SSN: An R Package for Spatial Statistical Modeling on Stream Networks

Jay M. Ver Hoef
NOAA, Alaska Fisheries
Science Center

Erin E. Peterson
CSIRO, Division of Mathematics,
Informatics and Statistics

David Clifford
CSIRO, Division of Mathematics,
Informatics and Statistics

Rohan Shah
CSIRO, Division of Mathematics,
Informatics and Statistics

Abstract

The **SSN** package for R provides a set of functions for modeling stream network data. The package can import geographic information systems data or simulate new data as a `SpatialStreamNetwork`, a new object class that builds on the spatial **sp** classes. Functions are provided that fit spatial linear models (SLMs) for the `SpatialStreamNetwork` object. The covariance matrix of the SLMs use distance metrics and geostatistical models that are unique to stream networks; these models account for the distances and topological configuration of stream networks, including the volume and direction of flowing water. In addition, traditional models that use Euclidean distance and simple random effects are included, along with Poisson and binomial families, for a generalized linear mixed model framework. Plotting and diagnostic functions are provided. Prediction (kriging) can be performed for missing data or for a separate set of unobserved locations, or block prediction (block kriging) can be used over sets of stream segments. This article summarizes the **SSN** package for importing, simulating, and modeling of stream network data, including diagnostics and prediction.

Keywords: spatial statistics, network graphs, geostatistics, generalized linear mixed models.

1. Introduction

New spatial statistical methods were recently developed to fit models to data collected on stream (river) networks (Ver Hoef and Peterson 2010). Stream networks, in our usage, are based on a mathematical topology that represents streams as line segments that converge downstream, or viewed conversely, that create dichotomous branching when moving upstream from an outlet (the most downstream location in the network). A number of packages [1] have been written in R to fit spatial statistical models that use geostatistical autocovariance functions (based on Euclidean distance), but they are not guaranteed to produce positive-definite covariance matrices when using an alternative distance measure, such as stream distance

[1] Including, but not limited to, **geoR** (Ribeiro and Diggle 2001), **spatial** (Venables and Ripley 2002), **geoRglm** (Christensen and Ribeiro 2002), **gstat** (Pebesma 2004), **fields** (Fields Development Team 2006), **spBayes** (Finley, Banerjee, and Carlin 2007), and **ramps** (Smith, Yan, and Cowles 2008).

(Ver Hoef, Peterson, and Theobald 2006). In this paper, we present the R package **SSN**, which allows users to fit autocovariance functions developed for stream networks (Ver Hoef and Peterson 2010). These models are unique because they use distance measured along the network, they incorporate flow direction, and they allow covariance weighting when segments converge (e.g., by volume of flowing water). We develop two classes of covariance models based on moving average constructions that we call the tail-up and tail-down models. These models may also be combined in a mixed model strategy that includes models based on Euclidean distance. Such geostatistical mixed models are important because they can account for multiple processes of spatial autocorrelation in stream systems, including those that occur within the stream and others that result from the straight-line distances due to the terrestrial environment (Ver Hoef and Peterson 2010). We note that there is another package **Rtop** for spatial prediction along stream networks (Skøien, Laaha, Koffler, Blöschl, Pebesma, Parajka, and Viglione 2012). After describing **SSN**, we make some comparisons to **Rtop** in Section 7.

The **SSN** package is available on CRAN, with more information and data sets available at https://www.fs.usda.gov/rm/boise/AWAE/projects/SpatialStreamNetworks.shtml, which contains additional documentation, tutorials and example data sets. Windows and Linux binaries are provided, as well as the source code. After installation, the package is ready for use in an R session after typing

```
library("SSN")

## Loading required package:  RSQLite
## Loading required package:  sp
```

at the R prompt. To ensure that you have full read and write permissions, and that you do not leave stray files on your computer, use the following

```
file.copy(system.file("lsndata/MiddleFork04.ssn", package = "SSN"),
  to = tempdir(), recursive = TRUE, copy.mode = FALSE)

## [1] TRUE

setwd(tempdir())
```

The rest of the paper is organized as follows: Section 2 provides a brief overview of spatial statistical modelling on stream networks. Section 3 provides an overview of the `S4 SpatialStreamNetwork` object, while Section 4 describes how to set up and manipulate the object. Section 5 describes the main statistical capabilities provided by the package, including exploratory data analysis, model fitting, model evaluation and diagnostics, as well as prediction. Section 6 shows how to simulate stream network data. Finally, Section 7 provides a brief discussion of future developments.

This document was compiled on 2023-04-19 using R version 4.2.2 Patched (2022-11-10 r83330).

2. Spatial statistical models on stream networks

Some new models for stream networks, based on moving average constructions, were initially described by Ver Hoef *et al.* (2006) and Cressie, Frey, Harch, and Smith (2006). The models used stream distance measures (e.g., Dent and Grimm 1999), where stream distance is defined as the shortest distance between two locations computed only along the stream network. This work was summarized by Ver Hoef and Peterson (2010), which provides more technical details. Here, we give a brief summary.

2.1. Background

The fact that streams are dichotomous and have flow creates a rich set of models not seen in either time-series or spatial statistics. The models are created using moving average constructions. If a moving average function starts at some location and is non-zero only upstream of that location, we call them "tail-up" models. Due to the dichotomous nature of streams, the moving average function may need to split as it goes upstream in order to produce stationary models, so some weighting must occur. If a moving average function starts at some location and is non-zero only downstream of that location, we call them "tail-down" models. A full development of these models is given in Ver Hoef and Peterson (2010). Consider two pairs of sites that have the same stream distance between them, but one pair is connected by flowing water (i.e., flow-connected), and the other pair is not connected by flowing water (i.e., flow-unconnected); in general the amount of autocorrelation will be different between them. For the following development, let r_i and s_j denote two locations on a stream network, and let h be the stream distance between them. Then the following covariance models have been developed and implemented in the **SSN** package.

2.2. Tail-up models

The moving average construction as described by Ver Hoef *et al.* (2006) is

$$C_u(r_i, s_j|\boldsymbol{\theta}_u) = \begin{cases} \pi_{i,j} C_t(h|\boldsymbol{\theta}_u) & \text{if } r_i \text{ and } s_j \text{ are flow-connected,} \\ 0 & \text{if } r_i \text{ and } s_j \text{ are flow-unconnected,} \end{cases} \tag{1}$$

where $\pi_{i,j}$ are weights due to branching characteristics of the stream, and the function $C_t(h|\boldsymbol{\theta}_u)$ can take the following forms:

- Tail-up Linear-with-Sill Model,

$$C_t(h|\boldsymbol{\theta}_u) = \sigma_u^2 \left(1 - \frac{h}{\alpha_u}\right) I\left(\frac{h}{\alpha_u} \leq 1\right),$$

- Tail-up Spherical Model,

$$C_t(h|\boldsymbol{\theta}_u) = \sigma_u^2 \left(1 - \frac{3}{2}\frac{h}{\alpha_u} + \frac{1}{2}\frac{h^3}{\alpha_u^3}\right) I\left(\frac{h}{\alpha_u} \leq 1\right),$$

- Tail-Up Exponential Model,

$$C_t(h|\boldsymbol{\theta}_u) = \sigma_u^2 \exp(-3h/\alpha_u),$$

- Tail-up Mariah Model,

$$C_t(h|\boldsymbol{\theta}_u) = \begin{cases} \sigma_u^2 \left(\frac{\log(90h/\alpha_u+1)}{90h/\alpha_u}\right) & \text{if } h > 0, \\ \sigma_u^2 & \text{if } h = 0, \end{cases}$$

- Tail-up Epanechnikov Model (Garreta, Monestiez, and Ver Hoef 2009),

$$C_t(h|\boldsymbol{\theta}_u) = \frac{\sigma_u^2(h-\alpha_u)^2 f_{eu}(h;\alpha_u)}{16\alpha_u^5} I\left(\frac{h}{\alpha_u} \leq 1\right).$$

where $f_{eu}(h;\alpha_u) = 16\alpha_u^2 + 17\alpha_u^2 h - 2\alpha_u h^2 - h^3$, $I(\cdot)$ is the indicator function (equal to one when the argument is true), $\sigma_u^2 > 0$ is an overall variance parameter (also known as the partial sill), $\alpha_u > 0$ is the range parameter, and $\boldsymbol{\theta}_u = (\sigma_u^2, \alpha_u)^\top$. Note the factors 3, and 90 for the exponential and mariah models, respectively, which cause the autocorrelation to be approximately 0.05 when h equals the range parameter. This helps compare range parameters (α_u) across models. (The distance at which autocorrelation reaches 0.05 is sometimes called the effective range when models approach zero asymptotically.)

2.3. Weights for tail-up models

For the weights, $\pi_{i,j}$, consider the following example. First associate a weight of 1 with each source (upper-most) segment in a stream network. When two segments converge at a junction the weight for the segment downstream of the junction is the sum of the two upstream weights, creating an additive function when moving downstream. Segment weights formed in this manner are known as Shreve's stream order (Shreve 1967). These additive weights are a property of whole stream segments, creating a step function along a stream network. Any point on a stream network has an additive function value obtained from the stream segment where it occurs. If we denote the value of the additive function as $\Omega(x)$ for some point x on the stream network, then for two flow-connected points where r_i is downstream from s_j,

$$\pi_{i,j} = \sqrt{\frac{\Omega(s_j)}{\Omega(r_i)}}.$$

More details can be found in Ver Hoef and Peterson (2010), including ways to create an additive function from arbitrary values associated with stream segments, such as flow volume or a proxy for flow volume (e.g., basin area).

2.4. Tail-down models

For tail-down models, we distinguish between the flow-connected and flow-unconnected situation. When two sites are flow-unconnected, let b denote the longer of the distances to the common downstream junction, and a denote the shorter of the two distances. If two sites are flow-connected, again use h to denote their separation distance via the stream network. The following are tail-down models:

- Tail-Down Linear-with-Sill Model, $b \geq a \geq 0$,

$$C_d(a,b,h|\boldsymbol{\theta}_d) = \begin{cases} \sigma_d^2 \left(1 - \frac{h}{\alpha_d}\right) I\left(\frac{h}{\alpha_d} \leq 1\right) & \text{if flow-connected,} \\ \sigma_d^2 \left(1 - \frac{b}{\alpha_d}\right) I\left(\frac{b}{\alpha_d} \leq 1\right) & \text{if flow-unconnected,} \end{cases}$$

- Tail-Down Spherical Model, $b \geq a \geq 0$,

$$C_d(a,b,h|\boldsymbol{\theta}_d) = \begin{cases} \sigma_d^2(1 - \frac{3}{2}\frac{h}{\alpha_d} + \frac{1}{2}\frac{h^3}{\alpha_d^3})I\left(\frac{h}{\alpha_d} \leq 1\right) & \text{if flow-connected,} \\ \sigma_d^2\left(1 - \frac{3}{2}\frac{a}{\alpha_d} + \frac{1}{2}\frac{b}{\alpha_d}\right)\left(1 - \frac{b}{\alpha_d}\right)^2 I\left(\frac{b}{\alpha_d} \leq 1\right) & \text{if flow-unconnected,} \end{cases}$$

- Tail-down Exponential Model,

$$C_d(a,b,h|\boldsymbol{\theta}_d) = \begin{cases} \sigma_d^2 \exp(-3h/\alpha_d) & \text{if flow-connected,} \\ \sigma_d^2 \exp(-3(a+b)/\alpha_d) & \text{if flow-unconnected,} \end{cases}$$

- Tail-down Mariah Model,

$$C_d(a,b,h|\boldsymbol{\theta}_d) = \begin{cases} \sigma_d^2\left(\frac{\log(90h/\alpha_d+1)}{90h/\alpha_d}\right) & \text{if flow-connected, } h > 0, \\ \sigma_d^2 & \text{if flow-connected, } h = 0, \\ \sigma_d^2\left(\frac{\log(90a/\alpha_d+1)-\log(90b/\alpha_d+1)}{90(a-b)/\alpha_d}\right) & \text{if flow-unconnected, } a \neq b, \\ \sigma_d^2\left(\frac{1}{90a/\alpha_d+1}\right) & \text{if flow-unconnected, } a = b, \end{cases}$$

- Tail-down Epanechnikov Model, $b \geq a \geq 0$,

$$C_d(a,b,h|\boldsymbol{\theta}_d) = \begin{cases} \frac{\sigma_d^2(h-\alpha_d)^2 f_{eu}(h;\alpha_d)}{16\alpha_d^5} I\left(\frac{h}{\alpha_u} \leq 1\right) & \text{if flow-connected,} \\ \frac{\sigma_d^2(b-\alpha_d)^2 f_{ed}(a,b;\alpha_d)}{16\alpha_d^5} I\left(\frac{b}{\alpha_u} \leq 1\right) & \text{if flow-unconnected,} \end{cases}$$

where $f_{ed}(a,b;\alpha_d) = 16\alpha_d^3 + 17\alpha_d^2 b - 15\alpha_d^2 a - 20\alpha_d a^2 - 2\alpha_d b^2 + 10\alpha_d ab + 5ab^2 - b^3 - 10ba^2$, $\sigma_d^2 > 0$ and $\alpha_d > 0$, and $\boldsymbol{\theta}_d = (\sigma_d^2, \alpha_d)^\top$. Although not necessary to maintain stationarity, the weights used in the tail-up models can be applied to the tail-down models as well. Note that h is unconstrained, because for model-building we imagine that the headwater and outlet segments continue to infinitiy, as first described by Ver Hoef *et al.* (2006). Also note that a does not appear in the tail-down linear-with-sill model, but is used indirectly because the model depends on the point that is farthest from the junction; i.e., b, and so a is the shorter of the two distances.

2.5. Euclidean distance models

The 2-D coordinate system is used to include Euclidean distance models, which are described in many textbooks (e.g., Cressie 1993; Chiles and Delfiner 1999). We include four models and show their parameterization in the **SSN** package. If site r_i has x,y-coordinates (x_i, y_i) and site s_j has x,y-coordinates (x_j, y_j), then the Euclidean distance is defined as $d = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$, and we have the following models:

- Euclidean Distance Cauchy Model,

$$C_e(d|\boldsymbol{\theta}_e) = \sigma_e^2\left(1 + 4.4(d/\alpha_e)^2\right)^{-1},$$

- Euclidean Distance Spherical Model,

$$C_e(d|\boldsymbol{\theta}_e) = \sigma_e^2\left(1 - \frac{3}{2}\frac{d}{\alpha_e} + \frac{1}{2}\frac{d^3}{\alpha_e^3}\right) I\left(\frac{d}{\alpha_e} \leq 1\right),$$

- Euclidean Distance Exponential Model,

$$C_e(d|\boldsymbol{\theta}_e) = \sigma_e^2 \exp(-3d/\alpha_e),$$

- Euclidean Distance Gaussian Model,

$$C_e(d|\boldsymbol{\theta}_e) = \sigma_e^2 \exp(-3(d/\alpha_e)^2).$$

Note the factors 3, 3, and 4.4 for the exponential, Gaussian, and Cauchy models, respectively, which cause the autocorrelation to be approximately 0.05 when d equals the range parameter.

2.6. Random effects models

The **SSN** package provides the functionality to fit random effect models associated with factor variables. Let $\gamma_k(x)$ denote the factor level at location x. Each level of γ_k is assumed to be a random quantity with mean 0 and variance σ_k^2 and these are independently and identically distributed. As such, sites with the same level of γ_k are correlated, and sites with different levels of γ_k are uncorrelated,

$$C_k(r_i, s_j) = \begin{cases} \sigma_i^2 & \text{if } \gamma_k(r_i) = \gamma_k(s_j), \\ 0 & \text{if } \gamma_k(r_i) \neq \gamma_k(s_j). \end{cases} \tag{2}$$

2.7. Spatial linear mixed models

The most general linear model that the **SSN** package considers is

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{z}_u + \mathbf{z}_d + \mathbf{z}_e + \mathbf{W}_1\boldsymbol{\gamma}_1 + \ldots + \mathbf{W}_p\boldsymbol{\gamma}_p + \boldsymbol{\epsilon}, \tag{3}$$

where $\mathbf{X}$ is a design matrix of fixed effects, $\boldsymbol{\beta}$ are parameters, the vector $\mathbf{z}_u$ contains spatially-autocorrelated random variables with a tail-up autocovariance, with $\text{var}(\mathbf{z}_u) = \sigma_u^2\mathbf{R}(\alpha_u)$ and $\mathbf{R}(\alpha_u)$ is a correlation matrix that depends on the range parameter α_u as described in Section 2.2; $\mathbf{z}_d$ contains spatially-autocorrelated random variables with a tail-down autocovariance, $\text{var}(\mathbf{z}_d) = \sigma_d^2\mathbf{R}(\alpha_d)$ as described in Section 2.4; $\mathbf{z}_e$ contains spatially-autocorrelated random variables with a Euclidean distance autocovariance, $\text{var}(\mathbf{z}_e) = \sigma_e^2\mathbf{R}(\alpha_e)$ as described in Section 2.5; $\mathbf{W}_k$ is a design matrix for random effects $\boldsymbol{\gamma}_k; k = 1, \ldots, p$ with $\text{var}(\boldsymbol{\gamma}_k) = \sigma_k^2\mathbf{I}$, and $\boldsymbol{\epsilon}$ contains independent random variables with $\text{var}(\boldsymbol{\epsilon}) = \sigma_0^2\mathbf{I}$. When used for spatial prediction, this model is referred to as "universal" kriging (Le and Zidek 2006, p. 107) , with "ordinary" kriging being the special case where the design matrix $\mathbf{X}$ is a single column of ones (Cressie 1993, p. 119). The most general covariance matrix considered by the **SSN** package is of the form

$$\text{cov}(\mathbf{Y}) = \boldsymbol{\Sigma} = \sigma_u^2\mathbf{R}(\alpha_u) + \sigma_d^2\mathbf{R}(\alpha_d) + \sigma_e^2\mathbf{R}(\alpha_e) + \sigma_1^2\mathbf{W}_1\mathbf{W}_1^\top + \ldots + \sigma_p^2\mathbf{W}_p\mathbf{W}_p^\top + \sigma_0^2\mathbf{I}. \tag{4}$$

The **SSN** package uses either maximum likelihood (ML) or restricted maximum likelihood (REML) to estimate $\boldsymbol{\beta}$ and some subset of the possible covariance parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_u^\top, \boldsymbol{\theta}_d^\top, \boldsymbol{\theta}_e^\top, \sigma_1^2, \ldots, \sigma_p^2, \sigma_0^2)^\top$, as specified by the user.

2.8. Spatial generalized linear mixed models

The **SSN** package estimates parameters for generalized linear mixed models using pseudo-models as described by Wolfinger and O'Connell (1993). For time series and other spatial covariance structures, pseudo-models are implemented in the `glmmPQL` function in the **MASS** package (Venables and Ripley 2002). At present, only binomial and Poisson distributions are supported in the **SSN** package. Consider the expectation of the response variable as a linear mixed model through a link function,

$$E(\mathbf{Y}|\boldsymbol{\gamma}) = g^{-1}(\mathbf{X}\boldsymbol{\beta} + \mathbf{W}\boldsymbol{\gamma}) = g^{-1}(\boldsymbol{\eta}) = \boldsymbol{\mu}, \tag{5}$$

where g is the link function. Suppose that we have a current estimates of $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$; call them $\tilde{\boldsymbol{\beta}}$ and $\tilde{\boldsymbol{\gamma}}$. Initial estimates could be obtained using the `glm()` function, which assumes independence among data. Then pseudo-data are formed as

$$\tilde{\mathbf{Y}} \equiv \tilde{\boldsymbol{\Delta}}^{-1}(\mathbf{Y} - g^{-1}(\mathbf{X}\tilde{\boldsymbol{\beta}} + \mathbf{W}\tilde{\boldsymbol{\gamma}})) + \mathbf{X}\tilde{\boldsymbol{\beta}} + \mathbf{W}\tilde{\boldsymbol{\gamma}}, \tag{6}$$

where

$$\tilde{\boldsymbol{\Delta}} \equiv \frac{\partial g^{-1}(\boldsymbol{\eta})}{\partial \boldsymbol{\eta}}$$

is a diagonal matrix evaluated at $\tilde{\boldsymbol{\beta}}$ and $\tilde{\boldsymbol{\gamma}}$. Then the pseudo-model is

$$\tilde{\mathbf{Y}} = \mathbf{X}\boldsymbol{\beta} + \mathbf{W}\boldsymbol{\gamma} + \boldsymbol{\epsilon}. \tag{7}$$

Note that the left-hand side of Equation 7 depends on $\tilde{\boldsymbol{\beta}}$ and $\tilde{\boldsymbol{\gamma}}$, so that suggests an iterative procedure for updating $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$. The covariance matrix specification of pseudo-data is assumed to be

$$\text{var}(\boldsymbol{\epsilon}) = \tilde{\boldsymbol{\Delta}}^{-1}\mathbf{A}^{\frac{1}{2}}\sigma^2\mathbf{R}\mathbf{A}^{\frac{1}{2}}\tilde{\boldsymbol{\Delta}}^{-1},$$

where $\mathbf{A}$ is a diagonal matrix and contains the variance functions of the model (e.g., McCullagh and Nelder 1989, Table 2.1). The matrix $\mathbf{R}$ is a correlation matrix as in Equation 4. The most general covariance matrix of the linear mixed pseudo-model considered by the **SSN** package is

$$\begin{aligned}\text{cov}(\tilde{\mathbf{Y}}) = \tilde{\boldsymbol{\Delta}}^{-1}\mathbf{A}^{\frac{1}{2}}[\sigma_u^2\mathbf{R}(\alpha_u) + \sigma_d^2\mathbf{R}(\alpha_d) + \sigma_e^2\mathbf{R}(\alpha_e) + \sigma_0^2\mathbf{I}]\mathbf{A}^{\frac{1}{2}}\tilde{\boldsymbol{\Delta}}^{-1} + \\ \sigma_1^2\mathbf{W}_1\mathbf{W}_1^\top + \ldots + \sigma_p^2\mathbf{W}_p\mathbf{W}_p^\top.\end{aligned} \tag{8}$$

The iterative algorithm of Wolfinger and O'Connell (1993) is implemented as the following steps: 1) the parameters $\tilde{\boldsymbol{\beta}}$ and $\tilde{\boldsymbol{\gamma}}$ and pseudo-data $\tilde{\mathbf{Y}}$ are updated using an iteratively weighted least squares algorithm (e.g., McCullagh and Nelder 1989, pg. 40) for a fixed covariance model (Equation 8), and then 2) the covariance parameters $\boldsymbol{\theta}$ of Equation 8 are updated using ML or REML for fixed $\tilde{\mathbf{Y}}$.

3. The S4 `SpatialStreamNetwork` object

3.1. Input data

The spatial data needed to fit a stream network model are non-trivial. Much of the spatial data editing, information generation, and formatting take place in ArcGIS version 9.3.1 using the Spatial Tools for the Analysis of River Systems (**STARS**) toolset (Peterson and Ver Hoef in review). When this pre-processing is complete, a new directory is created to store the data,

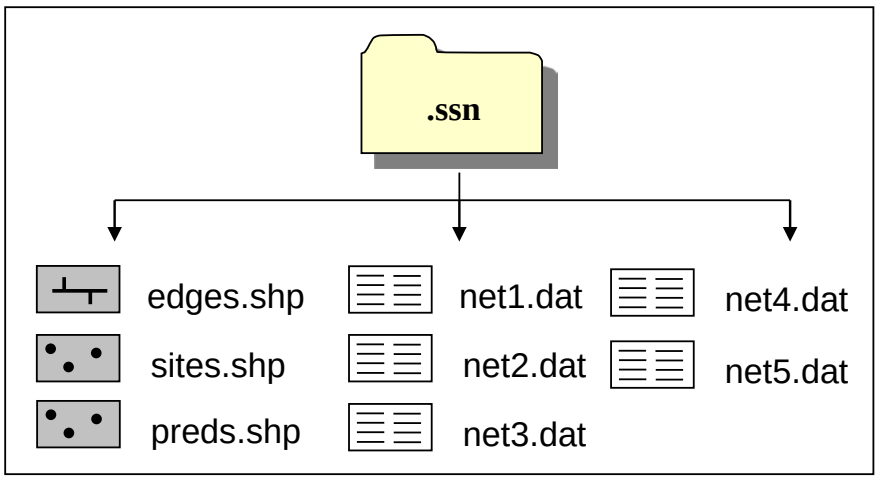

Figure 1: The `.ssn` directory contains the spatial, attribute, and topological information needed to create a `SpatialStreamNetwork` object in R. The *.shp files are of stream reaches (edges.shp), observed points on the stream (site.shp), and prediction points on the stream (preds.shp)(optional). The *.dat files contain topological relationship information for each distinct network; in this example, there are 5 networks.

hereafter referred to as the `.ssn` directory, which can then be imported into R. These data include the feature geometry, attribute data, and topological relationships of each point and line data set, see Figure 1.

The `.ssn` directory will always contain two shapefiles: edges and sites, which contain the geometry and attribute information for the stream network and the observed data locations. Multiple comma-delimited text files containing the topological information for each stream network within the edges data set will also be included. Here, a network is defined as a set of connected, directed line segments that share a common junction somewhere downstream. Note that the naming conventions for these files are implemented by the **STARS** toolset and must be preserved. Multiple shapefiles representing the prediction locations may also be included in the `.ssn` directory; however, the naming conventions for these data sets will vary. Please see Peterson and Ver Hoef (in review) for a detailed description of the **STARS** toolset, the methods used to generate the .ssn directory, and data therein.

3.2. Object structure

The `SpatialStreamNetwork` class is an `S4` object (Figure 2), which is the foundational spatial data class in the **SSN** package. Its structure adheres to the conventions for spatial data classes set out by Bivand, Pebesma, and Gomez-Rubio (2008). Yet, the `SpatialStreamNetwork` is unique because it contains both point and line features within the same `S4` object. It directly extends class `SpatialLinesDataFrame`, with additional slots included to store the spatial point data `SSNPoints`, a matrix containing information about the network coordinates of each line segment `network.line.coords`, as well as a string representing the filepath to the `.ssn`

directory. A new class, `SSNPoint`, has been defined and created to store spatial point data for observation and prediction locations. This class is similar to class `SpatialPointsDataFrame`, but is not a direct extension of it since common slot names contain the prefix "point." An additional matrix containing network coordinates, `network.point.coords`, has also been included. The object `SSNPoints` is a list of class `SSNPoints` that stores objects of class `SSNPoint`. It also contains a slot named `ID`, which holds a string identifier for each `SSNPoint` object.This allows identification within the `SpatialStreamNetwork` object when multiple data sets of prediction sites are included.

The purpose of the `.ssn` directory is to store information about spatial relationships generated in R, in addition to holding the input data. When the `SpatialStreamNetwork` class is created, the topological information contained in text files is imported into a SQLite database using the **RSQLite** package (James 2011). This database is named `binaryID.db` and contains a separate table for each network, which holds the topological information of the network, including reach identifiers (`rid`) and binary identifiers (`ID`). The `rid`s are unique to each line segment in the entire edges data set, while binary `ID`s are unique to each line segment within a network, see Figure 2.

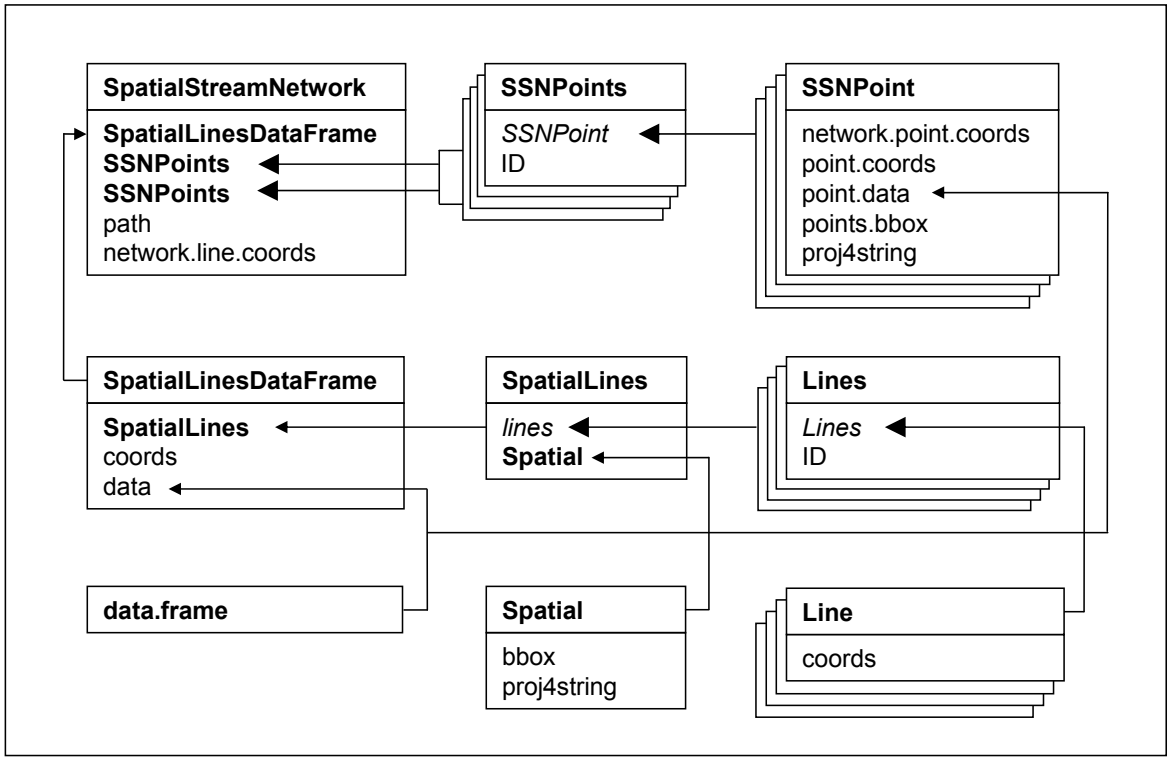

Figure 2: The `SpatialStreamNetwork` class and slots; arrows with small heads show sub-class extensions and arrows with large heads show the inclusion of lists of objects.

Peterson and Ver Hoef (in review) give additional details about the binary IDs and how they are used to assess spatial relationships within a network.

4. Manipulating the SpatialStreamNetwork object

The **SSN** package provides functions to import and export SpatialStreamNetwork objects. These objects may be created using the **STARS** ArcGIS custom toolbox for version 9.3.1 (Peterson and Ver Hoef in review). It is also possible to simulate a SpatialStreamNetwork with both observed and prediction locations in R. Regardless of the way the object is generated, it contains important spatial information about feature geometry, attribute data, and topological relationships. These relationships must be represented in R to create the spatial information needed to fit a spatial statistical model. These capabilities are further explored below.

4.1. Data available in the SSN package

A data set is included in the **SSN** package as well as code to simulate data on simulated stream networks. The data set is from the Middle Fork, a stream in Idaho, United States, and is a subset of a larger data set than can be freely accessed at https://www.fs.usda.gov/rm/boise/AWAE/projects/SpatialStreamNetworks.shtml. The data set consists of two stream networks with 45 total observations, 220 prediction locations on a 1 km spacing, 1273 prediction locations densely packed on a single stream called the Knapp, and another 654 prediction locations densely packed on a single stream called CapeHorn.

4.2. Importing and subsetting the SpatialStreamNetwork object

The function to import the data from the .ssn directory and create a SpatialStreamNetwork object is:

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

The only compulsory argument is Path, which is a string describing the filepath and basename of the .ssn directory. Documentation describing the input arguments is given by the command help(importSSN). For example, to import the MiddleFork04 data set provided in the **SSN** package:

```
mf04p <- importSSN("./MiddleFork04.ssn",
   predpts = "pred1km")

## binaryID.db already exists - no changes were made to binaryID.db table
```

Note that binaryID.db already existed, so it was not recomputed (because o.write = FALSE by default). The point.data data.frame (residing under SSNPoint in Figure 2) contains 45 observed data (rows) and 35 variables (columns). Using help("MiddleFork04.ssn") provides a description of each column in the point.data data.frames. The importSSN function allows one set of prediction points to be imported at a time into the SpatialStreamNetwork object, called pred1km in this example. It may be useful to have more than one prediction point data set associated with the SpatialStreamNetwork object and this is accomplished with the importPredpts function:

```
mf04p <- importPredpts(mf04p, "Knapp", "ssn")
mf04p <- importPredpts(mf04p, "CapeHorn", "ssn")
```

The real data sets have many lines so it is difficult to view the whole object using the `str` function on the `SpatialStreamNetwork` object. In Section 6, we show how to simulate very small, simple networks that can easily show a `SpatialStreamNetwork` object structure.

The `subsetSSN` function can be used to subset an existing `SpatialStreamNetwork` object from within the R environment. The subset selection is based on a logical expression, indicating which observed sites to keep, with an additional argument specifying whether the logical expression should also be applied to the edges and prediction locations. A new `.ssn` directory (Figure 1) is created, where the subset of spatial information is stored.

4.3. Generating an additive function value

When fitting tail-up models the moving average function "splits" at every junction, and this requires the computation of the values $\pi_{i,j}$ listed in Section 2.2. Recall that these are defined as

$$\pi_{i,j} = \sqrt{\frac{\Omega(s_j)}{\Omega(r_i)}}, \tag{9}$$

where typically

$$\Omega(x) = \prod_{k \in D_x} \omega_k \tag{10}$$

for some attribute ω_k of stream segment k (i.e., the `data data.frame`, contained within the `SpatialLinesDataFrame` in Figure 2), and D_x comprises all stream segments downstream of point x, including the segment containing x itself. The calculated values $\Omega(\cdot)$, known as additive function values, are then included as an additional column to the `point.data data.frame`s. These values are computed based on the observed covariate values for the line segments, and so they are highly dependent on the structure of the underlying network. This computation can be performed in ArcGIS, which is proprietary software that is not accessible to all end-users, using the **STARS** toolset. Additive function values can also be computed by `additive.function` provided by the **SSN** package:

```
additive.function(mf04p, VarName, afvName)
```

The function returns a `SpatialStreamNetwork` object, which is a modified version of the input object `ssn`. The modified object has an additional covariate named `afvName`, which contains the additive function values based on the covariate named `VarName`. A column names `afvName` is added to the `data data.frame`, contained within the `SpatialLinesDataFrame` and the `point.data data.frame`s contained in each `SSNpoint` within `SSNpoint`. The included data set can be used to demonstrate this. The additive-function-value column `afvArea` was pre-computed and included in the data set, based on `h2oAreaKm2`. First, notice the attributes for the line segments in the network:

```
names(mf04p@data)
```

```
##  [1] "COMID"      "FDATE"      "RESOLUTION" "GNIS_ID"    "GNIS_NAME"
##  [6] "LENGTHKM"   "REACHCODE"  "FLOWDIR"    "WBAREACOMI" "FTYPE"
## [11] "FCODE"      "CUMdrainAG" "MAXELEVSMO" "AREAWTMAP"  "SLOPE"
## [16] "HUC3"       "HUC4"       "h2oAreaKm2" "Shape_Leng" "rid"
## [21] "areaPI"     "afvArea"    "upDist"     "Length"     "netID"
```

and the values for `h2oAreaKm2` and `afvArea` in the point `data.frame`:

```
head(mf04p@data[, c("h2oAreaKm2", "afvArea")])
```

```
##   h2oAreaKm2   afvArea
## 0   1970.558 0.3792740
## 1   1799.024 0.3792740
## 2   1620.071 0.3764886
## 3  10671.358 1.0000000
## 4  10180.870 1.0000000
## 5   9603.091 0.9896855
```

Now, use `additive.function` to compute the additive function value based on `h2oAreaKm2`, and name it `computed.afv`:

```
mf04p <- additive.function(mf04p, "h2oAreaKm2", "computed.afv")
```

Notice that the `computed.afv` column has been added to `mf04p@data`:

```
names(mf04p@data)
```

```
##  [1] "rid"          "COMID"        "FDATE"        "RESOLUTION"
##  [5] "GNIS_ID"      "GNIS_NAME"    "LENGTHKM"     "REACHCODE"
##  [9] "FLOWDIR"      "WBAREACOMI"   "FTYPE"        "FCODE"
## [13] "CUMdrainAG"   "MAXELEVSMO"   "AREAWTMAP"    "SLOPE"
## [17] "HUC3"         "HUC4"         "h2oAreaKm2"   "Shape_Leng"
## [21] "areaPI"       "afvArea"      "upDist"       "Length"
## [25] "netID"        "computed.afv"
```

and values from the stream segments are passed to points on each segment:

```
head(mf04p@data[, c("h2oAreaKm2",
   "afvArea", "computed.afv")])
```

```
##   h2oAreaKm2   afvArea computed.afv
## 0   1970.558 0.3792740    0.3792740
## 1   1799.024 0.3792740    0.3792740
## 2   1620.071 0.3764886    0.3764886
## 3  10671.358 1.0000000    1.0000000
## 4  10180.870 1.0000000    1.0000000
## 5   9603.091 0.9896855    0.9896855
```

```
head(getSSNdata.frame(mf04p)[, c("afvArea", "computed.afv")])

##     afvArea computed.afv
## 1 0.6234092    0.6234092
## 2 0.1509203    0.1509203
## 3 0.1509203    0.1509203
## 4 0.1509203    0.1509203
## 5 0.1509203    0.1509203
## 6 0.1502166    0.1502166
```

where the `computed.afv` column replicated the existing `afvArea` column.

4.4. Calculating distance matrices

The covariance models of Sections 2.2 and 2.4 are based on network distance. Fitting these models requires a symmetric matrix $\mathbf{D}_{o,o}$, containing the network distances between all pairs of observed points. In fact for tail-down models we need more detailed information for every pair of observed points r_i and s_j, including the distance downstream from r_i to the first junction that connects it to s_j. This second set of distances can be organised into another matrix $\mathbf{N}_{o,o}$ that is not symmetric because it contains the distances from i to the junction with j, and from j to the junction with i as off-diagonal elements. If i and j are flow-connected sites, one of $\mathbf{N}_{o,o}[i,j]$ and $\mathbf{N}_{o,o}[j,i]$ is equal to zero (the one farther downstream; i.e., it is the common junction). Consequently, $\mathbf{D}_{o,o} = \mathbf{N}_{o,o} + \mathbf{N}_{o,o}^{\top}$, so that it is sufficient to compute $\mathbf{N}_{o,o}$.

Network distances between observation and prediction points are needed for prediction. These are computed as two matrices $\mathbf{N}_{o,p}$ and $\mathbf{N}_{p,o}$, corresponding to downstream distances from observed to prediction points and prediction to observed points, respectively. For block kriging and simulations (that include simulations of prediction points), a matrix of downstream distances between pairs of prediction points, $\mathbf{N}_{p,p}$, is also needed. If we denote the collection of all downstream distances by $\mathbf{N}$, then

$$\mathbf{N} = \left(\begin{array}{c|c} \mathbf{N}_{o,o} & \mathbf{N}_{o,p} \\ \hline \mathbf{N}_{p,o} & \mathbf{N}_{p,p} \end{array} \right). \tag{11}$$

The function `createDistMat` creates the relevant parts of the matrix $\mathbf{N}$ before fitting models or making predictions:

```
createDistMat(mf04p, predpts = "Knapp", o.write = TRUE,
        amongpreds = TRUE)
createDistMat(mf04p, predpts = "CapeHorn", o.write = TRUE,
        amongpreds = TRUE)
```

The matrix $\mathbf{N}_{o,o}$ is always computed. The input `predpts` gives the name of a set of prediction points; if this input is specified then the rectangular matrices $\mathbf{N}_{o,p}$ and $\mathbf{N}_{p,o}$ are also computed. A value of `predpts = NULL` indicates that only $\mathbf{N}_{o,o}$ should be computed. If `amongpreds = TRUE` then in addition the square matrix $\mathbf{N}_{p,p}$ is calculated, which is required if the `predpts` are used for block prediction, or for simulating at the prediction points as well as the observed

points. Input `o.write` determines whether the specified matrices will be recalculated if they already exist, with the default behaviour being to retain existing computations.

Using the `getStreamDistMat` function allows accessing the asymmetric stream distance matrix after it is created. In these matrices, the "from" sites are labeled across the top (column labels) and the "to" sites are labeled along the left (row labels). There may be multiple matrices if there are multiple networks, labeled sequentially starting with `.net1`, etc.

```
distObs <- getStreamDistMat(mf04p)
str(distObs)

## List of 2
##  $ dist.net1: num [1:13, 1:13] 0 2491 528 15877 15094 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:13] "31" "32" "33" "34" ...
##   .. ..$ : chr [1:13] "31" "32" "33" "34" ...
##  $ dist.net2: num [1:32, 1:32] 0 0 0 0 0 0 0 0 0 0 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:32] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:32] "1" "2" "3" "4" ...

distObs$dist.net1[1:5,1:5]

##            31       32        33 34       35
## 31     0.0000     0.00     0.000  0   0.0000
## 32  2491.3657     0.00  1962.963  0   0.0000
## 33   528.4024     0.00     0.000  0   0.0000
## 34 15876.6936 13385.33 15348.291  0 782.3891
## 35 15094.3044 12602.94 14565.902  0   0.0000
```

To obtain total in-stream distance, we take the asymmetric matrix plus its transpose.

```
strDistNet2 <- distObs$dist.net2 + t(distObs$dist.net2)
strDistNet2[5:10,5:10]

##           5        6        7        8        9       10
## 5     0.000 1018.962 2105.652 2345.758 3738.746 2979.006
## 6  1018.962    0.000 1086.690 1326.796 2719.784 3997.968
## 7  2105.652 1086.690    0.000  240.106 1633.094 5084.658
## 8  2345.758 1326.796  240.106    0.000 1392.988 5324.764
## 9  3738.746 2719.784 1633.094 1392.988    0.000 6717.751
## 10 2979.006 3997.968 5084.658 5324.764 6717.751    0.000
```

Likewise, we can obtain distance between observed sites and prediction locations. Here, two matrices are required per network, so the label "`.a`" indicates from prediction sites to observation sites, and the label "`.b`" indicates from observation sites to predictions sites.

```
distPred1km <- getStreamDistMat(mf04p, Name = "pred1km")
str(distPred1km)

## List of 4
##  $ dist.net1.a: num [1:13, 1:57] 0 0 0 1119 336 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:13] "31" "32" "33" "34" ...
##   .. ..$ : chr [1:57] "46" "47" "48" "49" ...
##  $ dist.net1.b: num [1:57, 1:13] 14758 15758 13384 11170 9900 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:57] "46" "47" "48" "49" ...
##   .. ..$ : chr [1:13] "31" "32" "33" "34" ...
##  $ dist.net2.a: num [1:32, 1:118] 0 0 0 0 0 0 0 0 0 0 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:32] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:118] "74" "75" "76" "77" ...
##  $ dist.net2.b: num [1:118, 1:32] 2638 0 0 0 0 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:118] "74" "75" "76" "77" ...
##   .. ..$ : chr [1:32] "1" "2" "3" "4" ...

distPred1km$dist.net1.a[1:5,1:5]

##           46       47       48       49       50
## 31    0.0000   0.0000    0.000    0.000    0.000
## 32    0.0000   0.0000    0.000    0.000    0.000
## 33    0.0000   0.0000    0.000    0.000    0.000
## 34 1118.5041 118.5041 2492.595 4706.651 5976.436
## 35  336.1149   0.0000 1710.206 3924.262 5194.047
```

It is also possible to get distances among just the prediction locations.

```
createDistMat(mf04p, predpts = "CapeHorn", o.write = TRUE,
  amongpreds = TRUE)
distCape <- getStreamDistMat(mf04p, Name = "CapeHorn")
str(distCape)

## List of 3
##  $ dist.net2.a: num [1:32, 1:654] 3658 2957 1996 966 572 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:32] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:654] "1494" "1495" "1496" "1497" ...
##  $ dist.net2.b: num [1:654, 1:32] 0 0 0 0 0 0 0 0 0 0 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:654] "1494" "1495" "1496" "1497" ...
##   .. ..$ : chr [1:32] "1" "2" "3" "4" ...
```

```
##  $ dist.net2  : num [1:654, 1:654] 0 10 20 30 40 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:654] "1494" "1495" "1496" "1497" ...
##   .. ..$ : chr [1:654] "1494" "1495" "1496" "1497" ...

distCape$dist.net2[1:5,1:5]

##          1494      1495 1496     1497 1498
## 1494  0.00000  0.000000    0 0.000000    0
## 1495 10.00001  0.000000    0 0.000000    0
## 1496 20.00000  9.999995    0 0.000000    0
## 1497 30.00000 19.999998   10 0.000000    0
## 1498 40.00000 29.999994   20 9.999995    0
```

5. Data analysis using SSN

In this section, we demonstrate a full data analysis using the **SSN** package, including exploratory data analysis, model fitting, model diagnostics, model selection, and prediction.

5.1. Exploratory data analysis

This section gives an example of exploratory data analysis using the Middle Fork data set, which was imported in Section 4.2, and distance matrices, which were computed in Section 4.4. Recall that information about the data set can be found by typing `help(MiddleFork04.ssn)`. The names of the variables in the `point.data data.frame` for each observed and prediction data set in `mf04` are obtained with:

```
names(mf04p)

## $Obs
##  [1] "rid"          "STREAMNAME"   "HUC3"         "HUC4"
##  [5] "COMID"        "CUMDRAINAG"   "AREAWTMAP"    "MAXELEVSMO"
##  [9] "SLOPE"        "NCEASID_"     "ELEV_DEM"     "Deployment"
## [13] "SampleYear"   "NumberOfDa"   "OriginalID"   "Source"
## [17] "Summer_mn"    "MaxOver20"    "C16"          "C20"
## [21] "C24"          "FlowCMS"      "AirMEANc"     "AirMWMTc"
## [25] "NEAR_FID"     "NEAR_DIST"    "NEAR_X"       "NEAR_Y"
## [29] "NEAR_ANGLE"   "ratio"        "afvArea"      "upDist"
## [33] "locID"        "netID"        "pid"          "computed.afv"
##
## $pred1km
##  [1] "rid"          "COMID"        "GNIS_NAME"    "CUMDRAINAG"
##  [5] "HUC3"         "HUC4"         "AREAWTMAP"    "MAXELEVSMO"
##  [9] "SLOPE"        "COMID_"       "ELEV_DEM"     "FlowCMS"
## [13] "AirMEANc"     "AirMWMTc"     "SampleYear"   "NEAR_FID"
```

```
## [17] "NEAR_DIST"    "NEAR_X"       "NEAR_Y"       "NEAR_ANGLE"
## [21] "ratio"        "afvArea"      "upDist"       "locID"
## [25] "netID"        "pid"          "computed.afv"
##
## $Knapp
##  [1] "rid"          "COMID"        "GNIS_NAME"    "CUMDRAINAG"
##  [5] "HUC3"         "HUC4"         "AREAWTMAP"    "MAXELEVSMO"
##  [9] "SLOPE"        "COMID_"       "ELEV_DEM"     "FlowCMS"
## [13] "AirMEANc"     "AirMWMTc"     "SampleYear"   "NEAR_FID"
## [17] "NEAR_DIST"    "NEAR_X"       "NEAR_Y"       "NEAR_ANGLE"
## [21] "ratio"        "afvArea"      "upDist"       "locID"
## [25] "netID"        "pid"          "computed.afv"
##
## $CapeHorn
##  [1] "rid"          "COMID"        "GNIS_NAME"    "CUMDRAINAG"
##  [5] "HUC3"         "HUC4"         "AREAWTMAP"    "MAXELEVSMO"
##  [9] "SLOPE"        "COMID_"       "ELEV_DEM"     "FlowCMS"
## [13] "AirMEANc"     "AirMWMTc"     "SampleYear"   "NEAR_FID"
## [17] "NEAR_DIST"    "NEAR_X"       "NEAR_Y"       "NEAR_ANGLE"
## [21] "ratio"        "afvArea"      "upDist"       "locID"
## [25] "netID"        "pid"          "computed.afv"
```

Any of these `data.frames` in the `SpatialStreamNetwork` object can be accessed or stored as a separate object using `getSSNdata.frame(mf04)` or `getSSNdata.frame(ssn, Name = preds1km)`. Then other useful functions can be used, such as generic functions `summary`, `apply`, etc., and exploratory graphical functions like `boxplot`, `hist`, `qqnorm`, etc.

For spatial data, it is useful to see mapped data. The default plotting function for a `SpatialStreamNetwork` object is a map (Figure 3):

```
plot(mf04p, lwdLineCol = "afvArea", lwdLineEx = 10, lineCol = "blue",
   pch = 19, xlab = "x-coordinate (m)", ylab = "y-coordinate (m)",
   asp = 1)
```

Complex stream networks make it difficult to find the outlet, to tell which stream segments lead to other segments, and which segments are small or large. The line width in the plot can be made proportional to a column in the `data data.frame` (Figure 2) using the `lwdLineCol` and `lwdLineEx` arguments. The color can also be set with `lineCol`.

Our example response variable, `Summer_mn`, is the average summer stream temperature. This response is plotted across the stream network with observation locations colored by their value (Figure 4):

```
brks <- plot(mf04p, "Summer_mn", lwdLineCol = "afvArea",
   lwdLineEx = 15, lineCol = "black", xlab =  "x-coordinate" ,
   ylab =  "y-coordinate", asp=1 )
```

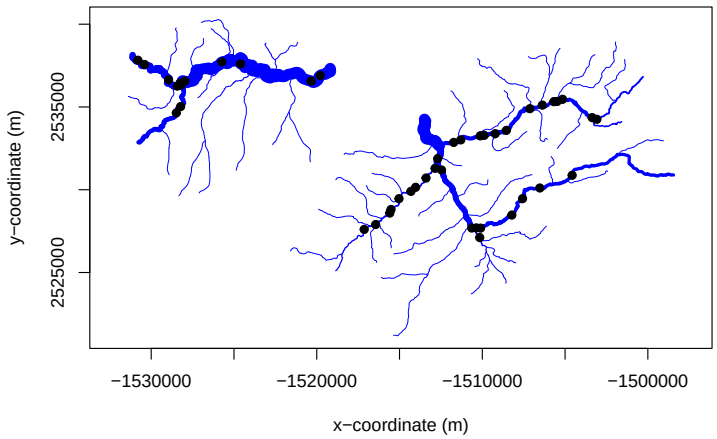

Figure 3: A plot of the Middle Fork stream network. The locations of observed data are shown as black points. The blue lines indicate the river network and the thickness of these lines indicates the size of the river for each segment.

The arguments `color.palette`, `breaktype` and `brks` control color scheme and break points. The plotting command returns a matrix of break points that can be used later for consistency of color meaning across multiple images.

It is also possible to use the plotting feature of the **sp** package by using the `as.SpatialLines`, `as.SpatialPoints`, and `as.SpatialPointsPolygons`, and the results are shown in Figure 5.

```
#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)
```

Many users of spatial statistics are familiar with empirical semivariogram plots, which are used to understand how covariance changes with Euclidean distance. A new but similar graphic, called a Torgegram, is used for stream network data. The Torgegram computes average squared-differences like an empirical semivariogram, except that it is based on stream distance with the semivariance plotted separately for flow-connected and flow-unconnected pairs (see Figure 6, where the size of the circles are proportional to the number of pairs for each binned distance class). Such figures were given in (Ver Hoef *et al.* 2006; Ver Hoef and Peterson 2010). The rationale behind a Torgegram is that autocorrelation may be evident

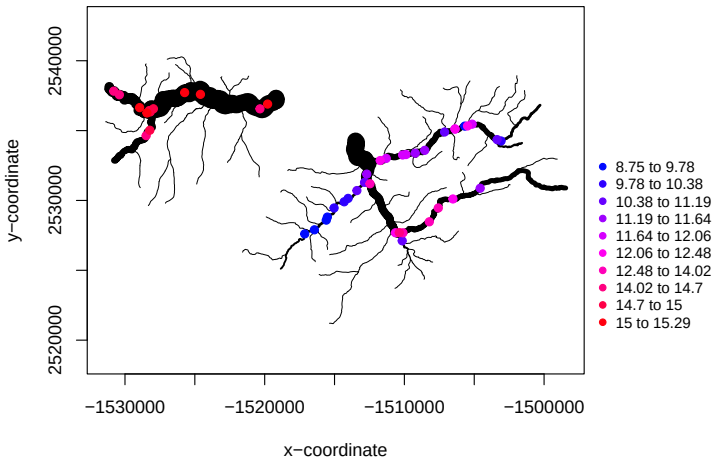

Figure 4: The generic plotting function of the `SpatialStreamNetwork` object for the Middle Fork data set. The observed values are represented by colored points and the stream network is shown in black, with the width of the lines proportional to the `afvArea` column.

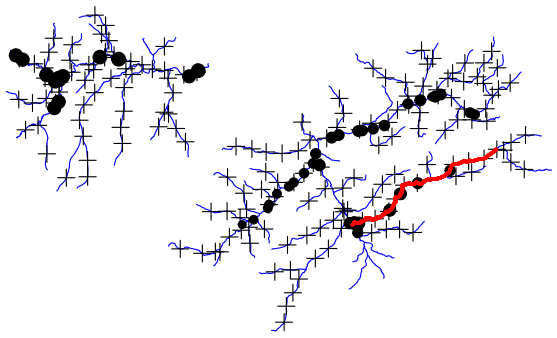

Figure 5: Plotting using **sp** classes.

only between flow-connected sites (e.g., for passive movement of chemical particles), while in other cases autocorrelation may be evident between flow-unconnected sites (e.g., for fish abundance because fish can move up-stream). Moreover, Section 2 showed models that are more natural for autocorrelation among flow-connected sites only (Subsection 2.2), those that also allow autocorrelation among flow-unconnected sites (Subsection 2.4), and variance component approaches that combine them, along with models based on Euclidean distance (Subsection 2.7). The Torgegram is a visual tool for evaluating autocorrelation separately for flow-connected and flow-unconnected sites, and can help inform the selection of candidate models for fitting. Further examples and discussion of the Torgegram are given in Peterson, Ver Hoef, Isaak, Falke, Fortin, Jordan, McNyset, Monestiez, Ruesch, Sengupta, Som, Steel, Theobald, Torgersen, and Wenger (2013).

Figure 6 is a plot of the Torgegram for the mean summer stream temperature:

```
mf04.Torg <- Torgegram(mf04p, "Summer_mn", nlag = 20, maxlag = 50000)
plot(mf04.Torg)
```

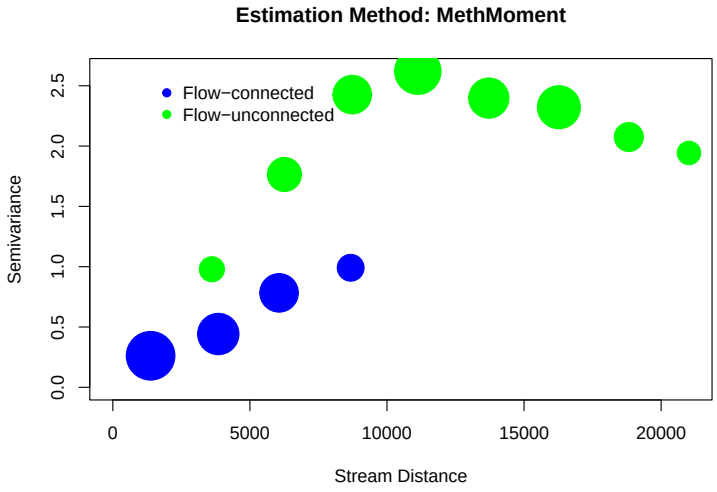

Figure 6: Torgegram of the mean summer temperature for the Middle Fork data set.

Close inspection of Figure 4 shows that neighboring locations on the stream network have very similar temperature values. Figure 6 shows that flow-connected sites have higher autocorrelation (lower semivariance) than flow-unconnected sites for the same distance when distances are short; however, both show high autocorrelation at short lags. Note that, for a stream network, there are longer in-stream distances when going from a head water to an outlet, and then back to a headwater (two flow-unconnected sites) than between a headwater and an outlet (two flow-connected sites). The availability of longer flow-unconnected distances is shown in Figure 4, where there are no flow-connected distances beyond 30000 m. In Figure 4, the semivariances increase to around 6 for flow-unconnected sites, and then start decreasing. Flow-unconnected distances of near 50000 are from headwater to outlet to headwater, and the Torgegram shows more similarity for these sites than those separated by 30000 m. This

suggests including a covariate such as elevation, or simply distance upstream. This makes sense for temperature which will be colder at the higher elevations of the headwaters. In summary, the Torgegram in Figure 6 suggests inclusion of elevation or distance upstream as a covariate in the linear model to account for an upstream trend in temperature, and based on the behavior of the Torgegram near the origin, both a tail-up and tail-down model will be necessary to capture the range of autocorrelation (Ver Hoef and Peterson 2010). Later, the Torgegram will be used on residuals for model diagnostics.

5.2. Model fitting

The `glmssn` function fits a spatial linear model (Equation 3) to a `SpatialStreamNetwork` object with a covariance structure shown in Equation 4. The ML or REML equations we optimized using the Nelder-Mead algorithm (Nelder and Mead 1965) in the `optim` function. Covariance matrices are inverted numerically, which means that the computing time for n observations increases proportionally to n^3, and is performed for each iteration of the ML or REML estimation method. This limits the size of data sets that can be fit by the `glmssn` function, and for most computing systems today we suggest sample sizes of less than 1000 observations.

An example of the `glmssn` function uses `Summer_mn` (mean summer stream temperature) as a response variable from in the Middle Fork data set. We include the covariates elevation and slope and various spatial covariance components. The response and fixed effects are specified using the `formula` argument, similar to `lm` and `glm`. The input data set is specified with the `data` argument, which must be a `SpatialStreamNetwork` object. The `CorModels` argument is a list of correlation models. These models should be of different types; i.e., at most one tail-up model, one tail-down model, and one Euclidean distance model. The `addfunccol` argument is the name of an additive function column, which is required if a tail-up correlation model is included in `CorModels`. More details can be found using `help(glmssn)`. A non-spatial model is fitted:

```
mf04.glmssn0 <- glmssn(Summer_mn ~ ELEV_DEM + SLOPE, mf04p,
   CorModels = NULL, use.nugget = TRUE)
summary(mf04.glmssn0)

##
## Call:
## glmssn(formula = Summer_mn ~ ELEV_DEM + SLOPE, ssn.object = mf04p,
##     CorModels = NULL, use.nugget = TRUE)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.3835 -1.1704  0.6205  0.9088  2.1930
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  62.839372  10.654190   5.898  < 2e-16 ***
## ELEV_DEM     -0.024955   0.005379  -4.639    3e-05 ***
## SLOPE       -88.019985  31.991343  -2.751  0.00872 **
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Covariance Parameters:
##  Covariance.Model Parameter Estimate
##            Nugget   parsill     1.81
##
## Residual standard error: 1.344093
## Generalized R-squared: 0.5625148
```

which can be compared to:

```
summary(lm(Summer_mn ~ ELEV_DEM + SLOPE, getSSNdata.frame(mf04p)))

##
## Call:
## lm(formula = Summer_mn ~ ELEV_DEM + SLOPE, data = getSSNdata.frame(mf04p))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.3835 -1.1704  0.6205  0.9088  2.1930
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  62.839372  10.654207   5.898 5.57e-07 ***
## ELEV_DEM     -0.024955   0.005379  -4.639 3.40e-05 ***
## SLOPE       -88.019985  31.991395  -2.751  0.00872 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.344 on 42 degrees of freedom
## Multiple R-squared:  0.5625,Adjusted R-squared:  0.5417
## F-statistic:    27 on 2 and 42 DF,  p-value: 2.886e-08
```

A spatial model, including a mixture of tail-up, tail-down, and Euclidean covariance models is fitted:

```
mf04.glmssn1 <- glmssn(Summer_mn ~ ELEV_DEM + SLOPE, mf04p,
   CorModels = c("Exponential.tailup", "Exponential.taildown",
      "Exponential.Euclid"), addfunccol = "afvArea")
summary(mf04.glmssn1)

##
## Call:
## glmssn(formula = Summer_mn ~ ELEV_DEM + SLOPE, ssn.object = mf04p,
##     CorModels = c("Exponential.tailup", "Exponential.taildown",
```

```
##         "Exponential.Euclid"), addfunccol = "afvArea")
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.1839 -1.8289 -0.4117  0.2991  1.3447
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  64.817126  10.793922   6.005   <2e-16 ***
## ELEV_DEM     -0.025756   0.005405  -4.765    2e-05 ***
## SLOPE       -27.325550  14.868674  -1.838   0.0732 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Covariance Parameters:
##      Covariance.Model Parameter     Estimate
##    Exponential.tailup   parsill      1.49392
##    Exponential.tailup     range 117789.73261
##  Exponential.taildown   parsill      0.00675
##  Exponential.taildown     range 117751.99113
##    Exponential.Euclid   parsill      0.07162
##    Exponential.Euclid     range  38068.34362
##                Nugget   parsill      0.01818
##
## Residual standard error: 1.261142
## Generalized R-squared: 0.4579287
```

It appears that slope is no longer a significant covariate when comparing fits of the independence model to that with spatial covariance. It is often true that autocorrelated models yield fewer factors with significant departures from zero.

The variable `MaxOver20` is a binary variable indicating if a stream temperature value was over 20^0 C. Here is an example of fitting a spatial generalized linear model (Section 2.8) to the binary data by using the `family = "binomial"` argument:

```
mf04.glmssnBin <- glmssn(MaxOver20 ~ ELEV_DEM + SLOPE, mf04p,
  CorModels = c("Mariah.tailup", "Spherical.taildown"),
  family = "binomial", addfunccol = "afvArea")
summary(mf04.glmssnBin)
```

```
##
## Call:
## glmssn(formula = MaxOver20 ~ ELEV_DEM + SLOPE, ssn.object = mf04p,
##     family = "binomial", CorModels = c("Mariah.tailup", "Spherical.taildown"),
##     addfunccol = "afvArea")
##
## Residuals:
```

```
##    Min     1Q Median     3Q    Max
## -3.001 -1.287 -1.070  1.312 11.614
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  59.16088   31.72727   1.865   0.0692 .
## ELEV_DEM     -0.02988    0.01607  -1.859   0.0701 .
## SLOPE       -74.43428  117.48688  -0.634   0.5298
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Covariance Parameters:
##    Covariance.Model Parameter   Estimate
##       Mariah.tailup   parsill    0.29556
##       Mariah.tailup     range    0.00213
##  Spherical.taildown   parsill    0.39666
##  Spherical.taildown     range 9627.05141
##              Nugget   parsill    0.29981
##
## Residual standard error: 0.9960069
## Generalized R-squared: 0.09534395
```

The variable `C16` represents the number of summer days when the stream temperature was greater than 16^o C. Here is an example of fitting a spatial generalized linear model (Section 2.8) to these count data by using the `family = "poisson"` argument:

```
mf04.glmssnPoi <- glmssn(C16 ~ ELEV_DEM + SLOPE, mf04p,
  CorModels = c("LinearSill.tailup", "LinearSill.taildown"),
  family = "poisson", addfunccol = "afvArea")
summary(mf04.glmssnPoi)
```

```
##
## Call:
## glmssn(formula = C16 ~ ELEV_DEM + SLOPE, ssn.object = mf04p,
##     family = "poisson", CorModels = c("LinearSill.tailup", "LinearSill.taildown"),
##     addfunccol = "afvArea")
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.00000 -0.18684 -0.01077  0.29539  1.25389
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  20.288563   6.763316   3.000  0.00453 **
## ELEV_DEM     -0.008512   0.003430  -2.482  0.01715 *
## SLOPE       -12.198263  12.545956  -0.972  0.33647
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Covariance Parameters:
##     Covariance.Model Parameter       Estimate
##    LinearSill.tailup   parsill     6.71742619
##    LinearSill.tailup     range 19572.57825779
##  LinearSill.taildown   parsill     0.98255198
##  LinearSill.taildown     range   216.29881981
##               Nugget   parsill     0.00000203
##
## Residual standard error: 2.774884
## Generalized R-squared: 0.179287
```

As one might expect, more high-temperature days are associated with lower elevations and lower slopes, although the slope covariate is not very significant.

5.3. Residuals and diagnostics

The `residuals` function computes a collection of residual diagnostics, including raw, standardized and Studentized residuals, as well as fitted values and spatial versions of leverage and Cook's D (Cook and Weisberg 1982). For all computations other than raw residuals, residual diagnostics use a modification of the classical formulas developed for independent data. Standard independence formulas are used for the linear model $\mathbf{y}^* = \mathbf{X}^*\boldsymbol{\beta} + \boldsymbol{\epsilon}^*$, where $\boldsymbol{\epsilon}^*$ are independent errors, after fitting the covariance matrix $\boldsymbol{\Sigma}$ and then "creating" independent data as $\mathbf{y}^* = \boldsymbol{\Sigma}^{-1/2}\mathbf{y}$ with a modified design matrix $\mathbf{X}^* = \boldsymbol{\Sigma}^{-1/2}\mathbf{X}$, where $\boldsymbol{\Sigma}$ was defined in Equation 4. The result of the `residuals` function is an `influenceSSN` object, which is an exact copy of the `glmssn` object, except that residual diagnostics are appended as new columns to the `point.data data.frame` containing the observed data. The default plotting method for an `influenceSSN` object is a map with color-coded raw residuals:

```
mf04.resid1 <- residuals(mf04.glmssn1)
names( getSSNdata.frame(mf04.resid1) )
```

```
##  [1] "pid"            "rid"            "STREAMNAME"
##  [4] "HUC3"           "HUC4"           "COMID"
##  [7] "CUMDRAINAG"     "AREAWTMAP"      "MAXELEVSMO"
## [10] "SLOPE"          "NCEASID_"       "ELEV_DEM"
## [13] "Deployment"     "SampleYear"     "NumberOfDa"
## [16] "OriginalID"     "Source"         "Summer_mn"
## [19] "MaxOver20"      "C16"            "C20"
## [22] "C24"            "FlowCMS"        "AirMEANc"
## [25] "AirMWMTc"       "NEAR_FID"       "NEAR_DIST"
## [28] "NEAR_X"         "NEAR_Y"         "NEAR_ANGLE"
## [31] "ratio"          "afvArea"        "upDist"
## [34] "locID"          "netID"          "computed.afv"
```

```
## [37] "obsval"          "_fit_"           "_resid_"
## [40] "_resid.stand_"   "_resid.student_" "_leverage_"
## [43] "_CooksD_"
```

```
plot(mf04.resid1)
```

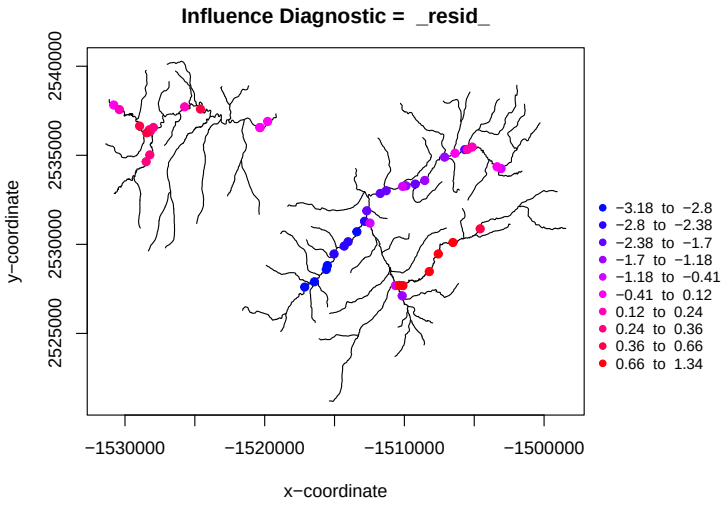

Figure 7: Plot of the raw residuals from fitting a model to the Middle Fork stream network. The legend on the right-hand side hints at the existence of residuals whose value are less than −3.

Columns added by the `residuals` function have names of the form `_*_`. Maps of other diagnostics computed by `residuals` can be plotted by using the `inflcol` argument.

Figure 7 indicates that there might be an outlier with a residual less than −3. Histograms can be plotted of the raw response variable and residuals. (Figure 8):

```
par(mfrow = c(1, 2))
hist(mf04.resid1)
hist(mf04p, "Summer_mn")
```

Let us treat the residual that is less than −3 as an outlier. We handle outliers by replacing their values with `NA`, which will allow us to predict it later. First extract the `data.frame` of observed data and identify the record associated with the outlier, insert an `NA` in the appropriate position for the response variable, and then put the `data.frame` back into the spatial stream network object:

```
ObsDFr <- getSSNdata.frame(mf04.resid1)
ObsDF <- getSSNdata.frame(mf04p)
indOutlier <- ObsDFr["_resid_"] < -3
```

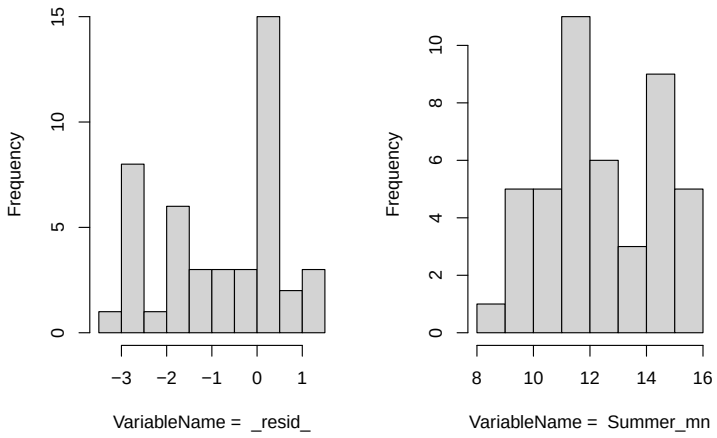

Figure 8: Histogram of the residuals (on the left) and a histogram of the average summer temperature values (on the right).

```
ObsDF[indOutlier, "Summer_mn"] <- NA
mf04c <- putSSNdata.frame(ObsDF, mf04p)
```

The new `SpatialStreamNetwork` object was renamed `mf04c`. Note that the outlier has been replaced in the `SpatialStreamNetwork` object, but not in the original data set in the `.ssn` directory. Having dealt with the outlier we refit the basic spatial model to the mean summer temperature, this time using ML rather than REML. We use ML because later we will use AIC for model selection, and REML cannot be used with AIC when fixed effects are changing (Verbeke and Molenberghs 2000, p. 75).

```
mf04c.glmssn0 <- glmssn(Summer_mn ~ ELEV_DEM + SLOPE, mf04c,
   CorModels = c("Exponential.tailup", "Exponential.taildown",
   "Exponential.Euclid"), addfunccol = "afvArea", EstMeth = "ML")
summary(mf04c.glmssn0)

##
## Call:
## glmssn(formula = Summer_mn ~ ELEV_DEM + SLOPE, ssn.object = mf04c,
##     CorModels = c("Exponential.tailup", "Exponential.taildown",
##         "Exponential.Euclid"), addfunccol = "afvArea", EstMeth = "ML")
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##      NA -1.7829 -0.3289  0.3392      NA
##
## Coefficients:
```

```
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  64.29461   10.31964   6.230   <2e-16 ***
## ELEV_DEM     -0.02551    0.00517  -4.934    1e-05 ***
## SLOPE       -23.25010   15.03325  -1.547     0.13
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Covariance Parameters:
##      Covariance.Model Parameter    Estimate
##    Exponential.tailup   parsill      1.3481
##    Exponential.tailup     range 110976.5015
##  Exponential.taildown   parsill      0.0485
##  Exponential.taildown     range 117357.4743
##    Exponential.Euclid   parsill      0.0180
##    Exponential.Euclid     range  23667.8553
##                Nugget   parsill      0.0177
##
## Residual standard error: 1.196797
## Generalized R-squared: 0.4416391
```

The partial sill associated with the exponential Euclidean model is quite low compared to the partial sills for the tail-up and tail-down models, as well as the nugget effect, and the fixed effect for `SLOPE` is also not significant, so they are dropped when refitting the model:

```
mf04c.glmssn1 <- glmssn(Summer_mn ~ ELEV_DEM, mf04c,
   CorModels = c("Exponential.tailup", "Exponential.taildown"),
   addfunccol = "afvArea", EstMeth = "ML")
summary(mf04c.glmssn1)

##
## Call:
## glmssn(formula = Summer_mn ~ ELEV_DEM, ssn.object = mf04c, CorModels = c("Exponential.t
##     "Exponential.taildown"), addfunccol = "afvArea", EstMeth = "ML")
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##      NA -1.9503 -0.4230  0.0913      NA
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 69.876077  10.001403   6.987   <2e-16 ***
## ELEV_DEM    -0.028273   0.005004  -5.650   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Covariance Parameters:
```

```
##      Covariance.Model Parameter      Estimate
##    Exponential.tailup   parsill      1.815899
##    Exponential.tailup     range 117791.352999
##  Exponential.taildown   parsill      0.000795
##  Exponential.taildown     range    226.290371
##                Nugget   parsill      0.004710
##
## Residual standard error: 1.349594
## Generalized R-squared: 0.420658
```

Leave-one-out cross validation (LOOCV) provides a good diagnostic for evaluating model performance. The function `CrossValidationSSN` computes LOOCV predictions and standard errors for a fitted `glmssn` object. The LOOCV predictions and standard errors are included by default when using the `residual` function; column names `_CrossValPred_` and `_CrossValStdErr_` are added to the observed `point.data data.frame`, in addition to the column `_resid.crossv_` which is the observed value subtracted from `_CrossValPred_`. These variables can be mapped as well. Figure 9 plots these LOOCV predictions and prediction standard errors against the observed data.

```
cv.out <- CrossValidationSSN(mf04c.glmssn1)
par(mfrow = c(1, 2))
plot(mf04c.glmssn1$sampinfo$z,
   cv.out[, "cv.pred"], pch = 19,
   xlab = "Observed Data", ylab = "LOOCV Prediction")
abline(0, 1)
plot( na.omit( getSSNdata.frame(mf04c)[, "Summer_mn"]),
   cv.out[, "cv.se"], pch = 19,
   xlab = "Observed Data", ylab = "LOOCV Prediction SE")
```

The function `CrossValidationStatsSSN` both computes and summarises the cross-validation statistics for a particular `glmssn` object. Bias, root-mean-squared prediction error, and confidence interval coverage are computed.

```
CrossValidationStatsSSN(mf04c.glmssn1)
```

```
##         bias   std.bias     RMSPE       RAV std.MSPE    cov.80
## 1 0.07892529 0.07445568 0.5467495 0.3680389 1.180716 0.7954545
##      cov.90    cov.95
## 1 0.8409091 0.9090909
```

The `GR2` function computes a generalised R-squared for the fitted `glmssn` object by computing the classical R-squared on the $\mathbf{y}^* = \mathbf{X}^*\boldsymbol{\beta} + \boldsymbol{\epsilon}^*$ model.

The `varcomp` function takes the proportion of variation not explained by the fixed effects (covariates) in `GR2`, and apportions it by the partial sill of each spatial covariance component.

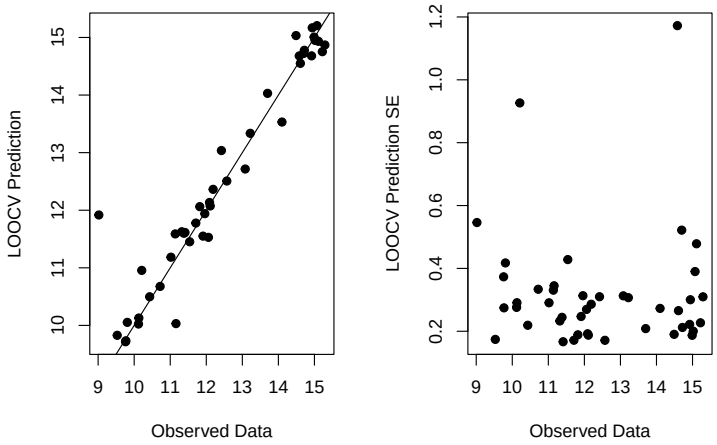

Figure 9: Leave-one-out cross validation predictions (on the left) and prediction standard errors (on the right) plotted against the observed response values.

```
GR2(mf04c.glmssn1)

##           [,1]
## [1,] 0.420658

varcomp(mf04c.glmssn1)

##                 VarComp  Proportion
## 1    Covariates (R-sq) 0.420658045
## 2   Exponential.tailup 0.577590714
## 3 Exponential.taildown 0.000252963
## 4               Nugget 0.001498278
```

Here the fixed effects explains about 42% of the variation in the data, with the tail-up model contributing 57% and very little from the tail-down and independent nugget component.

5.4. Model selection

Prior knowledge on the best model is rarely available, especially for covariance structures, and so we often fit several and then compare them in various ways. We fit `mf04c.glmssn0` and `mf04c.glmssn1` using ML so that they could be compared using the Akaike Information Criteria (AIC):

```
AIC(mf04c.glmssn0)

## [1] 72.24983
```

```
AIC(mf04c.glmssn1)

## [1] 66.74482
```

It is clear that `SLOPE` is not a significant covariate. Here, we re-fit `mf04c.glmssn0` with REML (the default, so the `EstMeth = "ML"` argument can be dropped), along with a few more covariance structures:

```
mf04c.glmssn1 <- glmssn(Summer_mn ~ ELEV_DEM, mf04c,
   CorModels = c("Exponential.tailup", "Exponential.taildown"),
   addfunccol = "afvArea")
mf04c.glmssn2 <- glmssn(Summer_mn ~ ELEV_DEM,  mf04c,
   CorModels = c("LinearSill.tailup", "Mariah.taildown"),
   addfunccol = "afvArea")
mf04c.glmssn3 <- glmssn(Summer_mn ~ ELEV_DEM , mf04c,
   CorModels =  c("Mariah.tailup", "LinearSill.taildown"),
   addfunccol = "afvArea")
mf04c.glmssn4 <- glmssn(Summer_mn ~ ELEV_DEM, mf04c,
   CorModels = c("Spherical.tailup", "Spherical.taildown"),
   addfunccol = "afvArea")
mf04c.glmssn5 <- glmssn(Summer_mn ~ ELEV_DEM, mf04c,
   CorModels = "Exponential.Euclid",
   addfunccol = "afvArea")
```

Several different spatial models can be compared using the `InfoCritCompare` command. This function extracts the AIC from each model fit, evaluates the cross validation statistics for each model and presents the results in table format:

```
options(digits = 4)
InfoCritCompare(list(mf04c.glmssn1, mf04c.glmssn2,
   mf04c.glmssn3, mf04c.glmssn4, mf04c.glmssn5))

##                formula EstMethod
## 1 Summer_mn ~ ELEV_DEM      REML
## 2 Summer_mn ~ ELEV_DEM      REML
## 3 Summer_mn ~ ELEV_DEM      REML
## 4 Summer_mn ~ ELEV_DEM      REML
## 5 Summer_mn ~ ELEV_DEM      REML
##                                  Variance_Components neg2LogL    AIC
## 1 Exponential.tailup + Exponential.taildown + Nugget    62.68  72.68
## 2       LinearSill.tailup + Mariah.taildown + Nugget    56.07  66.07
## 3       Mariah.tailup + LinearSill.taildown + Nugget    86.78  96.78
## 4     Spherical.tailup + Spherical.taildown + Nugget    57.42  67.42
## 5                         Exponential.Euclid + Nugget   126.54 132.54
##         bias   std.bias  RMSPE    RAV std.MSPE cov.80 cov.90 cov.95
## 1  0.0703594  0.0661302 0.5300 0.3865   1.0445 0.7955 0.8864 0.9318
```

```
## 2  0.0759843  0.0680604 0.5042 0.3797   0.9103 0.7955 0.9091 0.9318
## 3  0.0570522  0.0569083 0.5053 0.5044   0.7416 0.8409 0.9318 0.9545
## 4  0.0786681  0.0713283 0.5262 0.3668   1.1189 0.7727 0.8182 0.9091
## 5 -0.0008186 -0.0004457 0.7863 0.8548   0.9261 0.8864 0.9091 0.9318

options(digits = 7)
```

Note that AIC can be used with REML here because the fixed effects are not changing among models. The comparisons show that all pure stream network models do much better than one based on Euclidean distance. Based on the low AIC value and the low root-mean-square prediction error we decide on a final model that uses linear-with-sill tailup and mariah taildown covariance structures. Sample sizes are rather low, but prediction intervals seem appropriate. Here is summary of the final fitted model:

```
summary(mf04c.glmssn2)

##
## Call:
## glmssn(formula = Summer_mn ~ ELEV_DEM, ssn.object = mf04c, CorModels = c("LinearSill.ta
##     "Mariah.taildown"), addfunccol = "afvArea")
##
## Residuals:
##      Min       1Q   Median       3Q      Max
##       NA -2.06272 -0.55153 -0.01528       NA
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 69.500614   9.110772   7.628   <2e-16 ***
## ELEV_DEM    -0.028026   0.004546  -6.166   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Covariance Parameters:
##   Covariance.Model Parameter     Estimate
##  LinearSill.tailup   parsill      2.55170
##  LinearSill.tailup     range 116389.88234
##    Mariah.taildown   parsill      0.00145
##    Mariah.taildown     range  54764.16600
##             Nugget   parsill      0.02348
##
## Residual standard error: 1.605188
## Generalized R-squared: 0.5012196
```

In addition to outliers from a fitted model (i.e., global outliers), outliers may exist with respect to predictions that do not appear as large deviations from the fit. A histogram of standardized LOOCV prediction residuals can identify local prediction outliers (Figure 10):

```
mf04c.resid2 <- residuals(mf04c.glmssn2,
    cross.validation = TRUE)
mf04c.resid2.cv.std <-
    getSSNdata.frame(mf04c.resid2)[, "_resid.crossv_"] /
    getSSNdata.frame(mf04c.resid2)[, "_CrossValStdErr_"]
hist(mf04c.resid2.cv.std)
```

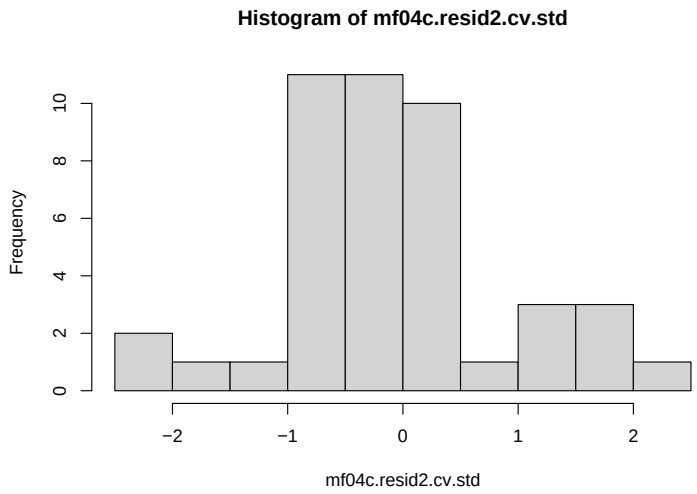

Figure 10: Histogram of standardized cross-validation residuals for model `mf04c.glmssn2` of mean summer stream temperature for the Middle Fork stream network with `ELEV_DEM` covariate and spherical tail-up, spherical tail-down and nugget covariance components.

When fitting covariates, spatial autocorrelation is modeled on the errors, so a Torgegram of the residuals helps visualize the spatial pattern after fitting the fixed effects. Note that this is imperfect because the fitted model is not a simple function of distance; it is complicated by weighting for tail-up models, and by asymmetry in distances for tail-down models (see Ver Hoef and Peterson 2010, Figure 7). A Torgegram on the residuals of `mf04c.glmssn2` is given in Figure 11:

```
plot(Torgegram(mf04c.resid2, "_resid_", nlag = 8, maxlag = 25000))
```

Figure 11 shows apparent autocorrelation for both flow-connected and flow-unconnected sites, although flow-unconnected sample sizes are small for short lags, supporting the decision to use both tail-up and tail-down models, even though the partial sill for tail-down model contributes little to the overall variability of the error. The overall sill is modeled to be slightly greater than 2.5.

5.5. Prediction

Two functions for prediction are included in the `SSN` package: `predict` and `BlockPredict`, which are also known as (universal) kriging and (universal) block kriging, respectively.

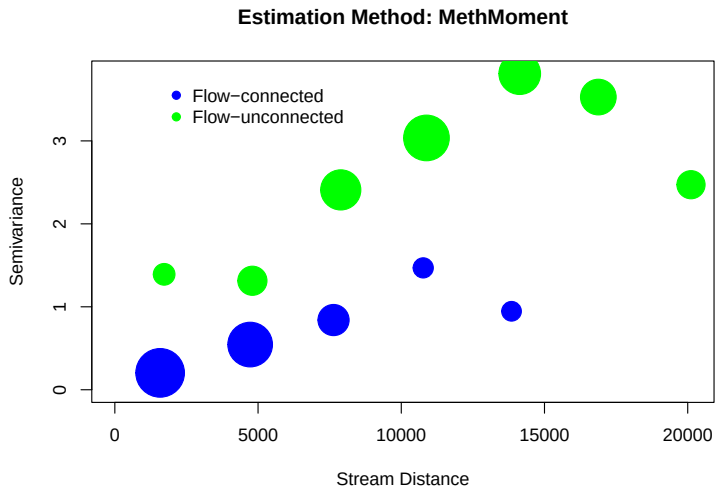

Figure 11: Torgegram of residuals for model `mf04c.glmssn2` of mean summer temperature in the Middle Fork stream network with `ELEV_DEM` covariate and spherical tail-up, spherical tail-down and nugget covariance components.

As a first example, we predict at point locations in the `pred1km` data set (Figure 12). The size of the prediction points indicates the prediction standard errors; larger points have lower prediction standard errors, so if we are more confident in a point it stands out more in the graphic. Previously defined breakpoints are used here, so the colours in Figure 12 match those found earlier in Figure 4.

```
mf04c.pred1km <- predict(mf04c.glmssn4, "pred1km")
plot(mf04c.pred1km, SEcex.max = 1, SEcex.min = .5/3*2,
     breaktype = "user", brks = brks)
```

The second prediction data set is made up of a dense set of points along a single tributary, called Knapp Creek, of the Middle Fork river. We predict all points using the `predict` function. In Figure 13 we first plotted the observed values with large open circles in the vicinity of the Knapp tributary, with values color-coded. We then added the Knapp predictions, which use the same break points as the first plot. Note that text is automatically added for the lowest and highest predicted values:

```
plot(mf04c, "Summer_mn", pch = 1, cex = 3,
   xlab = "x-coordinate", ylab = "y-coordinate",
   xlim = c(-1511000,-1500000), ylim = c(2525000,2535000))
mf04c.glmssn4.Knapp <- predict(mf04c.glmssn4, "Knapp")
plot(mf04c.glmssn4.Knapp, "Summer_mn", add = TRUE,
   xlim = c(-1511000,-1500000), ylim = c(2525000,2535000))
```

By using matching break points for observed data and predicted data, we can see that predictions seem reasonable given the observed data.

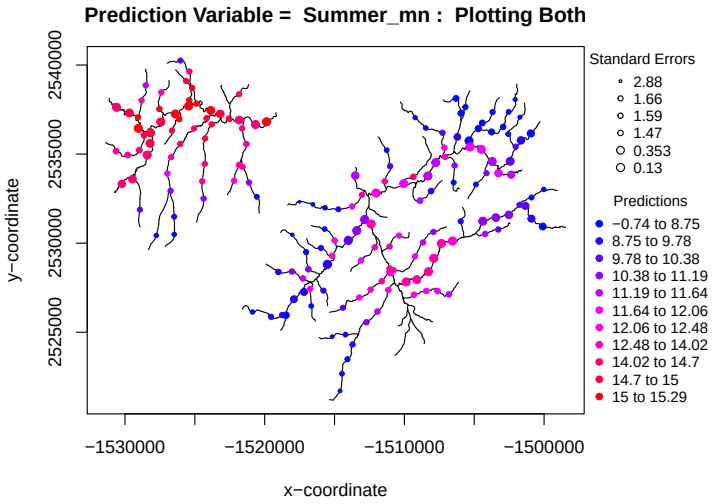

Figure 12: Predictions of mean summer temperature values for the Middle Fork stream network. Predicted values are indicated by the color of the circles. The size of the circles is inversely related to the prediction standard error.

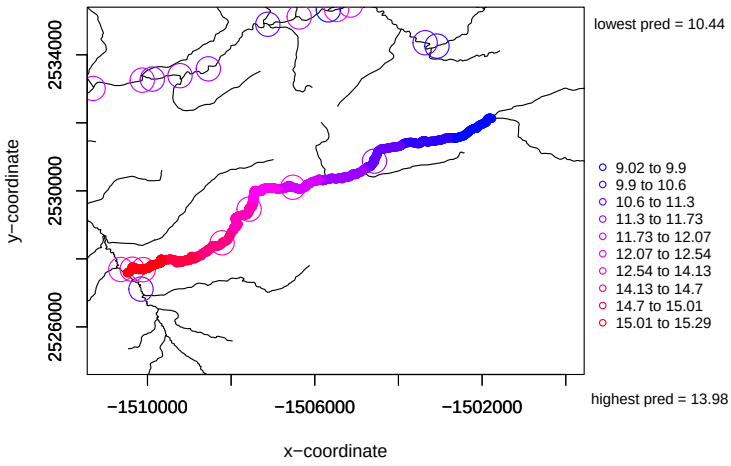

Figure 13: Predictions of mean summer stream temperature values along Knapp Creek. Predicted values are filled circles and observed data are large open circles. Color breaks are based on the full data set, but the map is zoomed-in to Knapp Creek.

The prediction sites are on a dense evenly-spaced grid because they are used to approximate the integrals involved with block prediction:

```
mf04c.glmssn4.BPKnapp <- BlockPredict(mf04c.glmssn4, "Knapp")
mf04c.glmssn4.BPKnapp

##   BlockPredEst BlockPredSE
## 1      12.3197  0.09913953
```

We can repeat this for another set of spatially dense locations on the Cape Horn tributary of the Middle Fork river:

```
mf04c.glmssn4.BPCapeHorn <- BlockPredict(mf04c.glmssn4, "CapeHorn")
mf04c.glmssn4.BPCapeHorn

##   BlockPredEst BlockPredSE
## 1     10.02122   0.0477452
```

The predicted block average values, along with the standard errors of block prediction, allow comparison between these two reaches.

Finally, recall that we replaced an outlier value with `NA` when creating `mf04c` from `mf04`. When fitting a model with `glmssn`, records with `NA` response values are used to create a new prediction data set, called `_MissingObs_`, in the fitted `glmssn` object. `_MissingObs_` is like any other prediction data set and can be used to predict the `NA`s. We compare the original outlier value with this prediction:

```
mf04c.missingobs <- predict(mf04c.glmssn4, "_MissingObs_")
getPreds(mf04c.missingobs, pred.type = "pred")

##      pid Summer_mn Summer_mnPredSE
## [1,]  29  9.668828       0.7183271

with(getSSNdata.frame(mf04p), Summer_mn[pid==29])

## [1] 8.75
```

Notice that the original value is within the 95% prediction interval at that point, so it probably was not an outlier.

6. Simulating stream network data

The **SSN** package can be used to simulate stream network data, which involves two steps: 1) creating a `SpatialStreamNetwork` object, and 2) simulating an autocorrelated response variable for a `SpatialStreamNetwork` object. These functions can be used for testing methods, comparing sampling methods, etc., and were used extensively for testing functions when developing the **SSN** package.

6.1. Creating a `SpatialStreamNetwork` object

The **SSN** package provides the function `createSSN` for constructing `SpatialStreamNetwork` objects based on randomly generated networks, with randomly generated observation and prediction points. There are two approaches to generating a random network. First, notice that stream networks are tree structures that are a special type of graph, so random graph functions in R can be used to generate such a structure. However, these functions are not spatially explicit so every vertex requires a position in the plane for a graph-drawing algorithm, which is an additional step. The second approach is to write more customised code that generates a network and assigns positions to vertices while generating the network.

The first approach has several downsides; few of the methods for generating random graphs can be used to generate tree structures, and existing graph-drawing algorithms tend to give a highly regular network structure or a structure with self-intersections, which we do not consider in our models of river systems. The second approach has none of these downsides but may be slower computationally. Both approaches are implemented in the **SSN** package.

The `createSSN` function has the form

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

The argument `n` is a integer vector where the length of the vector is the number of networks and each integer is the number of stream segments per network. The `path` argument gives the full path name where the `.ssn` directory associated with the new `SpatialStreamNetwork` object will be stored, and `importToR` determines whether the created object will be loaded and returned directly from this function. The arguments `obsDesign` and `predDesign` specify sampling design functions that allow the user to control how the observation and prediction points are generated. The simplest input is a single point for `obsDesign` and no prediction points `predDesign = noPoints`, which generates no points.

The argument `treeFunction` controls how the random tree structure is generated. There are currently two possible values, with the default of `igraphKamadaKawai` taking the first approach above and using random graph methods. We use the **igraph** package and the Kamada-Kawai graph drawing algorithm from the same package (Csardi and Nepusz 2006). The second possible value is `iterativeTreeLayout` which can produce more realistic networks and is guaranteed not to create any self-intersections. An example of the types of networks generated by this layout is given in Figure 14.

A number of design functions are provided, including

- `poissonDesign(lambda)`
- `hardCoreDesign(n, inhibition_region)`
- `systematicDesign(spacing)`
- `binomialDesign(n)`.

Input `lambda` is a numeric vector specifying (on a per-network basis) the rate of occurance of points for the Poisson process, while for the binomial design `n` specifies the number of points

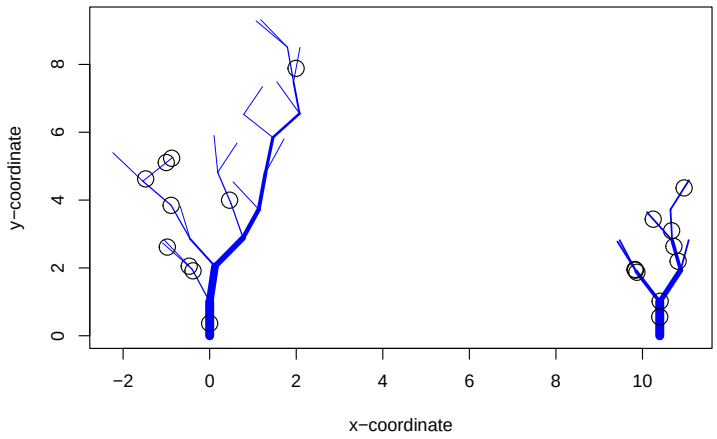

Figure 14: The network is generated using `iterativeTreeLayout` and the points are generated using the `hardCoreDesign` function.

to be generated from a uniform distribution across each network. The `systematicDesign` function is more complicated, and is intended to generate a set of regular points, especially useful for block prediction. Note that in our implementation for networks this means every point is a constant distance from the next most downstream point, so points just upstream of a junction may be very close to each other, and on visual inspection the points may not appear to be equally spaced. This constant spacing is specified by the input `inhibition_region`. The `hardCoreDesign` generates `n` randomly distributed points on each network, and then removes points until the remainder are all at least `inhibition_region` distant from each other. Examples of `systematicDesign` and `hardCoreDesign` are given in Figures 15 and 16 respectively.

```
set.seed(101)
raw.ssn <- createSSN(n = c(10, 10, 10),
   obsDesign = binomialDesign(c(40, 40, 40)),
   predDesign = systematicDesign(c(0.2, 0.4, 0.8)), importToR = TRUE,
   path = "./raw.ssn")
plot(raw.ssn, lwdLineCol = "addfunccol", lwdLineEx = 8,
   lineCol = "blue", cex = 2, xlab = "x-coordinate",
   ylab = "y-coordinate", pch = 1)
plot(raw.ssn, PredPointsID = "preds", add = TRUE, cex = .5, pch = 19,
   col = "green")
```

```
set.seed(13)
hardcore.ssn <- createSSN(n = c(10, 10),
   obsDesign = hardCoreDesign(c(200, 200), c(0.2, 0.4)),
   importToR = TRUE, path = "./SimHardcore.ssn")
```

```
plot(hardcore.ssn, lwdLineCol = "addfunccol", lwdLineEx = 8,
   lineCol = "blue", cex = 2, xlab = "x-coordinate",
   ylab = "y-coordinate", pch = 1)
plot(hardcore.ssn, PredPointsID = NULL, add = TRUE, cex = .5,
   pch = 19, col = "green")
```

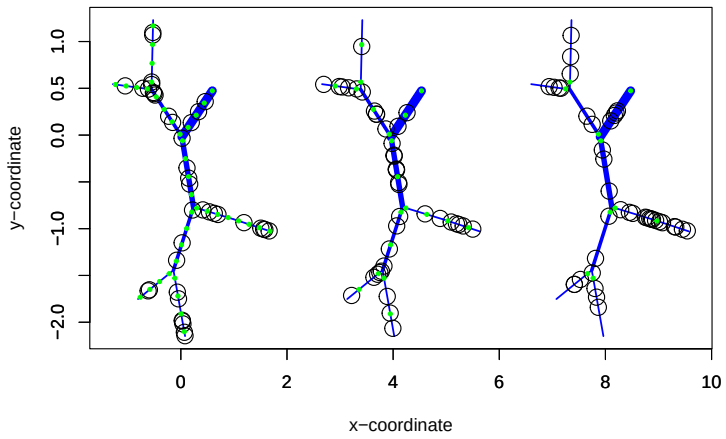

Figure 15: Three simulated stream networks. The blue lines get thinner farther up-stream. The open black circles are observed locations, and the smaller solid green points are prediction locations generated by `systematicDesign`.

While there are only four design functions built into the package, it is possible for the user to construct their own sampling design functions, and then generate location data using the `createSSN` function.

6.2. Simulating data on the `SpatialStreamNetwork` object

After creating a `SpatialStreamNetwork` object (e.g., Figure 15), we may want to simulate on both observed and prediction locations. This functionality is useful for developing and testing new statistical methods and code, as well as for model diagnostics for real data, such as examining realized patterns of hypothetical or fitted models. First, the distance matrices among all points is created:

```
createDistMat(raw.ssn, "preds", o.write=TRUE, amongpred = TRUE)
```

Note the `amongpred` argument. For simulation, we need a covariance matrix (computed from the distance matrix) among prediction locations, which is typically not needed for spatial modeling of observed locations or predicting at prediction locations. This matrix can be quite large so some care should be taken about its size, otherwise R will run out of memory or predictions will take a long time to compute. The first step is to extract the `point.data`

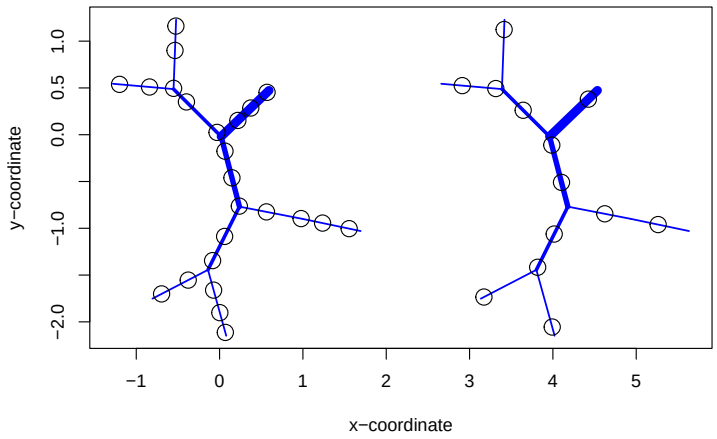

Figure 16: Points generated by `hardCoreDesign`, with varying inhibition regions. The blue lines get thinner farther up-stream. The open black circles are simulated locations, which have a more regular pattern than the spatially random locations shown by the open circles in Figure 15.

`data.frame`s for the observed and prediction locations from the `SpatialStreamNetwork` object:

```
rawDFobs <- getSSNdata.frame(raw.ssn, "Obs")
rawDFpred <- getSSNdata.frame(raw.ssn, "preds")
```

Continuous covariates are created in each `data.frame`:

```
rawDFobs[,"X1"] <- rnorm(length(rawDFobs[,1]))
rawDFpred[,"X1"] <- rnorm(length(rawDFpred[,1]))
rawDFobs[,"X2"] <- rnorm(length(rawDFobs[,1]))
rawDFpred[,"X2"] <- rnorm(length(rawDFpred[,1]))
```

Categorical covariates can also be created:

```
rawDFobs[,"F1"] <- as.factor(sample.int(4,length(rawDFobs[,1]),
   replace = TRUE))
rawDFpred[,"F1"] <- as.factor(sample.int(4,length(rawDFpred[,1]),
   replace = TRUE))
```

Random effects are created just like a categorical covariates; in this example one is created with three levels and another with four levels:

```
rawDFobs[,"RE1"] <- as.factor(sample(1:3,length(rawDFobs[,1]),
   replace = TRUE))
rawDFobs[,"RE2"] <- as.factor(sample(1:4,length(rawDFobs[,1]),
   replace = TRUE))
rawDFpred[,"RE1"] <- as.factor(sample(1:3,length(rawDFpred[,1]),
   replace = TRUE))
rawDFpred[,"RE2"] <- as.factor(sample(1:4,length(rawDFpred[,1]),
   replace = TRUE))
```

The list of column names for the `data.frames` now includes the new covariates:

```
names(rawDFobs)

##  [1] "locID"      "upDist"     "pid"        "netID"      "rid"
##  [6] "ratio"      "shreve"     "addfunccol" "X1"         "X2"
## [11] "F1"         "RE1"        "RE2"

names(rawDFpred)

##  [1] "locID"      "upDist"     "pid"        "netID"      "rid"
##  [6] "ratio"      "shreve"     "addfunccol" "X1"         "X2"
## [11] "F1"         "RE1"        "RE2"
```

For prediction, it is important to make sure that the observed and prediction `data.frames` have the same set of columns for any specification of random or fixed effects.

Note that the creation of the `SpatialStreamNetwork` object included Shreve's stream order, and the additive function value is based on Shreve's and is contained in the `addfunccol` column for each location, which is required for tail-up models as described in Section 2.2.

To simulate data on the SpatialStreamNetwork object, use the `SimulateOnSSN` function:

```
set.seed(102)
sim.out <- SimulateOnSSN(raw.ssn, ObsSimDF = rawDFobs,
   PredSimDF = rawDFpred, PredID = "preds",
   formula = ~ X1 + X2 + F1, coefficients = c(10,1,0,-2,0,2),
   CorModels = c("LinearSill.tailup", "Mariah.taildown",
   "Exponential.Euclid", "RE1", "RE2"), use.nugget = TRUE,
   CorParms = c(3, 10, 2, 10, 1, 5, 1, .5, .1),
   addfunccol = "addfunccol")
```

The `ObsSimDF` argument replaces the observed `point.data data.frame` in `raw.ssn` with `rawDFobs`. For that reason, it is best to use `getSSNdata.frame` to extract the original observed `point.data data.frame` in `raw.ssn` to make sure it complies with the object structure, and then modify its covariates if desired. Likewise, the `PredSimDF` argument replaces the prediction `point.data data.frame` in `raw.ssn` with `rawDFpred`. The function `SimulateOnSSN` simulates and stores data in a new column, `"Sim_Values"` in both `point.data data.frames`.

The one-sided formula specifies how the fixed effects are computed, which works just like an R formula in other functions. A design matrix $\mathbf{X}$ is created from the `formula` input, and the mean value is computed as $\boldsymbol{\mu} = \mathbf{X}\boldsymbol{\beta}$. Hence, coefficients $\boldsymbol{\beta}$ must be specified using the `coefficients` argument. Some understanding of how R creates design matrices is necessary in order to apply the coefficients properly. In our example, the first column of the design matrix $\mathbf{X}$ will be all ones for an overall intercept, as that is the default in a formula specification. The next two columns will contain the covariate values of `X1` and `X2`. Because there is an overall intercept, the model is over-parameterized for the categorical covariate and so the first level of `F1` is dropped and 0-1 dummy variables are created for the other three levels. The column names of the design matrix can be examined

```
with(rawDFobs, colnames(model.matrix( ~ X1 + X2 + F1)))

## [1] "(Intercept)" "X1"          "X2"          "F12"
## [5] "F13"         "F14"
```

or the whole matrix can be examined by removing the `colnames` function. The `coefficients` argument creates the $\boldsymbol{\beta}$ in the order given in the argument and multiplies it with the design matrix created by the `formula` argument.

Likewise, the rules for covariance matrix construction require careful consideration. Regardless of the order of `CorModel` specification, the subset of parameters will be in the order of $\boldsymbol{\theta} = (\sigma^2_u, \alpha_u, \sigma^2_d, \alpha_d, \sigma^2_e, \alpha_e, \sigma^2_1, \ldots, \sigma^2_p, \sigma^2_0)$, so the `CorParms` argument should be used to match this order. Note that there can only be a single tail-up model, a single tail-down model, and a single Euclidean distance model. Any factor variable can be used to create random effects, and the inclusion of the `nugget = TRUE` argument includes σ^2_0 in the covariance model.

The output of the `SimulateOnSSN` function is a list with three objects. Two of those objects simply verify that the `coefficients` and `CorParms` were used as intended. For our example:

```
sim.out$FixedEffects

##        Xnames Coefficient
## 1 (Intercept)          10
## 2          X1           1
## 3          X2           0
## 4         F12          -2
## 5         F13           0
## 6         F14           2

sim.out$CorParms

##             CorModel    type Parameter
## 1  LinearSill.tailup parsill       3.0
## 2  LinearSill.tailup   range      10.0
## 3    Mariah.taildown parsill       2.0
## 4    Mariah.taildown   range      10.0
## 5 Exponential.Euclid parsill       1.0
```

```
## 6 Exponential.Euclid   range       5.0
## 7                RE1 parsill       1.0
## 8                RE2 parsill       0.5
## 9             nugget parsill       0.1
```

The simulated SpatialStreamNetwork object can be extracted:

```
sim.ssn <- sim.out$ssn.object
```

and plotting yields Figure 17:

```
plot(sim.ssn, "Sim_Values",
   xlab = "x-coordinate", ylab = "y-coordinate",
   cex = 1.5)
```

which shows simulated values at the locations created in Figure 15. Note that response values were also simulated at prediction locations in Figure 15, but are not shown.

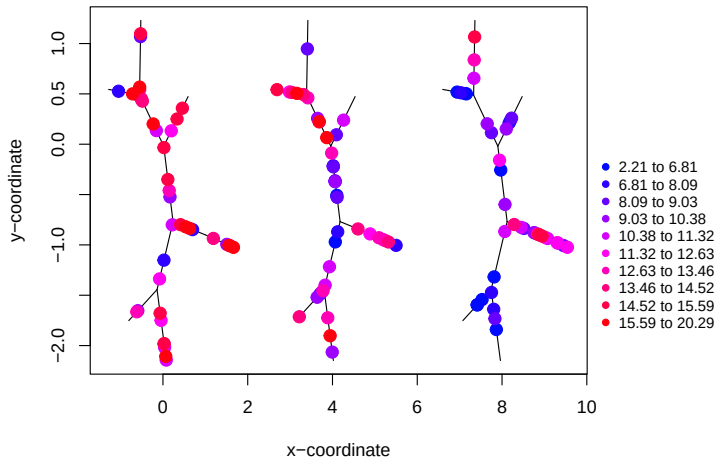

Figure 17: Values for the response variable were simulated at the simulated observed locations shown in Figure 15. The simulated data are colored by their values, showing an autocorrelated response.

To test the function, extract the observed and predicted `point.data data frames`, which now contain simulated values:

```
simDFobs <- getSSNdata.frame(sim.ssn, "Obs")
simDFpred <- getSSNdata.frame(sim.ssn, "preds")
```

To test prediction, we stored the simulated prediction values and replaced them with `NA`s:

```
simpreds <- simDFpred[,"Sim_Values"]
simDFpred[,"Sim_Values"] <- NA
sim.ssn <- putSSNdata.frame(simDFpred, sim.ssn, "preds")
```

We fit a model to the simulated data to see how well the simulation parameters are estimated:

```
glmssn.out <- glmssn(Sim_Values ~ X1 + X2 + F1, sim.ssn,
   CorModels = c("LinearSill.tailup", "Mariah.taildown",
   "Exponential.Euclid", "RE1", "RE2"),
   addfunccol = "addfunccol")
```

yielding output:

```
summary(glmssn.out)

##
## Call:
## glmssn(formula = Sim_Values ~ X1 + X2 + F1, ssn.object = sim.ssn,
##     CorModels = c("LinearSill.tailup", "Mariah.taildown", "Exponential.Euclid",
##         "RE1", "RE2"), addfunccol = "addfunccol")
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.4497 -1.6865  0.9792  2.9670 10.2007
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 10.51703    2.59457   4.053    9e-05 ***
## X1           0.95963    0.09235  10.392   <2e-16 ***
## X2           0.01186    0.07901   0.150    0.881
## F11          0.00000         NA      NA       NA
## F12         -2.01487    0.27220  -7.402   <2e-16 ***
## F13          0.07737    0.25425   0.304    0.761
## F14          1.92121    0.24332   7.896   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Covariance Parameters:
##    Covariance.Model Parameter   Estimate
##   LinearSill.tailup   parsill  3.2013087
##   LinearSill.tailup     range  2.6635313
##     Mariah.taildown   parsill  1.2166751
##     Mariah.taildown     range  2.9007278
##  Exponential.Euclid   parsill  8.2811507
##  Exponential.Euclid     range 24.1092269
```

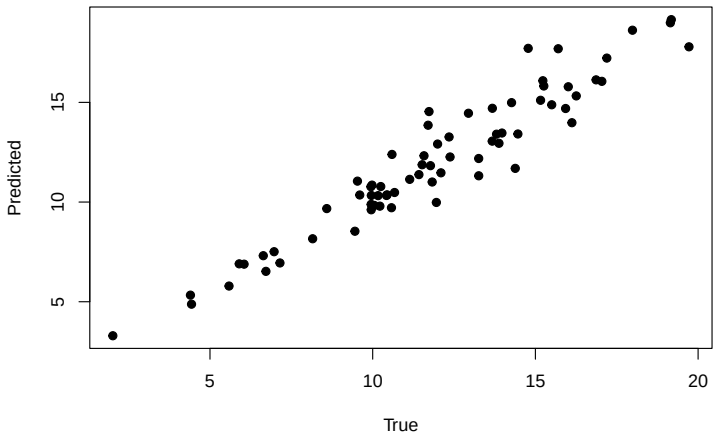

Figure 18: A comparison of true simulated data and predictions at locations after true values were removed.

```
##                 RE1   parsill  4.6092024
##                 RE2   parsill  0.1664536
##              Nugget   parsill  0.0000326
##
## Residual standard error: 4.18029
## Generalized R-squared: 0.7877686
```

If we compare the fixed-effects estimates to the coefficients specified in the `SimulateOnSSN` function, it appears that they are estimated quite well. The covariance parameters are not estimated as well. This is not surprising because there are quite a few of them, and even though they may individually be far from their simulation values, the actual covariance matrix based on them can be quite close to the original covariance matrix, especially near the origin, which is most critical (Stein 1988). Finally, the predictions from the fitted model are compared to the actual values that were simulated:

```
glmssn.pred <- predict(glmssn.out,"preds")
predDF <- getSSNdata.frame(glmssn.pred, "preds")
plot(simpreds, predDF[,"Sim_Values"], xlab = "True",
   ylab = "Predicted", pch = 19)
```

Figure 18 shows excellent prediction accuracy, which relied on both covariates and autocorrelation.

7. Discussion and future development

In the Introduction, we noted that the **Rtop** package can also be used for prediction on stream networks. The model, called topological kriging, was developed in Skøien, Merz, and Blöschl (2006). The basic idea is to create random variables as block means of a spatial random field (with point support) for all basin area above a point on a stream. This will create nested blocks for locations that share flow and create strong correlation among them, while locations that do not share flow will be correlated by proximity in the blocks. Because correlation is based on both nesting and proximity, Laaha, Skøien, and Blöschl (2012) argue that topological kriging should be better than the methods given in **SSN**. However, Ver Hoef and Peterson (2010) and the **SSN** as demonstrated here advance the idea that the best use of the models is to combine the tail-up, tail-down, and Euclidean distance models in a variance component approach. In fact, this approach allows for pure stream-distance models, to pure Euclidean distance models, and combinations thereof, so it should be more flexible than **Rtop**. Also, by using likelihood methods, **SSN** allows fitting linear models with valid inference on covariate effects, in addition to spatial prediction. Regardless of any conceptual argument, the availability of both **SSN** and **Rtop** should allow for interesting comparisons between the methods and hopefully spur development in both areas.

We plan to refine the **SSN** package, making function calls more flexible, which will give users more control over how a plot looks or even to define their own functions for parameter estimation. Currently, we have only implemented the Poisson and binomial families, but plan to extend this to many more families similar to other generalized linear model fitting functions in R. Similarly, data transformation such as Box-Cox (Box and Cox 1964) with lognormal and trans-gaussian kriging (Cressie 1993, p. 135-138) will be implemented, and newer methods such as nonparametric transformations (Gribov and Krivoruchko 2012) will be investigated. We also plan to extend the `createSSN` function so that observed and prediction locations may be generated on existing stream network shapefiles. Another major goal is to implement more efficient model-fitting algorithms for data sets with large numbers of observation sites (e.g., $>$ 2000), such as those collected using in-stream sensor networks (Porter, Hanson, and Lin 2012). The estimation of model parameters currently requires the iterative inversion of a matrix with dimensions given by the number of observation sites, which can be computationally intensive when the number of those sites is larger than 2000. R can be compiled with support for more advanced linear algebra packages such as the Intel Math Kernel Library (MKL) `http://software.intel.com/en-us/articles/intel-mkl/`, and this results in substantial speed improvements. However, new model-fitting algorithms will be needed before a geostatistical model can be fit to data sets on the order of 20000 or more observations. Currently, large numbers of prediction sites (>50000) are possible because prediction of individual sites only requires a single inverse of a matrix with dimensions given by the number of observation sites.

8. Acknowledgments

Support for this work was provided by the United States (US) Forest Service, the US Geological Survey, and the Oregon State Office of the Bureau of Land Management. Some of this work was conducted as part of the working group entitled "Spatial Statistics for Streams," supported by the National Center for Ecological Analysis and Synthesis, a Center funded by NSF (Grant #EF-0553768), the University of California, Santa Barbara, and the State of California. This project also received financial support from the CSIRO Water for a Healthy Country Flagship, and the NOAA's National Marine Fisheries Service to the Alaska Fisheries

Science Center. We also thank Devin Johnson and Jeff Laake for constructive reviews of a previous version of this manuscript. Reference to trade names does not imply endorsement by the National Marine Fisheries Service, NOAA. The findings and conclusions in the paper are those of the authors and do not necessarily represent the views of the National Marine Fisheries Service.

References

Bivand RS, Pebesma EJ, Gomez-Rubio V (2008). *Applied Spatial Data Analysis with R*. Springer, NY. URL http://www.asdar-book.org/.

Box GEP, Cox DR (1964). "An Analysis of Transformations." *Journal of the Royal Statistical Society, Series B*, **26**, 211–252.

Chiles JP, Delfiner P (1999). *Geostatistics: Modeling Spatial Uncertainty*. John Wiley & Sons, New York. ISBN 0-471-08315-1.

Christensen O, Ribeiro Paulo J J (2002). "geoRglm - a package for generalised linear spatial models." *R News*, **2**(2), 26–28. ISSN 1609-3631, URL http://cran.R-project.org/doc/Rnews.

Cook R, Weisberg S (1982). *Residuals and Influence in Regression*. Chapman and Hall, New York.

Cressie N, Frey J, Harch B, Smith M (2006). "Spatial Prediction on a River Network." *Journal of Agricultural, Biological, and Environmental Statistics*, **11**(2), 127–150.

Cressie NAC (1993). *Statistics for Spatial Data*. John Wiley & Sons, New York. ISBN 0-471-00255-0.

Csardi G, Nepusz T (2006). "The igraph software package for complex network research." *InterJournal*, **Complex Systems**, 1695. URL http://igraph.sf.net.

Dent CI, Grimm NB (1999). "Spatial Heterogeneity of Stream Water Nutrient Concentrations Over Successional Time." *Ecology*, **80**, 2283–2298.

Fields Development Team (2006). "fields: Tools for Spatial Data. National Center for Atmospheric Research, Boulder, CO." http://www.cgd.ucar.edu/Software/Fields.

Finley AO, Banerjee S, Carlin BP (2007). "spBayes: An R Package for Univariate and Multivariate Hierarchical Point-referenced Spatial Models." *Journal of Statistical Software*, **19**(4), 1–24. ISSN 1548-7660. URL http://www.jstatsoft.org/v19/i04.

Garreta V, Monestiez P, Ver Hoef JM (2009). "Spatial Modelling and Prediction on River Networks: Up model,down model, or hybrid." *Environmetrics*, **21**, 439–456, DOI: 10.1002/env.995.

Gribov A, Krivoruchko K (2012). "New Flexible Non-parametric Data Transformation for Trans-Gaussian Kriging." *Geostatistics Oslo 2012, Quantitative Geology and Geostatistics*, **17**, 51–65.

James DA (2011). *Applied Multivariate Statistical Analysis*. R package version 0.10.0, URL http://CRAN.R-project.org/package=RSQLite.

Laaha G, Skøien JO, Blöschl (2012). "Comparing Geostatistical Models for River Networks." *Geostatistics Oslo 2012, Quantitative Geology and Geostatistics*, **17**, 543–553.

Le ND, Zidek JV (2006). *Statistical Analysis of Environmental Space-Time Processes*. Springer, New York.

McCullagh P, Nelder JA (1989). *Generalized Linear Models, 2nd Edition*. Chapman & Hall Ltd. ISBN 0-412-31760-5.

Nelder JA, Mead R (1965). "A Simplex Method for Function Minimization." *Computer Journal*, **7**, 308–313.

Pebesma EJ (2004). "Multivariable geostatistics in S: the gstat package." *Computers & Geosciences*, **30**, 683–691.

Peterson EE, Ver Hoef JM, Isaak DJ, Falke JA, Fortin MJ, Jordan C, McNyset K, Monestiez P, Ruesch AS, Sengupta A, Som N, Steel A, Theobald DM, Torgersen CE, Wenger SJ (2013). "Stream networks in space: concepts, models, and synthesis." *Ecology Letters*, **16**, 707–719. doi:10.1111/ele.12084.

Porter H, Hanson P, Lin C (2012). "Staying afloat in the sensor data deluge." *Trends in Ecology and Evolution*, **27**, 121–129.

Ribeiro Paulo J J, Diggle PJ (2001). "geoR: a package for geostatistical analysis." *R-NEWS*, **1**(2), 14–18. ISSN 1609-3631, URL http://CRAN.R-project.org/doc/Rnews/.

Shreve RL (1967). "Infinite Topographically Random Channel Networks." *Journal of Geology*, **75**, 178–186.

Skøien JO, Laaha G, Koffler D, Blöschl G, Pebesma E, Parajka J, Viglione A (2012). "Rtop - an R package for interpolation along the stream network." *Geophysical Research Abstracts*, **14**, 1.

Skøien JO, Merz R, Blöschl (2006). "Top-Kriging – geostatistics on stream networks." *Hydrology and Earth System Sciences*, **10**, 277–287.

Smith BJ, Yan J, Cowles MK (2008). "Unified Geostatistical Modeling for Data Fusion and Spatial Heteroskedasticity with R Package ramps." *Journal of Statistical Software*, **25**(10), 1–21. ISSN 1548-7660. URL http://www.jstatsoft.org/v25/i10.

Stein ML (1988). "Asymptotically Efficient Prediction of a Random Field with a Misspecified Covariance Function." *The Annals of Statistics*, **16**, 55–63.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. Fourth edition. Springer, New York. ISBN 0-387-95457-0, URL http://www.stats.ox.ac.uk/pub/MASS4.

Ver Hoef JM, Peterson E (2010). "A Moving Average Approach for Spatial Statistical Models of Stream Networks (with discussion)." *Journal of the American Statistical Association*, **105**, 6–18. doi:10.1198/jasa.2009.ap08248; Rejoinder pgs. 22 - 24.

Ver Hoef JM, Peterson EE, Theobald D (2006). "Spatial Statistical Models That Use Flow and Stream Distance." *Environmental and Ecological Statistics*, **13**(1), 449–464.

Verbeke G, Molenberghs G (2000). *Linear Mixed Models for Longitudinal Data*. Springer-Verlag Inc. ISBN 0-387-98222-1.

Wolfinger R, O'Connell M (1993). "Generalized Linear Mixed Models: a Pseudo-likelihood Approach." *Journal of Statistical Computation and Simulation*, **48**, 233–243.

Affiliation:

Jay M. Ver Hoef
NOAA National Marine Mammal Laboratory
NMFS Alaska Fisheries Science Center
International Arctic Research Center, Room 351
University of Alaska Fairbanks, Fairbanks, AK 99775-7345
E-mail: jay.verhoef@noaa.gov
URL: http://sites.google.com/site/jayverhoef

Erin E. Peterson
Division of Mathematics, Informatics and Statistics
Commonwealth Scientific and Industrial Research Organisation (CSIRO)
PO Box 2583, Brisbane, QLD 4001
E-mail: Erin.Peterson@csiro.au
URL: http://www.csiro.au/org/CMIS.html

David Clifford
Division of Mathematics, Informatics and Statistics
Commonwealth Scientific and Industrial Research Organisation (CSIRO)
PO Box 2583, Brisbane, QLD 4001
E-mail: Erin.Peterson@csiro.au
URL: http://www.csiro.au/org/CMIS.html

Rohan Shah
Division of Mathematics, Informatics and Statistics
Commonwealth Scientific and Industrial Research Organisation (CSIRO)
PO Box 2583, Brisbane, QLD 4001
E-mail: Erin.Peterson@csiro.au
URL: http://www.csiro.au/org/CMIS.html