Package ‘SSN’

May 13, 2018

Type Package

Title Spatial Modeling on Stream Networks

Version 1.1.13

Date 2018-05-12

Depends R (>= 3.4.0), RSQLite (>= 1.1-2), sp

Imports MASS, igraph (>= 1.0.0), maptools, lattice, methods, Matrix, rgdal (>= 1.2-5), rgeos (>= 0.3-22)

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

Description Spatial statistical modeling and prediction for data on stream networks, including models based on in-stream distance (Ver Hoef, J.M. and Peterson, E.E., 2010. <DOI:10.1198/jasa.2009.ap08248>.) Models are created using moving average constructions. Spatial linear models, including explanatory variables, can be fit with (restricted) maximum likelihood. Mapping and other graphical functions are included.

License GPL-2

LazyLoad yes

LinkingTo BH

NeedsCompilation yes

Author Jay Ver Hoef [aut, cre], Erin Peterson [aut]

Repository CRAN

Date/Publication 2018-05-12 22:53:05 UTC

R topics documented:

SSN-package ................................................................. 3
additive.function ....................................................... 4
AIC ................................................................. 5
as.SpatialLines ......................................................... 6
binSp ............................................................... 8
BlockPredict .......................................................... 9
BLUP ............................................................... 10
R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>boxplot.SpatialStreamNetwork</td>
<td>12</td>
</tr>
<tr>
<td>copyLSN2temp</td>
<td>14</td>
</tr>
<tr>
<td>covparms</td>
<td>15</td>
</tr>
<tr>
<td>createDistMat</td>
<td>16</td>
</tr>
<tr>
<td>createSSN</td>
<td>18</td>
</tr>
<tr>
<td>CrossValidationSSN</td>
<td>21</td>
</tr>
<tr>
<td>CrossValidationStatsSSN</td>
<td>23</td>
</tr>
<tr>
<td>Design functions</td>
<td>25</td>
</tr>
<tr>
<td>EmpiricalSemivariogram</td>
<td>26</td>
</tr>
<tr>
<td>fitNS</td>
<td>29</td>
</tr>
<tr>
<td>fitRE</td>
<td>29</td>
</tr>
<tr>
<td>fitSimBin</td>
<td>30</td>
</tr>
<tr>
<td>fitSimGau</td>
<td>30</td>
</tr>
<tr>
<td>fitSimPoi</td>
<td>31</td>
</tr>
<tr>
<td>fitSp</td>
<td>31</td>
</tr>
<tr>
<td>fitSpBk</td>
<td>32</td>
</tr>
<tr>
<td>fitSpRE1</td>
<td>32</td>
</tr>
<tr>
<td>fitSpRE2</td>
<td>33</td>
</tr>
<tr>
<td>getConfusionMat</td>
<td>33</td>
</tr>
<tr>
<td>getPreds</td>
<td>34</td>
</tr>
<tr>
<td>getSSNdata.frame</td>
<td>36</td>
</tr>
<tr>
<td>getStreamDistMat</td>
<td>38</td>
</tr>
<tr>
<td>glmssn</td>
<td>40</td>
</tr>
<tr>
<td>glmssn-class</td>
<td>45</td>
</tr>
<tr>
<td>GR2</td>
<td>47</td>
</tr>
<tr>
<td>importPredpts</td>
<td>49</td>
</tr>
<tr>
<td>importSSN</td>
<td>51</td>
</tr>
<tr>
<td>influenceSSN-class</td>
<td>52</td>
</tr>
<tr>
<td>InfoCritCompare</td>
<td>53</td>
</tr>
<tr>
<td>mf04</td>
<td>55</td>
</tr>
<tr>
<td>mf04p</td>
<td>56</td>
</tr>
<tr>
<td>MiddleFork04.ssn</td>
<td>57</td>
</tr>
<tr>
<td>plot glmssn.predict</td>
<td>60</td>
</tr>
<tr>
<td>plot.influenceSSN</td>
<td>62</td>
</tr>
<tr>
<td>plot.SpatialStreamNetwork</td>
<td>64</td>
</tr>
<tr>
<td>plot.Torgegram</td>
<td>66</td>
</tr>
<tr>
<td>poiSp</td>
<td>68</td>
</tr>
<tr>
<td>predict glmssn</td>
<td>68</td>
</tr>
<tr>
<td>print summary glmssn</td>
<td>70</td>
</tr>
<tr>
<td>putSSNdata.frame</td>
<td>71</td>
</tr>
<tr>
<td>residuals glmssn</td>
<td>73</td>
</tr>
<tr>
<td>SimulateOnSSN</td>
<td>75</td>
</tr>
<tr>
<td>SpatialStreamNetwork-class</td>
<td>82</td>
</tr>
<tr>
<td>splitPredictions</td>
<td>84</td>
</tr>
<tr>
<td>subsetSSN</td>
<td>85</td>
</tr>
<tr>
<td>summary glmssn</td>
<td>87</td>
</tr>
<tr>
<td>Torgegram</td>
<td>88</td>
</tr>
<tr>
<td>updatePath</td>
<td>90</td>
</tr>
</tbody>
</table>
The SSN package provides tools to fit generalized linear models with spatial autocorrelation for stream network data. SSN uses normal likelihood methods (including REML) and quasi-likelihood for Poisson and Binomial families. The spatial formulation was originally described in Ver Hoef, Peterson, and Theobald (2006), with more details given by 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. A custom ArcGIS toolbox has been provided to format the data: Spatial Tools for the Analysis of River Systems (STARS) toolset (Peterson and Ver Hoef, 2014), and the SSN package is fully described in Ver Hoef, Peterson, Clifford, and Shah (2014).

**Author(s)**

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

**References**


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


### Examples

```r
library(SSN)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()

# NOT RUN
# Create a SpatialStreamNetwork object that also contains prediction sites
# mf04p <- importSSN(paste0(tempdir(),'MiddleFork04.ssn'),
#                    predxpts = "pred1km", overwrite = TRUE)
# use mf04p SpatialStreamNetwork object, already created
data(mf04p)
# for examples only, make sure mf04p has the correct path
# if you use importSSN(), path will be correct
mf04p <- updatePath(mf04p, paste0(tempdir(),'MiddleFork04.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"))
```

### Description

AIC.glmsn is a method that calculates AIC for fitted glmsn objects.

### Usage

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

### Arguments

- **object**: an object of class `glmsn`
- **...**: 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 glmsn objects.
Value

a numeric AIC value for the specified glmssn object

Author(s)

Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also

glmssn

Examples

```r
library(SSN)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetork object that also contains prediction sites
mf04p <- importSSN(paste0(tempdir(),'MiddleFork04.ssn'),
# predpts = "pred1km", o.write = TRUE)
# use mf04p SpatialStreamNetwork object, already created
data(mf04p)
# for examples only, make sure mf04p has the correct path
#if you use importSSN(), path will be correct
mf04p <- updatePath(mf04p, paste0(tempdir(),'MiddleFork04.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 = "afArea"
# for examples only, make sure fitSp has the correct path
#if you use importSSN(), path will be correct
fitSp$ssn.object <- updatePath(fitSp$ssn.object,
paste0(tempdir(),'MiddleFork04.ssn'))

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

Description

Converts SpatialStreamNetwork objects to sp objects.

Usage

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

Arguments

- `x` an SpatialStreamNetwork object to be converted to class SpatialLines, SpatialPoints, SpatialLinesDataFrame 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

```r
library(SSN)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetwork object that also contains prediction sites
```
#MF04p <- importSSN(paste0(tempdir(), '/MiddleFork04.ssn'),
# predpts = "pred1km", o.write = TRUE)
# use MF04p SpatialStreamNetwork object, already created
data(MF04p)
#if for examples only, make sure MF04p has the correct path
#if you use importSSN(), path will be correct
MF04p <- updatePath(MF04p, paste0(tempdir(), '/MiddleFork04.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 Prediction 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**


**Examples**

```r
## Not run:
library(SSN)
#for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
```
# Create a SpatialStreamNetwork object that also contains prediction sites
mf04p <- importSSN(paste0(tempdir(), '/MiddleFork04.ssn'),
                    predpts = "pred1km", o.write = TRUE)
use mf04p SpatialStreamNetwork object, already created
data(mf04p)
# for examples only, make sure mf04p has the correct path
# if you use importSSN(), path will be correct
mf04p <- updatePath(mf04p, paste0(tempdir(), '/MiddleFork04.ssn'))

# Not needed: already added,
# add densely gridded prediction points for two stream segments
mf04p <- importPredpts(mf04p, "Knapp", "ssn")
mf04p <- importPredpts(mf04p, "CapeHorn", "ssn")
names(mf04p)

# see densely gridded prediction points on stream
plot(mf04p, PredPointsID = "Knapp")

# you could fit the model
fitSpBk <- glmssn( Summer.mn ~ ELEV_DEM + netID,
                    ssn.object = mf04p, EstMeth = "REML", family = "Gaussian",
                    CorModels = c("Exponential.tailup","Exponential.taildown",
                                  "Exponential.Euclid"), addfunccol = "afVArea")

# or load the pre-fit model
data(modelfits)
fitSpBk$ssn.object <- updatePath(fitSpBk$ssn.object,
paste0(tempdir(), '/MiddleFork04.ssn'))

# one-at-a-time predictions for CapeHorn stream
## NOTE: need the amongpreds distance matrices for block prediction
createDistMat(mf04p, predpts = "CapeHorn", o.write = TRUE, amongpreds = TRUE)
fitSpPredC <- predict(fitSpBk, "CapeHorn")
# plot densely gridded prediction points on stream
plot(fitSpPredC, "Summer.mn")
# block prediction for CapeHorn stream
BlockPredict(fitSpBk, "CapeHorn")

## Another example
# one-at-a-time predictions for Knapp stream
createDistMat(mf04p, predpts = "Knapp", o.write = TRUE, amongpreds = TRUE)
fitSpPredK <- predict(fitSpBk, "Knapp")
# plot densely gridded prediction points on stream
plot(fitSpPredK, "Summer.mn")
# block prediction for Knapp stream
BlockPredict(fitSpBk, "Knapp")

## End(Not run)
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 \( \sigma^2 \cdot \text{Identity} \). We just plug in the REML estimate of \( \sigma^2 \) from the fitted glmssn model object.

Usage

\[
\text{BLUP}(\text{model}, \ RE = \text{NULL})
\]

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

```r
library(SSN)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetork object that also contains prediction sites
# mf04p <- importSSN(paste0(tempdir(),'/MiddleFork04.ssn'),
# # predpts = "pred1km", o.write = TRUE)
# use mf04p SpatialStreamNetwork object, already created
data(mf04p)
# for examples only, make sure mf04p has the correct path
# if you use importSSN(), path will be correct
mf04p <- updatePath(mf04p, paste0(tempdir(),'/MiddleFork04.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"))
# for examples only, make sure fitRE has the correct path
# if you use importSSN(), path will be correct
fitRE$ssn.object <- updatePath(fitRE$ssn.object, 
paste0(tempdir(), '/MiddleFork04.ssn'))

summary(fitRE)
# random effects details
fitRE$BLUP <- BLUP(fitRE)
str(fitRE$BLUP)
fitRE$BLUP$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")
# for examples only, make sure fitRE has the correct path
# if you use importSSN(), path will be correct
fitSpRE1$ssn.object <- updatePath(fitSpRE1$ssn.object, 
paste0(tempdir(), '/MiddleFork04.ssn'))

fitSpRE1$BLUP <- BLUP(fitSpRE1)
str(fitSpRE1$BLUP)
fitSpRE1$BLUP$Mean

### boxplot.SpatialStreamNetwork

`boxplot.SpatialStreamNetwork`  
*

#### Description

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

#### Usage

```r
## 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.SpatialStreamNetwork` is a generic function that has been adapted for `SpatialStreamNetwork-class` objects. 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

```r
library(SSN)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetork object that also contains prediction sites
```
copyLSN2temp

#mf04 <- importSSN(paste0(tempdir(), '/MiddleFork04.ssn'),
#  predpts = "pred1km", o.write = TRUE)
#use mf04 SpatialStreamNetwork object, already created
data(mf04)
#if for examples only, make sure mf04's path will be correct
mf04 <- updatePath(mf04, paste0(tempdir(), '/MiddleFork04.ssn'))

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

---

**copyLSN2temp**  
 Copies the lsn directory MiddleFork04.ssn to R's temporary directory

**Description**

Copies the lsn directory MiddleFork04.ssn to R's temporary directory so the examples do not write to library or any other places

**Usage**

```r
library(SSN)
copyLSN2temp()
```

**Details**

Copies the lsn directory MiddleFork04.ssn to R's temporary directory

**Value**

MiddleFork04.ssn on R's temporary directory

**Author(s)**

Jay Ver Hoef <support@SpatialStreamNetworks.com>

**Examples**

```r
library(SSN)
copyLSN2temp()
#getwd()
#setwd(tempdir())
#getwd()
#if unix-alike use: system('ls')
#if windows use: system('dir')```
**covparms**  

Get Covariance Parameters

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

**Usage**  
\[ \text{covparms}(\text{object}) \]

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

**Details**  
A `glmssn-class` object 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**

```r
library(SSN)  
# for examples, copy MiddleFork04.ssn directory to R's temporary directory  
copyLSN2temp()  
# NOT RUN  
# Create a SpatialStreamNetork object that also contains prediction sites  
# mf04p <- importSSN(paste0(tempdir(),'MiddleFork04.ssn'),  
# predpts = "pred1km", o.write = TRUE)  
# use mf04p SpatialStreamNetwork object, already created  
data(mf04p)  
# for examples only, make sure mf04p has the correct path  
# if you use importSSN(), path will be correct  
mf04p <- updatePath(mf04p, paste0(tempdir(),'MiddleFork04.ssn')))```
## createDistMat

### Description

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

### Usage

```r
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 sites.
(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.
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.
createSSN

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. 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 initial 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
#set.seed(5)
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 pair 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,
CrossValidationSSN

# 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)

ssn1@obspoints@SSNPoints[[1]]@point.data
ssn5@obspoints@SSNPoints[[1]]@point.data

---

CrossValidationSSN

**Compute Crossvalidation Values for glmsnn Objects**

**Description**

CrossValidationSSN operates on glmsnn 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 glmsnn-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 three columns, the point identifier "pid", predictions "cv.pred", and their standard errors "cv.se". 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)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetork object that also contains prediction sites
mf04 <- importSSN(paste0(tempdir(),'/MiddleFork04.ssn', o.write = TRUE))
# use mf04 SpatialStreamNetwork object, already created
data(mf04)
# for examples only, make sure mf04p has the correct path
# if you use importSSN(), path will be correct
mf04 <- updatePath(mf04, paste0(tempdir(),'/MiddleFork04.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.taulup","Exponential.taildown", 
# "Exponential.Euclid"), addfunccol = "afVArea")
#for examples only, make sure fitSp has the correct path
# if you use importSSN(), path will be correct
fitSp$ssn.object <- updatePath(fitSp$ssn.object, 
paste0(tempdir(),'/MiddleFork04.ssn'))

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$sampinfo$x, 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**

```r
library(SSN)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetork object that also contains prediction sites
# mf04 <- importSSN(paste0(tempdir(),'\MiddleFork04.ssn', o.write = TRUE))
# use mf04 SpatialStreamNetwork object, already created
data(mf04)
# for examples only, make sure mf04p has the correct path
# if you use importSSN(), path will be correct
mf04 <- updatePath(mf04, paste0(tempdir(),'\MiddleFork04.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.tailldown",
#    "Exponential.Euclid"), addfunccol = "afvArea")
#for examples only, make sure fitSp has the correct path
#if you use importSSN(), path will be correct
fitSp$ssn.object <- updatePath(fitSp$ssn.object,
paste0(tempdir(), '/MiddleFork04.ssn'))

CrossValidationStatsSSN(fitSp)
```
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)

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 certain 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

\[
\text{function}(\text{tree.graphs}, \text{edge.lengths}, \text{locations}, \text{edge.updist}, \text{distance.matrices})
\]

All inputs to the design function are lists of length \(n\) where \(n\) is the number of trees. Input \(\text{tree.graphs}[[i]]\) is an object of class \(\text{igraph}\) which represent the \(i\)th generated network in a graph theoretic sense; without any specific locations assigned to the vertices. \(\text{edge.lengths}[[i]]\) contains the lengths of the edges for the \(i\)th tree, in the same order as the edges appear in the corresponding \(\text{igraph}\) object. Input \(\text{locations}[[i]]\) is a matrix with \(n[i]\) rows and 2 columns giving the locations of the points on that network. \(\text{edge.updist}[[i]]\) 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 \(\text{igraph}\) object. \(\text{distance.matrices}[[i]]\) 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 \(\text{igraph}\) object. To summarise, on tree number \(i\) if edge number \(k\) has downstream point \(k_\) and edge number \(l\) has downstream point \(l_\), then \(\text{edge.lengths}[[i]][k]\) gives the length of edge number \(k\), \(\text{edge.updist}[[i]][k]\) gives the distance from point \(k_\) to the outlet of the stream network, and \(\text{distance.matrices}[[i]][k, l]\) 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
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)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetork object that also contains prediction sites
#mf04 <- importSSN(paste0(tempdir(),'/MiddleFork04.ssn', o.write = TRUE))
#use mf04 SpatialStreamNetwork object, already created
data(mf04)
#for examples only, make sure mf04p has the correct path
#if you use importSSN(), path will be correct
mf04 <- updatePath(mf04, paste0(tempdir(),'/MiddleFork04.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)
#for examples only, make sure fitSP has the correct path
#if you use importSSN(), path will be correct
fitSp$ssn.object <- updatePath(fitSp$ssn.object,
paste0(tempdir(),'/MiddleFork04.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 MiddleFork.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**

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


### fitRE

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

**Description**

The MiddleFork.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**

```r
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**

```r
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**

```r
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**

```r
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**

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

```r
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()

getConfusionMat

Extract confusion matrix for binary spatial model predictions.

Description

This function extracts the binary confusion matrix from 'influenceSSN' objects. The matrix values are based on a leave-one-out cross-validation of the observed dataset.

Usage

getConfusionMat(x, threshold = 0.5)

Arguments

x an object of class influenceSSN-class
threshold a numeric value used to classify binary spatial model predictions into 1’s and 0’s. The default is 0.5.

Value

getConfusionMat returns a 2x2 matrix containing information about the classification accuracy of the binary spatial model, based on the observations and the leave-one-out cross-validation predictions.

Author(s)

Erin E. Peterson <support@SpatialStreamNetworks.com>
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.
getPreds

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

```r
library(SSN)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetork object that also contains prediction sites
#mf04p <- importSSN(paste0(tempdir(), '/MiddleFork04.ssn'),
#   predpts = "pred1km", o.write = TRUE)
#use mf04p SpatialStreamNetwork object, already created
data(mf04p)
# for examples only, make sure mf04p has the correct path
#if you use importSSN(), path will be correct
#mf04p <- updatePath(mf04p, paste0(tempdir(), '/MiddleFork04.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,
   paste0(tempdir(),'/MiddleFork04.ssn'))
```
# Extract predictions and standard errors for the prediction sites
# make sure the distance matrix is there
createDistMat(mf84p, predpts = "pred1km", o.write = TRUE)

# create predictions
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**

```r
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.
getSSNdata.frame

Value
A data.frame.

Author(s)
Jay Ver Hoef <support@SpatialStreamNetworks.com>

See Also
putSSNdata.frame

Examples
library(hydroclust)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetwork object that also contains prediction sites
#mf04 <- importSSN(paste0(tempdir(),'/MiddleFork04.ssn', o.write = TRUE))
#use mf04 SpatialStreamNetwork object, already created
data(mf04)
# for examples only, make sure mf04 has the correct path
#if you use importSSN(), path will be correct
mf04 <- updatePath(mf04, paste0(tempdir(),'/MiddleFork04.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_DNM + netID,
#    ssn.object = mf04p, EstMeth = "REML", family = "Gaussian",
#    CorModels = c("Exponential.tailup","Exponential.taildown",
#    "Exponential.Euclid"), addfunccol = "afvArea")
# for examples only, make sure fitSp has the correct path
#if you use importSSN(), path will be correct
fitSp$ssn.object <- updatePath(fitSp$ssn.object,
paste0(tempdir(),'/MiddleFork04.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)
The `getStreamDistMat` function extracts the stream network distance matrix for either observation data or prediction data, from the specified `SpatialStreamNetwork` object.

Usage

```r
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

```r
library(SSN)
# For examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetwork object that also contains prediction sites
# mf04p <- importSSN(paste0(tempdir(), '/MiddleFork04.ssn'),
# predpts = "pred1km", o.write = TRUE)  
# use mf04p SpatialStreamNetwork object, already created
data(mf04p)
# for examples only, make sure mf04p has the correct path
# if you use importSSN(), path will be correct
mf04p <- updatePath(mf04p, paste0(tempdir(), '/MiddleFork04.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]

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

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 net-
work. This contains the variables used in the model.

family the error distribution and link function to be used in the model. This is a charac-
ter 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"; tail-
down models include: "Exponential.taildown" (default), "LinearSill.taildown",
"Spherical.taildown", "Mariah.taildown", "Epanech.taildown"; Euclidean dis-
tance models include: "Spherical.Euclid", "Gaussian.Euclid", "Exponential.Euclid"
(default), "Cauchy.Euclid". The first 4 tailup and tailldown 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 mod-
els 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 anisotropy 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 transformation

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 \texttt{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 \texttt{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 \texttt{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:

```r
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
V_i = V_i[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


Examples

library(SSN)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetwork object that also contains prediction sites
#mf04p <- importSSN(paste0(tempdir(),'/MiddleFork04.ssn'),
# predpts = "pred1km", o.write = TRUE)

#use mf04p SpatialStreamNetwork object, already created
data(mf04p)
# for examples only, make sure mf04p has the correct path
#if you use importSSN(), path will be correct
mf04p <- updatePath(mf04p, paste0(tempdir(),'/MiddleFork04.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")
# for examples only, make sure fitNS has the correct path
if you use importSSN(), path will be correct
fitNS$ssn.object <- updatePath(fitNS$ssn.object,
paste0(tempdir(),'/MiddleFork04.ssn'))

summary(fitNS)

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"))
# for examples only, make sure fitRE has the correct path
# if you use importSSN(), path will be correct
fitRE$ssn.object <- updatePath(fitRE$ssn.object,
paste0(tempdir(),'/MiddleFork04.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"))
# for examples only, make sure fitSpRE1 has the correct path
# if you use importSSN(), path will be correct
fitSpRE1$ssn.object <- updatePath(fitSpRE1$ssn.object,
paste0(tempdir(),'/MiddleFork04.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 = "afArea")
# for examples only, make sure fitSpRE2 has the correct path
# if you use importSSN(), path will be correct
fitSpRE2$ssn.object <- updatePath(fitSpRE2$ssn.object,
paste0(tempdir(),'/MiddleFork04.ssn'))

summary(fitSpRE2)

## 3 component spatial model
# glmssn-class

Class "glmssn"

```r
#fitSp <- glmssn(Summer_mn ~ ELEV_DEM + netID,
#  ssn.object = mf04p, EstMeth = "REML", family = "Gaussian",
#  CorModels = c("Exponential.tailup", "Exponential.taildown",
#  "Exponential.Euclid"), addfunccol = "afvArea")
#for examples only, make sure fitSp has the correct path
#if you use importSSN(), path will be correct
fitSp$ssn.object <- updatePath(fitSp$ssn.object,
paste0(tempdir(), '/MiddleFork04.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")
#for examples only, make sure binSp has the correct path
#if you use importSSN(), path will be correct
binSp$ssn.object <- updatePath(binSp$ssn.object,
paste0(tempdir(), '/MiddleFork04.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")
#for examples only, make sure poiSp has the correct path
#if you use importSSN(), path will be correct
poiSp$ssn.object <- updatePath(poiSp$ssn.object,
paste0(tempdir(), '/MiddleFork04.ssn'))
summary(poiSp)
```
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:

```r
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,
    cutXItoX2,
    RES,
    REModelmatrices,
  ),
  estimates = list(  # Estimated covariance parameters
    theta,
    nugget,
    )
)
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)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetork object that also contains prediction sites
# mf04p <- importSSN(paste0(tempdir(),'/MiddleFork04.ssn'),
# # predpts = "pred1km", o.write = TRUE)
# use mf04p SpatialStreamNetwork object, already created
data(mf04p)
# for examples only, make sure mf04p has the correct path
# if you use importSSN(), path will be correct
mf04p <- updatePath(mf04p, paste0(tempdir(),'/MiddleFork04.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")
# for examples only, make sure fitSp has the correct path
# if you use importSSN(), path will be correct
fitSp$ssn.object <- updatePath(fitSp$ssn.object,
paste0(tempdir(),'/MiddleFork04.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 pred-points slot is populated in the object

Author(s)
Erin E. Peterson <support@SpatialStreamNetworks.com>

References

See Also
importSSN, SpatialStreamNetwork-class, and glmssn-class
Examples

library(SSN)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()

# NOT RUN
# Create a SpatialStreamNetwork object that also contains prediction sites
mf04 <- importSSN(paste0(tempdir(),'/MiddleFork04.ssn', o.write = TRUE))
# use mf04 SpatialStreamNetwork object, already created
data(mf04)
# for examples only, make sure mf04p has the correct path
# if you use importSSN(), path will be correct
mf04p <- updatePath(mf04, paste0(tempdir(),'/MiddleFork04.ssn'))

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 glmsn object
# use models that have been created already
data(modelfits)
# for examples only, make sure fitSp has the correct path
# if you use importSSN(), path will be correct
fitSp$ssn.object <- updatePath(fitSp$ssn.object, paste0(tempdir(),'/MiddleFork04.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")
importSSN

#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 overwritten 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


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)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()

# Create a SpatialStreamNetwork object that does not contain prediction points
mf04 <- importSSN(paste0(tempdir(),'/MiddleFork04.ssn'), o.write = TRUE)

# Create a SpatialStreamNetwork object that also contains prediction sites
mf04p <- importSSN(paste0(tempdir(),'/MiddleFork04.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.crossov_** ## 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

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 \( \sqrt{\text{mean}((\text{obs} - \text{pred})^2)} \)
RAV  root average variance, computed as \( \sqrt{\text{mean}(\text{predSE}^2)} \). If the prediction standard errors are being estimated well, this should be close to RMSPE.
std.MSPE  standardized mean-squared prediction error, computed as \( \text{mean}((\text{obs} - \text{pred})/\text{predSE})^2 \). 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 \( \text{pred} \pm qt(.9, df)*\text{predSE} \), 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 \( \text{pred} \pm qt(.95, df)*\text{predSE} \), 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 \( \text{pred} \pm qt(.975, df)*\text{predSE} \), 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)
# for examples only, make sure all models have the correct path
# if you use importSSN(), path will be correct
fitNS$ssn.object <- updatePath(fitNS$ssn.object, 
paste0(tempdir(), '/MiddleFork04.ssn'))
fitRE$ssn.object <- updatePath(fitRE$ssn.object, 
paste0(tempdir(), '/MiddleFork04.ssn'))
fitSp$ssn.object <- updatePath(fitSp$ssn.object, 
paste0(tempdir(), '/MiddleFork04.ssn'))
fitSpRE1$ssn.object <- updatePath(fitSpRE1$ssn.object, 
paste0(tempdir(), '/MiddleFork04.ssn'))
fitSpRE2$ssn.object <- updatePath(fitSpRE2$ssn.object, 
paste0(tempdir(), '/MiddleFork04.ssn'))

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

```r
library(SSN)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
mf04 < - importSSN(paste0(tempdir(), '/MiddleFork04.ssn'), o.write = TRUE)
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

```r
library(SSN)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
mf04p < - importSSN(paste0(tempdir(), '/MiddleFork04.ssn'), predpts = "pred1km",
  o.write = TRUE)
mf04p < - importPredpts(target = mf04p, predpts = "CapeHorn", obj.type = "ssn")
mf04p < - importPredpts(target = mf04p, predpts = "Knapp", obj.type = "ssn")
names(mf04p)
```
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 identifier 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 stream network) to the uppermost location on the line segment
MiddleFork04.ssn

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 temperature of the deployment
MaxOver20: Binary variable: 1 represents the maximum summer temperature was greater than 20°C and 0 indicates that it was less than 20°C
C16: the number of times daily stream temperature exceeded 16°C
C20: the number of times daily stream temperature exceeded 20°C
C24: the number of times daily stream temperature exceeded 24°C
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
AirMMTc: Average maximum air temperature (°C) from July 15 - August 31, from 1980-2009 across 10 COOP air stations within the domain. MM = 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 watershed 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
ELEVDEM: 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 (h2AreaKm2)
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 1980-2000 across 10 COOP air stations within the domain
AirMWMTc: Average maximum air temperature (C) from July 15 - August 31, from 1980-2000 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)
#for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
mf04p <- importSSN(paste0(tempdir(),'/MiddleFork04.ssn'), predpts = "pred1km", o.write = TRUE)
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)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetwork object that also contains prediction sites
#mf04p <- importSSN(paste0(tempdir(),'/MiddleFork04.ssn'),
# predpts = "pred1km", o.write = TRUE)
#use mf04p SpatialStreamNetwork object, already created
data(mf04p)
# for examples only, make sure mf04p has the correct path
#if you use importSSN(), path will be correct
mf04p <- updatePath(mf04p, paste0(tempdir(),'/MiddleFork04.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")
# for examples only, make sure fitSp has the correct path
# if you use importSSN(), path will be correct
fitSp$ssn.object <- updatePath(fitSp$ssn.object,
paste0(tempdir(),'\MiddleFork04.ssn'))

# NOT RUN
# make sure the distance matrix is there
# createDistMat(mf04p, predpts = "pred1km", o.write = TRUE)

## 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)
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

```r
## 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.
- `colorpalette`: 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

# get some model fits stored as data objects
data(modelFits)
# NOT RUN use this one
# fitSp <- glmssn(Summer mn ~ ELEV DEM + netID,
# sn.object = mf04p, EstMeth = "REML", family = "Gaussian",
# CorModels = c("Exponential.tailup","Exponential.taildown",
# "Exponential.Euclid"), addfunccol = "afvArea")
# for examples only, make sure fitSp has the correct path
# if you use importSSN(), path will be correct
fitsp$sn.object <- updatePath(fitsp$sn.object,
paste0(tempdir(), '/MiddleFork04.ssn'))

resids <- residuals(fitSp,cross.validation = TRUE)
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_")
Arguments

- **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)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetork object that also contains prediction sites
mf04p <- importSSN(paste0(tempdir(),'/MiddleFork04.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 allows 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 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 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**

```r
library(SSN)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
```
# NOT RUN
# Create a SpatialStreamNetwork object that also contains prediction sites
mf04p <- importSSN(paste0(tempdir(), '/MiddleFork04.ssn'),
# predpts = "pred1km", o.write = TRUE)
#use mf04p SpatialStreamNetwork object, already created
data(mf04p)
#for examples only, make sure mf04p has the correct path
#if you use importSSN(), path will be correct
mf04p <- updatePath(mf04p, paste0(tempdir(), '/MiddleFork04.ssn'))

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

ESVF <- Torgegram(mf04p, "Summer_mm", 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
#make sure fitSp has the correct path, will vary for each users installation
fitSp$snn.object@path <- system.file("lsndata/MiddleFork04.ssn", package = "SSN")
resids <- residuals(fitSp)
names(resids$snn.object)
ESVF <- Torgegram(resids, ".resid_", maxlag = 20000,
nlag = 10)
plot(ESVF, xlim = c(0,10000))
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

```r
## 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)

# get some model fits stored as data objects
data(modelfits)
#NOT RUN use this one
#fitSp <- glmsn(Summer mn ~ ELEV DEM + netID,
#ssn.object = mf04p, EstMeth = "REML", family = "Gaussian",
#CorModels = c("Exponential.tailup","Exponential.taildown",
#"Exponential.Euclid"), addfunccol = "afVArea")
#for examples only, make sure fitSp has the correct path
#if you use importSSN(), path will be correct
fitSp$ssn.object <- updatePath(fitSp$ssn.object,
paste0(tempdir(),’/MiddleFork04.ssn'))

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

print.summary.glmsn  Print summary - S3 Method for Class 'glmsn'

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.glmsn.

Usage

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

Arguments

x an object of class summary.glmsn
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.glmsn function summarizes and prints the fitted model with a table of esti-
mates for the fixed effects and the covariance parameter estimates.
putSSNdata.frame

**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**

- glmsn, link{covparms}

**Examples**

```r
library(SSN)
data(modelFits)
# for examples only, make sure fitSp has the correct path
# if you use importSSN(), path will be correct
fitSp$ssn.object <- updatePath(fitSp$ssn.object, paste0(tempdir(),'/MiddleFork04.ssn'))
names(fitSp)
names(fitSp$ssn.object)

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

---

**Description**

Replace 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)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetwork object that also contains prediction sites
#mf04 <- importSSN(paste0(tempdir(),'/MiddleFork04.ssn', o.write = TRUE))
#use mf04 SpatialStreamNetwork object, already created
data(mf04)
#for examples only, make sure mf04p has the correct path
#if you use importSSN(), path will be correct
mf04 <- updatePath(mf04, paste0(tempdir(),'/MiddleFork04.ssn'))

# 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)
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=FALSE, ...)

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 FALSE. Setting cross.validation to TRUE may increase 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 \( \text{var}(e) = sV \), \( C \) is the square root inverse of \( V \), and \( s \) is an overall variance parameter. Under such a model, the hat matrix is \( H = CX^\ast\text{inv}(X'\text{C}'C)X'X'\text{C}' \). 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-class` object; 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

```r
library(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")
#for examples only, make sure fitSp has the correct path
#if you use importSSN(), path will be correct
fitSp$ssn.object <- updatePath(fitSp$ssn.object,
paste0(tempdir(),'/MiddleFork04.ssn'))

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

```r
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 `i`th 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 other models.

CorParms

- `CorParms` a numeric vector representing the parameters of the spatial autocorrelation model. It contains the regression parameters of the exponential model: `CorParms = c(1, 10000, 1, 10000, 1, 10000, 0.1)`.
SimulateOnSSN

"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.

addfunc[col 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.

correlation 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 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 following order, if specified in CorParms: tailup model (partial sill then range), taildown (partial 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


Examples

```
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
```
SimulateOnSSN

```r
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,
```
SimulateOnSSN

```r
# 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 R: binomial data, 1 network
#############################################################################
## NOT RUN takes about 10 seconds
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 Binomial data
#sim2.out <- SimulateOnSSN(raw2.ssn,
```
# ObsSimDF = raw2DFobs,
# PredSimDF = raw2DFpred,
# PredID = "peds",
# 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, 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")
## Simulate On SSN

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

# example S: 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")
#raw3Df pred <- getSSNdata.frame(raw3.ssn, "preds")

## add a continuous covariate randomly
#raw3Dfobs[,"X1"] <- rnorm(length(raw3Dfobs[,1]))
#raw3Df pred[,"X1"] <- rnorm(length(raw3Df pred[,1]))

## add a categorical covariate randomly
#raw3Dfobs[,"F1"] <- as.factor(sample.int(3, length(raw3Dfobs[,1]), replace = TRUE))
#raw3Df pred[,"F1"] <- as.factor(sample.int(3, length(raw3Df pred[,1]), replace = TRUE))

## simulate Poisson data
#sim3.out <- SimulateOnSSN(raw3.ssn,
#   # ObsSimDF = raw3Dfobs,
#   # PredSimDF = raw3Df pred,
#   # 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, 0.81),
#   # 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
# sim3DFpred <- 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 = "grey80")
# plot(pred3.ssn, color.palet = 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

splitPredictions

**Split Prediction Sets in a SpatialStreamNetwork Object**

**Description**

The `splitPredictions` 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**

```r
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 set based 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.
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 ex-
          port 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)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# Create a SpatialStreamNetwork object
ssn <- importSSN(paste0(tempdir(),'/MiddleFork04.ssn'), o.write = TRUE)

ssn.sub1 <- subsetSSN(ssn, filename =
    paste(tempdir(),"/subset1.ssn", sep = ""), subset = Summer_mm > 13)

# another example
# ssn.sub2 <- subsetSSN(ssn, filename =
#    paste(tempdir(),"/subset2.ssn", sep = ""), subset = netID == 1, clip = TRUE)
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

```r
## 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)
# 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")
# for examples only, make sure fitSp has the correct path
# if you use importSSN(), path will be correct
fitSp$ssn.object <- updatePath(fitSp$ssn.object,
paste0(tempdir(),'/MiddleFork04.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`.
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

```r
library(SSN)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetwork object that also contains prediction sites
-mf04p <- importSSN(paste0(tempdir(), '/MiddleFork04.ssn'),
  predpts = "pred1km", o.write = TRUE)
# use mf04p SpatialStreamNetwork object, already created
data(mf04p)
# for examples only, make sure mf04p has the correct path
# if you use importSSN(), path will be correct
mf04p <- updatePath(mf04p, paste0(tempdir(), '/MiddleFork04.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)"
)

# 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")
# for examples only, make sure fitSp has the correct path
# if you use importSSN(), path will be correct
fitSp$ssn.object <- updatePath(fitSp$ssn.object, paste0(tempdir(), '/MiddleFork04.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**

```r
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>

**Examples**

```r
library(SSN)

# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetwork object that also contains prediction sites
mf04p <- importSSN(paste0(tempdir(), '/MiddleFork04.ssn'),
  # predpts = "pred1km", o.write = TRUE)
# use mf04p SpatialStreamNetwork object, already created
data(mf04p)
# for examples only, make sure mf04p has the correct path
#if you use importSSN(), path will be correct
mf04p <- updatePath(mf04p, paste0(tempdir(), '/MiddleFork04.ssn'))

## UPDATE PATH FOR glmssn object
## get some model fits stored as data objects
data(modelFits)
## NOT RUN
## Fit a model to binary data
## binSp <- glmssn(MaxOver20 ~ ELEV_DEM + SLOPE, mf04p,
## CorModels = c("Mariah.tailup", "Spherical.taildown"),
## family = "binomial", addfunccol = "afArea")
## for examples only, make sure binSp has the correct path
## if you use importSSN(), path will be correct
binSp$ssn.object <- updatePath(binSp$ssn.object,
```
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)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetork object that also contains prediction sites
#mf04p <- importSSN(paste0(tempdir(),'/MiddleFork04.ssn'),
# predpts = "pred1km", o.write = TRUE)
#use mf04p SpatialStreamNetwork object, already created
writeSSN

```r
data(mf04p)
# for examples only, make sure mf04p has the correct path
# if you use importSSN(), path will be correct
mf04p <- updatePath(mf04p, paste0(tempdir(), '/MiddleFork04.ssn'))

# get some model fits stored as data objects
data(modelFits)
# NOT RUN use this one
# fitSp <- glmsn(Summer_mm ~ ELEV_DEM + netID,
# ssn.object = mf04p, EstMeth = "REML", family = "Gaussian",
# CorModels = c("Exponential.tailup","Exponential.taildown",
# "Exponential.Euclid"), addfunccol = "afVArea")
# for examples only, make sure fitSp has the correct path
# if you use importSSN(), path will be correct
fitSp$ssn.object <- updatePath(fitSp$ssn.object,
paste0(tempdir(), '/MiddleFork04.ssn'))

varcomp(fitSp)
```

---

**writeSSN**

**Write a SpatialStreamNetwork Object**

**Description**

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

**Usage**

```r
writeSSN(ssn, filename = '', o.write = 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
- `o.write` If TRUE, overwrite existing ssn. Defaults to FALSE.

**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)
# for examples, copy MiddleFork04.ssn directory to R's temporary directory
copyLSN2temp()
# NOT RUN
# Create a SpatialStreamNetwork object that also contains prediction sites
mf04p <- importSSN(paste0(tempdir(), '/MiddleFork04.ssn'),
    predpts = "pred1km", o.write = TRUE)

writeSSN(mf04p, filename = paste0(tempdir(), "//tempFile.ssn"))
Index

*Topic classes
  glmssn-class, 45

additive.function, 4
AIC, 5, 5, 55
as.SpatialLines, 6
as.SpatialLinesDataFrame
  (as.SpatialLines), 6
as.SpatialPoints (as.SpatialLines), 6
as.SpatialPointsDataFrame
  (as.SpatialLines), 6

binomialDesign, 19
binomialDesign (Design functions), 25
binSp, 8
BlockPredict, 9, 32
BLUP, 10
boxplot, 13
boxplot.SpatialStreamNetwork, 12
boxplot.stats, 13

copyLSN2temp, 14
covparms, 15, 92
createDistMat, 16, 38, 52
createSSN, 18, 18, 25, 26, 52
CrossValidationSSN, 21, 24
CrossValidationStatsSSN, 23, 55
data.frame, 37
Design functions, 25
design functions, 19

EmpiricalSemivariogram, 26

fitNS, 29
fitRE, 29
fitSimBin, 30
fitSimGau, 30
fitSimPoi, 31
fitSp, 31
fitSpBk, 32

fitSpRE1, 32
fitSpRE2, 33
formula, 41, 76

getConfusionMat, 33
getPreds, 34
getSSNdata.frame, 36, 72, 76
getStreamDistMat, 38

glmssn, 5, 6, 8, 9, 15, 18, 21, 24, 29–33, 40, 42, 47, 48, 52, 53, 55, 68, 69, 71, 73, 87, 92

glmssn-class, 9, 11, 15, 21, 36, 45, 49, 53, 54, 74, 92

GR2, 47

hardCoreDesign, 19
hardCoreDesign (Design functions), 25

igraph, 19, 20
igraphKamadaKawai (createSSN), 18
importPredpts, 49, 52
importSSN, 18, 20, 49, 51, 56
influenceSSN, 63
influenceSSN-class, 27, 33, 35, 36, 52, 63, 88
InfoCritCompare, 24, 53, 53, 54
iterativeTreeLayout (createSSN), 18

Line-class, 84
Lines, 83
Lines-class, 84
list, 38
lm, 41, 76

mf04, 55
mf04p, 56
MiddleFork04.ssn, 56, 57
model.matrix, 41, 76

noPoints (Design functions), 25
palette, 60, 63, 65
par, 60, 63, 65
plot, 66, 67, 90
plot.glmssn.predict, 60
plot.influenceSSN, 62
plot.SpatialStreamNetwork, 64, 64
plot.Torgegram, 66
poiSp, 68
poissonDesign, 19
poissonDesign (Design functions), 25
predict, 34, 35, 61
predict.glmssn, 35, 69
print.summary.glmssn, 70
putSSNdata.frame, 37, 71, 76
residuals, 53, 64
residuals.glmssn, 73
SimulateOnSSN, 20, 30, 31, 75
Spatial, 47, 83
SpatialLines, 7, 47
SpatialLines-class, 83
SpatialLinesDataFrame, 7, 83
SpatialPoints, 7
SpatialPointsDataFrame, 7
SpatialStreamNetwork, 40, 41, 46, 47,
51–53, 65, 75, 84–86, 91
SpatialStreamNetwork-class, 4, 7, 12, 13,
16–18, 27, 36, 38, 49, 51, 72, 82, 86,
88, 93, 94
splitPredictions, 84
spplot, 7
SSN (SSN-package), 3
SSN-package, 3
subsetSSN, 85, 85
summary.glmssn, 55, 70, 87
summary.SpatialStreamNetwork
(plot.SpatialStreamNetwork), 64
systematicDesign, 19
systematicDesign (Design functions), 25
Torgegram, 88
updatePath, 90
varcomp, 92
writeSSN, 93