# Spatial Statistics with S-Plus - Available Libraries and Functions*-

Tillmann Krahnke

Department of Statistics

University of Dortmund

44221 Dortmund, Germany

**Abstract**

An overview is given over the S-Plus libraries and modules for statistical analysis of spatial data that are currently available at the Department of Statistics, University of Dortmund. It is believed that this includes all libraries currently available on the internet.

Listings of functions show what statistical techniques are implemented, and where to find them. This facilitates the search for any particular function, and saves from re-programming of techniques that are already available. This overview may therefore also be viewed as a starting point for developing further analysis tools for spatial and spatio-temporal statistics in S-Plus.

KEY WORDS: Geostatistics, lattice data, modules, public libraries, S-Plus, spatial statistics, spatial point pattern.

*This work was supported by the Deutsche Forschungsgemeinschaft (DFG), Sonderforschungsbereich 475.

## 1 Introduction

The statistical analysis of spatio-temporal data is under current investigation at the Department of Statistics at the University of Dortmund[1]. By their very nature, the corresponding methods for this type of analysis largely draw on results both from spatial statistics and time series analysis. Many of these techniques are already implemented in the statistical software package S-Plus (Mathsoft,1998), some as part of the commercial add-on module *S+SpatialStats*, others as part of publicly available collections of functions, so-called *libraries*.

Before implementing newly developed methods, a first step should always be to review what statistical methodology is already encoded. This paper attempts to do so for functions from spatial statistics. It also shows how these functions can be made available on any PC within the department's computer network.

All code described here is written by authors outside of the University of Dortmund and published in the internet, or offered on a commercial basis by official S-Plus distributors.

The sections of the paper are divided as follows: An introduction in *Using Modules and Libraries* is given in section 2. Section 3 briefly reviews what *Types Of Data In Spatial Statistics* are encountered. Sections 4 to 8 describe the libraries *splancs*, *sgeostat*, *spatial* and `spatCSU`, and the module *S+SpatialStats*, respectively. Each of these five sections is subdivided into subsections as follows: *General Concepts and Sources* informs on the purpose the library was designed for, when it was written, and by whom. *Working with Data* mainly lists functions for data import and export. *Geostatstics*, *Lattice Data*, and *Spatial Point Patterns* all include listings of functions related to the corresponding data type. Short descriptions explain what the functions are used for. For completeness, *Data sets* lists the data sets included. Any *Other Features* are described in the last subsection.

Section 9 , *Other Functions for Spatial Statistics*, lists additional S-Plus functions that are already part of the base package. A *Summary* concludes the paper in section 10.

[1]See section on "The Analysis of Spatio-Temporal Data in Epidemiology and Ecology" in the Collaborative Research Centre 475, "Complexity Reduction In Multivariate Data Structures", funded by the Deutsche Forschungsgemeinschaft.

## 2 Using Modules And Libraries

This section describes how to access libraries over the network[2]. It also shows how modules and libraries can be installed locally if necessary, and given the necessary rights for system administration.

In general, S-Plus libraries should be accessed using the departments PC network. This saves time and disk space. Because there are only single user licenses for the module `S+SpatialStats`, this must always be installed locally.

### 2.1 Before Getting Started...

Before working on a particular project it is recommended to create a corresponding project directory first. This assures that all objects related to a single project will be stored in one directory. In S-Plus 4.x you create a project directory as follows:

1. Create a link to the S-Plus startup file. This is usually located in (SHOME)/cmd/splus.exe, where "(SHOME)" refers to the S-Plus home directory (e.g. c:/software/splus45). Move the link to any desired location, e.g. on the Windows desktop.
2. Left-click on the icon's name for highlighting. Left-click again for editing. Change the name to the project's name, e.g. "Myproject".
3. Right-click on the icon to edit its properties.
4. In the field "Target" in the property dialog, add the expression "S__Proj=c:/Myproject" to the path to the S-Plus startup file (if the project's files are to be saved in folder S__Proj=c:/Myproject).

If S-Plus is opened for the next time by clicking on the icon labeled "Myproject", you will be asked to allow for creation of two new directories "__Data" and "__Prefs". Answer yes. All objects and functions created will then be saved under c:/Myproject/__Data.

[2]Thanks to Uwe Ligges for his support when installing the libraries

## 2.2 Network Access

All public libraries discussed here are stored in the directory **//server01/software/s-plus/libraries**. You find this server in the NT Explorer under your *Network Neighborhood*. To use a library, do the following:

1. Create a project directory (if necessary) and start S-Plus
2. From the Commands window or a Script window, define the location of the additional libraries by submitting:
   `assign(x="lib.loc",where=0,`
   `value="//server01/software/s-plus/libraries")`
3. To attach the library called *spatial*, say, submit `library(spatial, first=T)` from the commands window, or chose FILE - LOAD LIBRARY - spatial from the menu.

If you plan to use certain libraries whenever you use S-Plus, it might be better to define something similar to the following function:

```
.First<-function(){ assign(x="lib.loc",where=0,
value="//server01/software/s-plus/libraries")
library(spatial, first=T )}
```

This function will be automatically called at every startup of the program. If you already defined a function `.First()`, simply add the appropriate lines. Now all installed libraries should be accessible by the local machine and can be loaded as described above.

## 2.3 Install Libraries Locally

Libraries in S-Plus are collections of functions and objects stored in a particular file or folder. The libraries `splancs`, *spatial* and *spatCSU* originally come as zipped files, but are already unpacked for public use on the departments server. *sgeostat* comes as compressed tar-file. Its contents is also already extracted and can be found on the server. Locations of all these files are given in the corresponding sections below. Although not recommended, you may want to install them locally on your PC. This section describes how to do that.

## 2.4 Installing *splancs*, *spatial* or *spatCSU*

For installation of *splancs*, *spatial* or *spatCSU* proceed as follows.

1. Download the zipped library file you need and copy it to the desired location. This is c:/splus45/library for the files `spatial.zip` and `VR5.zip`. For the the libraries `splancs` and *spatCSU* create new folders before unpacking, e.g. c:/splus45/library/splancs (assuming that c:/splus45 is the folder were your version of S-Plus resides).

2. Unzip the file in the folder where it was copied. Allow for creation of the corresponding folders!

3. Start S-Plus.

4. a) Load the library from menu using **File - Load Library...** and selecting the library name, or

   b) type, e.g.,

   ```
   library(spatial, first=T)
   ```

   in the commands window. The option `first=T` assures that S-Plus uses the library version of a function even if there comes a function with the same name with the S-Plus base distribution.

5. Libraries are automatically detached when S-Plus is closed. If you want to detach a library during a running session, look up its number in the search list and remove it using the command

   ```
   detach(libnumber)
   ```

   where *libnumber* is the number of the library in the search list.

## 2.5 Installing *sgeostat*

The library *sgeostat* was written for S-Plus for Unix and comes as compressed tar-file. You need to have access to Linux or Unix to uncompress it. To be more specific, proceed as follows:

1. Download file SGeoStat.tar.Z from **http://www.gis.iastate.edu/SGeoStat/homepage.html**.

2. Under Unix or Linux, uncompress SGeoStat.tar.Z and extract the files by

   ```
   uncompress SGeoStat.tar.Z
   tar -xvf SGeoStat.tar
   ```

   Copy the resulting *.s -files to your Windows PC.

3. Create a new library folder (SHOME)/library/sgeostat/__Data, where (SHOME) is the folder where S-Plus is installed. Edit all *.s -files and change the value of the `where`-argument of each call to `assign()` to:

   ```
   paste(getenv("SHOME"),"/library/sgeostat/__Data",sep="")
   ```

4. Create a project directory (if necessary) and open S-Plus from there. Attach the library folder by

   ```
   library(sgeostat,first=T)
   ```

5. Source in all *.s -files and make the new functions visible by calling `synchronize(database=2)`, if the new library folder is the second on the search list (This should be the case if you used `first=T` when attaching the library).

Now the library *sgeostat* should be available.

## 2.6 Install Modules Locally

Commercial versions of S-Plus libraries are called *Modules*. They are easily installed by running the corresponding setup.exe-file from disk. You need to have the appropriate version of the S-Plus main program installed on your computer (should be S-Plus 4.x). In addition, the necessary rights for system administration have to be set. If problems occur, check with your local system administrator.

Menus and dialog boxes lead the user through the installation process. To use the module after installation, start S-Plus and open the **File**-menu. You will find special help topics under **Help-S+SpatialStats** in the menu. Selection of **Load Module...** will open a dialog box that allows for selection of the module *S+SpatialStats* (choose *spatial*). Pressing OK will load the module.

If you want to start the module from the commands window, make this window active and type

```
    module(spatial)
```

To unload the module, type

```
    module(spatial, unload = T)
```

All modules used in the current session will be automatically unloaded when exiting S-Plus. If you want to re-open the module automatically whenever you start the program, you may do this by defining the following function in your working directory:

```
    .First <- function(){module(spatial)}
```

If you already defined a function `.First()`, simply add the appropriate lines. The `First()`-function will be executed at each startup of S-Plus and opens the module automatically.

## 2.7 Getting help

### 2.7.1 Help for Libraries

For general information on a library called *"libname"* submit the words `library(help=libname)` from the command line. To access help for functions and objects from this library, try

```
    help(name, library="libname")
```

or

```
    ?name
```

from the command line. Here *name* refers to any S-Plus object.

### 2.7.2 Help for Module Functions

If you are using a S-Plus module, help is available in the following ways:

- Open the menu selecting **Help - S+SpatialStats**. The usual Microsoft Windows dialog box appears. Type a name or select one from the list and press OK. A help page will be displayed.

- From the command line, type `help(module="spatial")` to get the same dialog box. Proceed as above.
- Alternatively, ask for help for a specific function or object from the commands window using the syntax

  ```
  ?name
  ```

  The help page for function *name* will be displayed.

## 3 Types of Data in Spatial Statistics

Spatial data can be collected in several different ways. Depending on the underlying data structures, spatial statistics may be divided into three major fields (Kaluzny et al., 1998; Cressie, 1993): *Geostatistics*, *Analysis of Lattice Data* and *Spatial Point Patterns (or: SPPs).* This is not the only possible partition, and there is some overlap in the statistical methodology. This applies particularly to Geostatistics and the Analysis of Lattice Data.

In all three fields, the data are generally modeled according to the decomposition

$$\text{data} = \text{large–scale variation} + \text{small–scale variation}$$

The large-scale variation may be interpreted as a (not necessarily linear) trend in the data, whereas the small-scale variation may be interpreted as measurement error, within region variability, inherent site variability and the like (c.f. Haining, 1990).

If the measurements are taken at some fixed locations, one deals with random field data or *Geostatistics.* The spatial coordinates are usually taken to be continuous. The actual measurement $z$, say, can be either continuous or discrete. Examples are: The amount of rain fall at some fixed weather stations (continuous case); or the number of members of a specified species collected at fixed sites in a forest.

The general stochastic model in spatial statistics can be formulated based on the notation given in the following table (Cressie, 1993; Kaluzny et al., 1998).

| The General Spatial Stochastic Model | |
|---|---|
| $s \in \mathcal{R}^d$ | Location of measurement in $\mathcal{R}^d$ |
| $Z(s)$ | Measurement at location $s$. |
| $\{Z(s) : s \in D\}$ | Multivariate random field. $D \subset \mathcal{R}^d$ index set. |

The interpretation in Geostatistics is as follows:

| The Geostatistical Model | |
|---|---|
| $D$ | is a fixed index set that encompasses a rectangular area. |
| $s$ | Set of continuous coordinates in D. |
| $Z(s)$ | Random vector of measurements at location $s$ in $D$. |

The situation changes if the data collected are related to *regions* of interest. Epidemiological data, for example, often refer to numbers of events within a certain area. The regions of interest are usually laid out in an irregular fashion, and in many cases additional neighborhood information is available. Data of this type are called *Lattice Data*. The general model now has a slightly different interpretation:

| The Lattice Model | |
|---|---|
| $D$ | Fixed set of countably many sites in $\mathcal{R}^d$. |
| $Z(s)$ | Measurement at location $s$ in D. |
| (others) | Neighboring information. |

The third type of data are *Spatial Point Patterns* (SPPs). Here the locations of measurement themselves are random. One is typically interested in the distribution of these locations, which may be completely at random, or show some kind of regularity or clustering. An example for SPP data are the locations of trees of a certain kind within a specified region. Typical questions then are if the trees are randomly scattered over the area, or if some structure can be observed.

If additional variables are measured, a *marked* Spatial Point Pattern is given. The extra variables may be used to improve the analysis of the underlying random processes. In the above example, one may collect additional data on soil quality or precipitation.

The model for a (marked) Spatial Point Pattern is given in the following table:

| **The (marked) SPP-Model** | |
|---|---|
| $D$ | Point pattern in (subset of) $\mathcal{R}^d$. D is random. |
| $Z(s)$ | equal to 1 (= observed) for the general statistical point pattern; or: |
| $Z(s)$ | Random vector at location $s \in D$ for a marked SPP. |

Since the type of data as well as the major goals of the analysis differ in some respect between the three fields given above, some specific statistical methods were developed. In what follows, the S-Plus functions for each module or library are therefore sorted into three different subsections as well: Functions for Geostatistics, for Lattice Data, and for Spatial Point Patterns will be described separately. Two preceding subsections will give some information on *General Concepts and Sources* of the particular library, or describe the *Working with Data* (mainly data im- and export). Data sets included and additional functionality are listed in two additional subsections *Data sets* and *Other Features*.

# 4 The Library *splancs*

## 4.1 General Concepts and Sources

The library *splancs* was written and first released by Barry Rowlingson and Peter Diggle from Lancaster University in 1991 as part of a project integrating spatial statistical analysis into a Geographical Information System (GIS). It is particularly designed for the analysis of Spatial Point Patterns. Some functionality has been added later, particularly dealing with space-time SPPs. It was last revised in 1997. For download see **http://www.maths.lancs.ac.uk/~rowlings/**. The `splancs` library was ported to Windows NT 4.0 and Windows 95 by Steven Reader from the University of South Florida in 1998 and is available as Version "2.00 Win" (see the copyright notes printed when calling `splancs()`). This Windows version is contained in the zipped file **/Florida/sflorida.zip** which can be found under the link above. A summary of all functions is given in file **/Splancs/splancs.ps**. Printable help files can be found in **/Splancs/Shelp.ps**.

The library *splancs* does *not* use the object oriented approach for programming in S-Plus. It therefore does *not* introduce new classes of S-Plus objects.

## 4.2 Working with Data

The general format used to store data within *splancs* is in a n by m (m $\geq 2$) array (matrix or dataframe), where the first two columns are assumed to contain x and y coordinates, respectively. Other columns may contain additional information. Read in the data by regular S-Plus functions like `scan()`, or `read.table()` for a marked SPP. The *splancs* format may be obtained by applying `as.points()` on any numeric S-Plus data object. Check for correctness using `is.points()`. Vectors are put into the right format by `spoints()`, taking the odd vector elements for x coordinates and the even elements for y coordinates. The number of points read in (i.e., the number of rows of the array) are counted by `npts()`. Note that the "points objects" in *splancs* do *not* have a special attribute and do *not* constitute a class!

| **Function** | **Action** |
|---|---|
| `as.points` | Create data object in spatial point format. |
| `is.points` | Test if data object is in spatial point format. |
| `npts` | Count number of points in data set. |
| `spoints` | Turn vector into matrix in spatial point format. |

Table 1: **Library *splancs* – Working With Data**

## 4.3 Geostatistics

The library *splancs* is not designed specifically for the analysis of geostatistical data. For some functions that might be applicable to this type of analysis compare the subsection on `Spatial Point Patterns` below.

## 4.4 Lattice Data

The library *splancs* is not designed for this type of analysis. For some functions that might be applicable to this type of analysis compare the subsection on *Spatial Point Patterns.*

## 4.5 Spatial Point Patterns

The strength of the *splancs* library is in the analysis of spatial point patterns. An important goal is to check on randomness of the observed points. This can be done using Ripley's K-function. It is estimated by `khat()`. For a bivariate version see `k12hat()`. Pointwise standard errors for differences between two K-functions are found with `secal()`. Envelopes for K-functions and their differences are calculated from simulated data by `Kenv.csr()`, `Kenvlabel()` and `Kenv.tor()`. The covariance matrix is found by `khvc()` or `khvmat()`.

The functions `Fhat()`, `Fzero()`, `Ghat()` and `n2dist()` deal with nearest neighbor distances from one or two patterns. The functions `kernel2d()`, `kernrat()` and `mse2d()` perfrom kernel smoothing to obtain estimates for local intensity. `pdens()` calculates the intensity for points in a polygon. The Diggle-Rowlingson raised-incidence model is fit by `tribble()`.

For space-time analysis, `kernel3d()` for kernel smoothing has been added. `stkhat()` gives the K-function in space and time, and `stvmat()` the covariance. Space-time clustering is analyzed using `stmctest()` and `stsecal()`. Random selections and simulations of data points are managed by `rlabel()`, `thin()` and `csr()`.

| **Function** | **Action** |
|---|---|
| `area` | Calculate area of polygon. |
| `bbox` | Find bounding box for set of points. |
| `csr` | Generate random pattern. |
| `dsquare` | Squared distances from points to sources. |
| `Fhat` | Empirical distribution function of origin-to-point nearest neighbor distances. |
| `Fzero` | Theoretical nearest neighbor distribution function for Poisson process. |
| `Ghat` | Empirical distribution function of point-to-point nearest neighbor distances. |
| `gridpts` | Create sampling origins. |
| `inout` | Logical vector. True if point is in polygon, False if not. |
| `inpip` | Return indices of points inside a polygon. |
| `k12hat` | Bivariate K-function. |
| `Kenv.csr` | Generate random pattern and estimate K-function. |
| `Kenv.label` | Envelope for difference of two K-functions. |
| `Kenv.tor` | Envelope from `k12hat()` using toroidal shifts. |
| `kernel2d` | Estimator of local intensity using quartic kernel. |
| `kernel3d` | Compute space-time kernel. |

Table 2: **Library *splancs* – SPP (cont.)**

| **Function** | **Action** |
|---|---|
| `kernrat` | Calculate ratio of local intensities. |
| `khat` | Ripley's K-function. |
| `khvc` | Covariance for difference between two K-functions. |
| `khvmat` | Covariance for difference between two K-functions under random labelling. |
| `mpoint` | Overlay point patterns. |
| `mse2d` | Find optimal kernel. |
| `n2dist` | Nearest neighbors for two point patterns. |
| `nndistF` | Calculate nearest neighbor differences for `Fhat()`. |
| `nndistG` | Calculate nearest neighbor differences for `Ghat()`. |
| `pdense` | Intensity of points in polygon. |
| `pip` | Return subset of points within (or without) a given polygon. |
| `plt` | Cumulative distribution as function of distance. |
| `print.ribfit` | Print fit from `tribble()`. |
| `ranpts` | Generate random points. |
| `rlabel` | Randomly label point sets. |
| `rtor.shift` | Perform random toroidal shift. |
| `sbox` | Bounding box plus extra space. |
| `secal` | Pointwise standard errors for difference of two K-functions. |
| `shift` | Shift point pattern along x and/or y axis. |
| `splancs` | Version number and author information. |
| `stdiagn` | Summary plots for space-time clustering analysis. |
| `stkhat` | Compute space-time K-functions. |
| `stmctest` | Monte-Carlo test of space-time clustering. |
| `stsecal` | Pointwise (in space and time) standard errors for space-time clustering. |
| `stvmat` | Four-dimensional Covariance matrix for space-time clustering. |
| `thin` | Randomly select n out of a set of points. |
| `tor.shift` | Perform toroidal shift. |
| `tribble` | Diggle-Rowlingson raised incidence model. |
| `triblik` | Log-likelihood for the Diggle-Rowlingson raised incidence model. |

Table 2: **Library *splancs* – SPP Data**

## 4.6 Data sets

There are no data sets included in library *splancs*.

## 4.7 Other Features

There are a couple of special features associated with graphics that come with library *splancs*: `pointmap()` results in a scatterplot of observed points with axes of equal length. Points can be added or deleted interactively by `addpoints()` and `delpoints()` (the Graphical parameter "plot type" should be set to "s" first if used in combination with `pointmap()`). `getpoly()` interactively creates a surrounding polygon by mouse click. If the polygon is stored as an S-Plus matrix, use `polymap()` with argument `add=T` to add it to an existing plot. For display of space-time data `kerview()` was added.

The authors added a map of boarders and counties which can be called by `uk()`.

| **Function** | **Action** |
|---|---|
| `addpoints` | Add points interactively. |
| `delpoints` | Delete points interactively. |
| `gen` | Generate random points in polygon. |
| `getpoly` | Create polygon interactively. |
| `kerview` | Display linked-window system for browsing space-time data. |
| `pointmap` | Plot new points or add to existing plot. |
| `polymap` | Plot new polygon or add to existing plot. |
| `uk` | Plot of the United Kingdom (without Northern Ireland). |
| `uk.coast` | Data set containing UK coast lines. |
| `uk.county` | Data set containing UK county lines. |
| `zoom` | Select and enlarge subregion from plot. |

Table 3: **Library *splancs* – Other Features**

# 5 The Library *sgeostat*

## 5.1 General Concepts and Sources

The library *sgeostat* was written by James J. Majure from Iowa State University in Ames, Iowa, USA. It consists of 26 functions and objects that are designed using the object oriented approach to programming. The focus is on the analysis of geostatistical data, i.e. exploratory data analysis, variogram modeling and kriging. The library can be downloaded from **http://www.gis.iastate.edu/SGeoStat/homepage.html**. This URL also points to the online help files, which refer quite frequently to Cressie (1993). Note that *sgeostat* does not come with regular S-Plus help files !

## 5.2 Working with Data

There are several newly defined classes of objects that are used in `sgeostat`, like `"point"`, `"pairs"`, `"variogram"`, and `"variogram.model"`. They represent observed data, neighborhood information used for variogram estimation, variogram estimates and fitted variogram models. In addition, `"trend.surface"` objects can be created to describe a trend surface model. The table below gives an overview. See the *sgeostat* online help on *Data Structures* for more details.

| **Object Class** | **Contents** | **Function** |
|---|---|---|
| `point` | Dataframe with x and y coordinates and observed data. | `point()` |
| `pairs` | List with neighborhood information, distances and lags. | `pairs()` |
| `trend.surface` | List of parameters and other components describing a trend surface model. | `fir.trend()` |
| `variogram` | Dataframe with lag information and empirical variogram estimates. | `variogram()` |
| `variogram.model` | List of parameters and function of variogram fit. | `fit.⋆()` |

Table 4: **Library *sgeostat* – Object Classes**

To read in data for use with *sgeostat*, use the regular S-Plus functions. Create a dataframe with two named columns containing the coordinates, and additional columns with the data. The function `point()` converts such a dataframe to a `point` -object.

## 5.3 Geostatistics

Central to the `sgeostat` library is variogram estimation and kriging. The Corresponding functions are `variogram()` for variogram estimation, plus several fitting tools called `fit.`*modeltype*`()`, where *modeltype* is `exponential`, `linear` or `wave`. Graphical tools for variogram estimation are a `plot`-method for variograms, and `spacecloud()` and `spacebox()` to plot for variogram clouds or boxplots. To fit a linear trend surface, use `fit.trend()`. Estimate the trend with `trend.value()` or `trend.matrix()`.

| **Function** | **Action** |
|---|---|
| `calcangle` | Find angle between 2 points in 2D. |
| `export.point` | Export `point`-object to file. |
| `fit.exponential` | Fit an exponential model to empirical variogram. |
| `fit.linear` | Fit a linear model to empirical variogram. |
| `fit.trend` | Fit a trend model to `point`-object. |
| `fit.wave` | Fit a wave model to empirical variogram. |
| `identify.point` | `identify()`-method for `point` -objects. |
| `krige` | Perform spatial prediction. |
| `krige.all` | Perform spatial prediction using all points. |
| `krige.maxdist` | Perform spatial prediction using points up to distance `maxdist`. |
| `lagplot` | Spatially lagged scatterplot. |
| `ls` | Redefinition of `ls()`. Use `objects()` instead. |
| `pairs` | Create a `pairs` -object containing neighborhood information for variogram estimation. |
| `pairs.aniso` | Create a list similar to a `pairs` -object for anisotropic model. Used in `pairs()`. |
| `pairs.iso` | Create a list similar to a `pairs` -object for isotropic model. Used in `pairs()`. **Need to change `stations` in function definition to `point.obj`!** |
| `pairs.newangle` | Create a `pairs`-object for isotropic model. Include only pairs in specified directions (given angle and angle plus 180 degrees), and within given tolerance angle. |
| `plot.point` | Plotting method for `point` -objects. |
| `plot.variogram` | Plot empirical variogram, optionally with fitted model. |
| `point` | Create `point` -object from dataframe. |

Table 5: **Library *sgeostat* – Geostatistics (cont.)**

| **Function** | **Action** |
|---|---|
| `print.pairs` | Print-method for `pairs` -object. |
| `print.point` | Print-method for `point` -object. |
| `spacebox` | Boxplots of variogram cloud. |
| `spacecloud` | Plot scatterplot of square-root or squared distances of pairs versus distance. |
| `trend.matrix` | Evaluate trend surface over grid given by coordinates of corners. |
| `trend.value` | Evaluate trend surface for set of points. |
| `variogram` | Calculate classic, robust and median variogram estimates. |

Table 5: **Library *sgeostat* – Geostatistics**

## 5.4 Lattice Data

The library *sgeostat* is not designed for the analysis of lattice data.

## 5.5 Spatial Point Patterns

The library *sgeostat* is not designed for the analysis of spatial point patterns.

## 5.6 Data sets

There are no data sets included in library *sgeostat*.

## 5.7 Other Features

As mentioned at the beginning of this section on *sgeostat*, there are online help files in html-format available. They may be copied from the original URL and saved on hard disk and read by your local web browser.

In addition to *sgeostat* for S-Plus there exists a modified version for use with the GNU software R. It was ported to R by Albrecht Gebhardt (from University of Klagenfurt, Austria), with contributions by Roger Bivand (Norwegian School of Economics and Business Administration, Bergen, Norway) and extended by functions for fitting gaussian or spherical variogram models.

# 6 The Library *spatial*

## 6.1 General Concepts and Sources

The library *spatial* for spatial statistics is available as supplement of the book "Modern Applied Statistics with S-Plus" (MASS, for short) by Venables and Ripley (1997). Some of the code presumably goes back to the work related to Ripley (1981) and Ripley (1988). The library is designed for spatial smoothing and analysis of spatial point patterns. It contains a Windows help file that can be accessed in the usual fashion using the `help()`-function.

The library *spatial* heavily draws on C code written by the authors. This source code may be obtained from **http://www.stats.ox.ac.uk /pub/MASS2/VR5.zip** which includes the UNIX distribution of the library. The library *spatial* does *not* use the object oriented approach for programming in S-Plus. It therefore does *not* introduce new classes of S-Plus objects. The library can be obtained from the following sources:

- As file spatial.zip from **http://lib.stat.cmu.edu/DOS/S/SWin/**.
- As part of the complete set of libraries used in MASS. See **http://www.stats.ox.ac.uk/pub/MASS2/Windows.shtml**.
- As part of **S-Plus 2000**. Here, the library *spatial* and some other slightly modified libraries from MASS are installed when installing the base software.

The version currently used at the University of Dortmund is downloaded from the second source. This source also makes available some additional functions and data sets as part of a library called `mass` that are used in the chapter on spatial statistics in Venables and Ripley (1997).

The "Statistics Online Complements" to Venables and Ripley (1997) include a few pages that give an idea of how analysis differs between the library *spatial* and the *S+SpatialStats* module. The complements can be found under **http://www.stats.ox.ac.uk/pub/MASS2**.

## 6.2 Working with Data

To read in data for geostatistical analysis, use the regular S-Plus functions like `read.table()` or `import.data()`. The `surf.*` -functions in library *spatial* expect either a dataframe with columns named x,y and z, or three vectors that contain the appropriate data. It therefore seems useful to create the appropriate column names when reading in the data.

When working with point patterns special functions are needed to read in the data and to initialize the domain for the process. `ppinit()` reads in ASCII data-files of special format and returns a list: In the first line, the file should contain the number n of observations to read in. The next line contains a header, usually the name of the data set. A third line gives the coordinates of the four corners for the (rectangular) domain where the process lives. These are the lower and upper possible values for x and y, respectively. A fifth number in this line describes the number by which the values from file must be divided to obtain the coordinates as used by S-Plus, i.e. a scaling factor.

Finally, the n pairs of x- and y-coordinates for the data points follow. There must be an even number of values in each line. When calling `ppinit()`, the domain of the point process is read in and stored internally. Other functions may rely on this for their computations (e.g., `Kenvl()`). To check for the domain currently set up, call `ppgetregion()` (with no arguments). To explicitly set up a domain, use `ppregion()`.

| **Function** | **Action** |
|---|---|
| `ppgetregion` | Extract corners of spatial domain after initialization with `ppinit()`. No arguments. |
| `ppinit` | Read in file with point process data. |
| `ppregion` | Initialize domain for SPP for use in C code. Arguments are either four numbers for the domain's corners, or a vector of length four. |

Table 6: **Library *spatial* – Working with Data**

## 6.3 Geostatistics

For smoothing, the main functions in the *spatial* library are `surf.ls()` and `surf.gls()` for least squares and general least squares fits, i.e. universal kriging. To evaluate a fit on a grid, use `trmat()` and `prmat()`, respectively. Standard errors for prediction are found by `semat()`. Important tools when modeling a kriging surface are the (co-) variogram and correlogram. Their empirical versions are found by `variogram()` and `correlogram()`. Exponential, gaussian or spherical theoretical covariance functions may be calculated by `expcov()`, `gaucov` or `spher.cov()`.

| **Function** | **Action** |
|---|---|
| `correlogram` | Compute and plot emperical spatial correlogram from `nint` bins. |
| `variogram` | Compute and plot emperical spatial(semi-) covariogram from `nint` bins. |
| `expcov` | Theoretical covariance based on exponential model. |
| `gaucov` | Theoretical covariance based on gaussian model. |
| `sphercov` | Theoretical covariance based on spherical model (up to 3D). |
| `surf.ls` | Fit polynomial trend surface using least squares. |
| `surf.gls` | Trend surface using generalized least squares (universal kriging). |
| `trmat` | Evaluate trend surface over grid. |
| `prmat` | Evaluate kriging surface over grid. |
| `semat` | Standard error of prediction for kriging. |

Table 7: **Library *spatial* – Geostatistics**

## 6.4 Lattice Data

There are no functions in the *spatial* library particularly designed for lattice data. For this type of analysis one may refer to the other libraries or modules described here.

## 6.5 Spatial Point Patterns

The functions for analyzing SPPs concentrate on Ripley's K-function, and on the simulation of particular spatial processes. The K-function for a particular data set may be estimated by `Kfn()`, whereas `Kaver()` and `Kenvl()` calculate the average K-function and envelope for a simulated process. It is possible to simulate binomial, Strauss and Matérn's sequential inhibition processes.

| **Function** | **Action** |
|---|---|
| `pplik` | Pseudo likelihood estimation for fitting a Strauss process. |
| `Kfn` | Estimate scaled version of Ripley's K function. |
| `Kenvl` | Averages and extremes from simulated K-functions ("envelope"). |
| `Kaver` | Average from simulated K-functions. |
| `Psim` | Simulate binomial process. |
| `SSI` | Simulate sequential inhibition process (Matérn). |
| `Strauss` | Simulate Strauss process. |

Table 8: **Library *spatial* – SPP**

## 6.6 Data sets

There are a couple of data sets that come with the *spatial* library. They are stored in the subdirectory ".../spatial" where "..." refers to the location where the library was installed. In the departments network, this is **//server01/software/s-plus/libraries**. The data files have extension *.dat and contain ASCII-Text in a format that can be read in by `ppinit()` (see *Working with Data* above).

The data sets `npr1`, `shkap` and `topo` are used in the examples on spatial statistics in chapter 16 of Venables and Ripley (1997), but are available only after installation and loading of the `mass`-library. It is part of the file VR-libc.exe (second source 2 mentioned above) and is also installed on the server.

Note that all data sets except `agter` in the *spatial* library are also contained in the library *spatCSU* (see below). For more specific information on sources for the contained data sets see the file pp.fil that comes with both of these libraries.

| | | | |
|---|---|---|---|
| agter.dat | caveolae.dat | cells.dat | davis.dat |
| drumlin.dat | eagles.dat | grocery.dat | hccells.dat |
| nztrees.dat | pairfn.dat | pereg.dat | pines.dat |
| redwood.dat | schools.dat | stowns1.dat | tokyo.dat |
| towns.dat | npr1.dat | shkap.dat | topo.dat |

Table 9: **Library *spatial* – Data sets**

## 6.7 Other Features

The libraries from MASS (including *spatial*) as well as `sgeostat` are ported to the freely available software package R. See **http://www.ci.tuwien.ac.at/R/bin/windows/windows-9x/contrib/** and VR5.3pl037.zip.

# 7 The Library *spatCSU*

## 7.1 General Concepts and Sources

The file spat98.zip contains a library of functions on spatial statistics collected at the Colorado State University (CSU), Fort Collins, Colorado, by Robin M. Reich and Richard Davis. Unfortunately, the authors refer to this library also as library *spatial*, apparently because Ripley's *spatial* library served as a starting point for *spatCSU*. In fact, essentially all functions from *spatial* can also be found in *spatCSU*. Three issues have to be kept in mind, however:

- There may be some differences in function names, like using upper case letters instead of lower case, or inserting dots into the names.
- Ripley's functions in *spatial* contain C code, whereas in *spatCSU* he used FORTRAN. This may result in some differences between functions.
- Since the *spatial* library is still maintained by B.D. Ripley, some changes may be introduced from time to time which are not necessarily incorporated immediately into *spatCSU*.

To avoid confusion with other libraries and modules on spatial statistics, the library compiled by Reich and Davis will here be referred to as library *spatCSU*. This name was chosen quite arbitrarily, but reflects the origin in some sense. The version used here was last updated on 8 February 1999, and downloaded from **http://www.stat.colostate.edu/∼rdavis/st523/**. It was used for a course on Quantitative Spatial Statistics held by R. Reich and R. Davis in 1998. A course manual is accessible as pdf-file via the link given above.

Contributors to *spatCSU* are R. Davis (CSU), R. M. Reich (CSU), R.D. Ripley (Oxford University), P. Mielke Jr. (CSU) and K. Metzger (CSU). See the file Readme.new to find out about the author of a particular function. Some of the functions contributed by Brian Ripley were already described above for library *spatial*. In spat98.zip you find slightly modified versions as in *spatial*: Whereas the former uses Fortran-code, the latter contains C-code.

## 7.2 Working with Data

Reading in data for use with *spatCSU* is very similar to reading in data with the *spatial* library. For example, the function `ppinit()` expects a certain file format for spatial point patterns. See section *Working with Data* for library *spatial* for details. In addition, *spatCSU* has special functions `spinput()` and `swinput()` to read in ASCII files with data or neighborhood information.

Files in ARC/INFO format may be imported or exported with `arcexprt()` and `arcinput()`. A description how to do this is given in the file arc.txt which can be found under the URL for *spatCSU* given above.

| **Function** | **Action** |
|---|---|
| `arcexprt` | Import ARC/INFO ASCII-file and create S-Plus grid. |
| `arcinput` | Export a Splus grid into ASCII text file for input into ARC/INFO. |
| `ppinit` | Read in file with point process data. |
| `spctg` | Read in ASCII file for use with `spwtctg()`. |
| `spinput` | Read in ASCII file into matrix. |
| `swinput` | Read in ASCII file with neighborhood information. |

Table 10: **Library *spatCSU* – Working with Data**

## 7.3 Geostatistics

There are a couple of additional functions in *spatCSU* that do not appear in library *spatial*. For variogram estimation, use `fitvar()` in combination with `expvar()`, `gauvar()`, `linvar()` or `sphervar()`. `correlogram()` and `variogram()` from *spatial* are replaced by `corrgram()` and `variogrm()` which are also attributed to Ripley. In addition, cross-variograms can be estimated by `crossvar()`.

Polynomial trend surface estimation may be done by `ols()`, `trendls()` and `trend()`. `mpolish()` does median polishing. For kriging, use `krig()`, `cokrig()` and `cokrig1()`. Results can be evaluated with `crossval()`, `vcokrig()` or `vcokrig1()`, which perform cross-validation on a (co-)kriged surface. The surface itself is predicted by `predkrg()`, `predcok()` and `predcok1()`.

| **Function** | **Action** |
|---|---|
| `cokrig` | Ordinary cokriging with non-bias constraint. |
| `cokrig1` | Ordinary cokriging with non-bias constraint. |
| `corrgram` | Spatial correlogram as a function of distance. |
| `crossvar` | Spatial cross-variogram. |
| `crossval` | Crossvalidation of kriged surface. |
| `expcov` | Theoretical covariance based on exponential model. |

Table 11: **Library *spatCSU* – Geostatistics (cont.)**

| **Function** | **Action** |
|---|---|
| `expvar` | Theor. variogram based on exponential model. |
| `fexp` | Support for `fitvar()`. |
| `fitvar` | fits an exponential, Gaussian, spherical, or model to a sample variogram or cross-variogram (Reich). |
| `fmat` | Support for `surfls()` and `surfgls()`. |
| `gau` | Support for `fitvar()`. |
| `gaucov` | Theoretical covariance based on gaussian model. |
| `gauvar` | Theoretical variogram based on gaussian model. |
| `krig` | Ordinary (block) kriging. |
| `lin` | Support for `fitvar()`. |
| `linvar` | Theoretical variogram based on linear model. |
| `mpolish` | Median polish on data matrix. |
| `ols` | Fit linear model and test residuals from spatial autocorrelation using Moran's I. |
| `pred` | Find point estimate from kriged surface. |
| `predcok` | Spatial prediction using cokriging. |
| `predcok1` | Spatial prediction using cokriging. |
| `predkrg` | Spatial prediction using kriging. |
| `predval` | Support for `prmat()`. |
| `prmat` | Evaluate kriging surface over grid. |
| `semat` | Standard error of prediction for kriging. |
| `serror` | Standard error of point estimate from kriged surface. |
| `seval` | Support for `semat()`. |
| `sph` | Support for `fitvar()`. |
| `sphercov` | Theoretical covariance based on spherical model. |
| `sphervar` | Theoretical variogram based on spherical model. |
| `surfls` | Fit polynomial trend surface using least squares. |
| `surfgls` | Trend surface using generalized least squares (universal kriging). |
| `trendls` | Fit polynomial trend surface using least squares. Provides more info than `surfls()`. |
| `trend` | Evaluate trend surface over grid using output from `trendls()`. |
| `trmat` | Evaluate trend surface over grid using output from `surfls()` or `surfgls()`. |
| `trval` | Support for `prmat()`. |
| `unflip` | Reverses order of columns of a dataframe or matrix. |
| `variogrm` | Empirical variogram in 2D. |
| `vcokrig` | Crossvalidation of cokriging model from `cokrig()`. |
| `vcokrig1` | Crossvalidation of cokriging model from `cokrig1()`. |

Table 11: **Library *spatCSU* – Geostatistics**

## 7.4 Lattice Data

When examining lattice data, spatial correlation is of major interest. The special data structure makes it necessary, though, to allow for weight matrices (proximity matrices) that describe neighborhood relations between points. Helpful functions to create such matrices are `spwtctg()` (using data set with information on neighboring polygons), `spwtjoin()` (using chess moves) and `spwtdist()` (using distances). Rescaling of the weights is handled by `rescalew()`. The neighborhood information used by `spwtctg()` should come in an ASCII file with the total number of polygons given in the first line. For *each* polygon two more lines follow, stating the polygon ID and the number of neighbors, followed by the IDs of the neighbors.

Moran's I and Geary's C can be estimated by functions `morani()`, `bimorani()` or `gearyc()`. For measures of spatial correlation on data of the chess board type, see `bb()`, `bw()` or `ww()`.

A couple of functions deal with spatial AR models. These are `spatar()` and `spatlag()` for modelling and `spatt()` for testing.

| **Function** | **Action** |
|---|---|
| `bb` | Black-black join statistic. |
| `bimorani` | Moran's I to test for spatial cross-correlation. |
| `bw` | Black-white join statistic. |
| `gearyc` | Geary's C to test for spatial auto-correlation. |
| `join` | Compute join count statistics for lattice data. |
| `morani` | Moran's I. |
| `rescalew` | Rescale spatial weights matrix to rowsum 1. |
| `spatar` | Fit spatial autoregressive model. |
| `spatlag` | Fit spatial lag model. |
| `spatt` | Spatial t-test. |
| `spwtctg` | Generate spatial weights matrix using polygons. |
| `spwtjoin` | Generate spatial weights matrix using chess moves. |
| `spwtdist` | Generate spatial weights matrix using distances. |
| `ww` | White-white join statistic. |

Table 12: **Library *spatCSU* – Lattice Data**

## 7.5 Spatial Point Patterns

There is a variety of functions available for the analysis of spatial point patterns. Before working with data, the domain should be set by `ppregion()` or equivalently `ppset()`. The values can be checked for by `nppset()`.

In addition to what is possible in Ripley's *spatial* library, *spatCSU* allows for estimation of the K-function for *two* populations with envelope by `kfn2()` from data, or by `kenvl2()` from simulated processes. Tests for (non-)randomness can be performed calculating Clark and Evans' nearest neighbor index (`cenn()`, Pielou's index of non-randomness, or applying permutation procedures (`mrpp()`, `mrppa()`). The Cramer-von Mises goodness-of-fit test may be used to test for Neyman-Scott clustering or a Strauss process (`cramera()` and `cramerst()`). Fitting functions are `fitfreq()`, `ftstrau()`,or `neyman()`.

The intensity of a process can be estimated by kernel estimates or using the k-th nearest neighbor approach, or even a combination of the two. Take a look at `intker()`, `intknn()`, and `intknn2()`. An alternative is to use quadrat sampling, as is done by `quad()` and `quadrat()`.

Extending Ripley's functions, it is possible to simulate Matérn's clustering process (`agg()`), a poisson inhibition process (`inhom()`), and a Strauss softcore process (`softcore()`). Use `ppsetsd()` to set the random generator explicitly.

| **Function** | **Action** |
|---|---|
| `agg` | Simulates Matérn's clustering process. |
| `assoc` | Contingency table of nearest neighbors. |
| `cenn` | Clark and Evans' nearest neighbor index. |
| `cramera` | Test for Neyman-Scott clustering using Cramer-von Mises goodness-of-fit test. |
| `cramerr` | Test complete spatial randomness using Cramer-von Mises goodness-of-fit test. |
| `cramerst` | Test for Strauss process using Cramer-von Mises goodness-of-fit test. |
| `dmap` | Compute Kth nearest neighbor distances over a grid of equally spaced points and compare with distance when CSR holds. |
| `fitfreq` | Fits quadrat count data to various models. |
| `ftstrau` | Fit a Strauss process. |
| `inhom` | Simulates inhomogeneous Poisson point process. |

Table 13: **Library *spatCSU* – SPP (cont.)**

| **Function** | **Action** |
|---|---|
| `intker` | Kernel estimate of intensity function. |
| `intknn` | K-th nearest neighbor approach to estimate intensity. |
| `intknn2` | Estimates the intensity using a combination of the k-th nearest neighbors and kernel estimator. |
| `kaver` | Average from simulated K-functions. |
| `kenvl` | Averages and extremes from simulated K-functions ("envelope"). |
| `kenvl2` | Like kenvl(), but for two species populations. |
| `kern` | Support for kernel estimation. |
| `kfn` | Estimate scaled version of Ripley's K function. |
| `kfn2` | Like kfn(), but for two species populations. |
| `knear` | Support for kernel estimation. |
| `knear2` | Support for kernel estimation. |
| `knearn` | Support for kernel estimation. |
| `knearn2` | Support for kernel estimation. |
| `mindst` | Minimum distance within a point pattern. |
| `mrpp` | Multi-response permutation procedures testing for nonrandomness. |
| `mrppa` | Multi-response permutation procedures testing for spatial association. |
| `neyman` | Fit a Neyman-Scott clustering process. |
| `neyman1` | Support for `neyman()`. |
| `nppset` | Invisibly returns x and y coordinates of currently defined region. |
| `pielou` | Pielou's index on nonrandomness. |
| `pplik` | Pseudo likelihood estimation for fitting a Strauss process. |
| `ppsetsd` | Sets seed for Fortran random number generator. |
| `predker` | Kernel estimate of intensity function. |
| `predknn` | Kth nearest neighbor estimates of intensity. |
| `predknn2` | Kth nearest neighbor estimates of intensity. |
| `ppregion` | Initialize variables in Fortran code to set up domain for SPP. |
| `ppset` | See `ppregion()`. |
| `psim` | Simulate binomial process. |
| `qcir` | Quadrat sampling with circular plot. |
| `qsquare` | Quadrat sampling with rectangular plot. |
| `quad` | Estimates density and spatial distribution of a point pattern using quadrat sampling. |
| `quadadd` | Produce additional rectangles or circles for sampling. |

Table 13: **Library *spatCSU* – SPP (cont.)**

| **Function** | **Action** |
|---|---|
| `quadencr` | Check if rectangles (or circles) overlap. |
| `quadgen` | Generate random center points of new rectangles or circles. |
| `quadplot` | Plot ractanlges or circles from quadrat sampling. |
| `quadrat` | Estimates density and spatial distribution of a point pattern using quadrat sampling. |
| `quadsmpl` | Check how many points fall into sampled rectangles (or circles) and eventually plot data and sampled regions. |
| `softcore` | Simulates regular spatial pattern using a soft- core interaction. |
| `ssi` | Simulate sequential inhibition process (Matérn). |
| `strau1` | Support for `ftstrau()`. |
| `strauss` | Simulate Strauss process. |

Table 13: **Library *spatCSU* – SPP**

## 7.6 Data Sets

The library *spatCSU* contains all data sets that come with the library *spatial* described above, plus three more that come with the `mass` library. The latter are used for illustrative purposes in the chapter on spatial statistics in Venables and Ripley (1997). All these data sets are stored as ASCII-files with extension *.dat in the directory were spat98.zip was unzipped. Information on data sources can be found in file pp.fil in the same directory. Additional data sets are given in the table below.

| | | | |
|---|---|---|---|
| buffalo.asc | buffspa.asc | col.asc | colspa.asc |
| denv1.asc | denv2.asc | denv3.asc | denv4.asc |
| denv5.asc | denv6.asc | denv7.asc | hardwood.asc |
| denver.asc | grass.asc | buffalo.asc | buffspa.asc |
| col.asc | colspa.asc | denv1.asc | denv2.asc |
| denv3.asc | denv4.asc | denv5.asc | denv6.asc |
| denv7.asc | hardwood.asc | denver.asc | grass.asc |
| loblolly.dat | bb.dat | range.dat | |

Table 14: **Library *spatCSU* – Data Sets**

## 7.7 Other Features

There is a very comprehensive manual describing functions and underlying theory. See **http://www.stat.colostate.edu/~rdavis/protected/manual.pdf** for further information.

# 8 The Module S+SpatialStats

## 8.1 General Concepts

The version of S-Plus currently used at the statistics department is version 4.5, release 2 under Windows NT 4.0. There is a commercial add-on module available for spatial statistics called *S+SpatialStats* (Mathsoft, 1997), version 1.1, release 1. It is described in detail in Kaluzny et al. (1998).

The *S+SpatialStats* module makes intensive use of the concepts of object oriented programming implemented in S-Plus. It is designed particularly for geographers. A manual is available written by Kaluzny et al. (1998). It contains a special chapter to describe the use of *S+SpatialStats* together with Geographical Information Systems, exemplified by ARC/INFO.

The following subsections describe what functions are available in the *S+SpatialStats* module.

## 8.2 Working with Data

The table below shows the functions available for im- and export of data in the *S+SpatialStats* module[3] . The basic function to use is `read.neighbor()` which reads in an ASCII-file with one line for each region containing at least a region identifier and neighborhood information. There are also functions `read.geoeas()` and `write.geoeas()` to read and write files in GEO-EAS format.

| **Function** | **Action** |
|---|---|
| `read.neighbor` | Reads in Ascii-Files containing neighboring information. |
| `read.geoeas` | Read files in the GEO-EAS file format. |
| `write.geoeas` | Write data out to file in GEO-EAS file format. |

Table 15: **S+SpatialStats – Working with Data**

Other S-Plus base functions for data import and export are `scan()`, `write()`, `read.table()`, `import.data()` and `export.data()`. The advantage of `read.neighbor()` is that it directly creates an object of class `spatial.neighbor`, which is required by several other functions in

[3]To transfer data between S-Plus and ARC/INFO use the module S+GISLINK. This module is not described here.

*S+SpatialStats*. If base functions are used to read in data, objects of class `spatial.neighbor` must be created explicitly using the function `spatial.neighbor()`.

## 8.3 Geostatistics

Geostatistics basically comes down to smoothing data over a grid. A crucial role play the spatial correlations observed in the data. The are analyzed using theoretical tools like the variogram, covariogram and correlogram (see functions with corresponding names). *S+SpatialStats* allows for calculations in predefined directions to check for geometric anisotropy. In many cases, median polishing or some other technique is used for detrending. Theoretical covariance models like the exponential, gaussian or spherical model can be fit. The function `model.variogram()` enables the user to change the fitting parameters interactively. The results can be saved in objects of class `variogram`, `covariogram` or `correlogram` which have their own plotting methods. If geostatistical data are to be simulated, use `rfsim()`. This function creates an isotropic random field.

**Note** that the models in *S+SpatialStats* are usually stated in terms of so-called S-Plus formulas. Within the formula, the `loc()` function may be used to identify the location variables and to correct for geometric anisotropy.

The following table summarizes what is available in *S+SpatialStats* for geostatistical data analysis.

| **Function** | **Action** |
|---|---|
| `boxplot.vgram.cloud` | Boxplot of variogram cloud, binned by distance. |
| `correlogram` | Estimate empirical correlogram. |
| `covariogram` | Estimate empirical covariogram. |
| `exp.cov` | Theoretical covariance based on exponential model. |
| `exp.vgram` | Theoretical variogram based on exponential covariances. |
| `gauss.cov` | Theoretical covariance based on gaussian model. |
| `gauss.vgram` | Theoretical variogram based on gaussian covariances. |
| `identify.vgram.cloud` | `identify()`-method for class `vgram.cloud`. |
| `krige` | Ordinary and universal kriging in 2D. |

Table 16: **S+SpatialStats – Geostatistics (cont.)**

| **Function** | **Action** |
|---|---|
| `linear.vgram` | Theoretical isotropic linear variogram. |
| `loc` | Correct for spatial anisotropy. |
| `model.variogram` | Plots empirical `variogram` object with theoretical fit. allows for interaction. |
| `panel.gamma0` | Panel function to add loees curves. |
| `predict.krige` | Compute kriging predictions. |
| `print.krige` | Print-method for class `krige`. |
| `plot.correlogram` | Plot-method for correlograms (class `correlogram`). |
| `plot.covariogram` | Plot-method for covariograms (class `covariogram`). |
| `plot.variogram` | Plot-method for variograms (class `variogram`). |
| `plot.vgram.cloud` | Plot-method for variogram clouds (class `vgram.cloud`). |
| `power.vgram` | Theoretical isotropic power variogram. |
| `rfsim` | Simulation of geostatistical data. |
| `scaled.plot` | Scatterplot with axes scaled to a given ratio (default is 1). |
| `spher.cov` | Theoretical covariance based on spherical model. |
| `spher.vgram` | Theoretical isotropic spherical variogram. |
| `summary.variogram` | Call and parameters for `variogram` object. |
| `twoway` | Estimate row, column and grand effects. Default estimates from median polishing. |
| `twoway.default` | See `twoway()`. |
| `twoway.formula` | Formula-method of `twoway()`. |
| `variogram` | Empirical variogram in 2D. Generic. |
| `variogram.cloud` | Compute all pairwise differences. |
| `variogram.default` | See `variogram()`. |
| `variogram.formula` | See `variogram()`. |

Table 16: **S+SpatialStats – Geostatistics**

## 8.4 Lattice Data

Basic assumptions for the analysis of Lattice Data in *S+SpatialStats* are multivariate normality and stationarity. Data including neighborhood information can be read into a neighborhood matrix with `read.neighbor()`. If this information is to be added later, use `neighbor.grid()` or `find.neighbor()`

and `quad.tree()`. Neighborhood information should be stored in objects of class `"spatial.neighbor"`. The function with the corresponding name allows for construction of such objects. Redundancy may be removed by `spatial.condense()`. A `spatial.neighbor` -object can contain several neighborhood matrices at once. Since neighborhood information is stored in sparse matrix form, some special functions are needed to perform mathematical operations. These are `spatial.cg.solve()`, `spatial.solve()`, `spatial.determinant()`, `spatial.multiply()` and `spatial.sum()`.

Moran's and Geary's measures of spatial correlation are found by `spatial.cor()`. Spatial linear models are fit by `slm()`. The class of the returned object is of the same name ("`slm`") and may be tested in a likelihood ratio test using `lrt.slm()`. Possible models for the covariance structure are given by conditional autoregression (CAR), simultaneous autoregression (SAR) or moving average (MA) models.

The functions for variogram estimation are also helpful for analysis of Lattice Data. See section*Geostatistics* above for a list.

| **Function** | **Action** |
|---|---|
| `check.islands` | Detects isolated regions in object of class `spatial.neighbor`. |
| `find.neighbor` | Find nearest neighbor in `quad.tree` -object. |
| `lrt.slm` | Likelihood ratio test for spatial linear model. |
| `neighbor.grid` | Create `spatial.neighbor` -object from regular grid. |
| `print.lrt.slm` | Print-method for LRT on `slm` -object. |
| `print.slm` | Print-method for `slm` -object. |
| `print.spatial.cor` | Print-method for `spatial.cor` -object. |
| `print.spatial.neighbor` | Print-method for `spatial.neighbor` -object. |
| `print.summary.slm` | Print-method for object of class `summary.slm`. |
| `quad.tree` | Reorder matrix to become a `quad.tree` -object. |
| `rayplot` | Add rays to existing plot. |
| `slm` | Fit a spatial linear model. |
| `slm.nlminb` | Fit a profile likelihood. |
| `spatial.cg.solve` | Solves linear system with sparse matrix. |
| `spatial.condense` | Remove redundant elements of `spatial .neighbor` -object. |
| `spatial.cor` | Compute measures of spatial correlation. |

Table 17: **S+SpatialStats – Lattice Data (cont.)**

| **Function** | **Action** |
|---|---|
| `spatial.determinant` | Find determinant of `spatial.neighbor` -matrix. |
| `spatial.multiply` | Matrix multiplication with `spatial .neighbor` -matrix. |
| `spatial.neighbor` | Creates objects of class `spatial.neighbor`. |
| `spatial.solve` | Solve linear system with `spatial.neighbor` -matrix. |
| `spatial.subset` | Extract subset of spatial units from `spatial.neighbor` -object. |
| `spatial.sum` | Sum up two objects of class `spatial .neighbor`. |
| `spatial.weights` | Find matrix of spatial weights. |
| `summary.slm` | Summary of a spatial linear regression model object. |
| `triangulate` | Delaunay's triangulation. |

Table 17: **S+SpatialStats** – **Lattice Data**

## 8.5 Spatial Point Patterns

Data on Spatial Point Patterns should be stored in an object of class `"spp"`. This is essentially a data frame containing columns of coordinates plus some additional attributes. The constructing function `spp()` is accompanied by `is.spp()` and `as.spp()`.

Several functions are related to the construction or checks on polygons, like `bbox()`, `is.convex.poly()`, `points.in.poly()`, `poly.expand()` and `poly.grid()`. In terms of statistical analysis, complete spatial randomness may be tested using nearest neighbor methods. The empirical distribution functions of *point-to-point* or *origin-to-point* nearest neighbor distances are calculated and plotted by `Ghat()` and `Fhat()`, respectively. Ripley's K-function is calculated in `Khat()` and `Kenv()`. Its scaled version (which is linear for a homogeneous Poisson process) is implemented in `Lhat()` and `Lenv()`.

| **Function** | **Action** |
|---|---|
| `Fhat` | EDF for origin-to-point nearest neighbor distances, $\hat{\mathrm{F}}$. |
| `Ghat` | EDF for point-to-point nearest neighbor distances, $\hat{\mathrm{G}}$. |
| `Khat` | Ripley's K function. |
| `Kenv` | Compute simulations for Khat. |
| `Lhat` | Ripley's K function. |
| `Lenv` | Compute simulations for `Khat()`. |
| `as.spp` | Create object of class `spp`. |
| `bbox` | Bounding box for object of class `spp`. |
| `intensity` | Estimate the intensity of a spatial point pattern. |
| `is.spp` | Check for class `spp`. |
| `is.convex.poly` | Tests for convexity of polygon. |
| `kern2d` | Kernel smoother in 2D. |
| `make.pattern` | Generate random points in 2D. |
| `plot.spp` | Plot-method for `spp`-objects. |
| `points.in.poly` | Checks if xy-coordinates given are in polygon. |
| `points.spp` | Add points to plot from `plot.spp()`. |
| `poly.area` | Compute area within polygon. |
| `poly.expand` | Expand polygon by small fraction to truly contain all points. |
| `poly.grid` | Determine lattice of points within polygon. |
| `spp` | Create `spp`-object which represents a spatial point pattern. |
| `summary.spp` | Summary of a spatial point pattern object. |

Table 18: **S+SpatialStats – Point Patterns**

## 8.6 Data Sets

The following data sets come with the *S+SpatialStats* module as S-Plus-objects. Short descriptions follow Kaluzny (1997, Appendix B).

| **Data set** | **Contents** |
|---|---|
| `aquifer` | Wolfcamp aquifer data. |
| `bramble` | Bramble cane data. |
| `coal.ash` | Coal ash data. |
| `iron.ore` | Iron ore data. |
| `lansing` | Lansing Woods tree data. |
| `quakes.bay` | Bay area earthquakes. |

Table 19: **S+SpatialStats – Data Sets (cont.)**

| **Data set** | **Contents** |
|---|---|
| `quakes.wash` | Washington State earthquakes in 1980. |
| `scallops` | Scallop abundance data. |
| `sids` | Sudden infant death syndrome data 1974-1978. |
| `sids2` | Sudden infant death syndrome data 1979-1984. |
| `sids.neighbor` | Neighbors for sids data. |
| `wheat` | Wheat grain and straw yield. |

Table 19: **S+SpatialStats – Data Sets**

## 8.7 Other Features

There are a couple of other functions in *S+SpatialStats* not related directly to any of the topics above, but still valuable tools in statistical analysis. They are mostly related to mathematical calculations or to plotting, e.g. hexagonal binning. An exception is `lrt()` which performs a likelihood ratio test on fitted objects.

| **Function** | **Action** |
|---|---|
| `hessian` | Hessian of a function of several parameters. |
| `hexbin` | Creates object of class `hexbin` for hexagonal binning. |
| `lrt` | Perform likelihood ratio test. |
| `plot.hexbin` | Plots `hexbin` object. |
| `powers` | Find powers of variables in term (i.e. product) given. |

Table 20: **S+SpatialStats – Other Features**

# 9 Other Functions for Spatial Statistics

There are a few functions in the S-Plus base distribution that may be also used for spatial statistics. The `loess()` function may be used to fit local trend surfaces in 2D. The function `interp()` uses a method proposed by Akima (c.f. Akima,1978). The library delauney (see statlib) computes the Dirichlet-tessalation and the Delaunay triangulation. For some more detail, see Venables and Ripley (1997). Besides that, the usual graphical functions are a valuable tool in any spatial analysis.

## 10 Summary

This paper gives an overview of the functions available for spatial statistics in S-Plus. Most of these are part of the public libraries *splancs* (by B. Rowlingson and P. Diggle), *sgeostat* (by J.J. Majure), *spatial* (By B.D. Ripley) and *spatCSU* (compiled by R.Reich and R.Davis). Others are implemented in the commercial add-on module *S+SpatialStats*.

In terms of the number of functions available, the smallest library is *sgeostat*. It concentrates on geostatistics, i.e. on variogram estimation and surface fitting. This includes trend estimation as well as kriging. The library *sgeostat* is the newest public library currently available and the only one that makes use of the concept of object oriented programming.

The field of application of *splancs* are spatial point patterns. The library offers many functions for analysis involving one or two populations. Also, some methods for space-time analysis are implemented. Several graphical functions come as convenient extra features.

The most comprehensive public library is *spatCSU*. The name *spatCSU* is chosen somewhat ad lib, but reflects the location where it was collected and partly written. The library was originally based on Ripley's *spatial* library, but is extended considerably by now. It offers not only the tools from *spatial* for geostatistics and spatial point patterns, but extends them, and adds functionality for the analysis of lattice data. It is even more comprehensive than the the module *S+SpatialStats*. In addition, this library is well described in a detailed manual.

The module *S+SpatialStats* also covers all three major areas of spatial statistics. It makes intensive use of object oriented programming and is well structured and easy to use. The well-defined classes and methods allow for extension by the user. The manual (Kaluzny et al., 1998) describes the handling very well and gives many examples and references. It includes many references to statistical texts, notably Cressie (1993).

In summary, there is a wide range of functions and methods already available for the analysis of spatial data in libraries and modules. Some additional work should be done in implementing methods for spatio-*temporal* data analysis to close a gap that is still open.

## 11 References

Akima, H. (1978). *A method of bivariate interpolation and smooth surface fitting for irregularly distributed data points.* ACM Transactions on Mathematical Software 4, 148-159.

Cressie, N. (1993). *Statistics for Spatial Data*, Wiley.

Haining, R. (1990). *Spatial Data Analysis in the Social and Environmental Sciences*, Cambridge University Press, Cambridge.

Kaluzny, S.P., Vega, S.C., Cardoso, T.P. and Shelly,A.A. (1998). *S+SpatialStats: User's manual for Windows and Unix*, Springer.

Mathsoft (1997). *S+SpatialStats 1.1*, Release 1. Mathsoft, Inc., Seattle.

Mathsoft (1998). *S-Plus 4.5*, Release 2. Mathsoft, Inc., Seattle.

Ripley, B.D. (1981). *Spatial Statistics.* John Wiley and Sons, New York.

Ripley, B.D. (1988). *Statistical Inference for Spatial Processes.* Cambridge University Press, Cambridge.

Venables, W.N. and Ripley, B.D. (1997). *Modern applied statistics with S-Plus*, 2nd edition. Springer, New York.

# List of Tables