

# Package ‘sgeostat’

September 21, 2009

**Title** An Object-oriented Framework for Geostatistical Modeling in S+

**Version** 1.0-23

**Author** S original by James J. Majure <majure@iastate.edu> Iowa State University, R port +  
extensions by Albrecht Gebhardt <albrecht.gebhardt@uni-klu.ac.at>

**Maintainer** Albrecht Gebhardt <albrecht.gebhardt@uni-klu.ac.at>

**Description** An Object-oriented Framework for Geostatistical Modeling in S+

**Depends** stats, grDevices, graphics

**License** file LICENSE

**Repository** CRAN

**Date/Publication** 2009-09-21 18:43:32

## R topics documented:

est.variogram . . . . .	2
fit.trend . . . . .	3
fit.variogram . . . . .	4
identify.point . . . . .	6
in.chull . . . . .	7
in.polygon . . . . .	8
krige . . . . .	9
lagplot . . . . .	10
maas . . . . .	12
maas.bank . . . . .	12
pair . . . . .	13
plot.point . . . . .	14
plot.variogram . . . . .	15
point . . . . .	16
print.pair . . . . .	17
print.point . . . . .	18
sgeostat-internal . . . . .	19
spacebox . . . . .	19
spacecloud . . . . .	20

---

est.variogram      *Variogram Estimator*

---

## Description

Calculate empirical variogram estimates.

An object of class `variogram` contains empirical variogram estimates generated from a point object and a pair object. A variogram object is stored as a data frame containing six columns: `lags`, `bins`, `classic`, `robust`, `med`, and `n`. The length of each vector is equal to the number of lags in the pair object used to create the variogram object, say  $l$ . The `lags` vector contains the lag numbers for each lag, beginning with one (1) and going to the number of lags ( $l$ ). The `bins` vector contains the spatial midpoint of each lag. The `classic`, `robust`, and `med` vectors contain the classical,

$$\gamma_c(h) = \frac{1}{n} \sum_{(i,j) \in N(h)} (z(x_i) - z(x_j))^2$$

robust,

$$\gamma_m(h) = \frac{(\frac{1}{n} \sum_{(i,j) \in N(h)} (\sqrt{|z(x_i) - z(x_j)|}))^4}{0.457 + \frac{0.494}{n}}$$

and median

$$\gamma_m(h) = \frac{(\text{median}_{(i,j) \in N(h)} (\sqrt{|z(x_i) - z(x_j)|}))^4}{0.457 + \frac{0.494}{|N(h)|}}$$

variogram estimates for each lag, respectively (see Cressie, 1993, p. 75). The `n` vector contains the number  $|N(h)|$  of pairs of points in each lag  $N(h)$ .

## Usage

```
est.variogram(point.obj, pair.obj, a1, a2)
```

## Arguments

<code>point.obj</code>	a point object generated by <code>point()</code>
<code>pair.obj</code>	a pair object generated by <code>pair()</code>
<code>a1</code>	a variable to calculate semivariogram for
<code>a2</code>	an optional variable name, if entered cross variograms will be created between <code>a1</code> and <code>a2</code>

**Value**

A variogram object:

lags	vector of lag identifiers
bins	vector of midpoints of each lag
classic	vector of classic variogram estimates for each lag
robust	vector of robust variogram estimates for each lag
med	vector of median variogram estimates for each lag
n	vector of the number of pairs in each lag

**References**

<http://www.gis.iastate.edu/SGeoStat/homepage.html>

**See Also**

[point](#), [pair](#)

**Examples**

```
maas.v<-est.variogram(maas.point,maas.pair,'zinc')
```

---

fit.trend

*Fit polynomial trend functions*

---

**Description**

Fits a polynomial trend function to a [point](#) object. Similar to functions in B. Ripley's spatial library.

**Usage**

```
fit.trend(point.obj, at, np=2, plot.it=TRUE)
```

**Arguments**

point.obj	<a href="#">point</a> object
at	name of dependent variable in <code>point.obj</code>
np	degree of polynom to be fitted
plot.it	switches generation of a contour plot

**Value**

beta	estimated parameters
...	

## References

<http://www.gis.iastate.edu/SGeoStat/homepage.html>

---

fit.variogram      *Variogram Model Fit*

---

## Description

Fit variogram models (exponential, spherical, gaussian, linear) to empirical variogram estimates.

An object of class `variogram.model` represents a fitted variogram model generated by fitting a function to a variogram object. A `variogram.model` object is composed of a list consisting of a vector of parameters, `parameters`, and a semi-variogram model function, `model`.

## Usage

```
fit.variogram(model="exponential", v.object, nugget, sill,
              range, slope, ...)
fit.exponential(v.object, c0, ce, ae, type='c',
                iterations=10, tolerance=1e-06, echo=FALSE, plot.it=FALSE, weighted=TRUE)
fit.gaussian(v.object, c0, cg, ag, type='c',
             iterations=10, tolerance=1e-06, echo=FALSE, plot.it=FALSE, weighted=TRUE)
fit.spherical(v.object, c0, cs, as, type='c', iterations=10,
              tolerance=1e-06, echo=FALSE, plot.it=FALSE, weighted=TRUE,
              delta=0.1, verbose=TRUE)
fit.wave(v.object, c0, cw, aw, type='c',
          iterations=10, tolerance=1e-06, echo=FALSE, plot.it=FALSE, weighted=TRUE)
fit.linear(v.object, type='c', plot.it=FALSE, iterations=1, c0=0, c1=1)
```

## Arguments

<code>model</code>	only available for <code>fit.variogram</code> , switches what kind of model should be fitted ("exponential", "wave", "gaussian", "spherical", "linear").
<code>v.object</code>	a variogram object generated by <code>est.variogram()</code>
<code>nugget, sill, range, slope</code>	only available for <code>fit.variogram</code> , initial estimates for specified variogram model (slope only for <code>fit.linear</code> )
<code>c0</code>	initial estimate for nugget effect, valid for all variogram types, partial sill ( $c_X$ ) and (asymptotical) range ( $a_X$ ) as follows:
<code>ce, ae</code>	initial estimates for the exponential variogram model
<code>cg, ag</code>	initial estimates for the gaussian variogram model
<code>cs, as</code>	initial estimates for the spherical variogram model
<code>cw, aw</code>	initial estimates for the periodical variogram model
<code>c1</code>	initial estimates for the linear variogram model (slope)

type	one of 'c' (classic), 'r' (robust), 'm' (median). Indicates to which type of empirical variogram estimate the model is to be fit.
iterations	the number of iterations of the fitting procedure to execute.
tolerance	the tolerance used to determine if model convergence has been achieved.
delta	initial stepsize (relative) for pseudo Newton approximation, applies only to fit.spherical
echo	if TRUE, be verbose.
verbose	if TRUE, be verbose (show iteration for spherical model fit).
plot.it	if TRUE, the variogram estimate will be plotted each iteration.
weighted	if TRUE, the fit will be done using weighted least squares, where the weightes are given in Cressie (1991, p. 99)
...	only fit.variogram: additional parameters to hand through to specific model fit functions

**Value**

A variogram.model object:

parameters	vector of fitted model parameters
model	function implementing a valid variogram model

**Note**

fit.exponential, fit.gaussian and fit.wave use an iterative, Gauss-Newton fitting algorithm to fit to an exponential or gaussian variogram model to empirical variogram estimates. fit.spherical uses the same algorithm but with differential quotients in place of first derivatives. When weighted is TRUE, the regression is weighted by  $n(h)/\gamma(h)^2$  where the numerator is the number of pairs of points in a given lag.

Setting iterations to 0 means no fit procedure is applied. Thus parameter values from external sources can be plugged into a variogram model object.

**References**

<http://www.gis.iastate.edu/SGeoStat/homepage.html>

**See Also**

[est.variogram](#)

**Examples**

```
#
# automatic fit:
#
maas.vmod<-fit.gaussian(maas.v, c0=60000, cg=110000, ag=800, plot.it=TRUE,
iterations=30)
#
```

```
# iterations=0, means no fit, intended for "subjective" fit
#
maas.vmod.fixed<-fit.variogram("gaussian",maas.v,nugget=60000,sill=110000,range=800,plot.it=
```

---

identify.point      *Identify points on a Point Object*

---

## Description

Plot variable values next to locations after the `plot.point()` function.

## Usage

```
## S3 method for class 'point':
identify(x, v, ...)
```

## Arguments

x	a point object generated by <code>point()</code>
v	use values of variable "v" as labels
...	additional arguments to <code>identify</code>

## Value

An integer vector containing the indexes of the identified points.

## References

<http://www.gis.iastate.edu/SGeoStat/homepage.html>

## See Also

[plot.point](#)

## Examples

```
plot(maas.point)
# use indices as labels:
identify(maas.point)
# use values as labels:
identify(maas.point,v="zinc")
```

---

`in.chull`*Convex hull test*

---

**Description**

Checks if points are in the interior of a convex hull.

**Usage**

```
in.chull(x0, y0, x, y)
```

**Arguments**

<code>x0</code>	coordinates of points to check
<code>y0</code>	see <code>x0</code>
<code>x</code>	coordinates defining the convex hull
<code>y</code>	see <code>x</code>

**Details**

Uses a simple points-in-polygon check combined with the `chull` function.

**Value**

<code>comp1</code>	Description of 'comp1'
<code>comp2</code>	Description of 'comp2'

**Author(s)**

Albrecht Gebhardt <agebhard@uni-klu.ac.at>

**References**

Follows an idea from algorithm 112 from CACM (available at <http://www.netlib.org/tomspdf/112.pdf>)

**See Also**

[in.convex.hull](#), [chull](#)

**Examples**

```
in.chull(c(0,1),c(0,1),c(0,1,0,-1),c(-1,0,1,0))  
# should give: TRUE FALSE
```

---

in.polygon	<i>In-Polygon test</i>
------------	------------------------

---

### Description

Checks if points are in the interior of a polygon.

### Usage

```
in.polygon(x0, y0, x, y)
```

### Arguments

x0	coordinates of points to check
y0	see x0
x	coordinates defining the polygon
y	see x

### Details

Uses a simple points-in-polygon check combined with the [polygon](#) function. Polygon is closed automatically.

### Value

comp1	Description of 'comp1'
comp2	Description of 'comp2'

### Author(s)

Albrecht Gebhardt <agebhard@uni-klu.ac.at>

### References

Follows an idea from algorithm 112 from CACM (available at <http://www.netlib.org/tomspdf/112.pdf>)

### See Also

[in.convex.hull](#), [polygon](#), [in.chull](#)

### Examples

```
in.polygon(c(0,1), c(0,1), c(0,1,0,-1), c(-1,0,1,0))  
# should give: TRUE FALSE
```

---

krige	<i>Kriging</i>
-------	----------------

---

### Description

Carry out spatial prediction (or kriging).

### Usage

```
krige(s, point.obj, at, var.mod.obj, maxdist=NULL, extrap=FALSE, border)
```

### Arguments

<code>s</code>	a point object, generated by <code>point()</code> , at which prediction is carried out
<code>point.obj</code>	a point object, generated by <code>point()</code> , containing the sample points and data
<code>at</code>	the variable, contained in <code>point.obj</code> , for which prediction will be carried out
<code>var.mod.obj</code>	variogram object
<code>maxdist</code>	an optional maximum distance. If entered, then only sample points (i.e. in <code>point.obj</code> ) within <code>maxdist</code> of each prediction point will be used to do the prediction at that point. If not entered, then all <code>n</code> sample points will be used to make the prediction at each point.
<code>extrap</code>	logical, indicates if prediction outside the convex hull of data points should be done, default <code>FALSE</code>
<code>border</code>	optional polygon (list with two components <code>x</code> and <code>y</code> of same length) representing a (possibly non convex) region of interest to be used instead of the convex hull. Needs <code>extrap=TRUE</code> .

### Value

A point object which is a copy of the `s` object with two new variables, `zhat` and `sigma2hat`, which are, respectively, the predicted value and the kriging variance.

### References

<http://www.gis.iastate.edu/SGeoStat/homepage.html>

### See Also

[est.variogram](#), [fit.variogram](#)

**Examples**

```

# a single point:
prdpnt <- point(data.frame(list(x=180000,y=331000)))
prdpnt <- krige(prdpnt, maas.point, 'zinc', maas.vmod)
prdpnt

# kriging on a grid (slow!)
grid <- list(x=seq(min(maas$x),max(maas$x),by=100),
            y=seq(min(maas$y),max(maas$y),by=100))
grid$xr <- range(grid$x)
grid$xs <- grid$xr[2] - grid$xr[1]
grid$yr <- range(grid$y)
grid$ys <- grid$yr[2] - grid$yr[1]
grid$max <- max(grid$xs, grid$ys)
grid$xy <- data.frame(cbind(c(matrix(grid$x, length(grid$x), length(grid$y)),
                                c(matrix(grid$y, length(grid$x), length(grid$y), byrow=TRUE))))
colnames(grid$xy) <- c("x", "y")
grid$point <- point(grid$xy)
data(maas.bank)
grid$krige <- krige(grid$point,maas.point,'zinc',maas.vmod,
                    maxdist=1000,extrap=FALSE,border=maas.bank)
op <- par(no.readonly = TRUE)
par(pty="s")
plot(grid$xy, type="n", xlim=c(grid$xr[1], grid$xr[1]+grid$max),
      ylim=c(grid$yr[1], grid$yr[1]+grid$max))
image(grid$x,grid$y,
       matrix(grid$krige$zhat, length(grid$x), length(grid$y)),
       add=TRUE)
contour(grid$x,grid$y,
        matrix(grid$krige$zhat, length(grid$x), length(grid$y)),
        add=TRUE)
data(maas.bank)
lines(maas.bank$x,maas.bank$y,col="blue")
par(op)

```

---

lagplot

*Lag Scatter Plot*


---

**Description**

Create a spatially lagged scatter plot, e.g. plot  $z(s)$  versus  $z(s+h)$ , where  $h$  is a lag in a pair object.

**Usage**

```
lagplot(point.obj, pair.obj, a1, a2, lag=1, std=FALSE, query.a, xlim, ylim)
```

**Arguments**

<code>point.obj</code>	a point object generated by <code>point()</code>
<code>pair.obj</code>	a pair object generated by <code>pair()</code>
<code>a1</code>	a variable to plot
<code>a2</code>	an optional variable name, if entered the plot will be created between <code>a1</code> and <code>a2</code>
<code>lag</code>	the lag to plot
<code>std</code>	a logical variable indicating whether the data should be standardized to their means and standard deviations before plotting
<code>query.a</code>	an optional variable name, if entered, the value of the variable will be displayed on the graphics device for points identified by the user.
<code>xlim</code>	a vector of length 2 indicating the x limits of the graphics page
<code>ylim</code>	a vector of length 2 indicating the y limits of the graphics page

**Value**

NULL

**Note**

When `query.a` is entered, the user will be prompted to identify points on the display device. Because each point in the plot represents a pair of locations, the user must identify each point twice, once for the "from" point and once for the "to" point. Querying is ended by pressing the middle mouse button on the mouse while the cursor is in the display window.

**References**

<http://www.gis.iastate.edu/SGeoStat/homepage.html>

**See Also**

[point](#), [pair](#)

**Examples**

```
opar <- par(ask = interactive() && .Device == "X11")
lagplot(maas.point, maas.pair, 'zinc')
# with identifying pairs:
lagplot(maas.point, maas.pair, 'zinc', lag=2, query.a='zinc')
par(opar)
```

*maas**maas- zinc measurements*

---

**Description**

Zinc measurements as groundwater quality variable.

**Usage**

*maas*

**Value**

list with components *x,y* and *zinc*.

**References**

gstat E.J Pebesma (E.J.Pebesma@frw.uva.nl) <http://www.frw.uva.nl/~pebesma/gstat/>

**See Also**

[maas.bank](#)

---

*maas.bank**maas.bank - coordinates*

---

**Description**

Coordinates of maas bank. To be used together with [maas](#).

**Usage**

*maas.bank*

**Value**

list with components *x* and *y*.

**References**

gstat E.J Pebesma (E.J.Pebesma@frw.uva.nl) <http://www.frw.uva.nl/~pebesma/gstat/>

**See Also**

[maas](#)

pair

*Pair Object***Description**

Create a pair object from a point object.

A pair object contains information defining pairs of points contained in a point object. A pair object is a list containing five vectors: `from`, `to`, `lags`, `dist`, and `bins`. The length of each of these vectors (except `bins`) is equal to the number of pairs of points being represented, say  $k$ . The vectors `from` and `to` contain pointers into the vectors of a point object, pointing to each member of the pair of points (e.g., `from[k]` points to  $s_i$  and `to[k]` points to  $s_j$ ). The vector `dist` contains the distance between the pairs of points. The vector `lags` contains the lag number to which each pair of points has been assigned. The vector `bins` contains the spatial midpoint between each lag and is used for plotting.

**Usage**

```
pair(point.obj,num.lags=10,type='isotropic', theta=0, dtheta=5, maxdist)
```

**Arguments**

<code>point.obj</code>	a point object generated by <code>point()</code>
<code>num.lags</code>	the number of lags into which to divide the pairs of points in the pair object. The lags are all of equal size.
<code>type</code>	either <code>'isotropic'</code> or <code>'anisotropic'</code> . If <code>'isotropic'</code> then all $\binom{n}{2}$ possible pairs of points are represented in the pair object. If <code>'anisotropic'</code> , then the arguments <code>theta</code> and <code>dtheta</code> are used to determine which pairs of points to include.
<code>theta</code>	an angle, measured in degrees from the horizontal x axis, that determines pairs of points to be included in the pair object (see Notes below).
<code>dtheta</code>	a tolerance angle, around <code>theta</code> , measured in degrees that determines pairs of points to be included in the pair object (see Notes below).
<code>maxdist</code>	the distance beyond which not to consider pairs of points. A large number of spatial locations can cause the <code>pair</code> function to consume a considerable amount of computation time. In most cases, spatial dependence can be adequately characterized without examining the entire spatial extent of the data set.

**Value**

A pair object:

<code>from</code>	vector of indices into the point object for "from" point
<code>to</code>	vector of indices into the point object for "to" point
<code>lags</code>	vector of spatial lags of each pair
<code>dist</code>	vector of distances between each pair
<code>bins</code>	vector of spatial midpoints of each lag (used for plotting)

**NOTE**

Name of this function changed from `pairs` to `pair` to avoid conflicts with R's `pairs` function!!

**Note**

When creating an anisotropic pair object, the assumption is that the direction, as well as the distance, between pairs of points is important in describing the variation. Using the `theta` and `dtheta` arguments, pairs of points that meet direction requirements can be selected. A pair of points will be included when the angle between the positive x axis and the vector formed by the pair of points falls within the tolerance angle given by  $(\theta - d\theta, \theta + d\theta)$

**References**

<http://www.gis.iastate.edu/SGeoStat/homepage.html>

**See Also**

`point`

**Examples**

```
maas.pair <- pair(maas.point,num.lags=10,maxdist=2000)
maas.pair25 <- pair(maas.point,num.lags=10,type='anisotropic',
                    theta=25,maxdist=500)
```

---

plot.point

*Plot Point Objects*

---

**Description**

Plot the spatial locations in a point object, optionally coloring by quantile.

**Usage**

```
## S3 method for class 'point':
plot(x, v, legend.pos=0, axes=TRUE, xlab='', ylab='', add=FALSE, ...)
```

**Arguments**

<code>x</code>	a point object generated by <code>point()</code>
<code>v</code>	an optional variable name, if entered will divide the points into quantiles and color using 4 colors
<code>legend.pos</code>	position of legend (0 - none, 1 - bottom-left, 2 - bottom-right, 3 - top-right, 4 - top-left), requires <code>Lang(v)</code>
<code>axes</code>	logical, whether to plot axes
<code>xlab, ylab</code>	axes labels, default none

```
add          usefull for overlaying images with a point plot
...         additional arguments for plot
```

**Value**

NULL

**References**

<http://www.gis.iastate.edu/SGeoStat/homepage.html>

**See Also**

[point](#)

**Examples**

```
plot(maas.point)
plot(maas.point,v='zinc')
plot(maas.point,v='zinc',xlab='easting',ylab='northing',axes=TRUE,legend.pos=4)
# plot additionally the maas bank:
data(maas.bank)
lines(maas.bank)
```

---

plot.variogram      *Plot Variogram*

---

**Description**

Plot empirical variogram estimates, optionally plotting a fitted variogram model.

**Usage**

```
## S3 method for class 'variogram':
plot(x, var.mod.obj, title.str, ylim, type='c', N=FALSE, ...)
```

**Arguments**

```
x          a variogram object generated by est.variogram()
var.mod.obj a variogram model object generated by a model fitting routine.
title.str  optional: an user supplied plot title
type      optional: which type of variogram model to plot, 'c' = classical, 'r' = robust,
           'm' median
N         logical, toggles printing of absolute pair counts per lag
ylim      optonal user supplied y dimension for the plot
...       additional arguments for plot
```

**Value**

NULL

**References**<http://www.gis.iastate.edu/SGeoStat/homepage.html>**See Also**[est.variogram](#)**Examples**

```
# two plots
oldpar <- par(mfrow=c(2,1))
plot(maas.v)
plot(maas.v, var.mod.obj=maas.vmod)
par(oldpar)
```

---

point

*Point Object*

---

**Description**

Create an object of class point from a data frame.

An object of class point represents the observed data of a spatial process. This includes the spatial location of sampling sites and the values observed at those sites. A point object is stored as a data frame. The data frame must contain one column for the X coordinate and one column for the Y coordinate of each point, as well as any number of columns representing data observed at the points.

**Usage**

```
point(dframe, x='x', y='y')
```

**Arguments**

dframe	a data frame containing the x and y coordinates for each point and the variables observed at each point
x	the name of the column in dframe that contains the x coordinate
y	the name of the column in dframe that contains the y coordinate

**Value**

A point object:

x	vector of x coordinates
y	vector of ycoordinates
var1	vector of the first variable
...	...
varm	vector of the mth variable

**References**

<http://www.gis.iastate.edu/SGeoStat/homepage.html>

**See Also**

[point](#)

**Examples**

```
data(maas)
maas.point <- point(maas)
```

---

`print.pair`                      *Pairs Object Description*

---

**Description**

Print descriptive information about a pair object.

**Usage**

```
## S3 method for class 'pair':
print(x, ...)
```

**Arguments**

x	a pair object generated by <code>pair()</code>
...	additional arguments for <code>print</code>

**Value**

NULL

**References**

<http://www.gis.iastate.edu/SGeoStat/homepage.html>

**See Also**[pair](#)**Examples**

```
print(maas.pair)
# gives:
# Pairs object: maas.pair
#
#      Type:          isotropic
#      Number of pairs: 8370
#      Number of lags: 10
#      Max h:         1999.867
```

---

print.point                    *Point Object Description*

---

**Description**

Print descriptive information about a point object.

**Usage**

```
## S3 method for class 'point':
print(x, ...)
```

**Arguments**

x	a point object generated by <code>point()</code>
...	additional arguments for <code>print</code>

**Value**

NULL

**References**

<http://www.gis.iastate.edu/SGeoStat/homepage.html>

**See Also**[point](#)

## Examples

```
print.point(maas.point)
# gives
# Point object: maas.point
#
#   Locations: 155
#
#   Attributes:
#       x
#       y
#       zinc
```

---

sgeostat-internal *Internal sgeostat functions*

---

## Description

Internal sgeostat functions

## Details

These functions are not intended to be called by the user.

The `krige` function interfaces to `krige.*`, `pair` to `pair.*` and `fit.trend` to `trend.*`.

---

spacebox *Boxplot of Variogram Cloud*

---

## Description

Create boxplots of square-root or squared differences between variable values at pairs of points versus the distance between the points.

## Usage

```
spacebox(point.obj, pair.obj, a1, a2, type='r')
```

## Arguments

<code>point.obj</code>	a point object generated by <code>point()</code>
<code>pair.obj</code>	a pairs object generated by <code>pair()</code>
<code>a1</code>	a variable to plot
<code>a2</code>	an optional variable name, if entered the plot will be created between <code>a1</code> and <code>a2</code>
<code>type</code>	either 'r' for square-root differences or 's' for squared differences

**Value**

NULL

**References**<http://www.gis.iastate.edu/SGeoStat/homepage.html>**See Also**[point](#), [pair](#)**Examples**

```
spacebox(maas.point, maas.pair, 'zinc')
```

---

spacecloud

*Variogram Cloud*

---

**Description**

Create a scatter plot of square-root or squared differences between variable values at pairs of points versus the distance between the points.

**Usage**

```
spacecloud(point.obj, pair.obj, a1, a2, type='r', query.a, ...)
```

**Arguments**

<code>point.obj</code>	a point object generated by <code>point()</code>
<code>pair.obj</code>	a pair object generated by <code>pair()</code>
<code>a1</code>	a variable to plot
<code>a2</code>	an optional variable name, if entered the plot will be created between <code>a1</code> and <code>a2</code>
<code>type</code>	either <code>'r'</code> for square-root differences or <code>'s'</code> for squared differences
<code>query.a</code>	an optional variable name, if entered, the value of the variable will be displayed on the graphics device for points identified by the user.
<code>...</code>	additional arguments for <code>plot</code>

**Value**

NULL

**Note**

When `query.a` is entered, the user will be prompted to identify points on the display device. Because each point in the plot represents a pair of locations, the user must identify each point twice, once for the "from" point and once for the "to" point. Querying is ended by pressing the middle mouse button on the mouse while the cursor is in the display window.

**References**

<http://www.gis.iastate.edu/SGeoStat/homepage.html>

**See Also**

`point`, `pair`

**Examples**

```
opar <- par(ask = interactive() && .Device == "X11")
spacecloud(maas.point,maas.pair,'zinc')
# identify some points:
spacecloud(maas.point,maas.pair,'zinc',query.a='zinc')
par(opar)
```

# Index

## \*Topic **datasets**

maas, 11  
maas.bank, 11

## \*Topic **spatial**

est.variogram, 1  
fit.trend, 3  
fit.variogram, 3  
identify.point, 5  
in.chull, 6  
in.polygon, 7  
krige, 8  
lagplot, 10  
pair, 12  
plot.point, 13  
plot.variogram, 14  
point, 15  
print.pair, 16  
print.point, 17  
sgeostat-internal, 18  
spacebox, 18  
spacecloud, 19

calcangle(*sgeostat-internal*), 18  
chull, 6, 7

est.variogram, 1, 5, 9, 15  
export.point(*sgeostat-internal*),  
18

fit.exponential(*fit.variogram*), 3  
fit.gaussian(*fit.variogram*), 3  
fit.linear(*fit.variogram*), 3  
fit.spherical(*fit.variogram*), 3  
fit.trend, 3, 18  
fit.variogram, 3, 9  
fit.wave(*fit.variogram*), 3

identify.point, 5  
in.chull, 6, 8  
in.convex.hull, 7, 8

in.polygon, 7

krige, 8, 18  
krige.all(*sgeostat-internal*), 18  
krige.maxdist  
(*sgeostat-internal*), 18

lagplot, 10

maas, 11, 11, 12  
maas.bank, 11, 11

pair, 2, 10, 12, 17–20  
pair.aniso(*sgeostat-internal*), 18  
pair.iso(*sgeostat-internal*), 18  
pair.newangle  
(*sgeostat-internal*), 18

pairs, 13  
plot.point, 6, 13  
plot.variogram, 14  
point, 2, 3, 10, 13, 14, 15, 16, 18–20  
polygon, 7, 8

prediction.matrix  
(*sgeostat-internal*), 18

print.pair, 16  
print.point, 17

sgeostat-internal, 18  
spacebox, 18  
spacecloud, 19

trend.matrix(*sgeostat-internal*),  
18

trend.value(*sgeostat-internal*),  
18

which.na(*sgeostat-internal*), 18