreds = TRUE)

# NOT RUN see densely gridded prediction points on stream
# plot(mf04p, PredPointsID = "Knapp")

# NOT RUN   fit the model first
#fitSpBk <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#    ssn.object = mf04p, EstMeth = "REML", family = "Gaussian",
#    CorModels = c("Exponential.tailup","Exponential.taildown",
#    "Exponential.Euclid"), addfunccol = "afvArea")
data(modelFits)
fitSpBk$ssn.object <- updatePath(fitSpBk$ssn.object,
system.file("lsndata/MiddleFork04.ssn", package = "SSN"))
# one-at-a-time predictions for CapeHorn stream
fitSpPredC <- predict(fitSpBk, "CapeHorn")
# NOT RUN plot densely gridded prediction points on stream
# plot(glmssn.BPCapeHorn, "Summer_mn")
# block prediction for CapeHorn stream
```

```
BlockPredict(fitSpBk, "CapeHorn")
## NOT RUN Another example
# one-at-a-time predictions for Knapp stream
#fitSpPredK <- predict(fitSpBk, "Knapp")
# NOT RUN plot densely gridded prediction points on stream
# plot(fitSpPredK, "Summer_mn")
# block prediction for Knapp stream
#BlockPredict(fitSpBk, "Knapp")
```

BLUP *Compute the joint mean, variance and covariance of any random effects in a glmssn model conditional on the data*

Description

Compute the joint mean, variance and covariance of any random effects in a glmssn model conditional on the data. This assumes each random effect has a Gaussian distribution with mean zero and covariance matrix sigma^2 * Identity. We just plug in the REML estimate of sigma^2 from the fitted glmssn model object.

Usage

```
BLUP(model, RE = NULL)
```

Arguments

model	An object of class glmssn-class
RE	Names of random effects (RE), defaults to all REs in the glmssn object, if any

Details

Similar to BLUP in the regress package.

Value

Mean	A vector of means for each Random Effect
Variance	A vector of variances for each Random Effect
Covariance	A variance-covariance matrix for the Random Effects

Author(s)

David Clifford <support@SpatialStreamNetworks.com>

Examples

```
library(SSN)
# NOT RUN
# mf04 <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#         package = "SSN"), o.write = TRUE)
# use SpatialStreamNetwork object mf04 that was already created
data(mf04)
#make sure mf04 has the correct path, will vary for each users installation
mf04 <- updatePath(mf04, system.file("lsndata/MiddleFork04.ssn", package = "SSN"))

## NOT RUN Distance Matrix has already been created
## createDistMat(mf04)

# The models take a little time to fit, so they are NOT RUN
# Uncomment the code to run them
# Alternatively, you can load the fitted models first to look at results
data(modelFits)

## Random effect model using STREAMNAME as our random effect
#fitRE <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#    ssn.object = mf04, EstMeth = "REML", family = "Gaussian",
#    CorModels = c("STREAMNAME"))
summary(fitRE)
## random effects details
fitREBLUP <- BLUP(fitRE)
str(fitREBLUP)
fitREBLUP$Mean

## spatial stream model with a random effect
#fitSpRE1 <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#    ssn.object = mf04, EstMeth = "REML", family = "Gaussian",
#    CorModels = c("STREAMNAME","Exponential.tailup"),
#    addfunccol = "afvArea")
fitRE1BLUP <- BLUP(fitSpRE1)
str(fitRE1BLUP)
fitRE1BLUP$Mean
```

`boxplot.SpatialStreamNetwork` *Box-and-whisker plots for data within SpatialStreamNetwork objects.*

Description

The `boxplot` function creates box-and-whisker plots for data within SpatialStreamNetwork-classobjects.

Usage

```
## S3 method for class 'SpatialStreamNetwork'
boxplot(x, variable, ...)
```

Arguments

x	an object of class SpatialStreamNetwork-class
variable	the variable (in quotes) for which the boxplots are being created, or a formula, such as y ~ grp, where y is a numeric vector of data values and grp is a grouping of variables (usually a factor).
...	see boxplot for additional arguments

Details

boxplot is a generic function that has been adapted for SpatialStreamNetwork-classobjects. Use names to get a list of the variable names within the SpatialStreamNetwork object; the boxplot boxplot will only work for the observed data set.

Value

A graph is produced, and a list with the following components:

stats	a matrix, each column contains the extreme of the lower whisker, the lower hinge, the median, the upper hinge and the extreme of the upper whisker for each group/plot. If all the inputs have the same class attribute, then so will this component.
n	a vector with the number of observations in each group.
conf	a matrix where each column contains the lower and upper extremes of the notch.
out	the values of any data points which lie beyond the extremes of the whiskers.
group	a vector of the same length as out whose elements indicate to which group the outlier belongs.
names	a vector of names for the groups

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

References

see boxplot

See Also

boxplot.stats

Examples

```
library(SSN)
# NOT RUN
# mf04 <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#         package = "SSN"), o.write = TRUE)
# use SpatialStreamNetwork object mf04 that was already created
data(mf04)

boxplot(mf04, "Summer_mn")
boxplot(mf04, Summer_mn ~ STREAMNAME, main = "Summer_mn ~ STREAMNAME",
      col = "gray", xlab = "STREAMNAME", ylab = "Summer Mean Temperature")
```

`covparms` *Get Covariance Parameters*

Description

Displays the covariance parameter estimates for the autocovariance function(s) in the `glmssn` object.

Usage

```
covparms(object)
```

Arguments

`object`	An object of class glmssn-class created using the `glmssn` function.

Details

A glmssn-classobject allows up to three autocovariance functions, as well as random effects. The `covparms` function displays the sill and range for each model, as well as the nugget.

Value

Returns a `data.frame` containing the names of the autocovariance functions and random effects, the parameter names, and their corresponding estimates.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

`glmssn`

Examples

```
library(SSN)
# NOT RUN
# mf04 <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#         package = "SSN"), o.write = TRUE)
# use SpatialStreamNetwork object mf04 that was already created
data(mf04)
#make sure mf04 has the correct path, will vary for each users installation
mf04 <- updatePath(mf04, system.file("lsndata/MiddleFork04.ssn", package = "SSN"))

## NOT RUN Distance Matrix has already been created
## createDistMat(mf04)

# The models take a little time to fit, so they are NOT RUN
# Uncomment the code to run them
# Alternatively, you can load the fitted models first to look at results
data(modelFits)

## 3 component spatial model
#fitSp <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#    ssn.object = mf04, EstMeth = "REML", family = "Gaussian",
#    CorModels = c("Exponential.tailup","Exponential.taildown",
#    "Exponential.Euclid"), addfunccol = "afvArea")

## Look at variance components in more detail
covparms(fitSp)
```

`createDistMat` *Calculate Hydrologic Distances for a SpatialStreamNetwork Object*

Description

Creates a collection of (non-symmetric) matrices containing pairwise downstream hydrologic distances between sites in a SpatialStreamNetwork object

Usage

```
createDistMat(ssn, predpts = NULL, o.write = FALSE, amongpreds = FALSE)
```

Arguments

`ssn`	a SpatialStreamNetwork-class object
`predpts`	a valid predpoints ID from the ssn
`o.write`	If TRUE, overwrite existing distance matrices. Defaults to `FALSE`.
`amongpreds`	If TRUE, compute the distances between the prediction sites i. Defaults to `FALSE`.

Details

A distance matrix that contains the hydrologic distance between any two sites in SpatialStreamNetwork object is needed to fit a spatial statistical model using the tail-up and tail-down autocovariance functions described in Ver Hoef and Peterson (2010). These models are implemented in R via `glmssn` in the SSN package. The hydrologic distance information needed to model the covariance between flow-connected (i.e. water flows from one location to the other) and flow-unconnected (i.e. water does not flow from one location to the other, but they reside on the same network) locations differs. The total hydrologic distance is a directionless measure; it represents the hydrologic distance between two sites, ignoring flow direction. The hydrologic distance from each site to a common downstream stream junction is used when creating models for flow-unconnected pairs, which we term downstream hydrologic distance. In contrast, the total hydrologic distance is used for modeling flow-connected pairs, which we term total hydrologic distance.

A downstream hydrologic distance matrix provides enough information to meet the data requirements for both the tail-up and tail-down models. When two locations are flow-connected, the downstream hydrologic distance from the upstream location to the downstream location is greater than zero, but it is zero in the other direction. When two locations are flow-unconnected the downstream hydrologic distance will be greater than zero in both directions. A site's downstream hydrologic distance to itself is equal to zero. The format of the downstream hydrologic distance matrix is efficient because distance information needed to fit both the tail-up and tail-down models is only stored once. As an example, a matrix containing the total hydrologic distance between sites is easily calculated by adding the downstream distance matrix to its transpose.

The downstream hydrologic distances are calculated based on the binaryIDs and stored as matrices. The matrices are stored in a directory named 'distance', which is created by the createDistMat function within the .ssn directory. The distance directory will always contain at least one directory named 'obs', which contains a number of .RData files, one for each network that has observed sites residing on it. The naming convention for the files is based on the netID number (e.g. dist.net1.RData). Each matrix in the 'obs' folder contains the information to form a square matrix, which contains the downstream hydrologic distance between each pair of observed sites on the network. Direction is preserved, with columns representing the FROM site and rows representing the TO site. Row and column names correspond to the pid attribute for each site.

If the argument `predpts` is specified in the call to the function, the downstream hydrologic distances between the observed and prediction sites will also be computed. A new directory is created within the distance directory, with the name corresponding to the predpoints ID (e.g. "preds"). A sequence of .RData files is created within this directory, similar to the structure for the observed sites, except that two objects are stored for each network that contains *both* observed and prediction sites. The letters `a` and `b` are used in the naming convention to distinguish between the two objects (e.g. dist.net1.a and dist.net1.b). The matrices that these objects represent are not necessarily square. In matrices of type `a`, rows correspond to observed locations and columns to prediction locations. In contrast, rows correspond to prediction locations and columns to observed locations in matrices of type `b`. Direction is also preserved, with columns representing the FROM site and rows representing the TO site in both object types. Again, row and column names correspond to the pid attribute for each site.

If the argument `amongpreds` is set to TRUE, the downstream hydrologic distances will also be computed between prediction sites, for each network. Again these are stored within the distance directory with the name corresponding to the predpoints ID. The naming convention for these prediction to prediction site distance matrices is the same as the distance matrices stored in the 'obs' directory (e.g. dist.net1.RData). These extra distance matrices are needed to perform block Kriging

using the glmssn

Value

The createDistMat function creates a collection of hierarchical directories in the ssn@path directory, which store the pairwise distances between sites associated with the SpatialStreamNetwork-class object. See details section for additional information.

Author(s)

Erin E. Peterson & Rohan Shah <support@SpatialStreamNetworks.com>

References

Ver Hoef, J.M. and Peterson, E.E. (2010) A moving average approach to spatial statistical models of stream networks. The Journal of the American Statistical Association, ***105(489)****, 22–24*

See Also

SpatialStreamNetwork-class, importSSN, createSSN, glmssn

Examples

```
library(SSN)
# NOT RUN
# mf04 <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#         package = "SSN"), o.write = TRUE)
# use SpatialStreamNetwork object mf04 that was already created
data(mf04)
mf04 <- updatePath(mf04, system.file("lsndata/MiddleFork04.ssn",
package = "SSN"))

# create distance matrix among observed data points
createDistMat(mf04, o.write = TRUE)

# create distance matrix among observed data points
#     and between observed and prediction points
data(mf04p)
mf04p <- updatePath(mf04p, system.file("lsndata/MiddleFork04.ssn",
package = "SSN"))
createDistMat(mf04p, predpts = "pred1km", o.write = TRUE)

# NOT RUN include prediction to prediction site distances
# createDistMat(mf04p, predpts = "pred1km", o.write = TRUE, amongpreds = TRUE)
```

createSSN *Create an SpatialStreamnetwork Object*

Description

Generates a random tree structure, with observed and prediction locations, and stores as an object of class SpatialStreamNetwork-class.

Usage

```
createSSN(n, obsDesign, predDesign = noPoints, path, importToR = FALSE,
    treeFunction = igraphKamadaKawai)
```

Arguments

n	A numeric vector, where the generated SpatialStreamNetwork object will consist of length(n) distinct random tree structures, with the ith tree structure consisting of n[i] straight line segments.
obsDesign	A function representing a sampling strategy. It is used to select observation points on the network. Input obsDesign is required and cannot have value noPoints, as there must be at least one observed point generated. point generated. At present the only implemented designs are binomialDesign, systematicDesign, systematicDesign, poissonDesign and hardCoreDesign. For further details, which will allow users to to write your their own design function, please see the design functions.
predDesign	A function having the same signature as the obsDesign input, but this time the function generates the prediction points. This defaults to noPoints, which generates no prediction points. Otherwise any of the design functions which can be used for input obsDesign can also be used for input predDesign.
path	The path where the new .ssn directory is to be stored.
importToR	If TRUE then a call to importSSN is made immediately and the imported SpatialStreamNetwork object is returned. If FALSE then no value is returned.
treeFunction	An input function that is used to generate the tree structure. This function must have the signature function(n) Where n is the desired number of edges for the generated network. This function must return a list with four entries. Entry graph must be an igraph object representing the generated network. Entry locations must be a numeric matrix giving the locations of all the points, in order. That is, the first row contains the coordinates for point 0, the second the coordinates for point 1, etc. Entry initialPoint gives the number of the inital point in that network. The two possible values for this input are iterativeTreeLayout and igraphKamadaKawai. igraphKamadaKawai is the default and uses the graph.tree function from the igraph package, with the Kamada-Kawai layout function. iterativeTreeLayout generates more natural looking tree structures but is slower and can fail to function.

Details

This function generates random tree structure using the igraph package and then turns these into an SpatialStreamNetwork object with prediction and observation sites generated by the obsDesign and predDesign functions. The main difficulty is assigning locations to the vertices of the random trees, in such a way that the result has the sort of layout that we want. This is a graph layout / embedding problem, more specifically a tree layout problem. For now we are using the layout.kamada.kawai function of the igraph package to construct this layout. Unlike some of the other layouts available, it still gives interesting layouts when applied to trees (some of the others tend to give highly structured layouts for such a simple graph. The downside is that it the resulting layout can have self intersections, and often does.

Value

An SpatialStreamNetwork object if importToR is TRUE, otherwise NULL.

Author(s)

Rohan Shah and Pascal Monestiez <support@SpatialStreamNetworks.com>

See Also

SimulateOnSSN, importSSN, igraph

Examples

```
library(SSN)
#Simulate three networks, the first consisting of ten straight line segments,
#the second of 20 and the third of 30. There are two observed points on the first
#network, four on the second and six on the third. All the observed points are
#distributed uniformly. The default for prediction points is no prediction points.
ssn1 <- createSSN(c(10, 20, 30), obsDesign = binomialDesign(c(2,4,6)),
  path=paste(tempdir(),"/simulated1.ssn", sep = ""), importToR = TRUE)
#NOT RUN plot(ssn1)

#Same as above, but using iterativeTreeLayout
ssn2 <- createSSN(c(10, 20, 30), obsDesign = binomialDesign(c(2,4,6)),
  path=paste(tempdir(),"/simulated2.ssn", sep = ""), importToR = TRUE,
  treeFunction = iterativeTreeLayout)
#NOT RUN plot(ssn2)

#Simulate the same number of line segments per network, but this time the observed
#points have the distribution of a Poisson process with rates 2, 1 and 0.5
#respectively. Again there are no prediction points.
ssn3 <- createSSN(c(10, 20, 30), obsDesign = poissonDesign(c(2,1,0.5)),
  path=paste(tempdir(),"/simulated3.ssn", sep = ""), importToR = TRUE)
#NOT RUN plot(ssn3)

#Simulate the same number of line segments per network, but this time the observed
#points have a hard-core process distribution. Two hundred points are placed on
#every network according to the binomial process, and then points are removed
```

```
#until every poir of points is at least a distance 0.5 apart on the first network,
#0.25 on the second and 0.1 on the third. Again there are no prediction points.
ssn4 <- createSSN(c(10, 20, 30), obsDesign = hardCoreDesign(200, c(0.5, 0.25, 0.1)),
  path=paste(tempdir(),"/simulated4.ssn", sep = ""), importToR = TRUE)
#NOT RUN plot(ssn4)

#This time there are the same number of observed points on each of the networks,
#but there are ten prediction sites on each network.
ssn5 <- createSSN(c(10, 20, 30), obsDesign = binomialDesign(c(2, 4, 6)),
  predDesign = binomialDesign(c(10, 10, 10)),
  path=paste(tempdir(),"/simulated5.ssn", sep = ""),
  importToR = TRUE)
#NOT RUN plot(ssn5)

#This time the observed and prediction points are a regular grid, spacing 0.5
ssn6 <- createSSN(c(10, 20, 30), obsDesign = systematicDesign(0.5),
  predDesign = systematicDesign(0.5),
  path=paste(tempdir(),"/simulated6.ssn", sep = ""),
  importToR = TRUE)
#NOT RUN plot(ssn6)

#Same as example number 5, but this time the observed (but not predicted) points
#are replicated twice with different time values
ssn7 <- createSSN(c(10, 20, 30), obsDesign = binomialDesign(c(2, 4, 6),
  replications=2),
  predDesign = binomialDesign(c(10, 10, 10)),
path=paste(tempdir(),"/simulated7.ssn", sep = ""),
  importToR = TRUE)
#NOT RUN plot(ssn7)

ssn7@obspoints@SSNPoints[[1]]@point.data
ssn5@obspoints@SSNPoints[[1]]@point.data
```

`CrossValidationSSN` *Compute Crossvalidation Values for glmssn Objects*

Description

`CrossValidationSSN` operates on glmssn objects. The response values are removed one at a time and the estimated model is used to predict each of the removed values along with the standard errors of prediction.

Usage

```
CrossValidationSSN(object)
```

Arguments

`object`	an object of class glmssn-class

Details

This function removes the response values one at a time. Then it uses the estimated model to predict each of the removed values along with the standard errors of prediction.

Value

Output is a data.frame with two columns, the predictions "cv.pred" and their standard errors "cv.se", and the data are in the same order as the data in the glmssn object.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

Examples

```
library(SSN)
# NOT RUN
# mf04 <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#         package = "SSN"), o.write = TRUE)
# use SpatialStreamNetwork object mf04 that was already created
data(mf04)
#make sure mf04 has the correct path, will vary for each users installation
mf04 <- updatePath(mf04, system.file("lsndata/MiddleFork04.ssn", package = "SSN"))

## NOT RUN Distance Matrix has already been created
## createDistMat(mf04)

# The models take a little time to fit, so they are NOT RUN
# Uncomment the code to run them
# Alternatively, you can load the fitted models first to look at results
data(modelFits)

## 3 component spatial model
#fitSp <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#    ssn.object = mf04, EstMeth = "REML", family = "Gaussian",
#    CorModels = c("Exponential.tailup","Exponential.taildown",
#    "Exponential.Euclid"), addfunccol = "afvArea")

fitSpCrVal <- CrossValidationSSN(fitSp)
str(fitSpCrVal)
# NOT RUN
# data are sorted by netID, then pid within netID.  This is different that
# the original data order, so get the sorted values of the response variable
# plot(fitSp$sampinfo$z, fitSpCrVal[,"cv.pred"], pch = 19)
```

`CrossValidationStatsSSN` *Compute Summary Statistics on Crossvalidation Values for glmssn Objects*

Description

`CrossValidationStatsSSN` operates on glmssn objects and uses the `CrossValidationSSN` function to create a data.frame of crossvalidation predictions and standard errors. Then it computes summary statistics such as bias and confidence interval coverage based on cross-validation.

Usage

```
CrossValidationStatsSSN(object)
```

Arguments

`object`	an object of class 'glmssn'

Details

This function uses the `CrossValidationSSN` function to create a data.frame of crossvalidation predictions and standard errors. Then it computes summary statistics on bias, root mean-squared prediction errors (RMSPE), and confidence interval coverage based on cross-validation. Output is a data.frame with with a single entry for the columns as describe below. In the descriptions, obs is an observed data value, pred is its prediction using crossvalidation, and predSE is the prediction standard error using crossvalidation.

bias Bias, computed as mean(obs - pred).

std.bias Standardized bias, computed as mean((obs - pred)/predSE).

RMSPE Root mean-squared prediction error, computed as sqrt(mean((obs - pred)^2))

RAV Root average variance, computed as sqrt(mean(predSE^2)). If the prediction standard errors are being estimated well, this should be close to RMSPE.

std.MSPE standardized mean-squared prediction error, computed as mean(((obs - pred)/predSE)^2). If the prediction standard errors are being estimated well, this should be close to 1.

cov.80 The proportion of times that obs was within the prediction interval formed from pred +- qt(.9, df)*predSE, where qt is the quantile t function, and df is the number of degrees of freedom. If there is little bias and the prediction standard errors are being estimated well, this should be close to 0.8 for large sample sizes.

cov.90 The proportion of times that obs was within the prediction interval formed from pred +- qt(.95, df)*predSE, where qt is the quantile t function, and df is the number of degrees of freedom. If there is little bias and the prediction standard errors are being estimated well, this should be close to 0.9 for large sample sizes.

cov.95 The proportion of times that obs was within the prediction interval formed from pred +- qt(.975, df)*predSE, where qt is the quantile t function, and df is the number of degrees of freedom. If there is little bias and the prediction standard errors are being estimated well, this should be close to 0.95 for large sample sizes.

Value

Output is a data.frame with with a single entry for the columns as listed above.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

InfoCritCompare, glmssn, CrossValidationSSN

Examples

```
library(SSN)
# NOT RUN
# mf04 <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#         package = "SSN"), o.write = TRUE)
# use SpatialStreamNetwork object mf04 that was already created
data(mf04)
#make sure mf04 has the correct path, will vary for each users installation
mf04 <- updatePath(mf04, system.file("lsndata/MiddleFork04.ssn", package = "SSN"))

## NOT RUN Distance Matrix has already been created
## createDistMat(mf04)

# The models take a little time to fit, so they are NOT RUN
# Uncomment the code to run them
# Alternatively, you can load the fitted models first to look at results
data(modelFits)

## 3 component spatial model
#fitSp <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#    ssn.object = mf04, EstMeth = "REML", family = "Gaussian",
#    CorModels = c("Exponential.tailup","Exponential.taildown",
#    "Exponential.Euclid"), addfunccol = "afvArea")
CrossValidationStatsSSN(fitSp)
```

Design functions *Design functions*

Description

Functions to generate points on a network.

Usage

```
binomialDesign(n, replications=1, rep.variable = "Time", rep.values)
poissonDesign(lambda, replications=1, rep.variable = "Time", rep.values)
hardCoreDesign(n, inhibition_region, replications=1, rep.variable = "Time", rep.values)
systematicDesign(spacing, replications=1, rep.variable = "Time", rep.values)
noPoints
```

Arguments

`n`	A numeric vector having length 1 or the same length as the number of networks. This represents the number of points to be spread across a network.
`lambda`	A numeric vector having length 1 or the same length as the number of networks. This represents the rate at which points occur on a network.
`inhibition_region`	A numeric vector having length 1 or the same length as the number of networks. This represents the size of the inhibition region on a network.
`spacing`	A numeric vector having length 1 or the same length as the number of networks. This represents the desired spacing for the regular grid of points.
`replications`	The number of replications of each point.
`rep.variable`	The name of the variable that will distinguish between the replicated points.
`rep.values`	The values that will be given to the variable named `rep.variable`. `rep.values[1]` will be assigned for replication 1, `rep.values[2]` for replication 2, etc. Defaults to `1:replications` if no value is given.

Details

These design functions are intended to be used in the `obsDesign` or `predDesign` inputs of the createSSN function. The `binomialDesign` function represents a binomial process - A number `n[i]` of points are distributed randomly and uniformly across network `i` (or `n` points if `n` is a single number).

The `poissonDesign` function represents a poisson process, where points occur at rate `lambda[i]` on network `i` (or `lambda` if `lambda` is a single number).

The `hardCoreDesign` function represents a hard-core process where `n[i]` (or `n` if `n` has length 1) points are distributed uniformly and randomly on network `i`, and then points are removed until all points are at least `inhibition_region[i]` distant from each other (or `inhibition_region` if `inhibition_region` has length 1).

The `systematicDesign` function gives a deterministic and regular set of points. Starting from the outlet points are placed upwards along the network, at a fixed distance from the previous point. Note that while the generated grids are regular in a certain sense, they can appear non-regular at certains points from visual inspection. This is because it is impossible to generate a grid of truly equal-spaced points on a network.

The noPoints function simply generates zero points across all networks. Note that this cannot be used as the design for the observed points as there must be at least one observed point. Also this is used directly without any parameters, unlike the other design functions.

A design function must have the form

```
function(tree.graphs, edge_lengths, locations, edge_updist, distance_matrices)
```

All inputs to the design function are lists of length `n` where `n` is the number of trees. Input `tree.graphs[[i]]` is an object of class `igraph` which represent the ith generated network in a graph theoretic sense; without any specific locations assigned to the vertices. `edge_lengths[[i]]` contains the lengths of the edges for the ith tree, in the same order as the edges appear in the corresponding `igraph` object. Input `locations[[i]]` is a matrix with `n[i]` rows and 2 columns giving the locations of the points on that network. `edge_updist[[i]]` is a numeric vector which gives the upstream distance from the downstream point of every stream segment, in the same order as these edges appear in the corresponding `igraph` object. `distance_matrices[[i]]` is a matrix with `n[i]` rows and columns, giving the network distance between the downstream points of a pair of edges, where the edges are ordered in the same way as in the `igraph` object. To summarise, on tree number `i` if edge number `k` has downstream point `k_` and edge number `l` has downstream point `l_` then `edge_lengths[[i]][k]` gives the length of edge number `k`, `edge_updist[[i]][k]` gives the distance from point `k_` to the outlet of the stream network, and `distance_matrices[[i]][k, l]` gives the network distance between points `k_` and `l_`. Note that some of these inputs may have associated row or column names, but these should be ignored.

Value

A design function having the signature mentioned above.

Author(s)

Rohan Shah <support@SpatialStreamNetworks.com>

See Also

createSSN

EmpiricalSemivariogram

Empirical Semivariogram Based on Euclidean Distance

Description

`EmpiricalSemivariogram` computes the empirical semivariogram from the data based on Euclidean distance.

Usage

```
EmpiricalSemivariogram(ssn.object, varName, nlag = 20, directions = c(0, 45, 90, 135),
   tolerance = 22.5, inc = 0, maxlag = 1e+32, nlagcutoff = 1, EmpVarMeth = "MethMoment")
```

Arguments

ssn.object	an object of class SpatialStreamNetwork-class or influenceSSN-class
varName	a response or residual variable name in the data.frame of observed data in the SpatialStreamNetwork or influenceSSN object.
nlag	the number of lag bins to create, by direction if directions are specified. The distance between endpoints that define a bin will have equal lengths for all bins. The bin sizes are then determined from the minimum lag in the data, and the specification of maxlag.
directions	directions in degrees clockwise from north that allow lag binning to be directional. Default is c(0, 45, 90, 135). Values should be between 0 and 180, as there is radial symmetry in orientation between two points.
tolerance	the angle on either side of the directions to determine if a pair of points falls in that direction class. Note, a pair of points may be in more than one lag bin if tolerances for different directions overlap.
inc	the distance increment for each bin class. Default is 0, in which case maxlag and nclasses determine the distance increments.
maxlag	the maximum lag distance to consider when binning pairs of locations by the hydrologic distance that separates them. If the specified maxlag is larger than the maximum distance among pairs of points, then maxlag is set to the maximum distance among pairs. If inc is greater than 0, then maxlag is disregarded.
nlagcutoff	the minimum number of pairs needed to estimate the semivariance for a bin. If the sample size is less than this value, the semivariance for the bin is not calculated.
EmpVarMeth	method for computing semivariances. The default is "MethMoment", the classical method of moments, which is just the average difference-squared within bin classes. "Covariance" computes covariance rather than semivariance, but may be more biased because it subtracts off the simple mean of the response variable. "RobustMedian" and "RobustMean" are robust estimators proposed by Cressie and Hawkins (1980). If v is a vector of all pairwise square-roots of absolute differences within a bin class, then RobustMedian computes median(v)^4/.457. "RobustMean" computes mean(v)^4/(.457 + .494/length(v)).

Value

A list of six vectors. The lengths of all vectors are equal, which is equal to nlag*(number of directions) - (any missing lags due to nlagcutoff).

distance	the mean Euclidean distance separating pairs of sites used to calculate the semivariance for each bin
gamma	the estimated semivariance for each bin, based on EmpVarMeth
np	the number of pairs of sites used to calculate the semivariance for each bin
azimuth	the azimuth, equivalent to the direction, used for the bin class
hx	the x-coordinate of the center of the bin lag.
hy	the y-coordinate of the center of the bin lag.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

Examples

```
library(SSN)
# NOT RUN
# mf04 <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#         package = "SSN"), o.write = TRUE)
# use SpatialStreamNetwork object mf04 that was already created
data(mf04)
#make sure mf04 has the correct path, will vary for each users installation
mf04 <- updatePath(mf04, system.file("lsndata/MiddleFork04.ssn", package = "SSN"))

# Compute the empirical semivariogram for the raw data.
# the number of pairs used to estimate the semivariance
EVout <- EmpiricalSemivariogram(mf04, "Summer_mn", directions = 0, tolerance = 180,
    nlag = 10)
# Plot it and set the point size relative to the number of pairs used to estimate
# the semivariance
plot(EVout$distance, EVout$gamma, pch = 19, cex = EVout$np/100, ylim = c(0,6),
    ylab = "Gamma", xlab = "Distance", col = "blue",
    main = "Empirical Semivariogram - Raw Data")

# generate and plot an empirical semivariogram based on model residuals
data(modelFits)
#make sure fitSP has the correct path, will vary for each users installation
fitSp$ssn@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")
fitSpRes <- residuals(fitSp)
names(getSSNdata.frame(fitSpRes))
EVresid <- EmpiricalSemivariogram(fitSpRes, "_resid_", directions = 0,
    tolerance = 180, nlag = 7,maxlag = 15000,)
plot(EVresid$distance, EVresid$gamma, pch = 19, cex = EVresid$np/50, ylim = c(0,6),
    ylab = "Gamma", xlab = "Distance", main = "Empirical Semivariogram - Residuals")
```

---

fitNS *Fitted glmssn object for example data set MiddleFork.ssn*

Description

The MiddleFork04.ssn data folder contains the spatial, attribute, and topological information needed to construct a spatial stream network object using the SSN package. This is a fitted model using the glmssn function.

Details

See the help for glmssn for how the model was created.

Examples

```
library(SSN)
data(modelFits)
ls()
```

`fitRE` *Fitted glmssn object for example data set MiddleFork.ssn*

Description

The MiddleFork04.ssn data folder contains the spatial, attribute, and topological information needed to construct a spatial stream network object using the SSN package. This is a fitted model using the `glmssn` function.

Details

See the help for `glmssn` for how the model was created.

Examples

```
library(SSN)
data(modelFits)
ls()
```

`fitSimBin` *Fitted glmssn object for simulated data*

Description

This is a fitted model using the `glmssn` function after simulating data.

Details

See the help for `SimulateOnSSN` for how the model was created.

Examples

```
library(SSN)
data(modelFits)
ls()
```

fitSimGau *Fitted glmssn object for simulated data*

Description

This is a fitted model using the glmssn function after simulating data.

Details

See the help for SimulateOnSSN for how the model was created.

Examples

```
library(SSN)
data(modelFits)
ls()
```

fitSimPoi *Fitted glmssn object for simulated data*

Description

This is a fitted model using the glmssn function after simulating data.

Details

See the help for SimulateOnSSN for how the model was created.

Examples

```
library(SSN)
data(modelFits)
ls()
```

fitSp — *Fitted glmssn object for example data set MiddleFork.ssn*

Description

The MiddleFork04.ssn data folder contains the spatial, attribute, and topological information needed to construct a spatial stream network object using the SSN package. This is a fitted model using the `glmssn` function.

Details

See the help for `glmssn` for how the model was created.

Examples

```
library(SSN)
data(modelFits)
ls()
```

fitSpBk — *Fitted glmssn object for example data set MiddleFork.ssn*

Description

The MiddleFork04.ssn data folder contains the spatial, attribute, and topological information needed to construct a spatial stream network object using the SSN package. This is a fitted model using the `glmssn` function. It is used for the block prediction example.

Details

See the help for `glmssn` for how the model was created, and `BlockPredict` for usage in block prediction.

Examples

```
library(SSN)
data(modelFits)
ls()
```

fitSpRE1 *Fitted glmssn object for example data set MiddleFork.ssn*

Description

The MiddleFork04.ssn data folder contains the spatial, attribute, and topological information needed to construct a spatial stream network object using the SSN package. This is a fitted model using the glmssn function.

Details

See the help for glmssn for how the model was created.

Examples

```
library(SSN)
data(modelFits)
ls()
```

fitSpRE2 *Fitted glmssn object for example data set MiddleFork.ssn*

Description

The MiddleFork04.ssn data folder contains the spatial, attribute, and topological information needed to construct a spatial stream network object using the SSN package. This is a fitted model using the glmssn function.

Details

See the help for glmssn for how the model was created.

Examples

```
library(SSN)
data(modelFits)
ls()
```

getPreds *Extract Predictions with associated standard errors.*

Description

This function extracts predictions and standard errors from objects of class 'glmssn.predict' or 'influenceSSN'. Predictions are extracted for unobserved locations, while cross-validation predictions are extracted for observed locations.

Usage

```
getPreds(x, pred.type = "cv")
```

Arguments

x	an object of class predict.glmssn or influenceSSN-class
pred.type	prediction type, either "pred" or "cv". The "pred" option indicates that a 'glmssn.predict' object is being accessed and a text file containing predictions and standard errors for the predictions is exported. When the "cv" option is used, objects of class influenceSSN are accessed and cross-validation predictions and standard errors are exported.

Value

getPreds returns a matrix containing the point identifier (pid), the predictions, and the standard errors for the predictions.

Author(s)

Erin E. Peterson <support@SpatialStreamNetworks.com>

See Also

predict, influenceSSN-class

Examples

```
library(SSN)
# NOT RUN
#mf04p <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#  package = "SSN"), predpts = "pred1km", o.write = TRUE)
#  use SpatialStreamNetwork object mf04p that was already created
data(mf04p)
#make sure mf04p has the correct path, will vary for each users installation
mf04p@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")

# get some model fits stored as data objects
data(modelFits)
```

```
#NOT RUN use this one
#fitSp <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#    ssn.object = mf04p, EstMeth = "REML", family = "Gaussian",
#    CorModels = c("Exponential.tailup","Exponential.taildown",
#    "Exponential.Euclid"), addfunccol = "afvArea")
#Update the path in fitSp, will vary for each users installation
fitSp$ssn.object <- updatePath(fitSp$ssn.object,
system.file("lsndata/MiddleFork04.ssn", package = "SSN"))

# Extract predictions and standard errors for the prediction sites
fitSpPred <- predict(fitSp, predpointsID = "pred1km")
class(fitSpPred)
fitSpgetPreds <- getPreds(fitSpPred, pred.type = "pred")
head(fitSpgetPreds)

# Extract cross-validation predictions for the observed sites in two ways:
fitSpRes <- residuals(fitSp)
class(fitSpRes)

# Extract from the influenceSSN class object
fitSpResGetCV <- getPreds(fitSpRes, pred.type = "cv")
head(fitSpResGetCV)

# Extract from the glmssn.predict class object
fitSpResGetCV2 <- getPreds(fitSpPred, pred.type = "cv")
# These values are identical
identical(fitSpResGetCV,fitSpResGetCV) ## TRUE
```

`getSSNdata.frame` *Extract data from SSN objects as a data.frame*

Description

The `getSSNdata.frame` function extracts the points data data.frame, either observation data or prediction data, from the specified SSN object.

Usage

```
getSSNdata.frame(x, Name = "Obs")
```

Arguments

`x`	an object of class SpatialStreamNetwork-class, influenceSSN-class, glmssn-class, or "glmssn.predict".
`Name`	the internal name of the data set in the object `x`. For observed values, this will always be "Obs", the default.

Details

The internal Name for observed data in objects of class SpatialStreamNetwork is "Obs" and it is the default. If another Name is specified, it must represent a prediction data set in the SpatialStreamNetwork-class, influenceSSN-class, glmssn-class, or "glmssn.predict" object. For SpatialStreamNetwork objects, these names are obtained using the call ssn@predpoints@ID. For all other object classes, the names are obtained using the call object$ssn.object@predpoints@ID. See examples for additional details.

Value

A data.frame.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

putSSNdata.frame

Examples

```
library(SSN)
# NOT RUN
#mf04 <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#  package = "SSN"), o.write = TRUE)
#  use SpatialStreamNetwork object mf04p that was already created
data(mf04)
#Update path in mf04, will vary for each users installation
mf04 <- updatePath(mf04, system.file("lsndata/MiddleFork04.ssn", package = "SSN"))

obsDF <- getSSNdata.frame(mf04)
head(obsDF)

# get some model fits stored as data objects
data(modelFits)
#NOT RUN use this one
#fitSp <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#    ssn.object = mf04p, EstMeth = "REML", family = "Gaussian",
#    CorModels = c("Exponential.tailup","Exponential.taildown",
#    "Exponential.Euclid"), addfunccol = "afvArea")
#Update path for fitSP, will vary for each users installation
fitSp$ssn.object <- updatePath(fitSp$ssn,system.file("lsndata/MiddleFork04.ssn", package = "SSN"))

# Get the data.frame from an influenceSSN object and plot the residuals
fitSpRes <- residuals(fitSp)
fitSpResDF <- getSSNdata.frame(fitSpRes)
# NOT RUN
#plot(fitSpResDF[,"_resid.crossv_"],fitSpResDF[,"_resid_"], pch = 19,
#  ylab = "Cross-validation Residuals", xlab = "Raw Residuals")
```

```
# Get the data.frame for the prediction locations
fitSpPred <- predict(fitSp, predpointsID = "pred1km")
predNames<- fitSpPred$ssn.object@predpoints@ID
fitSpPredDF <- getSSNdata.frame(fitSpPred, predNames[1])
head(fitSpPredDF)
```

`getStreamDistMat` *Extract the stream network distance matrix from SSN objects*

Description

The `getStreamDistMat` function extracts the stream network distance matrix for either observation data or prediction data, from the specified `SpatialStreamNetwork` object.

Usage

```
getStreamDistMat(x, Name = "obs")
```

Arguments

`x`	an object of class SpatialStreamNetwork-class. Note that the createDistMat function needs to be run on an imported `SpatialStreamNetwork` object first in order to create the distance matrix.
`Name`	the internal name of the data set in the object `x`. For observed values, this will always be "Obs", the default. To get a stream network distance matrix for a prediction data set, the name of the data set must be given, in quotes.

Details

The internal `Name` for observed data in objects of class `SpatialStreamNetwork` is "Obs" and it is the default. If another `Name` is specified, it must represent a prediction data set in the `SpatialStreamNetwork-class`. For `SpatialStreamNetwork` objects, these names are obtained using the call ssn@predpoints@ID.

Note that these are not traditional distance matrices because they are asymmetric. The matrices contain the distance from one point to the common junction of both points, so they are asymmetric. For example, if two points are flow-connected, the distance from the point lower in the network to the one higher in the network is 0, while the distance from the higher point to the lower point is > 0. The convention is that the "from" point, to the common junction, is along the top of the matrix (with the column labels), and the "to" point, to the common junction, is along the left side of the matrix (with the row labels). From this matrix, it is possible to get total stream distance between any two points, an indicator matrix of flow-connectedness, etc. See examples for additional details.

Value

A list of matrices. Note that distances are only computed within networks. For "Obs" data, a matrix of distances is returned for each network, labeled "dist.net1", "dist.net2", etc., for the first and second network, etc. For prediction matrices, there are "from" and "to" matrices for both observed sites and predictions sites. The convention is that "from" are again the columns, and "to" are again the rows, but the label "a" is for from prediction sites to observation sites, and the label "b" is for from observation sites to predictions sites. Thus, the list of prediction matrices are labeled "dist.net1.a" for distance to common junction from prediction sites along the columns, to observation sites along the rows, for the first network. A prediction matrix labeled "dist.net1.b" contains distances to the common junction from observation sites along the columns to prediction sites along the rows, for the first network. If the argument `amongPreds = TRUE` was used for the function `createDistMat`, then the distance to common junction among prediction sites is returned, using the same labelling convention as for among observation sites. That is, the matrices for each network will be labeled "dist.net1", "dist.net2", etc., for the first and second network, etc.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

Examples

```
library(SSN)
# NOT RUN
#mf04 <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#  package = "SSN"), o.write = TRUE)
#  use SpatialStreamNetwork object mf04p that was already created
data(mf04p)
#Update path in mf04, will vary for each users installation
mf04p <- updatePath(mf04p, system.file("lsndata/MiddleFork04.ssn", package = "SSN"))

names(mf04p)

distObs <- getStreamDistMat(mf04p)
str(distObs)
distObs$dist.net1[1:5,1:5]

# get total in-stream distance between all pairs of points
strDistNet2 <- distObs$dist.net2 + t(distObs$dist.net2)
strDistNet2[5:10,5:10]

# maximum distance to common junction between two sites
a.mat <- pmax(distObs$dist.net2,t(distObs$dist.net2))
a.mat[5:10,5:10]

# minimum distance to common junction between two sites
# sites with 0 minimum distance are flow-connected
b.mat <- pmin(distObs$dist.net2,t(distObs$dist.net2))
b.mat[5:10,5:10]

# get distance matrices between observed sites and prediction sites
distPred1km <- getStreamDistMat(mf04p, Name = "pred1km")
```

```
str(distPred1km)
distPred1km$dist.net1.a[1:5,1:5]

# create distance matrix among prediction sites
# note these sites only occur on the second network
# this is useful for block prediction
createDistMat(mf04p, predpts = "CapeHorn", o.write = TRUE, amongpreds = TRUE)
distCape <- getStreamDistMat(mf04p, Name = "CapeHorn")
str(distCape)
distCape$dist.net2[1:5,1:5]
```

`glmssn` *Fitting Generalized Linear Models for Spatial Stream Networks*

Description

This function works on objects of class SpatialStreamNetwork to fit generalized linear models with spatially autocorrelated errors using normal likelihood methods (including REML) and quasi-likelihood for Poisson and Binomial families. The spatial formulation is described in Ver Hoef and Peterson (2010) and Peterson and Ver Hoef (2010).

Usage

```
glmssn(formula, ssn.object, family = "Gaussian", CorModels =
  c("Exponential.tailup","Exponential.taildown","Exponential.Euclid"),
  use.nugget = TRUE, use.anisotropy = FALSE, addfunccol = NULL, trialscol = NULL,
  EstMeth = "REML", useTailDownWeight = FALSE, trans.power = NULL,trans.shift = 0,
   control = list(max.range.factor = 4, trunc.pseudo = NULL,
   maxiter.pseudo = 20, beta.converge = 1e-05))
```

Arguments

`formula`	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`ssn.object`	an object of class SpatialStreamNetwork, representing a spatial stream network. This contains the variables used in the model.
`family`	the error distribution and link function to be used in the model. This is a character string that is either "Gaussian", "Poisson", or "Binomial."
`CorModels`	a vector of spatial autocorrelation models for stream networks. The individual models should be of different "types" tail-up, tail-down, Euclidean, or NULL for a non-spatial model. The tailup models include: "Exponential.tailup" (default), "LinearSill.tailup", "Spherical.tailup", "Mariah.tailup" "Epanech.tailup"; taildown models include: "Exponential.taildown" (default), "LinearSill.taildown", "Spherical.taildown", "Mariah.taildown", "Epanech.taildown"; Euclidean distance models include: "Spherical.Euclid", "Gaussian.Euclid", "Exponential.Euclid" (default), "Cauchy.Euclid". The first 4 tailup and taildown models are described

in Ver Hoef and Peterson (2010), and the "Epanech" models are described in Garreta, Monestiez, and Ver Hoef (2010), and the 4 Euclidean distance models are standard spatial covariance models. If this is NULL, then use.nugget = TRUE will impose independence between observations, or a classical regression analysis non-spatial model. Basic random effects can be included in the model here also. See examples below.

`use.nugget` add a nugget effect, default is TRUE. This can be thought of as a variance component for independent errors, adding a variance component only along the diagonal of the covariance matrix.

`use.anisotropy` use anistropy for the Euclidean distance based spatial model in CorModels

`addfunccol` the name of the variable in the `SpatialStreamNetwork` object that is used to define spatial weights. For the tailup models, weights need to be used for branching. This is an additive function and is described in Ver Hoef and Peterson (2010). See example below.

`trialscol` name of the variable in the `SpatialStreamNetwork` object that contains the sample size when a binomial distribution is used. If NULL, a sample size of 1 is assumed, and the response variable must be binary (0 or 1).

`EstMeth` Estimation method; either "ML" for maximum likelihood, or "REML" for restricted maximum likelihood (default).

`useTailDownWeight`
use stream segment weighting in the tail-down model? Default is FALSE. Weighting is same as for tail-up models, based on an additive function.

`trans.power` power transformation for the response variable in case of Gaussian data. It must be between 0 and 0.5, and if 0, a natural log is used.

`trans.shift` a shift (addition or subtraction) applied to the response variable prior to the power tranformation

`control` a list of control parameters, consisting of four items: 1) max.range.factor; this sets the maximum range as a function of the maximum distance among observed data locations, 2) trunc.pseudo; this sets a truncation value for pseudo-data for the quasi-models (family binomial and poisson). Because the data are modeled on a log or logit scale, exponentiation can cause numerical overflows, so this sets an upper bound, 3) maxiter.pseudo; this sets the maximum number of iterations when creating pseudo data for quasi-models. 4)beta.converge; this sets convergence criteria on fixed effect estimates. When all changes in the fixed effect estimates are less than beta.converge during an iteratively reweighted least squares update, then iteration stops. The default setting for control is control = list(max.range.factor = 4, trunc.pseudo = NULL, maxiter.pseudo = 20, beta.converge = 1e-5)

Details

Models for glmssn are specified symbolically, similar to `lm` and other models in R. A typical model has the form response ~ terms where response is the (numeric) response vector and terms is a series of fixed effect linear predictors for the response. A terms specification of the form first + second indicates all the terms in first together with all the terms in second with duplicates removed. A specification of the form first:second indicates the set of terms obtained by taking the interactions

of all terms in first with all terms in second. The specification first*second indicates the cross of first and second. This is the same as first + second + first:second. See model.matrix for further details. The terms in the formula will be re-ordered so that main effects come first, followed by the interactions, all second-order, all third-order and so on. A formula has an implied intercept term. To remove this use either y ~ x - 1 or y ~ 0 + x. See formula for more details of allowed formulae.

The spatial formulation is described in Ver Hoef and Peterson (2010) and Peterson and Ver Hoef (2010).

Value

args	Information on arguments used in the function call to glmssn
ssn.object	a copy of the input object of class SpatialStreamNetwork, so that the model fit is directly tied to an SpatialStreamNetwork object
sampinfo	sample information
estimates	Estimates of the covariance parameters
optimOutput	Output from last call to optim to enable the user to check for correct convergence

glmssn returns an object of class "glmssn". This is a list of 5 objects, with the following structure:

```
outpt <- list(
args = list( ## stores all arguments used in function call
formula = formula,
zcol = dataXY.out$respvecs$response.col, # response column
family = family,
CorModels = CorModels,
useTailDownWeights = useTailDownWeights,
use.nugget = use.nugget,
use.anisotropy = use.anisotropy,
addfunccol = addfunccol,
trialscol = trialscol,
EstMeth = EstMeth,
trans.power = trans.power,
trans.shift = trans.shift
),

ssn.object = ssn.object, # input object of class "SpatialStreamNetwork"

sampinfo = list( # sample information
# indicator vector for non-missing response values
ind.obs = ind[order(data[,"pid"])],
sample.size = n.all, # total number of records in the data frame
# number of records with non-missing response values
obs.sample.size = n.allxy,
missing.sample.size = n.all - n.allxy, # number of missing response values
rankX = p, # rank of X
# vector of the response variable
z = zt[order(dataXY.out$datasets$data2[,"pid"])],
X = X2[order(dataXY.out$datasets$data2[,"pid"]),], # design matrix
```

```
effnames = dataXY.out$Xmats$effnames,
setzero = dataXY.out$indvecs$setzero,
setNA = dataXY.out$indvecs$setNA,
setNA2 = dataXY.out$indvecs$setNA2,
cutX1toX2 = dataXY.out$indvecs$cutX1toX2,
StdXDataFrame = dataXY.out$Xmats$StdXDataFrame
),

estimates = list(
theta=parmest, # estimated covariance parameters
# estimated covariance matrix
V = V[order(dataXY.out$datasets$data2[,"pid"]),
order(dataXY.out$datasets$data2[,"pid"])],
# inverse of estimated covariance matrix
Vi = Vi[order(dataXY.out$datasets$data2[,"pid"]),
order(dataXY.out$datasets$data2[,"pid"])],
betahat = b.hat, # estimated fixed effects
covb = covb, # estimated covariance matrix of estimated fixed effects
# inverse of estimated covariance matrix of estimated fixed effects
covbi = covbi,
m2LL = m2LL # -2 times log-likelihood
),

optimOutput=parmest.out
)
```

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

References

Garreta, V., Monestiez, P. and Ver Hoef, J.M. (2010) Spatial modelling and prediction on river networks: up model, down model, or hybrid? Environmetrics ***21(5)***, *439–456.*

Peterson, E.E. and Ver Hoef, J.M. (2010) A mixed-model moving-average approach to geostatistical modeling in stream networks. Ecology ***91(3)***, *644–651.*

Ver Hoef, J.M. and Peterson, E.E. (2010) A moving average approach for spatial statistical models of stream networks (with discussion). Journal of the American Statistical Association ***105***, *6–18. DOI: 10.1198/jasa.2009.ap08248. Rejoinder pgs. 22–24.*

Examples

```
library(SSN)
# NOT RUN
# mf04p <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#   package = "SSN1.1"), predpts = "pred1km", o.write = TRUE)
#   use SpatialStreamNetwork object mf04p that was already created
```

```
data(mf04p)
#make sure mf04p has the correct path, will vary for each users installation
mf04p@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")

# The models take a little time to fit, so they are NOT RUN
# Uncomment the code to run them
# Alternatively, you can load the fitted models first to look at results
data(modelFits)

## Non-spatial model
# fitNS <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#    ssn.object = mf04p, CorModels = NULL,
#    EstMeth = "REML", family = "Gaussian")
#make sure fitSN has the correct path, will vary for each users installation
fitNS$ssn.object@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")
summary(fitNS)

## Random effect model using STREAMNAME as our random effect
#fitRE <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#    ssn.object = mf04p, EstMeth = "REML", family = "Gaussian",
#    CorModels = c("STREAMNAME"))
#make sure fitRE has the correct path, will vary for each users installation
fitRE$ssn.object@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")
summary(fitRE)
## random effects details
fitREBLUP <- BLUP(fitRE)
str(fitREBLUP)
fitREBLUP$Mean

## Basic spatial model with a random effect
#fitSpRE1 <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#    ssn.object = mf04p, EstMeth = "REML", family = "Gaussian",
#    CorModels = c("STREAMNAME","Exponential.Euclid"))
#make sure fitSpRE1 has the correct path, will vary for each users installation
fitSpRE1$ssn.object@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")
summary(fitSpRE1)

## Spatial stream tail-up model with a random effect
#fitSpRE2 <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#    ssn.object = mf04p, EstMeth = "REML", family = "Gaussian",
#    CorModels = c("STREAMNAME","Exponential.tailup"),
#    addfunccol = "afvArea")
#make sure fitSpRE2 has the correct path, will vary for each users installation
fitSpRE2$ssn.object@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")
summary(fitSpRE2)

## 3 component spatial model
#fitSp <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#    ssn.object = mf04p, EstMeth = "REML", family = "Gaussian",
#    CorModels = c("Exponential.tailup","Exponential.taildown",
#    "Exponential.Euclid"), addfunccol = "afvArea")
#make sure fitSp has the correct path, will vary for each users installation
fitSp$ssn.object@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")
```

```
## Summarise last model
summary(fitSp)

## AIC for last model
AIC(fitSp)

## Generalised R-squared for last model
GR2(fitSp)

## Look at variance components in more detail
covparms(fitSp)
varcomp(fitSp)

## Compare models
InfoCritCompare(list(fitNS, fitRE, fitSpRE1, fitSpRE2, fitSp))

## Fit a model to binary data
#binSp <- glmssn(MaxOver20 ~ ELEV_DEM + SLOPE, mf04p,
#  CorModels = c("Mariah.tailup", "Spherical.taildown"),
#  family = "binomial", addfunccol = "afvArea")
#make sure binSp has the correct path, will vary for each users installation
binSp$ssn.object@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")
summary(binSp)

## Fit a model to count data
#poiSp <- glmssn(C16 ~ ELEV_DEM + SLOPE, mf04p,
#  CorModels = c("LinearSill.tailup", "LinearSill.taildown"),
#  family = "poisson", addfunccol = "afvArea")
#make sure poiSp has the correct path, will vary for each users installation
poiSp$ssn.object@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")
summary(poiSp)
```

`glmssn-class` *Class "glmssn"*

Description

a class that holds a fitted model for an object of class `SpatialStreamNetwork`

Objects from the Class

Objects can be created by calls in the form `glmssn(...)`.

List Objects

An object of class "glmssn" is a list of 4 objects, with the following structure:

```
  outpt <- list(
args = list(  ## stores all arguments used in function call
formula,
zcol, ## response column name
family,
CorModels,
use.nugget,
use.anisotropy,
addfunccol,
trialscol,
EstMeth,
trans.power,
trans.shift,
algorithm,
),
ssn.object = ssn.object, ## input object of class "SpatialStreamNetwork"
sampinfo = list( ## sample information
ind.obs, ## indicator vector for non-missing response values
ind.RespNA, ## indicator vector for non-missing response values
sample.size, ## Total number of records in the data frame
obs.sample.size, ## Number of records with non-missing response values
missing.sample.size, ## Number of missing response values
rankX, ## rank of X
z, ## vector of response values
trialsvec,
X, ## design matrix
effnames,
setzero,
setNA,
setNA2,
cutX1toX2,
REs,
REmodelmatrices,
),
estimates = list(
theta, ## Estimated covariance parameters
  nugget,
V ## Estimated covariance matrix
Vi ## Inverse of V
betahat ## Estimate of fixed effects
covb ## Estimated covariance matrix of fixed effects
covbi ## Inverse of covb
m2LL ## -2 times log likelihood
Warnlog ## List of warnings
),
loglik.surface=get("RESULT",loglik.environment),
        optimOutput ## output from optim
)
```

Extends

Class SpatialStreamNetwork, directly.

Class SpatialLines, by class "SpatialLinesDataFrame", distance 2.

Class Spatial, by class "SpatialLinesDataFrame", distance 3.

Methods

No methods defined with class "glmssn" in the signature.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

glmssn

GR2 *Generalised R2*

Description

GR2 is a generic function that calculates and displays the generalised R2 value for fitted glmssn objects.

Usage

```
GR2(object)
```

Arguments

object	an object of class glmssn

Details

The generalised R2 value, or 'coefficient of determination', lies somewhere between 0 and 1, and is a general measure of the predictive power of a model. In this instance, it relates to the proportion of the variability in the data that is explained by the fixed effects.

Value

a numeric value equal to the GR2 for the specified glmssn object

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

glmssn

Examples

```
library(SSN)
# NOT RUN
#mf04p <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#  package = "SSN"), predpts = "pred1km", o.write = TRUE)
#  use SpatialStreamNetwork object mf04p that was already created
data(mf04p)
#make sure mf04p has the correct path, will vary for each users installation
mf04p@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")

# get some model fits stored as data objects
data(modelFits)
#NOT RUN use this one
#fitSp <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#    ssn.object = mf04p, EstMeth = "REML", family = "Gaussian",
#    CorModels = c("Exponential.tailup","Exponential.taildown",
#    "Exponential.Euclid"), addfunccol = "afvArea")
#make sure fitSP has the correct path, will vary for each users installation
fitSp$ssn@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")

GR2(fitSp)
```

`importPredpts` *Import Prediction Points into a SpatialStreamNetwork or glmssn Object*

Description

Prediction points residing in a .ssn directory are imported into an existing object of class SpatialStreamNetwork-class or glmssn-class

Usage

```
importPredpts(target, predpts, obj.type)
```

Arguments

`target`	a `SpatialStreamNetwork-class` or `glmssn-class` object
`predpts`	Prediction points shapefile name, enclosed in quotes. When writing, omit the .shp extension. Prediction points must reside in the .ssn directory and be generated from the same landscape network as the other spatial data in the `SpatialStreamNetwork` object
`obj.type`	the class of the target. For a `SpatialStreamNetwork` object, `target = "ssn"`. For a `glmssn-class` object, `target = "glm"`

Details

importPredpts imports a shapefile of prediction points residing in the .ssn directory into an existing SpatialStreamnetwork or glmssn-class object. The spatial datasets residing the .ssn folder are generated in a geographic information system using the Spatial Tools for the Analysis of River Systems (STARS) tools for ArcGIS version 9.3.1. A detailed description of the spatial data format is provided in Peterson (2011).

Value

importPredpts returns an object of class "SpatialStreamNetwork" or "glmssn". An additional predpoints slot is populated in the object

Author(s)

Erin E. Peterson <support@SpatialStreamNetworks.com>

References

Peterson E.E.(2011)STARS: Spatial Tools for the Analysis of River Systems: A tutorial. CSIRO Technical Report EP111313. 42p.

See Also

importSSN, SpatialStreamNetwork-class, and glmssn-class

Examples

```
library(SSN)
#mf04 <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#  package = "SSN"), o.write = TRUE)
#  use SpatialStreamNetwork object mf04 that was already created
data(mf04)
#make sure mf04 has the correct path, will vary for each users installation
mf04@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")
names(mf04)

mf04p <- mf04
# add existing prediction points on 1 km spacing
mf04p <- importPredpts(target = mf04p, predpts = "pred1km", obj.type = "ssn")
# get names and verify that pred1km has been added
names(mf04p)

# add dense set of prediction points from Knapp stream
mf04p <- importPredpts(target = mf04p, predpts = "Knapp", obj.type = "ssn")
# get names and verify that Knapp has been added
names(mf04p)

# add dense set of prediction points from CapeHorn stream
mf04p <- importPredpts(target = mf04p, predpts = "CapeHorn", obj.type = "ssn")
# get names and verify that CapeHorn has been added
```

```
names(mf04p)

# create distance matrices, needed for prediction with stream network models
# NOT RUN
#createDistMat(mf04p, "pred1km", o.write = TRUE)
# for block prediction, we need distance among prediction points
#createDistMat(mf04p, "Knapp", o.write = TRUE, amongpreds = TRUE)
#createDistMat(mf04p, "CapeHorn", o.write = TRUE)

# Add prediction points to a glmssn object
# use models that have been created already
data(modelFits)
#make sure mf04 has the correct path, will vary for each users installation
fitSp$ssn@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")

#use model named fitSp; NOT RUN; already imported
#fitSp <- importPredpts(target = fitSp, predpts = "pred1km",
#  obj.type = "glm")
# now we can make predictions; make sure distance matrix for "pred1km" has
# been created
# NOT RUN
#fitSpPred <- predict(fitSp,"pred1km")
#plot(fitSpPred)
#fitSp <- importPredpts(target = fitSp, predpts = "Knapp",
#  obj.type = "glm")
# NOT RUN
#fitSpPredKnapp <- predict(fitSp,"Knapp")
#plot(fitSpPredKnapp)
```

`importSSN` *Import an object of class SpatialStreamNetwork*

Description

The `importSSN` function reads spatial data from a .ssn folder and to create a SpatialStreamNetwork object

Usage

```
importSSN(filepath, predpts = NULL, o.write = FALSE)
```

Arguments

`filepath`	path name to the .ssn folder, in string format including quotes. Also include the .ssn folder in the path name
`predpts`	default = NULL. Prediction points shapefile name enclosed in quotes. When writing, omit the .shp extension. Prediction points must reside in the .ssn directory and be generated from the same landscape network as the other spatial data in the .ssn directory

o.write default = FALSE. If TRUE, overwrite existing binaryID.db files

Details

The importSSN function imports spatial data from a .ssn folder to create a SpatialStreamNetwork-class object. The information contained in the .ssn folder is generated in a geographic information system using the Spatial Tools for the Analysis of River Systems (STARS) tools for ArcGIS version 9.3.1. A detailed description of the spatial data format is provided in Peterson (2011).

The information contained in the netID text files is imported into an SQLite database, binaryID.db, which is stored in the .ssn directory. This information is used internally by `createDistMat` and `glmssn` to calculate the data necessary to fit a spatial statistical model to stream network data. If o.write = TRUE (o.write = FALSE is the default) and a binaryID.db file already exists within the .ssn directory, it will be overwriten when the `SpatialStreamNetwork` object is created.

A `SpatialStreamNetwork` object may contain multiple sets of prediction points, which are contained in separate shapefiles. One prediction point shapefile may be imported using importSSN. The `importPredpts` function allows users to import additional sets of prediction sites to a an existing `SpatialStreamNetwork` object. The prediction points ID is set to the base name the shapefile.

Value

`importSSN` returns an object of class `SpatialStreamNetwork`. It also creates and stores a SQLite database, binaryID.db, within the .ssn directory.

Author(s)

Erin E. Peterson <support@SpatialStreamNetworks.com>

References

Peterson E.E.(2011)STARS: Spatial Tools for the Analysis of River Systems: A tutorial. CSIRO Technical Report EP111313. 42p.

See Also

`importPredpts` for adding prediction points after a SpatialStreamNetwork object has been created. `createDistMat` to create distance matrices among points, both among observed, between observed and predicted, and among predicted. `createSSN` for creating SSN objects from scratch for simulation purposes.

Examples

```
library(SSN)
# Create a SpatialStreamNetwork object that does not contain prediction points
mf04 <- importSSN(system.file("lsndata/MiddleFork04.ssn",
package = "SSN"), o.write = TRUE)

# Create a SpatialStreamNetork object that also contains prediction sites
mf04p <- importSSN(system.file("lsndata/MiddleFork04.ssn",
package = "SSN"), predpts = "pred1km", o.write = TRUE)
```

```
names(mf04p)

#NOT RUN Some graphics to explore imported object
#hist(mf04, "Summer_mn")
#boxplot(mf04, "Summer_mn")
#plot(mf04, cex = .8, xlab = "x", ylab = "y")
#plot(mf04p, PredPointsID = "pred1km", add = TRUE, pch = 1) #add to previous plot
```

`influenceSSN-class` *Class "influenceSSN"*

Description

A class that extends the results of generalized linear models, `glmssn` objects, for spatial stream networks by adding influence diagnostics and cross-validation predictions to each observation.

Objects from the Class

Objects can be created by functions in the form `residual(x)`, where `x` is a glmssn-class object.

Class Structure

Objects of class `influenceSSN` contain 4 list items and have the exact same structure as glmssn-class objects. A `influenceSSN` object retains the corresponding `SpatialStreamNetwork` object as the second list item. When `residuals(x)` is used for a glmssn object, the data for which the model was fit is stored in point.data data.frame of the observed points. This data.frame contains the response variable for the model, and is appended by the following columns:

```
obsval            ## The response value that was used to fit the model
_fit_
_resid_           ## The raw residuals
_resid.stand_     ## Standardized residuals; calculated by dividing the raw
                  ## residuals by the corresponding standard errors
_resid.student_   ## Studentized residuals
_leverage_        ## Leverage
_CooksD_          ## Cook's D
_resid.crossv_    ## Cross-validation residuals
_CrossValPred_    ## Cross-validation predictions
_CrossValStdErr_  ## Estimated cross-validation standard errors.
```

Extends

Class "`glmssn`", directly.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

residuals,glmssn

InfoCritCompare *Compare* glmssn *Information Criteria*

Description

InfoCritCompare displays important model criteria for each object of class glmssn object in the model list.

Usage

```
InfoCritCompare(model.list)
```

Arguments

model.list	a list of fitted glmssn-class model objects in the form list(model1, model2, ...)

Details

InfoCritCompare displays important model criteria that can be used to compare and select spatial statistical models. For instance, spatial models can be compared with non-spatial models, other spatial models, or both.

Value

InfoCritCompare returns a data.frame of the model criteria for each specified glmssn-class object. These are useful for comparing and selecting models. The columns in the data.frame are described below. In the description below 'obs' is an observed data value, 'pred' is its prediction using cross-validation, and 'predSE' is the prediction standard error using cross-validation.

formula model formula

EstMethod estimation method, either maximum likelihood (ML) or restricted maximum likelihood (REML)

Variance_Components names of the variance components, including the autocovariance model names, the nugget effect, and the random effects.

neg2Log -2 log-likelihood. Note that the neg2LogL is only returned if the Gaussian distribution (default) was specified when creating the glmssn object.

AIC Akaike Information Criteria (AIC). Note that AIC is only returned if the Gaussian distribution (default) was specified when creating the glmssn object.

bias bias, computed as mean(obs - pred).

std.bias standardized bias, computed as mean((obs - pred)/predSE).

RMSPE root mean-squared prediction error, computed as sqrt(mean((obs - pred)^2))

RAV root average variance, computed as sqrt(mean(predSE^2)). If the prediction standard errors are being estimated well, this should be close to RMSPE.

std.MSPE standardized mean-squared prediction error, computed as mean(((obs - pred)/predSE)^2). If the prediction standard errors are being estimated well, this should be close to 1.

cov.80 the proportion of times that the observed value was within the prediction interval formed from pred +- qt(.9, df)*predSE, where qt is the quantile t function, and df is the number of degrees of freedom. If there is little bias and the prediction standard errors are being estimated well, this should be close to 0.8 for large sample sizes.

cov.90 the proportion of times that observed value was within the prediction interval formed from pred +- qt(.95, df)*predSE, where qt is the quantile t function, and df is the number of degrees of freedom. If there is little bias and the prediction standard errors are being estimated well, this should be close to 0.9 for large sample sizes.

cov.95 the proportion of times that the observed value was within the prediction interval formed from pred +- qt(.975, df)*predSE, where qt is the quantile t function, and df is the number of degrees of freedom. If there is little bias and the prediction standard errors are being estimated well, this should be close to 0.95 for large sample sizes.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

glmssn, summary.glmssn, AIC, CrossValidationStatsSSN

Examples

```
library(SSN)
data(modelFits)

  compare.models <- InfoCritCompare(list(fitNS, fitRE, fitSp, fitSpRE1, fitSpRE2))

  # Examine the model criteria
  compare.models

  # Compare the AIC values for all models with random effects
  compare.models[c(2,4,5),c("Variance_Components","AIC")]

  # Compare the RMSPE for the spatial models
  compare.models[c(3,4,5),c("Variance_Components","RMSPE")]

  # Compare the RMSPE between spatial and non-spatial models
  compare.models[c(1,3),c("formula","Variance_Components", "RMSPE")]
```

mf04 *imported SpatialStreamNetwork object from MiddleFork04.ssn data folder*

Description

The MiddleFork04.ssn data folder contains the spatial, attribute, and topological information needed to construct a spatial stream network object using the SSN package. This is the representation by using the importSSN function.

Details

See the MiddleFork04.ssn for details of data set, importSSN on how to get it into SSN.

Source

See the MiddleFork04.ssn

Examples

```
library(SSN)
mf04 <- importSSN(system.file("lsndata/MiddleFork04.ssn", package = "SSN"))
names(mf04)
```

mf04p *Imported SpatialStreamNetwork object from MiddleFork04.ssn data folder*

Description

The MiddleFork04.ssn data folder contains the spatial, attribute, and topological information needed to construct a spatial stream network object using the SSN package. This is the representation by using the importSSN function.

Details

See the MiddleFork04.ssn for details of data set, importSSN on how to get it into SSN.

Source

See the MiddleFork04.ssn

Examples

```
library(SSN)
mf04p <- importSSN(system.file("lsndata/MiddleFork04.ssn", package = "SSN"),
  predpts = "pred1km")
mf04p <- importPredpts(target = mf04p, predpts = "CapeHorn", obj.type = "ssn")
mf04p <- importPredpts(target = mf04p, predpts = "Knapp", obj.type = "ssn")
names(mf04p)
```

MiddleFork04.ssn *MiddleFork04.ssn data folder*

Description

The MiddleFork04.ssn data folder contains the spatial, attribute, and topological information needed to construct a spatial stream network object using the SSN package.

Details

```
The MiddleFork04.ssn folder contains five spatial datasets:
  edges:    polyline shapefile representing the stream network
  sites:    point shapefile representing the observed site locations
  CapeHorn: point shapefile representing prediction site locations on the Cape
            Horn River
  Knapp:    point shapefile representing prediction site locations on the Knapp
            River
  pred1km:  point shapefile representing the prediction site locations

There is also 1 text file, netID1.txt, which contain the topological information
for the stream network in the Middle Fork dataset.

Attribute data is also stored within each of the spatial datasets. The column
names are defined as follows:

edges:
  COMID: Common identifier of an NHD feature or relationship
  GNIS_ID: Geographic Names Information System identifier
  GNIS_Name:  Feature name as found in the Geographic Names Information System
  LENGTHKM: Length (km)
  REACHCODE: Unique identifier for a reach. The first 8 digits contain
             the identfier for the HUC8 and the last 6 digits are a
             unique within-HUC8 identifier for the reach
  FCODE: Numeric code that contains the feature type and it's attributes
   as found in the NHDFCode lookup table
  CUMdrainAG: Cumulative drainage area (km2) for the lowermost location on the
   edge
  AREAWTMAP:  Area weighted mean annual precipitation (mm) at the lowermost
   location on the edge
```

```
  SLOPE: Slope of the edge (cm/cm)
 h2oAreaKm2: Watershed area (km2) for the lowermost location on the line segment
  rid:        Reach identifier
  areaPI:     Segment proportional influence value, calculated using watershed
              area (h2oAreaKm2)
  afvArea: Additive function value, calculated using areaPI
  upDist:     Distance from the stream outlet (most downstream location in the
             the stream network) to the uppermost location on the line segment
  netID: Network identifier
  Length:     Length of line segment (m)


sites:
  STREAMNAME: Stream name
  COMID: Common identifier of an NHD feature or relationship
  CUMDRAINAG: Cumulative drainage area (km2)
 AREAWTMAP:  Area weighted mean annual precipitation (mm) at lowermost location
 on the line segment where the site resides
  SLOPE: Slope of the line segment (cm/cm) where the site resides
  ELEV_DEM: Elevation at the site based on a 30m DEM
  Deployment: Unique identifier of the site by year of deployment
 NumberOfDa: Number of days sampled between the dates of July 15 and August 31.
 Maximum value is 48.
  Source: Source of the data - relates to the ID field of the source table
  Summer_mn: Overall summer mean termperature of the deployment
  MaxOver20: Binary variable: 1 represents the maximum summer temperature was
   greater than 20C and 0 indicates that it was less than 20C
  C16: the number of times daily stream temperature exceeded 16C
  C20: the number of times daily stream temperature exceeded 20C
  C24: the number of times daily stream temperature exceeded 24C
  FlowCMS: Average stream flow (cubic meters per sec) for August, by year,
   from 1950-2010 across 9 USGS gauges in the region
  AirMEANc:  Average mean air temperature (C) from July 15 - August 31, from
   1980-2009 across 10 COOP air stations within the domain
 AirMWMTc: Average maximum air temperature (C) from July 15 - August 31, from
 1980-2009 across 10 COOP air stations within the domain. MWMT =
 maximum 7-day moving average of the maximum daily temperature
 (i.e. maximum of all the 7-day maximums)
  NEAR_FID: the FID of the nearest edge feature
 NEAR_DIST: the distance between the original site coordinates and the nearest
     edge
  NEAR_X: x coordinate
  NEAR_Y: y coordinate
  NEAR_ANGLE: the angle between the original site location and the nearest
 point on the closest edge
  rid: Reach identifier of the edge the site resides on
  ratio: Site ratio value; provides the proportional distance along the
   edge to the site location
```

```
 upDist: Distance upstream from the stream outlet (m)
afvArea: Additive function value calculated using waterhsed area (h2oAreaKm2)
 locID:      Location identifier
 netID:      Stream network identifier
 pid:        Point identifier


pred1km, CapeHorn, & Knapp:
  COMID: Common identifier of an NHD feature or relationship
  CUMDRAINAG: Cumulative drainage area (km2)
 AREAWTMAP:  Area weighted mean annual precipitation (mm) at lowermost location
 on the line segment where the site resides
  SLOPE: Slope of the line segment (cm/cm) where the site resides
  ELEV_DEM: Elevation at the site based on a 30m DEM
  NEAR_FID: the FID of the nearest edge feature
 NEAR_DIST: the distance between the original site coordinates and the nearest
    edge
  NEAR_X: x coordinate
  NEAR_Y: y coordinate
  NEAR_ANGLE: the angle between the original site location and the nearest
 point on the closest edge
  rid: Reach identifier of the edge the site resides on
  ratio: Site ratio value; provides the proportional distance along the
 edge to the site location
  upDist: Distance upstream from the stream outlet (m)
 afvArea: Additive function value calculated using watershed area (h2oAreaKm2)
  locID:      Location identifier
  netID:      Stream network identifier
  pid:        Point identifier
  FlowCMS: Average stream flow (cubic meters per sec) for August, by year,
 from 1950-2010 across 9 USGS gauges in the region
  AirMEANc:  Average mean air temperature (C) from July 15 - August 31, from
 1980-2009 across 10 COOP air stations within the domain
  AirMWMTc: Average maximum air temperature (C) from July 15 - August 31, from
 1980-2009 across 10 COOP air stations within the domain. MWMT =
 maximum 7-day moving average of the maximum daily temperature
 (i.e. maximum of all the 7-day maximums)
```

Source

edges: modified version of the National Hydrography Dataset (http://nhd.usgs.gov/) sites, pred1km, CapeHorn, & Knapp: U.S. Forest Service, unpublished data.

Examples

```
library(SSN)
mf04p <- importSSN(system.file("lsndata/MiddleFork04.ssn", package = "SSN"),
```

```
  predpts = "pred1km")
mf04p <- importPredpts(target = mf04p, predpts = "CapeHorn", obj.type = "ssn")
mf04p <- importPredpts(target = mf04p, predpts = "Knapp", obj.type = "ssn")
names(mf04p)
```

`plot.glmssn.predict` *Plot glmssn.predict Objects*

Description

`plot.glmssn.predict` is a generic plot function that has been adapted for objects of class `glmssn.predict`.

Usage

```
## S3 method for class 'glmssn.predict'
plot(x, VariableName = NULL, VarPlot = "Both",
color.palette = rainbow(nclasses, start = 0.66, end = 0.99), nclasses =
10, breaktype = "quantile", dec.dig = 2, SEcex.min = 0.5, SEcex.max = 2,
brks = NULL, add = FALSE, ...)
```

Arguments

`x`	an object of class `glmssn.predict`
`VariableName`	name of variable to be plotted
`VarPlot`	a character argument that must be one of "Both", "Predictions", or "Standard Errors". Default is "Both", which colors predictions by their values and makes their size inversely proportional to the prediction standard errors.
`breaktype`	the method for breaking the predictions (or standard errors) into classes for coloring while plotting. A character argument that must be one of "quantile" (default), "even", or "user".
`brks`	if breaktype = "user", the break values must be specified here as a vector or matrix using c(...) or cbind(...). The sorted unique values are used as break points (together with the min and max of the variable being plotted if required)
`nclasses`	the number of classes for coloring the predictions (or standard errors) according to their value. The default is 10. If brks = c(...) is specified, then nclasses is automatically set to the number of breaks + 1.
`color.palette`	a color palette for plotting points. The default is rainbow(nclasses, start = .66, end = .99). The number of colors should equal to the number of classes. See `palette` for many ways to create palettes.
`SEcex.min`	if VarPlot = "both", the minimum cex value when making point sizes is inversely proportional to the prediction standard errors. See `par` for more on cex. Also see details below. Default is 1.
`SEcex.max`	if VarPlot = "both", the maximum cex value when making point sizes inversely proportional to the prediction standard errors. See `par` for more on cex. Also see details below. Default is 3.

dec.dig	the number of decimal places to print in the legend. Default is 2.
add	Logical value indicating whether the predictions should be added to an existing plot, such as a plot of colored values for observed data. Default is FALSE.
...	Arguments to be passed to methods, such as graphical parameters (see par).

Details

The plot.glmssn.predict function creates a map showing color-coded predictions or prediction standard error values. When VarPlot = "Both", predictions values are colored according to breaks. The size of the points is inversely proportional to the prediction standard errors. If SE is the standard error for a prediction, then the size of the plotted point will be SEcex.max - (SEcex.max - SEcex.min)*(SE - min(SE))/(max(SE) - min(SE)), where mins and maxs are over all SEs in the prediction set. This is simply a linear interpolator between SEcex.max and SEcex.min, specified by the user, with larger points for smaller standard errors. So large points reflect the fact that you have more confidence in those values and small points reflect the fact that you have less confidence in the values. Two plot legends are included in this case - one based on size and one on colour.

If the predictions are added to an existing plot, the printing of a second legend is suppressed, but the minimum predicted value is added as text to the top of the legend area, and the maximum predicted value is added as text to the bottom of the legend area. This option only makes sense if the breaks are matched to those when plotting the observed values. See the example below.

Value

Maps of stream networks with prediction and prediction standard error values.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

predict

Examples

```
library(SSN)
data(modelFits)
#make sure fitSp has the correct path, will vary for each users installation
fitSp$ssn.object@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")

## create predictions
ssnpred <- predict(fitSp, "pred1km")

##default graph
plot(ssnpred)

## max maximum size smaller
plot(ssnpred, SEcex.max = 1.5)
```

```
## predictions only
plot(ssnpred, VarPlot = "Predictions", breaktype = "quantile")

## change line width
plot(ssnpred, VarPlot = "Predictions", breaktype = "quantile", lwd = 2)

## change line type
plot(ssnpred, VarPlot = "Predictions", breaktype = "quantile", lwd = 2, lty = 2)

## standard errors only
plot(ssnpred, VarPlot = "Standard Errors", breaktype = "quantile")

## use even spacing for breaks
plot(ssnpred, VarPlot = "Standard Errors", breaktype = "even")

## use custom breaks and colors - specify four break points and four
## colours, but the actual range of standard errors requires a fifth break
## point and a fifth colour (see legend on RHS) chosen by default
plot(ssnpred, VarPlot = "Standard Errors", breaktype = "user",
 brks = seq(0,2.4,by=0.6), color.palette = c("darkblue", "purple",
   "green", "red"))

## add predictions to colored observations
brks <- as.matrix(plot(fitSp$ssn.object, "Summer_mn", cex = 2))
plot(ssnpred, add = TRUE, breaktype = "user", brks = brks, nclasses=length(brks),
    SEcex.max = 1)
```

`plot.influenceSSN` *Plotting Method for influenceSSN Objects*

Description

plot.influenceSSN is a generic plot function that has been adapted for influenceSSN-class objects that have been created from the residuals function.

Usage

```
## S3 method for class 'influenceSSN'
plot(x, color.palette = NULL, nclasses = NULL, inflcol = "_resid_",
breaktype = "quantile", brks=NULL, pch = 19, ...)
```

Arguments

x	an object of class influenceSSN.
color.palette	a color palette for plotting points. The default is rainbow(nclasses, start = .66, end = .99). The number of colors should equal the number of classes. See palette for many ways to create palettes.

nclasses	the number of classes for coloring the predictions (or standard errors) according to their value. The default is 10.
inflcol	an influence diagnostic or cross-validation variable name in the influenceSSN object. If NULL (default), just locations are plotted. If a variable is specified, it will be colored according to its value.
breaktype	The method for breaking the response values into classes for coloring while plotting. A character argument that must be one of "quantile" (default), "even", or "user".
brks	if breaktype = "user", the break values must be specified here as a vector or matrix using c(...) or cbind(...). The sorted unique values are used as break points (together with the min and max of the variable being plotted if required)
pch	either an integer specifying a symbol or a single character to be used as the default in plotting points. See link{points} for possible values and their interpretation. Note that only integers and single-character strings can be set as a graphics parameter (and not NA nor NULL).
...	arguments to be passed to methods, such as graphical parameters (see par).

Details

The plot.influenceSSN function creates a map showing data locations that can be color-coded according to the values of the diagnostic or influence variables.

Value

Maps of stream networks, with the spatial distribution of the influence or cross-validation variables shown.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

influenceSSN-class, residuals, plot.SpatialStreamNetwork

Examples

```
library(SSN)
data(modelFits)
#make sure fitSp has the correct path, will vary for each users installation
fitSp$ssn.object@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")

resids <- residuals(fitSp)
plot(resids)

## plot using user-defined breakpoints
brks <- seq(-3,2,by=1)
plot(resids, nclasses = 6, inflcol = "_resid_",
```

```
    breaktype = "user", brks = brks, pch = 3)

## plot crossvalidation residuals
plot(resids, nclasses = 6, inflcol = "_resid.crossv_")
```

---

plot.SpatialStreamNetwork

Plotting Method for SpatialStreamNetwork Objects

Description

plot.SpatialStreamNetwork is a generic plot function that has been adapted for SpatialStreamNetwork objects that have been created in SSN.

Usage

```
## S3 method for class 'SpatialStreamNetwork'
plot(x, VariableName=NULL, color.palette= NULL,
  nclasses = NULL, breaktype = "quantile", brks = NULL, PredPointsID = NULL,
  add = FALSE, addWithLegend=FALSE, lwdLineCol = NULL, lwdLineEx = 1,
  lineCol = "black", ...)
```

Arguments

x	an object of class SpatialStreamNetwork.
VariableName	a response variable name in the data.frame of observed data in the SpatialStreamNetwork object. If NULL (default), just locations are plotted. If a variable is specified, it will be colored according to its value.
color.palette	a color palette for plotting points. The default is rainbow(nclasses, start = .66, end = .99). The number of colors should equal the number of classes. See palette for many ways to create palettes.
nclasses	the number of classes for coloring the predictions (or standard errors) according to their value. The default is 10. If brks = c(...) is specified, then nclasses is automatically set to the number of unique breaks + 1.
breaktype	the method for breaking the response values into classes for coloring while plotting. A character argument that must be one of "quantile" (default), "even", or "user".
brks	if breaktype = "user", the break values must be specified here as a vector or matrix using c(...) or cbind(...). The sorted unique values are used as break points (together with the min and max of the variable being plotted if required).
PredPointsID	a string representing the internal name of the prediction sites data set, which will be added to the plot. Default is NULL.
add	logical indicating whether the predictions should be added to an existing plot, such as a plot of the stream network and observed locations. Use this if there is no legend. Default is FALSE.

addWithLegend	logical indicating whether the predictions should be added to an existing plot, such as a plot of colored values for observed data. Use this when there is a legend. Default is FALSE.
lwdLineCol	a column name in the lines data frame to be used for line width expansion. This will most likely be the name of the additive function column, but others could be used.
lwdLineEx	an expansion multiplier to create line widths for the values contained in lwdLineCol.
lineCol	a color for the lines forming the stream network. Default is "black".
...	arguments to be passed to methods, such as graphical parameters (see par).

Details

The plot.SpatialStreamNetwork function creates a map showing data locations that can be color-coded according to the values of observed variables. Prediction locations can also be added to existing plots of observed values.

Value

Maps of stream networks

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

SpatialStreamNetwork-class, plot

Examples

```
library(SSN)
# Create a SpatialStreamNetork object that also contains prediction sites
#undebug(importSSN)
mf04p <- importSSN(system.file("lsndata/MiddleFork04.ssn",
package = "SSN"), predpts = "pred1km", o.write = TRUE)
names(mf04p)
summary(mf04p)

#generic plotting includes stream network and observed locations
plot(mf04p)

#plot including the color coding the response variable
plot(mf04p, "Summer_mn")

#plot using user-defined breakpoints
plot(mf04p, "Summer_mn", breaktype = "user", brks = seq(8,16,by=1))

#pass normal plotting arguments, such as xlab and ylab, to plot
```

```
plot(mf04p, "Summer_mn", xlab = "x-coordinate", ylab = "y-coordinate")

# plot observations and prediction values
plot(mf04p, "Summer_mn", cex = 2, xlab = "x", ylab = "y")
plot(mf04p, PredPointsID = "pred1km", addWithLegend = TRUE)
```

`plot.Torgegram` *Plotting Method for Torgegram Objects*

Description

`plot.Torgegram` is a generic plot function that has been adapted for Torgegram objects, which are created using the Torgegram function. A Torgegram object stores information used to construct an empirical semivariogram based on hydrologic distance. The `plot.Torgegram` function allwos the results to be presented separately for flow-connected and flow-unconnected sites.

Usage

```
## S3 method for class 'Torgegram'
plot(x, sp.relationship = c("fc", "fu"),
  min.cex = 1.5, max.cex = 6, leg.auto = TRUE, main = "", ylab = "",
  xlab = "Stream Distance", ... )
```

Arguments

`x`	an object of class Torgegram
`sp.relationship`	a string or character vector representing the in-stream spatial relationship to be plotted. "fc" specifies plotting of only flow-connected, and "fu" specifies plotting of only flow-unconnected. Default is both.
`min.cex`	Minimum character expansion size of the plotting symbols. Symbols are scaled according to how many pairs of points went into computing each bin of the semivariogram. The bin with the smallest sample size will be be plotted with this symbol size. The default is 1.5.
`max.cex`	Maximum character expansion size of the plotting symbols. Symbols are scaled according to how many pairs of points went into computing each bin of the semivariogram. The bin with the largest sample size will be be plotted with this symbol size. The default is 6.
`leg.auto`	Logical, default to TRUE. Include a legend.
`main`	Title for plot
`ylab`	Label for y-axis
`xlab`	Label for x-axis
`...`	Other plotting arguments

Details

The Torgegram function creates a list of distances and empirical semivariogram values, along with number of pairs of points in each bin, for both flow-connected and flow-unconnected sites. Flow-connected locations lie on the same stream network (share a common downstream junction) and water flows from one location to the other. Flow-unconnected locations also lie on the same stream network, but do not share flow. The output is of class Torgegram. This is the default plotting method for this class.

Value

Plot of empirical semivariogram values

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

SpatialStreamNetwork-class, plot

Examples

```
library(SSN)
data(mf04p)
#make sure mf04p has the correct path, will vary for each users installation
mf04p@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")

ESVF <- Torgegram(mf04p, "Summer_mn")
plot(ESVF)

ESVF <- Torgegram(mf04p, "Summer_mn", maxlag = 20000, nlag = 10)
plot(ESVF, sp.relationship = "fc", col = "red", main = "Flow-connected Torgegram")
plot(ESVF, sp.relationship = "fu", min.cex = .4, max.cex = 8,
     main = "Flow-unconnected Torgegram")
plot(ESVF, min.cex = .4, max.cex = 8, col = c("darkgray", "black"),
     main = "", xlab = "Stream Distance (m)")

# generate and plot an empirical semivariogram based on model residuals
data(modelFits)
#make sure fitSp has the correct path, will vary for each users installation
fitSp$ssn.object@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")
resids <- residuals(fitSp)
names(resids$ssn.object)
ESVF <- Torgegram(resids, "_resid_", maxlag = 20000,
    nlag = 10)
plot(ESVF, xlim = c(0,10000))
```

poiSp — *Fitted glmssn object for example data set MiddleFork.ssn*

Description

The MiddleFork04.ssn data folder contains the spatial, attribute, and topological information needed to construct a spatial stream network object using the SSN package. This is a fitted model using the `glmssn` function.

Details

See the help for `glmssn` for how the model was created.

Examples

```
library(SSN)
data(modelFits)
ls()
```

predict.glmssn — *Calculate Predictions for Prediction Sites*

Description

The `predict.glmssn` function calculates prediction values for prediction sites based on the results stored within an object of class glmssn

Usage

```
## S3 method for class 'glmssn'
predict(object, predpointsID, ...)
```

Arguments

`object`	the glmssn object
`predpointsID`	the prediction points ID
`...`	other arguments passed to predict

Details

`predict.glmssn` is a generic function that has been modified for `glmssn` objects. Two new columns are added to the prediction points data.frame (point.data) within the existing `glmssn` object. The first column contains prediction values and has the same name as the response variable. The second column <response name>.predSE contains the standard errors for the predictions.

Value

The overall structure is the same as an object of class glmssn, except the prediction points data.frame in the SpatialStreamNetwork object (list-item 2) is appended by the following columns:

```
<response name> # The prediction value for each prediction site
<response name>.predSE # The standard error of the prediction value
```

Details of this object structure can be found using the names command.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

glmssn

Examples

```
library(SSN)
data(modelFits)
#Update the fitSp path, will vary for each users installation
fitSp$ssn.object <- updatePath(fitSp$ssn.object,
  system.file("lsndata/MiddleFork04.ssn", package = "SSN"))
names(fitSp)
names(fitSp$ssn.object)

ssnpred <- predict(fitSp, predpointsID = "pred1km")
names(ssnpred)
names(ssnpred$ssn.object) ## Note additional predicted columns in pred1km
plot(ssnpred)
```

---

print.summary.glmssn *Print summary - S3 Method for Class 'glmssn'*

Description

print is a generic function that prints output summaries of fitted models in the SSN package. In particular, the function invokes methods for objects of class summary.glmssn.

Usage

```
## S3 method for class 'summary.glmssn'
print(x, digits = max(3L, getOption("digits") - 3L),
   signif.stars = getOption("show.signif.stars"), ...)
```

Arguments

x	an object of class summary.glmssn
digits	the number of significant digits to use when printing.
signif.stars	logical. If 'TRUE', 'significance stars' are printed for each coefficient.
...	other arguments passed to print

Details

The print.summary.glmssn function summarizes and prints the fitted model with a table of estimates for the fixed effects and the covariance parameter estimates.

Value

Prints the summary beginning with call and arguments used, plus:

Residuals	a summary of the min, max, and quartiles of the usual residuals.
Coefficients	a p x 4 matrix with columns for the estimated coefficient, its standard error, t-statistic and corresponding (two-sided) p-value. Aliased coefficients are omitted.
Covariance Parameters	a list of covariance parameter estimates for each covariance model.
Residual standard error	the square-root of the sum of all of the variance (partial sill) parameters.
Generalized R-squared	the generalized R-squared value of the fitted model

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

glmssn,link{covparms}

Examples

```
library(SSN)
data(modelFits)
#make sure fitSp has the correct path, will vary for each users installation
fitSp$ssn.object@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")
names(fitSp)
names(fitSp$ssn.object)

  print(summary(fitSp))
  #the same effect is achieved with this
  summary(fitSp)
  #or this
  fitSp
```

putSSNdata.frame *putSSNdata.frame*

Description

Replacet the data.frame in an Object of Class SpatialStreamNetwork

Usage

```
putSSNdata.frame(DataFrame, x, Name = "Obs")
```

Arguments

DataFrame	data.frame to be placed into the SpatialStreamNetwork-class object
x	an object of class SpatialStreamNetwork, influenceSSN, glmssn, or glmssn.predict
Name	the internal name of the data set in the object x. For observed values, this will always be "Obs", the default.

Details

The internal Name for observed data in objects of class SpatialStreamNetwork is "Obs" and it is the default. If another Name is specified, it must represent a prediction data set in the SpatialStreamNetwork-class, influenceSSN-class, glmssn-class, or "glmssn.predict" object. For SpatialStreamNetwork objects, these names are obtained using the call ssn@predpoints@ID. For all other object classes, the names are obtained using the call object$ssn.object@predpoints@ID. See examples for additional details.

Note that, the DataFrame must be of the same dimensions as the original data.frame in the object x.

Value

Returns an object of the same class as x.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

getSSNdata.frame, SpatialStreamNetwork-class, influenceSSN-class, glmssn-class

Examples

```
library(SSN)
mf04 <- importSSN(system.file("lsndata/MiddleFork04.ssn",
package = "SSN"), o.write = TRUE)

# Take out the data.frame, make a change and put it back
```

```
obs.df <- getSSNdata.frame(mf04)
obs.df$Year_cat <- as.factor(obs.df$SampleYear)
mf04 <- putSSNdata.frame(obs.df, mf04)
```

`residuals.glmssn` *Compute Model Residuals for glmssn Objects*

Description

`residuals.glmssn` is a generic function that has been modified for `glmssn` objects. It produces residuals from glmssn spatial models.

Usage

```
## S3 method for class 'glmssn'
residuals(object, cross.validation=TRUE, ...)
```

Arguments

`object`	an object of class glmssn
`cross.validation`	logical value indicating whether leave-one-out cross-validation residuals will be computed. The default is TRUE. Setting `cross.validation` to FALSE may decrease processing times for large datasets.
`...`	Other arguments

Details

When using `residual(x)` on a glmssn object, the data for which the model was fit is contained in the obspoints slot @SSNPoints@point.data. This data frame contains the response variable for the model, so it is appended with the following columns,

obsval The response value used for fitting the model

fit For a model z = Xb + e, where X is a design matrix for fixed effects and e contains all random components, then the fit is Xb, where b contains the estimated fixed effects parameters.

resid The raw residuals. The observed response value minus the fitted value using only fixed effect estimates (no random effects are included).

resid.stand Standardized residuals, calculated by dividing the raw residuals by the corresponding estimated standard errors

resid.student Studentized residuals. From a model z = Xb + e, we can create uncorrelated data by taking a model Cz = CXb + Ce, where var(e) = sV, C is the square root inverse of V, and s is an overall variance parameter. Under such a model, the hat matrix is H = CX*inv(X'(C'C)X)*X'C'. Then, the variance of a residual is s(1-H[i,i]), and so the studentized residual is r[i]/sqrt(s(1-H[i,i]), where r[i] is the ith raw residual.

leverage Leverage. H[i,i] as described for Studentized residuals.

CooksD Cook's D, using the method of creating uncorrelated data as for Studentized residuals, and then applying Cook's D.

resid.crossv Cross-validation residuals, obtained from leave-one-out-at-a-time and taking the difference between the observed response value and that predicted after removing it. Only computed if `cross.validation` was set to TRUE.

CrossValPred The leave-one-out cross-validation predictions. Only computed if `cross.validation` is set to TRUE.

CrossValStdErr Estimated standard errors for the leave-one-out cross-validation predictions. Only computed if `cross.validation` is set to TRUE.

Value

The returned object is of class `influenceSSN-class`. It similar to a glmssn-classobject; the main difference is that additional columns (described in the details section) have been added to the observed points data.frame.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

Examples

```
library(SSN)
data(modelFits)
#make sure fitSp has the correct path, will vary for each users installation
fitSp$ssn.object@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")
names(fitSp)
names(fitSp$ssn.object)

  resids <- residuals(fitSp)
  class(resids)
  names(resids)
  plot(resids)
  hist(resids, xlab = "Raw Residuals")
  qqnorm(resids)

  resids.df <- getSSNdata.frame(resids)
  plot(resids.df[,"_resid_"], ylab = "Raw Residuals")
```

`SimulateOnSSN` *Simulating Data on Spatial Stream Networks*

Description

This function works on objects of class `SpatialStreamNetwork` to simulate data with spatially autocorrelated errors from models as described in Ver Hoef and Peterson (2010) and Peterson and Ver Hoef (2010). It works with simulated or real stream networks. It can simulate from Gaussian (normal), Poisson and binomial distributions.

Usage

```
SimulateOnSSN(ssn.object, ObsSimDF, PredSimDF = NULL, PredID = NULL,
  formula, coefficients,
  CorModels = c("Exponential.tailup", "Exponential.taildown", "Exponential.Euclid"),
  use.nugget = TRUE, use.anisotropy = FALSE,
  CorParms = c(1, 10000, 1, 10000, 1, 10000, 0.1),
  addfunccol = NULL, useTailDownWeight = FALSE, family = "Gaussian", mean.only=FALSE)
```

Arguments

ssn.object	an object of class SpatialStreamNetwork
ObsSimDF	a data frame used to replace the existing observed sites data frame in ssn.object. It is safest to first extract the point.data data.frame from ssn.object, then add covariate values to the extracted data.frame. See the examples section.
PredSimDF	a data frame used to replace the existing prediction site data frame in ssn.object It is safest to first extract the point.data data.frame from ssn.object, then add covariate values to the extracted data frame. See the examples section. The covariate names should match those from ObsSimDF.
PredID	a string representing the ID (name) of the prediction slot in the ssn.object. The ith name is accessed using the call ssn.object@predpoints@ID[i].
formula	a one-sided formula to the right of, and including, the ~. This is similar to linear model formula but works in reverse. It will create a design matrix based on the formula and covariates in the ObsSimDF and PredSimDF.
coefficients	a vector of numeric values representing the coefficients. The formula creates the design matrix, and these coefficients are multiplied by the columns in the design matrix. If the design matrix is X, and coefficients are beta, then the mean values are created as X %*% beta. Note that this presumes some knowledge about how R will create design matrices from formulas.
CorModels	a character vector of spatial autocorrelation model names for stream networks. The individual models should be of different "types". It can be "NULL" for a non-spatial model, or it can contain any of the tailup models: "Exponential.tailup" (default), "LinearSill.tailup", "Spherical.tailup", "Mariah.tailup", and/or one of the taildown models: "Exponential.taildown" (default), "LinearSill.taildown", "Spherical.taildown", "Mariah.taildown", or one of the Euclidean distance models: "Spherical.Euclid", "Gaussian.Euclid", "Exponential.Euclid" (default), "Cauchy.Euclid". The 4 tailup and taildown models are described in Ver Hoef and Peterson (2010) and the 4 Euclidean distance models are standard spatial autocorrelation models. If this is NULL, then use.nugget = TRUE will impose independence between observations, or a classical non-spatial linear model.
use.nugget	logical. Add a nugget effect, default is TRUE. This can be thought of as a variance component for independent errors, adding a variance component only along the diagonal of the covariance matrix.
use.anisotropy	logical. Use anistropy for the Euclidean distance based spatial model in CorModels. Not implemented at the current time.
CorParms	a vector of numeric covariance parameters. Each of the CorModels will generally have two parameters, a partial sill and a range (in that order, and in the

order as specified by CorModels). If use.nugget = TRUE, then a final CorParms parameter should be added for the nugget effect.

`addfunccol` for the tailup models, weights are need to be used to account for dendritic branching in the network. This is achieved using an additive function and is described in Ver Hoef and Peterson (2010). The name of the variable in the ssn.object that is to be used to define weights should be given here. See example below.

`useTailDownWeight`
Use weighting in the tail-down models in the same way as for tail-up models. Logical that defaults to FALSE.

`family` the error distribution and link function to be used in the model. This is a character string that is either "Gaussian" (default), "Poisson", or "Binomial."

`mean.only` Logical that defaults to FALSE.

Details

Models are specified symbolically in a manner similar to `lm` and other model-fitting functions in R, but here the formula is right-handed (e.g. ~ x1 + x2 + x3, where x1, x2, x3 are the 'terms'). If the formula is specified as ~ terms, data will be simulated as Sim_Values ~ terms, where Sim_Values is the (numeric) response vector and terms is a series of fixed effect linear predictors for Sim_Values. A terms specification of the form first + second indicates all the terms in first together with all the terms in second with duplicates removed. A specification of the form first:second indicates the set of terms obtained by taking the interactions of all terms in first with all terms in second. The specification first*second indicates the cross of first and second. This is the same as first + second + first:second. See `model.matrix` for additional details. The terms in the formula are re-ordered so that main effects come first, followed by the interactions, all second-order, all third-order and so on. A formula has an implied intercept term. To remove this use either ~ x - 1 or ~ 0 + x. See `formula` for more details about allowable formulae.

The observed data data.frame used for simulating is contained in the slot ssn.object@obspoints@SSNPoints[[1]]@point.data and can be easily accessed using `getSSNdata.frame`. The function `putSSNdata.frame` can be used to to put it back after it has been modified. Likewise, the predicted data data.frame used for simulating is contained in stored in ssn.object@predpoints@SSNPoints[[i]]@point.data, where i is the ith prediction data set within `ssn.object`; generally i = 1, but is not a limit on the number of prediction datasets that may be included. Calls to `getSSNdata.frame` and `putSSNdata.frame` may be used to access the prediction site data.frames as well.

Value

Output from SimulateOnSSN contains three list items.

`ssn.object` the input SSN that now has simulated data in the observed and/or prediction data.frames. Within these data.frames, the simulated data have a column heading called "Sim_Values"

`FixedEffects` a data.frame of the ordered column names for the design matrix that was created. The first column is the column name of the design matrix, and the second column is the coefficient used for that fixed effect for simulation. This can be used to ensure that the `coefficients` are being used in the way that they were intended.

CorParms — a data.frame of the ordered variance component model parameters. No matter the order of the Corparms input argument, the covariance parameters are applied in the follwoing order, if specifed in CorParms: tailup model (partial sill then range), taildown (partiall sill then range), Euclidean model (partial sill then range), random effects variance components ordered alphanumerically, and finally the nugget. This can be used to ensure that the CorParms are being applied in the way that they were intended.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

References

Peterson, E. E. and Ver Hoef, J. M. (2010) A mixed-model moving-average approach to geostatistical modeling in stream networks. Ecology ***91(3)***, *644–651.*

Ver Hoef, J. M. and Peterson, E. E. (2010) A moving average approach for spatial statistical models of stream networks (with discussion). Journal of the American Statistical Association ***105***, *6–18. DOI: 10.1198/jasa.2009.ap08248. Rejoinder pgs. 22 - 24.*

Examples

```
#####################################
## example 1: Gaussian data, 2 networks
#####################################

library(SSN)
set.seed(101)
## simulate a SpatialStreamNetwork object
raw1.ssn <- createSSN(n = c(10,10),
    obsDesign = binomialDesign(c(50,50)), predDesign = binomialDesign(c(100,100)),
importToR = TRUE, path = paste(tempdir(),"/sim1", sep = ""))
plot(raw1.ssn)

## create distance matrices, including between predicted and observed
createDistMat(raw1.ssn, "preds", o.write=TRUE, amongpred = TRUE)

## look at the column names of each of the data frames
names(raw1.ssn)

## extract the observed and predicted data frames
raw1DFobs <- getSSNdata.frame(raw1.ssn, "Obs")
raw1DFpred <- getSSNdata.frame(raw1.ssn, "preds")

## add a continuous covariate randomly
raw1DFobs[,"X1"] <- rnorm(length(raw1DFobs[,1]))
raw1DFpred[,"X1"] <- rnorm(length(raw1DFpred[,1]))

## add a categorical covariate randomly
raw1DFobs[,"F1"] <- as.factor(sample.int(3,length(raw1DFobs[,1]), replace = TRUE))
raw1DFpred[,"F1"] <- as.factor(sample.int(3,length(raw1DFpred[,1]), replace = TRUE))
```

```
## simulate Gaussian data
sim1.out <- SimulateOnSSN(raw1.ssn,
ObsSimDF = raw1DFobs,
PredSimDF = raw1DFpred,
PredID = "preds",
formula = ~ X1 + F1,
coefficients = c(1, .5, -1, 1),
CorModels = c("Exponential.tailup", "Exponential.taildown"),
use.nugget = TRUE,
use.anisotropy = FALSE,
CorParms = c(2, 5, 2, 5, 0.1),
addfunccol = "addfunccol")

## Columns of design matrix, coefficients argument applied to these
sim1.out$FixedEffects

## extract the ssn.object
sim1.ssn <- sim1.out$ssn.object

## extract the observed and predicted data frames, now with simulated values
sim1DFobs <- getSSNdata.frame(sim1.ssn, "Obs")
sim1DFobs[,"Sim_Values"]
sim1DFpred <- getSSNdata.frame(sim1.ssn, "preds")
sim1DFpred[,"Sim_Values"]

## plot the simulated observed values
plot(sim1.ssn, "Sim_Values")

## store simulated prediction values, and then create NAs in their place
sim1preds <- sim1DFpred[,"Sim_Values"]
sim1DFpred[,"Sim_Values"] <- NA
sim1.ssn <- putSSNdata.frame(sim1DFpred, sim1.ssn, "preds")

# NOT RUN, IT TAKES A MINUTE OR SO
## fit a model to see how well we estimate simulation parameters
#fitSimGau <- glmssn(Sim_Values ~ X1 + F1, ssn.object = sim1.ssn,
#  CorModels = c("Exponential.tailup", "Exponential.taildown"),
#  addfunccol = "addfunccol")
# LOAD A STORED VERSION INSTEAD
data(modelFits)
#make sure fitSimGau has the correct path, will vary for each users installation
#predictions depend on distance matrix created earlier with createDistMat function
#path of this lsn directory was created with createSSN
fitSimGau$ssn.object@path <- paste(tempdir(),"/sim1", sep = "")

summary(fitSimGau)

## make predictions
pred1.ssn <- predict(fitSimGau,"preds")
par(bg = "grey60")
plot(pred1.ssn, color.palette = terrain.colors(10))
par(bg = "white")
```

```
## compare predicted values to simulated values
pred1DF <- getSSNdata.frame(pred1.ssn, "preds")
plot(sim1preds, pred1DF[,"Sim_Values"], xlab = "True", ylab = "Predicted",
pch = 19)

#####################################
## example 2: Binomial data, 1 network
#####################################

set.seed(102)
## simulate a SpatialStreamNetwork object
raw2.ssn <- createSSN(n = 20,
obsDesign = binomialDesign(100), predDesign = binomialDesign(200),
importToR = TRUE, path = paste(tempdir(),"/sim2", sep = ""))
plot(raw2.ssn)

## create distance matrices, including between predicted and observed
createDistMat(raw2.ssn, "preds", o.write=TRUE, amongpred = TRUE)

## look at the column names of each of the data frames
names(raw2.ssn)

## extract the observed and predicted data frames
raw2DFobs <- getSSNdata.frame(raw2.ssn, "Obs")
raw2DFpred <- getSSNdata.frame(raw2.ssn, "preds")

## add a continuous covariate randomly
raw2DFobs[,"X1"] <- rnorm(length(raw2DFobs[,1]))
raw2DFpred[,"X1"] <- rnorm(length(raw2DFpred[,1]))

## add a categorical covariate randomly
raw2DFobs[,"F1"] <- as.factor(sample.int(3,length(raw2DFobs[,1]), replace = TRUE))
raw2DFpred[,"F1"] <- as.factor(sample.int(3,length(raw2DFpred[,1]), replace = TRUE))

## simulate Poisson data
sim2.out <- SimulateOnSSN(raw2.ssn,
ObsSimDF = raw2DFobs,
PredSimDF = raw2DFpred,
PredID = "preds",
formula = ~ X1 + F1,
coefficients = c(0, .5, -1, 1),
CorModels = c("Exponential.tailup", "Exponential.taildown",
"Exponential.Euclid"),
use.nugget = TRUE,
use.anisotropy = FALSE,
CorParms = c(.5, 5, .5, 5, .5, 2, 0.01),
addfunccol = "addfunccol",
family = "Binomial")

## Columns of design matrix, coefficients argument applied to these
sim2.out$FixedEffects
```

```
## extract the ssn.object
sim2.ssn <- sim2.out$ssn.object

## extract the observed and predicted data frames, now with simulated values
sim2DFobs <- getSSNdata.frame(sim2.ssn, "Obs")
sim2DFobs[,"Sim_Values"]
sim2DFpred <- getSSNdata.frame(sim2.ssn, "preds")
sim2DFpred[,"Sim_Values"]

## plot the simulated observed values
plot(sim2.ssn, "Sim_Values", nclasses = 2, color.palette = c("blue","red"),
breaktype = "user", brks = cbind(c(-.5,.5),c(.5, 1.5)))

## store simulated prediction values, and then create NAs in their place
sim2preds <- sim2DFpred[,"Sim_Values"]
sim2DFpred[,"Sim_Values"] <- NA
sim2.ssn <- putSSNdata.frame(sim2DFpred, sim2.ssn, "preds")

# NOT RUN, IT TAKES A MINUTE OR SO
## fit a model to see how well we estimate simulation parameters
#fitSimBin <- glmssn(Sim_Values ~ X1 + F1,
#  ssn.object = sim2.ssn, EstMeth = "REML", family = "Binomial",
#  CorModels = "Exponential.taildown",
#  addfunccol = "addfunccol")
# LOAD A STORED VERSION INSTEAD
data(modelFits)
#make sure fitSimBin has the correct path, will vary for each users installation
#predictions depend on distance matrix created earlier with createDistMat function
#path of this lsn directory was created with createSSN
fitSimBin$ssn.object@path <- paste(tempdir(),"/sim2", sep = "")
summary(fitSimBin)

## make predictions
predSimBin <- predict(fitSimBin,"preds")
par(bg = "grey60")
plot(predSimBin, color.palette = terrain.colors(10))
par(bg = "white")

## compare predicted values to simulated values
pred2DF <- getSSNdata.frame(predSimBin, "preds")
table(sim2preds, (pred2DF[,"Sim_Values"]>0)*1)

####################################
## example 3: Poisson data, 1 network
####################################
## NOT RUN Similar to Binomial Data
#set.seed(104)
## simulate a SpatialStreamNetwork object
#raw3.ssn <- createSSN(n = 20,
# obsDesign = binomialDesign(100), predDesign = binomialDesign(200),
# importToR = TRUE, path = paste(tempdir(),"/sim3", sep = ""))
#plot(raw3.ssn)
```

```
## create distance matrices, including between predicted and observed
#createDistMat(raw3.ssn, "preds", o.write=TRUE, amongpred = TRUE)

## look at the column names of each of the data frames
#names(raw3.ssn)

## extract the observed and predicted data frames
#raw3DFobs <- getSSNdata.frame(raw3.ssn, "Obs")
#raw3DFpred <- getSSNdata.frame(raw3.ssn, "preds")

## add a continuous covariate randomly
#raw3DFobs[,"X1"] <- rnorm(length(raw3DFobs[,1]))
#raw3DFpred[,"X1"] <- rnorm(length(raw3DFpred[,1]))
## add a categorical covariate randomly

#raw3DFobs[,"F1"] <- as.factor(sample.int(3,length(raw3DFobs[,1]), replace = TRUE))
#raw3DFpred[,"F1"] <- as.factor(sample.int(3,length(raw3DFpred[,1]), replace = TRUE))

## simulate Poisson data
#sim3.out <- SimulateOnSSN(raw3.ssn,
# ObsSimDF = raw3DFobs,
# PredSimDF = raw3DFpred,
# PredID = "preds",
# formula = ~ X1 + F1,
# coefficients = c(1, .5, -1, 1),
# CorModels = c("Exponential.taildown"),
# use.nugget = TRUE,
# use.anisotropy = FALSE,
# CorParms = c(.5, 5, 0.01),
# addfunccol = "addfunccol",
# family = "Poisson")

## Columns of design matrix, coefficients argument applied to these
#sim3.out$FixedEffects

## extract the ssn.object
#sim3.ssn <- sim3.out$ssn.object

## extract the observed and predicted data frames, now with simulated values
#sim3DFobs <- getSSNdata.frame(sim3.ssn, "Obs")
#sim3DFobs[,"Sim_Values"]
#sim3DFpred <- getSSNdata.frame(sim3.ssn, "preds")
#sim3DFpred[,"Sim_Values"]

## plot the simulated observed values
#plot(sim3.ssn, "Sim_Values")

## store simulated prediction values, and then create NAs in their place
#sim3preds <- sim3DFpred[,"Sim_Values"]
#sim3DFpred[,"Sim_Values"] <- NA
#sim3.ssn <- putSSNdata.frame(sim3DFpred, sim3.ssn, "preds")

# NOT RUN, IT TAKES A MINUTE OR SO
```

```
## fit a model to see how well we estimate simulation parameters
#fitSimPoi <- glmssn(Sim_Values ~ X1 + F1,
#  ssn.object = sim3.ssn, EstMeth = "REML", family = "Poisson",
#  CorModels = "Exponential.taildown",
#  addfunccol = "addfunccol")
# LOAD A STORED VERSION INSTEAD
#data(modelFits)
#make sure fitSimPoi has the correct path, will vary for each users installation
#predictions depend on distance matrix created earlier with createDistMat function
#path of this lsn directory was created with createSSN
#fitSimPoi$ssn.object@path <- paste(tempdir(),"/sim3", sep = "")
#summary(fitSimPoi)

## make predictions
#pred3.ssn <- predict(fitSimPoi,"preds")
#par(bg = "grey60")
#plot(pred3.ssn, color.palette = terrain.colors(10))
#par(bg = "white")

## compare predicted values to simulated values
#pred3DF <- getSSNdata.frame(pred3.ssn, "preds")

#plot(log(sim3preds+.1), pred3DF[,"Sim_Values"], xlab = "True", ylab = "Estimated",
# pch = 19)
```

`SpatialStreamNetwork-class` *Class "SpatialStreamNetwork"*

Description

A class that holds spatial features (points and lines), attributes, and topological information for stream networks.

Objects from the Class

SpatialStreamNetwork objects can be created by calls in the form `importSSN(x)`.

Slots

`network.line.coords`**:** Object of class `"data.frame"`; columns include `NetworkID` (a factor identifying individual networks that are assumed independent for modeling purposes), `SegmentID` (a unique identifier for each stream segment), and `DistanceUpstream` (the cumulative distance from the network outlet, the most downstream point on a network, to the uppermost point of that stream segment). Row names are set to `SegmentID` values.

`obspoints`**:** Object of class `"SSNPoints"` with 2 slots

```
@ SSNPoints: List of SSNPoint objects with 5 slots
  @ network.point.coords: object of class "data.frame". Row names
      represent point identifiers (pid) stored in the point.data
      data.frame.
       $ NetworkID: factor identifying the NetworkID of that point
     $ SegmentID: factor identifying the unique stream segment of that point
      $ DistanceUpstream: numeric value representing the cumulative
        distance from the network outlet, the most downstream point
          on a network, to that point
    @ point.coords: numeric matrix or "data.frame" with x- and y-
     coordinates (each row is a point); row names represent point
      identifiers (pid) stored in the point.data data.frame.
  @ point.data: object of class "data.frame"; the number of rows in
      data should equal the number of points in the
      network.point.coords object; row names are set to the pid
      attribute.
    @ points.bbox: Object of class "matrix"; see Spatial-class
    @ proj4string: Object of class "CRS"; see CRS-class
@ ID: character string representing the name of the observation points
```

`predpoints`: Object of class `"SSNPoints"`. See description of object format under slot obspoints

`path`: Object of class `"character"` representing the file location of the SSN dataset

`data`: Object of class `"data.frame"`. The number of rows in data should equal the number of lines in the lines object. Row names correspond to `SegmentID` values

`lines`: Object of class `"list"`; list members are all of class Lines. Each list item represents a polyline segment and the number of list items should be equal to the number of rows in the `network.line.coords` object

`bbox`: Object of class `"matrix"`; see Spatial

`proj4string`: Object of class `"CRS"`; see `CRS`

Extends

Class SpatialLinesDataFrame, directly. Class SpatialLines-class, by class "SpatialLinesDataFrame", distance 2. Class Spatial, by class "SpatialLinesDataFrame", distance3.

See Also

Line-class, Lines-class

`splitPredictions` *Split Prediction Sets in a SpatialStreamNetwork Object*

Description

The `splitPrediction` function is used to split prediction sets in a SpatialStreamNetwork object into smaller prediction sets. It returns a SpatialStreamNetwork object with additional prediction sets based on equal interval splits, a factor value stored within the prediction set, or a logical expression.

Usage

```
splitPredictions(ssn, predpointsID, chunksof, by, subset, new.id)
```

Arguments

ssn	a SpatialStreamNetwork object
predpointsID	a character string representing the prediction points ID
chunksof	numeric value representing the size of the new prediction sets. The existing prediction set is split equally to produce multiple prediction sets of this size
by	character string representing the column name of type factor or integer that the split will be based on
subset	logical expression indicating which elements or rows to keep; missing values are taken as false
new.id	character string representing the new prediction points ID. This value is only specified when the subset method is used

Details

Three methods have been provided to split prediction sets: chunksof, by, and subset. The chunksof method is used to split the existing prediction set into multiple equally-sized prediction sets. The by method is used if the prediction set is to be split into multiple new prediction sets based on an existing column of type factor or integer. The subset method is used to create one new prediction setbased on a logical expression. When the subset method is used, the prediction points ID may be specified using the new.id argument. Note that, only one method may be specified when the splitPredictions function is called.

Value

The splitPredictions function returns an object of class SpatialStreamNetwork that contains one or more new prediction sets. Shapefiles of the new prediction sets are written to the .ssn directory designated in the ssn@path slot. Distances matrices for the predicted and observed locations are also calculated and stored in the .ssn directory.

Author(s)

Rohan Shah <support@SpatialStreamNetworks.com>

See Also

SpatialStreamNetwork, subsetSSN

Examples

```
library(SSN)

## Create temporary .ssn directory to work with
old.wd <- getwd()
setwd(system.file("lsndata/MiddleFork04.ssn",package = "SSN"))
```

```
file.list <- list.files()
dir.create("tmp.ssn")

file.copy(file.list, "tmp.ssn", recursive = TRUE)

## Import the SpatialStreamNetwork object
mf04 <- importSSN("tmp.ssn", predpts = "pred1km")

# Split using the chunksof method
split1 <- splitPredictions(mf04, "pred1km", chunksof = 100)
summary(split1)

## Split using the by method
split2 <- splitPredictions(mf04, "pred1km", by = "netID")
summary(split2)

## Split using the subset method
split3 <- splitPredictions(mf04, "pred1km", subset = netID == 1,
     new.id="netID-1")
summary(split3)

## Split the predictions after creating a glmssn object
mf04.glmssn <- glmssn(Summer_mn ~ ELEV_DEM + SLOPE, mf04,
    CorModels = c("Exponential.tailup", "Exponential.taildown",
    "Exponential.Euclid"),addfunccol = "afvArea")
mf04.glmssn$ssn.object <- splitPredictions(mf04.glmssn$ssn.object, "pred1km",
    subset = netID == 1, new.id="netID-1")
pred.split<- predict(mf04.glmssn, "netID-1")
plot(pred.split)

## Delete temporary files and return to workspace
unlink("tmp.ssn", recursive = TRUE)
setwd(old.wd)
```

---

subsetSSN *Subset a SpatialStreamNetwork Object*

Description

The subsetSSN function returns an SpatialStreamnetwork object that has been subset based on a logical expression.

Usage

```
subsetSSN(ssn, filename = "", subset, clip = FALSE)
```

Arguments

ssn	a SpatialStreamNetwork-class object

filename	the file path to the new .ssn folder, in string format. When writing, include the .ssn folder in the path
subset	logical expression indicating which elements or rows to keep: missing values are taken as false
clip	default = FALSE. If TRUE, create a subset of the edges and prediction sites, based on the same logical expression used to subset the observed sites, and export the subset data to the new .ssn folder

Value

subsetSSN returns an object of class SpatialStreamNetwork. It also creates and stores an SQLite database, binaryID.db, within the new .ssn directory.

Author(s)

Erin E. Peterson <support@SpatialStreamNetworks.com>

See Also

SpatialStreamNetwork-class

Examples

```
library(SSN)

ssn <- importSSN(system.file("lsndata/MiddleFork04.ssn", package = "SSN"),
    predpts = "pred1km")

ssn.sub1 <- subsetSSN(ssn, filename =
  paste(tempdir(),"/subset1.ssn", sep = ""), subset = Summer_mn > 13)
ssn.sub2 <- subsetSSN(ssn, filename =
  paste(tempdir(),"/subset2.ssn", sep = ""), subset = netID == 1, clip = TRUE)
```

---

summary.glmssn *Summary - S3 Method for Class 'glmssn'*

Description

summary is a generic function that produces output summaries of fitted models in the SSN package. In particular, the function invokes methods for objects of class glmssn.

Usage

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

Arguments

object	an object of class glmssn
...	other arguments passed to summary

Details

The summary.glmssn function summarizes the fitted model with a table of estimates for the fixed effects and the covariance parameter estimates. It also provides a warning log.

Value

An object with a 3-item list:

fixed.effects.estimates	a data.frame with columns FactorLevel (effect & level) Estimate, std.err, t.value, and prob.t (p-value) for the fixed effects
covariance.parameter.estimates	a list of covariance parameter estimates for each covariance model
Warnings	a list of warnings associated with the object

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

glmssn,link{covparms}

Examples

```
library(SSN)
data(modelFits)
#make sure fitSp has the correct path, will vary for each users installation
fitSp$ssn.object@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")
names(fitSp)
names(fitSp$ssn.object)

  summary(fitSp)
```

Torgegram *Empirical Semivariogram Based on Hydrologic Distance and flow connection*

Description

Torgegram computes the empirical semivariogram from the data based on hydrologic distance. The results are presented separately for flow-connected and flow-unconnected sites.

Usage

```
Torgegram(object, ResponseName, maxlag = NULL, nlag = 6,
inc = 0, nlagcutoff = 15, EmpVarMeth = "MethMoment")
```

Arguments

object	an object of class SpatialStreamNetwork-class or influenceSSN-class
ResponseName	a response or residual variable name in the data.frame of observed data in the SpatialStreamNetwork or influenceSSN object.
maxlag	the maximum lag distance to consider when binning pairs of locations by the hydrologic distance that separates them. The default is the median distance between all pairs of locations.
nlag	the number of lag bins to create. The distance between endpoints that define a bin will have equal lengths for all bins. The bin sizes are then determined from the minimum lag in the data, and the specification of maxlag.
inc	the bin distance between endpoints. It is possible to specify the bin distance rather than nlag. In this case, the number of bins is determined by the bin distance and the distance between the mininum and maximum (maxlag) lag in the data
nlagcutoff	the minimum number of pairs needed to estimate the semivariance for a bin. If the sample sizes is less than this value, the semivariance for the bin is not calculated.
EmpVarMeth	method for computing semivariances. The default is "MethMoment", the classical method of moments, which is just the average difference-squared within bin classes. "Covariance" computes covariance rather than semivariance, but may be more biased because it subtracts off the simple mean of the response variable. "RobustMedian" and "RobustMean" are robust estimators proposed by Cressie and Hawkins (1980). If v is a vector of all pairwise square-roots of absolute differences within a bin class, then "RobustMedian" computes median(v)^4/.457. "RobustMean" computes mean(v)^4/(.457 + .494/length(v)).

Details

The Torgegram function creates a list of hydrologic distances and empirical semivariogram values, along with number of pairs of points in each bin, for both flow-connected and flow-unconnected

sites. Flow-connected locations lie on the same stream network (share a common downstream junction) and water flows from one location to the other. Flow-unconnected locations also lie on the same stream network, but do not share flow. The output is of class Torgegram.

Value

A list of six vectors describing the semivariance values for each bin and the hydrologic distances and number of pairs used to estimate those values. These data are presented separately for flow-connected and flow-unconnected sites.

`distance.connect`
: the mean hydrologic distance separating pairs of flow-connected sites used to calculate the semivariance for each bin

`gam.connect`
: the mean semivariance for flow-connected sites in each bin

`np.connect`
: the number of pairs of flow-connected sites used to calculate the semivariance for each bin

`distance.unconnect`
: the mean hydrologic distance separating pairs of flow-connected sites used to calculate the semivariance for each bin

`gam.unconnect`
: the mean semivariance for flow-connected sites in each bin

`np.unconnect`
: the number of pairs of flow-connected sites used to calculate the semivariance for each bin

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

A generic `plot` operates on the object created here.

Examples

```
library(SSN)
data(mf04p)
#make sure mf04p has the correct path, will vary for each users installation
mf04p@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")

ESVF <- Torgegram(mf04p, "Summer_mn")
plot(ESVF)

ESVF <- Torgegram(mf04p, "Summer_mn", maxlag = 20000, nlag = 10)
plot(ESVF, sp.relationship = "fc", col = "red", main = "Flow-connected Torgegram")
plot(ESVF, sp.relationship = "fu", min.cex = .4, max.cex = 8,
   main = "Flow-unconnected Torgegram")
plot(ESVF, min.cex = .4, max.cex = 8, col = c("darkgray", "black"),
   main = "", xlab = "Stream Distance (m)")

# generate and plot an empirical semivariogram based on model residuals
```

```
data(modelFits)
#make sure fitSp has the correct path, will vary for each users installation
fitSp$ssn.object@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")
resids <- residuals(fitSp)
names(resids$ssn.object)
ESVF <- Torgegram(resids, "_resid_", maxlag = 20000,
  nlag = 10)
plot(ESVF, xlim = c(0,10000))
```

---

updatePath *Update Path Slot in SpatialStreamNetwork Object*

Description

Updates the path slot in an existing SpatialStreamNetwork object based on a user-defined filepath.

Usage

```
updatePath(ssn, filepath)
```

Arguments

ssn	a SpatialStreamNetwork object
filepath	path name to the .ssn folder, in string format including quotes. Also include the .ssn folder in the path name

Details

At times, it may be necessary to move a .ssn directory, which is linked to a SpatialStreamNetwork object in an R workspace. If the .ssn directory is moved, the path slot must be updated before using the glmssn function. The updatePath function serves this purpose.

Author(s)

Erin E. Peterson <support@SpatialStreamNetworks.com>

varcomp *Variance Components for glmssn Objects*

Description

The varcomp function displays the variance proportions for the autocovariance functions in the glmssn object.

Usage

```
varcomp(object)
```

Arguments

object	a glmssn-class object

Details

The varcomp function displays the variance component for each autocovariance function, as well as the Covariates component and the nugget.

Value

A data frame with two columns. The first column is the name of the variance component and the second column is the proportion for each variance component.

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

glmssn, covparms

Examples

```
library(SSN)
# NOT RUN
#mf04p <- importSSN(system.file("lsndata/MiddleFork04.ssn",
#  package = "SSN"), predpts = "pred1km", o.write = TRUE)
#  use SpatialStreamNetwork object mf04p that was already created
data(mf04p)
#make sure mf04p has the correct path, will vary for each users installation
mf04p@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")

# get some model fits stored as data objects
data(modelFits)
#NOT RUN use this one
```

```
#fitSp <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#    ssn.object = mf04p, EstMeth = "REML", family = "Gaussian",
#    CorModels = c("Exponential.tailup","Exponential.taildown",
#    "Exponential.Euclid"), addfunccol = "afvArea")
#make sure fitSP has the correct path, will vary for each users installation
fitSp$ssn@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")

varcomp(fitSp)
```

`writeSSN` *Write a SpatialStreamNetwork Object*

Description

The `writeSSN` function writes a SpatialStreamnetwork object to a new .ssn directory.

Usage

```
writeSSN(ssn, filename = "")
```

Arguments

`ssn`	a SpatialStreamNetwork-class object
`filename`	the file path to the new .ssn folder, in string format. When writing, include the .ssn folder in the path

Value

`writeSSN` does not return an object. Instead, it creates a new .ssn directory that contains all of the information in the specified `SpatialStreamNetwork` object.

Author(s)

Erin E. Peterson <support@SpatialStreamNetworks.com>

See Also

SpatialStreamNetwork-class

Examples

```
library(SSN)
ssn <- importSSN(system.file("lsndata/MiddleFork04.ssn", package = "SSN"),
    predpts = "pred1km")

writeSSN(ssn, filename = paste(tempdir(),"/tempFile.ssn", sep = ""))
```

Index

∗Topic **classes**
glmssn-class, 41

additive.function, 4
AIC, *5*, 5, *50*
as.SpatialLines, 6
as.SpatialPoints *(*as.SpatialLines*)*, 6
as.SpatialPointsDataFrame *(*as.SpatialLines*)*, 6

binomialDesign, *17*
binomialDesign *(*Design functions*)*, 22
binSp, 8
BlockPredict, 8, *29*
BLUP, 10
boxplot, *12*
boxplot.SpatialStreamNetwork, 11
boxplot.stats, *12*

covparms, 13, *85*
createDistMat, 14, *34*, *47*
createSSN, *16*, 17, *23*, *24*, *47*
CrossValidationSSN, 19, *22*
CrossValidationStatsSSN, 21, *50*

data.frame, *33*
Design functions, 22
design functions, *17*

EmpiricalSemivariogram, 24

fitNS, 26
fitRE, 27
fitSimBin, 27
fitSimGau, 28
fitSimPoi, 28
fitSp, 29
fitSpBk, 29
fitSpRE1, 30
fitSpRE2, 30
formula, *38*, *70*

getPreds, 31
getSSNdata.frame, 32, *66*, *70*
getStreamDistMat, 34
glmssn, *5*, *6*, *8*, *13*, *16*, *19*, *22*, *26–30*, 36, *38*, *43*, *44*, *48–50*, *63–65*, *67*, *80*, *81*, *85*
glmssn-class, *8*, *10*, *13*, *19*, *32*, 41, *44*, *48*, *49*, *68*, *85*
GR2, 43

hardCoreDesign, *17*
hardCoreDesign *(*Design functions*)*, 22

igraph, *17*, *18*
igraphKamadaKawai *(*createSSN*)*, 17
importPredpts, 44, *47*
importSSN, *16*, *18*, *45*, 46, *51*
influenceSSN, *58*
influenceSSN-class, *25*, *31*, *32*, 48, *57*, *82*
InfoCritCompare, *22*, *49*, 49
iterativeTreeLayout *(*createSSN*)*, 17

Line-class, *77*
Lines, *77*
Lines-class, *77*
list, *35*
lm, *37*, *70*

mf04, 51
mf04p, 51
MiddleFork04.ssn, *51*, 52
model.matrix, *38*, *70*

noPoints *(*Design functions*)*, 22

palette, *55*, *57*, *59*
par, *55*, *56*, *58*, *60*
plot, *60*, *62*, *83*
plot.glmssn.predict, 55
plot.influenceSSN, 57
plot.SpatialStreamNetwork, *58*, 59
plot.Torgegram, 61

poiSp, 63
poissonDesign, *17*
poissonDesign *(*Design functions*)*, 22
predict, *31*, *56*
predict.glmssn, *31*, 63
print.summary.glmssn, 64
putSSNdata.frame, *33*, 66, *70*

residuals, *49*, *58*
residuals.glmssn, 67

SimulateOnSSN, *18*, *27, 28*, 68
Spatial, *43*, *77*
SpatialLines, *7*, *43*
SpatialLines-class, *77*
SpatialLinesDataFrame, *77*
SpatialPoints, *7*
SpatialPointsDataFrame, *7*
SpatialStreamNetwork, *36, 37*, *41*, *43*, *46–48*, *59*, *68, 69*, *77, 78*, *80*, *84*
SpatialStreamNetwork-class, *4*, *7*, *11, 12*, *14*, *16, 17*, *25*, *32*, *34*, *44*, *47*, *66*, 76, *79, 80*, *82*, *86*
splitPredictions, 77
spplot, *7*
SSN *(*SSN-package*)*, 3
SSN-package, 3
subsetSSN, *78*, 79
summary.glmssn, *50*, *64*, 80
summary.SpatialStreamNetwork *(*plot.SpatialStreamNetwork*)*, 59
systematicDesign, *17*
systematicDesign *(*Design functions*)*, 22

Torgegram, 82

updatePath, 84

varcomp, 85

writeSSN, 86