

S-PLUS Short Course

— Part 3 —

S+SPATIALSTATS

These notes are mostly based on

- S+SPATIALSTATS User's Manual, version 1.0 (1996). MathSoft, Inc.

The S+SPATIALSTATS Module

S+SPATIALSTATS is an *add-on* module to the S-PLUS system (i.e., it is bought separately). It has methods for the three main types of spatial data:

- *Geostatistical* data.
- *Lattice* data.
- *Spatial Point Patterns*

On the other hand, version 1.5 (current) is designed for two-dimensional data and does not include any methods for the analysis of *spatio-temporal* data.

Starting S+SPATIALSTATS

To start the spatial module in S-PLUS type at the S-PLUS prompt:

```
> module(spatial)
and the module is available. It can be unloaded
(deactivated) by the command:
> module(spatial, unload=T)
```

The Schedule

The third part of this S-PLUS Short Course will cover:

EDA of Spatial Data Exploratory Data Analysis through graphics in S-PLUS. Two datasets will be used; the `scallops` (non-gridded) data will be our primary data and the `coal.ash` (gridded) secondary.

Trend Trend estimation and removal.

Variogram Empirical variograms, detecting and correcting for anisotropy, and fitting theoretical variogram models.

Kriging Computing kriging predictions and standard errors, and plotting.

scallops EDA

The scallops Data

“The scallops data frame is a spatial data set listing the catch of scallops from a 1990 National Marine Fisheries Service trawl survey in the Atlantic Ocean. The survey area runs from the Delmarva Peninsula off the coast of Virginia and Maryland up to the George Banks.” (from the S-PLUS help-file for this data.)

Source: From Ecker and Heltshé (1994), who present a geostatistical analysis of the data.

Description

strata A factor indicating the National Marine Fisheries Service (NMFS) 4 digit strata designator in which the sample was taken.

sample Sample number per year ranging from 1 to approximately 450.

lat Location in terms of latitude of each sample in the Atlantic Ocean.

long Location in terms of negative longitude of each sample in the Atlantic Ocean.

tcatch total number of scallops caught at the *i*th sample location. This is `prerec + recruits`.

prerec Number of scallops whose shell length is smaller than 70 millimeters.

recruits Number of scallops whose shell length is 70 millimeters or larger.

Locations of Sites

Plot location of sites (and USA map):

```
> attach(scallops) ## attach the data:
> library('maps') ## use the map library in S-PLUS:
```

Warning messages:

```
The functions and datasets in library section
maps are not supported by MathSoft. in:
  library("maps")
> trellis.device()
> map('usa', xlim=c(-74, -71), ylim=c(38.2, 41.5))
> points(long, lat)
```

For more information on the `map` function, type `help(map)` at the prompt.

EDA of Total Catch

The variable of most interest is `tcatch` (total catch).

Histogram

The distribution of `tcatch` is highly right-skewed, as a *histogram* of `tcatch` shows:

```
> histogram(tcatch, nint=40)
```

The argument `nint` specifies the number of intervals (bars) to use.

By looking only at `tcatch < 100`, a *peak* at 0 is observed:

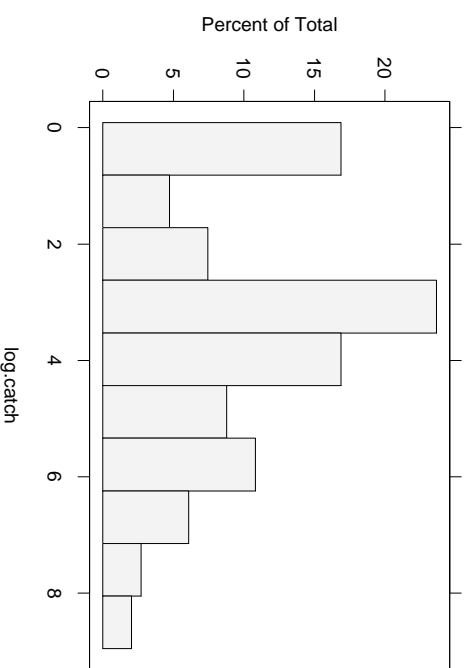
```
> histogram(tcatch[tcatch<100], nint=15)
```

This is simply because at some sites there are no scallops. A bad 'solution' (and one very often used) to this bimodal behavior of the data is to "log-plus-one" transform the data:

```
> log.catch <- log(tcatch+1)
```

```
> histogram(log.catch, nint=10)
```

It doesn't fix the problem, just diminishes it slightly.



Interpolation; Levelplot and Contours

The `scallops` data is non-gridded data.

The functions:

`interp()` *Interpolates* non-gridded data to a grid (see the

help-file for more details on the construction of the grid).

`image()` Creates a levelplot (image, pixelplot). The output from `interp` can be used as input.

`contour()` Draws contour-lines. The output from `interp` can be used as input.

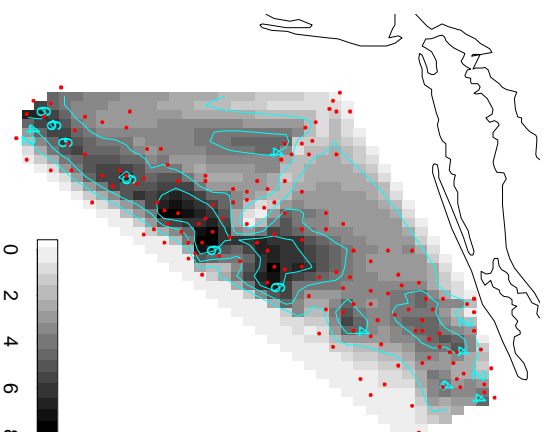
Example

```
> int.data <- interp(long, lat, log.catch)
> map('usa', xlim=c(-73.5, -71), ylim=c(38.2, 41.5))
> image(int.data, add=T)
> image.legend(int.data, x=-72.75, y=38.8, s=c(2, 0.2))
> contour(int.data, add=T, col=4)
> points(long, lat, pch=16, col=8, cex=0.8)
```

The argument `add=T` *adds* both image and contour lines to current plot.

Here are the actual commands used to create the postscript file with the image shown below:

```
> trellis.device(postscript, file='2dim_scallops.ps',
+               horizontal=F, width=6, height=6, color=T)
> par(mar=c(0, 0, 0, 0))
> map('usa', xlim=c(-73.5, -71), ylim=c(38.2, 41.1))
> image(int.data, add=T)
> image.legend(int.data, x=-72.75, y=38.8, s=c(2, 0.2))
> contour(int.data, add=T, col=2)
> points(long, lat, pch=16, col=8, cex=0.4)
> dev.off()
```



Detecting and Removing Trends

Before exploring and estimating the *variogram*, we need to remove any *large-scale* trends to meet the requirement of *intrinsic stationarity*. That is, decompose the *signal* into a large-scale trend and a small-scale random variation.

Fitting Trend Surfaces

Fit a *smooth* trend surface to the scallops data using the *loess* function (*local regression model*):

```
> trend <- loess(log.catch ~ long*lat, span=0.75)
> trend
Call:
loess(formula = log.catch ~ long * lat, span = 0.75)

Number of Observations:      148
Equivalent Number of Parameters:  8.8
Residual Standard Error:      1.781
Multiple R-squared:           0.4
Residuals:
    min     1st Q   median     3rd Q    max
-4.825 -1.242 -0.1849  0.9972  4.725
>
```

where the `span` argument specifies the amount of *smoothing*, the fraction of the data used to fit locally at a given point (value of 0.50, 0.25, and 0.15 were also tried). Other methods include *median polishing* (for gridded data), see the help on the function `twoway`.

Visualizing Trend Surfaces

Predicting

For prediction, we will create a 50×50 rectangular *grid* using the range of the data:

```
> long.g <- seq(min(long),max(long),le=50)
> lat.g <- seq(min(lat),max(lat),le=50)
> sc.grid <- expand.grid(long=long.g,lat=lat.g)
```

The `sc.grid` object is a `data.frame`.

Area Of Interest (AOI)

Our rectangular grid covers the original sampling sites, but also *extrapolates* outside the area covered by the scallop sampling sites. For plotting trend surfaces, and later for kriging prediction, we are *only* interested in area(s) that the data *represent*.

One such AOI is a *convex hull* around the sites: (DON'T DO IN

LAB)

```
> def.hull <- chull(long, lat) ## T if site def hull
> my.hull <- list(x=long[def.hull], y=lat[def.hull])
> in.hull <- points.in.poly(sc.grid$long, sc.grid$lat,
+ my.hull)
```

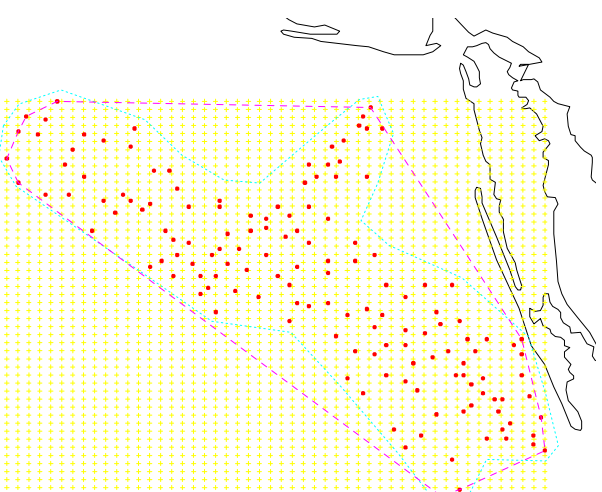
`in.hull` is then a *logical* vector specifying which grid sites are within the convex hull.

I have also defined an area (a polygon using the `locator` function) and the definition is in the list object `my.area` in `/home/gardar/pub/`. Data. To find out which grid points are within `my.area`: (DON'T DO IN LAB)

```
> in.area <- points.in.poly(sc.grid$long, sc.grid$lat,
+ my.area)
```

To see these two AOI: (DON'T DO IN LAB)

```
> map('usa', xlim=c(-73.5, -71), ylim=c(38.2, 41.1))
> points(sc.grid$long, sc.grid$lat, col=7, cex=0.4, pch=3)
> points(long, lat, pch=16, col=8, cex=0.4)
> polygon(my.area, lty=2, d=0, col=2) ## the AOI
> polygon(my.hull, lty=3, d=0, col=3)
```



Predict and Plot

Now, predict using the loess-trend on the `sc.grid` grid:

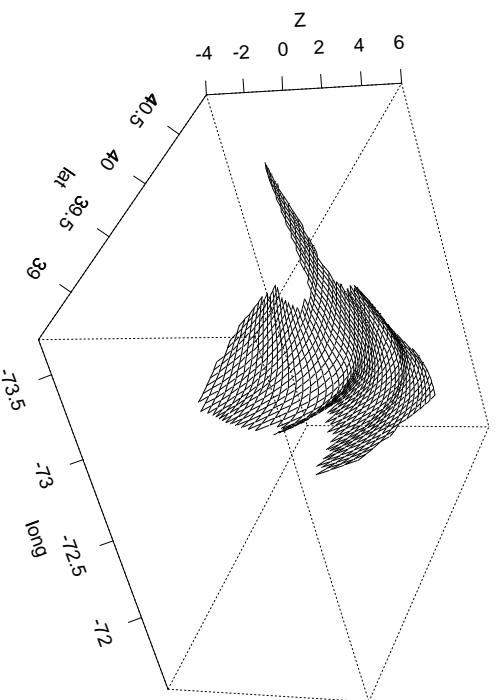
```
> pred <- predict(trend, sc.grid)
my.area:
```

```
> pred[!in.area] <- NA ## NA outside of AOI
```

Plot the trend:

```
> persp(x=long.g,y=lat.g,pred,xlab="long",ylab="lat")
```

In this case, we used the `persp` function, but we could have used the `image` function (as for the interpolated data).



Variogram

Empirical Variogram

The main `S-PLUS` function to compute *empirical* variogram is `variogram`. Its arguments (and defaults) are:

```
variogram(formula, data, subset, na.action,
           lag, nlag=20, tol.lag=lag/2, azimuth=0,
           tol.azimuth=90, bandwidth=1e21, maxdist,
           minpairs=6, method="classical")
```

where (see also the help-file for `variogram`):

formula Formula defining the response and the predictors.

data An optional data frame in which to find the objects mentioned in formula.

lag A numeric value, the width of the lags. If missing, lag is set to `maxdist/nlag`.

nlag An integer, the maximum number of lags to calculate.

tol.lag A numeric value, the distance tolerance.

azimuth A vector of direction angles in degrees, measured clockwise from North. A separate variogram will be estimated for each direction.

tol.azimuth Angle tolerance in degrees. A tol.azimuth of 90 or greater (the default) results in an omnidirectional variogram.

maxdist The maximum distance to include in the returned output. The default is half the maximum distance in the data.

minpairs The minimum number of pairs of points (minimum value for np) that must be used in calculating a variogram value. If np is less than minpairs then that value is dropped from the variogram.

method A character string to select the method for estimating the variogram. The possible values are "classical" for Matheron's (1963) estimate and "robust" for Cressie and Hawkins's (1980) robust estimator. Only the first character of the string needs to be given.

Note 1: The function `variogram` computes the *semivariogram* ($\gamma(\cdot)$), but not the variogram ($2\gamma(\cdot)$).

Note 2: In addition to `variogram`, there are both `covariogram`, for covariance estimation, and `correlogram`, for correlation.

The scallops Data

Omnidirectional Variogram

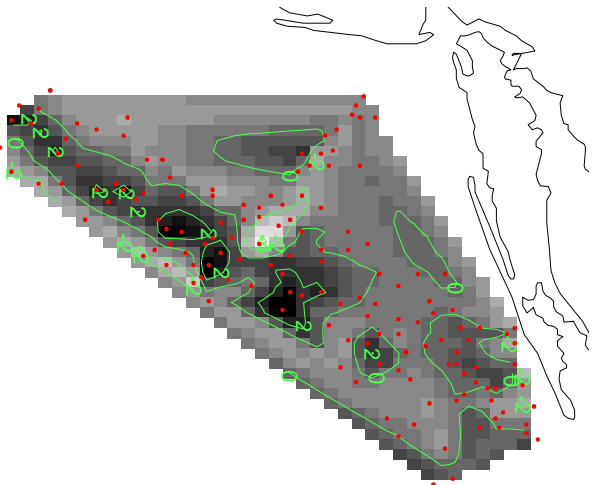
As we saw in *EDA* of the `scallops` data, there is a *trend* that needs to be removed to make the `log(tcatch+1)` process ('more') *intrinsically stationary*:

```
> log.catch.res <- log(tcatch+1) - predict(trend)
```

Let's plot `log.catch.res` to view the *small-scale* random process of interest: (**DON'T DO IN LAB**)

```
> int.res <- interp(long,lat,log.catch.res)
> map('usa',xlim=c(-73.5,-71),ylim=c(38.2,41.1))
> image(int.res,add=T)
> contour(int.res,add=T,col=4)
> points(long,lat,pch=16,col=8,cex=0.8)
```

You will notice little trend, but spatial pattern (areas of low and high residuals).



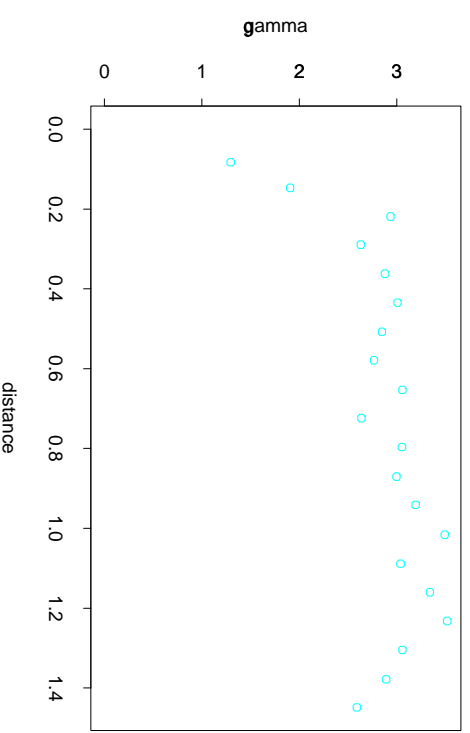
Assume an *isotropic* spatial variation and compute, and plot, the *empirical variogram*:

```
> v1 <- variogram(log.catch.res ~ loc(long,lat),
+                 method='r')
> v1[1:5,] ## show the first 5 lines
      distance      gamma np azimuth
1 0.08295816 1.295506 155      0
2 0.14690005 1.905280 290      0
```

```
3 0.21910418 2.935881 404      0
4 0.28922939 2.630932 458      0
5 0.36211783 2.880160 541      0
> plot(v1,main='After trend-removing') ## plot it
```

What is going on?!

After trend-removing



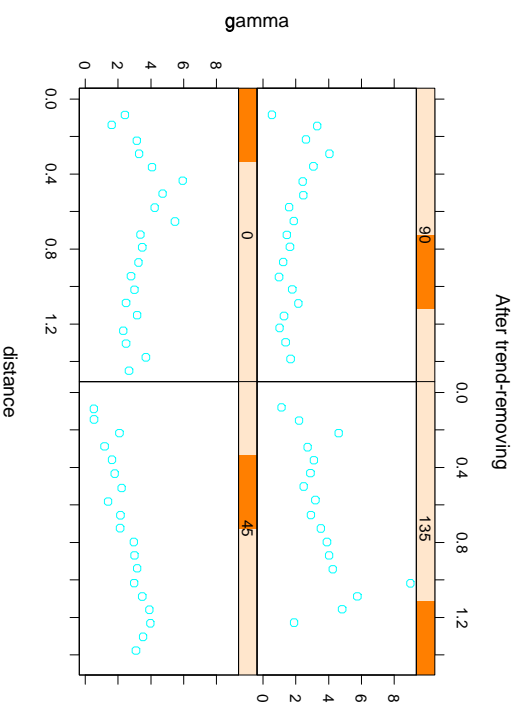
Consider how this looks for $\log(\text{tcatch}+1)$:

```
> v2 <- variogram(log(tcatch+1) ~ loc(long,lat),
+                 method='r')
> plot(v2) ## not shown
```

Directional Variogram

The variogram function can be used to compute empirical *directional* variograms by specifying the *azimuth* argument (vector of angles): (DON'T DO IN LAB)

```
> v3 <- variogram(log.catch.res ~ loc(long,lat),
+                 azimuth=c(0,45,90,135),
+                 tol.azimuth=11.25, method='r')
> plot(v3,main='After trend-removing')
```

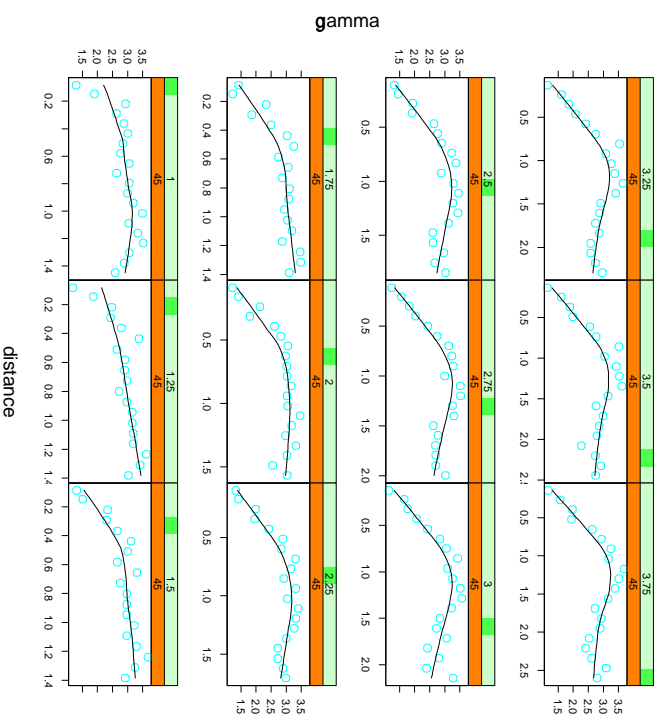


Note the slight difference in the directional variograms.

We need to correct for *anisotropy* — rotate and scale the axes. Checking for anisotropy can be done with the function `anisotropy.plot`. As an example, let's rotate the axis by 45 degrees and try different scaling: (DON'T DO IN LAB)

```
> anisotropy.plot(log.catch.res ~ loc(long,lat),
+                 angle=2*22.5,
+                 ratio=seq(1,by=0.25,le=12),
+                 method='r', layout=c(3,4))
```

A rotation of 45 degrees and a scaling of 2 seams to give satisfying result (see figure on next page).



Correcting for Anisotropy

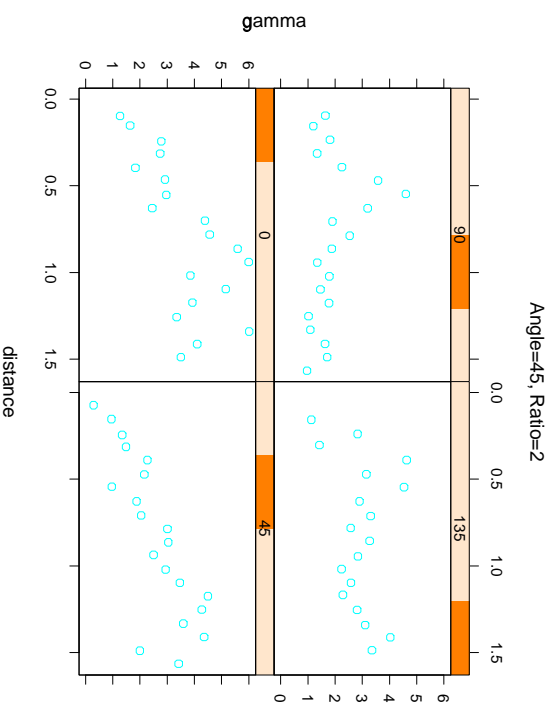
The general format of the `loc` function used in `variogram` is:

```
loc(x, y, angle=0, ratio=1)
```

The `angle` and `ratio` arguments can be used to correct

for anisotropy. The input spatial locations are rotated, scaled, and rotated back to correct for geometric anisotropy defined by angle of anisotropy and the ratio. For example, using an angle of 45 degrees and a ratio of 2, results in the following directional variograms:

```
> v.f <- variogram(log.catch.res~loc(long,lat,a=45,r=2),
+                 azimuth=c(0,45,90,135),
+                 tol.azimuth=45/4, method='r')
> plot(v.f,main='Angle=45, Ratio=2')
```



The empirical variogram that we will use is then:

```
> v.use <- variogram(log.catch.res~Loc(Long, lat, a=45, r=2),
+ method='r')
> plot(v.use) ## not shown in notes
```

Fitting Theoretical Variograms

S+SPATIAL STATS provides functions for common theoretical variograms:

exp.vgram An exponential variogram.

spher.vgram A spherical variogram.

gaus.vgram A gaussian variogram.

linear.vgram A linear variogram.

power.vgram A power variogram.

For example, the **spher.vgram** function is:

```
> spher.vgram
function(distance, range, sill = 1, nugget = 0)
{
  distance <- distance/range
  nugget + sill * ifelse(distance < 1,
    (1.5 * distance - 0.5 * distance^
    3), 1)
}
```

That is, the function has arguments **distance**, **range**, **sill** (by default 1), and **nugget** (by default 0).

The model.variogram Function

It is possible to fit a variogram model to an empirical variogram *interactively* (i.e., by specifying the parameters

interactively through visual matching of the theoretical variograms to the empirical variogram. This is done by using the function `model.variogram` (see the help-file).

Fitting Using Nonlinear Least Squares

Use the *weighted least squares* estimator of Cressie (1985), given by minimizing

$$\sum_{j=1}^K |N(h(j))| \left(\frac{\gamma(h(j))}{\gamma(h(j); \hat{\theta})} - 1 \right)^2$$

The first step in using this approach to *fit* a theoretical variogram is to write a function to compute the *residuals* (the square root of the terms in the sum). For the *spherical* variogram, this function is:

```
spher.wfun <- function(gamma, distance, np,
                       range, sill, nugget)
{
  gamma.hat <- spher.vgram(distance, range, sill, nugget)
  sqrt(np) * (gamma/gamma.hat - 1)
}
```

Then, use the `nls` function to fit a non-linear model by minimizing residual sum of squares:

```
> fit.var <- nls(~spher.wfun(gamma, distance,
```

```
+           np, range, sill, nugget),
+           data=v.use, start=list(range=0.6,
+                                 sill=2, nugget=1))
>
> summary(fit.var)
```

```
Formula: ~ spher.wfun(gamma, distance, np, range, sill,
nugget)
```

Parameters:

Value	Std. Error	t value
range 0.681931	0.0725497	9.39951
sill 2.386900	0.2201930	10.84000
nugget 0.701372	0.2178590	3.21939

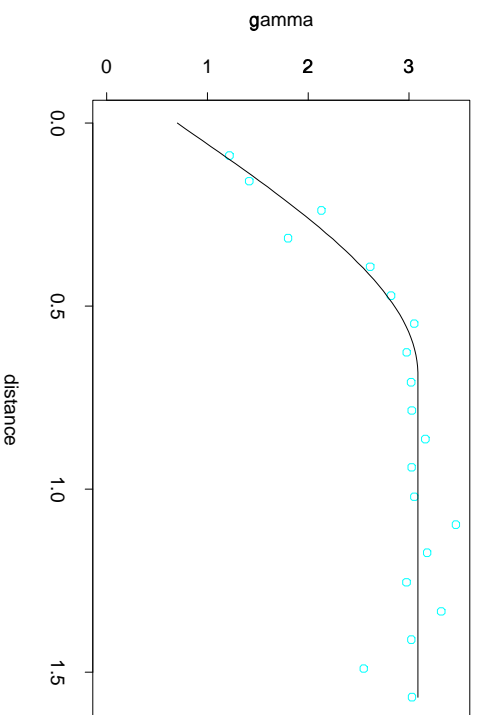
Residual standard error: 1.54461 on
17 degrees of freedom

Correlation of Parameter Estimates:

range	sill
sill -0.545	
nugget 0.676 -0.958	

Plot the fitted variogram:

```
> plot(v.use)
> dist.use <- seq(0, max(v.use$dis), le=100)
> lines(dist.use, spher.vgram(dist.use,
+                             range=0.6819, sill=2.387, nugget=0.7013))
```



Note: It is always an issue as to what should be taken as deterministic *trend* and what is a random process (with spatial dependence). A non-flexible trend leaves behind a lot of structure to be modeled as a random process that could violate the intrinsic stationary assumption of the process. Conversely, a very flexible trend surface doesn't leave very much for the random process.

Kriging

S+SPATIAL STATS uses the two functions `krige` and `predict.krige` to do both *ordinary* and *universal* kriging.

Fitting

Based on the example of the scallops, we fit:

```
> k.fit <- krige(log.catch.res ~ loc(long,lat,a=45,r=2),
+               covfun=spher.cov,
+               range=0.6819,sill=2.387,nugget=0.7013)
> k.fit
Call:
krige(formula = log.catch.res ~ loc(long, lat,
      a = 45, r = 2), covfun = spher.cov, range
      = 0.6819, sill = 2.387, nugget = 0.7013)

Coefficients:
constant
-0.2965647

Number of observations: 148
```

Recall that other covariance functions are available in

S+SPATIALSTATS, such as `exp.cov` and `gauss.cov`.(The user can also construct his *own* variogram and covariance functions.)**Predicting**

The function `predict.krige` predicts, by default, on a 30×30 grid defined by the range of the data. To predict on the `sc.grid` grid, created before, enter:

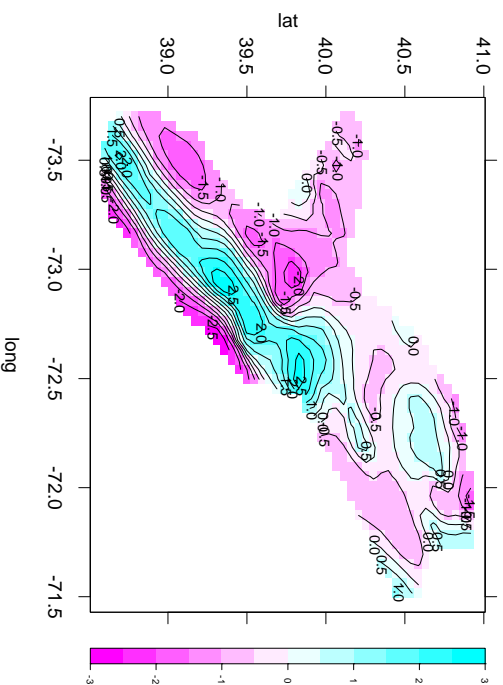
```
> k.pred <- predict(k.fit,sc.grid)
> summary(k.pred)
      Long      Lat
Min.: -73.70  Min.: 38.60
1st Qu.: -73.17 1st Qu.: 39.17
Median: -72.61  Median: 39.76
Mean: -72.61   Mean: 39.76
3rd Qu.: -72.05 3rd Qu.: 40.35
Max.: -71.52   Max.: 40.92

      fit      se.fit
Min.: -2.7890  Min.: 0.5013
1st Qu.: -0.6021 1st Qu.: 1.1990
Median: -0.2966  Median: 1.5540
Mean: -0.2865   Mean: 1.4950
3rd Qu.: -0.1374 3rd Qu.: 1.7920
Max.:  2.9410   Max.:  1.7920
```

This function gives both fitted values (`fit`) and standard errors (see `fit`). The object `k.pred` is a `data.frame`.

It is easy to plot the fitted values (e.g., using a function from the *trellis* plotting library):

```
> k.pred$fit[!in.area] <- NA ## NA outside of AOI
> levelplot(fit ~ long*lat, data=k.pred,
+           contour=T, pretty=T)
```



For the standard errors (adding the sites' locations), enter:
(DON'T DO IN LAB)

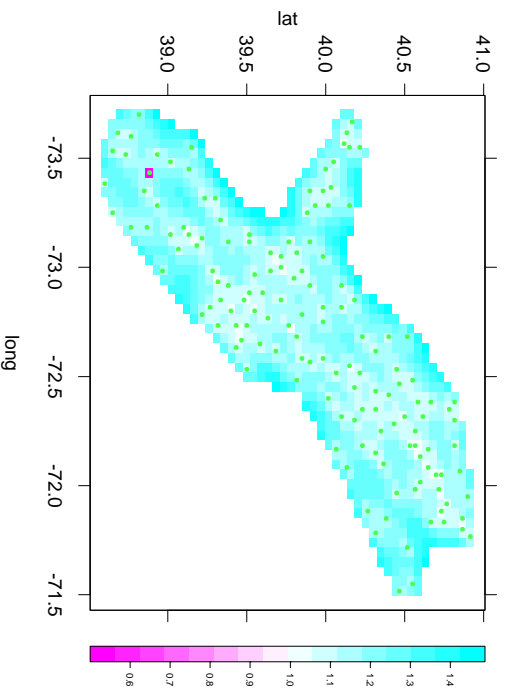
```
> k.pred$se.fit[!in.area] <- NA
```

```

> levelplot(se.fit ~ long*lat, data=k.pred,
+           panel=function(x,Y,...) {
+             panel.levelplot(x,Y,...)
+             panel.xyplot(long,lat,col=4,cex=1,pch=16)
+           })

```

(Note the one grid point with much lower SE than the others.)



To predict $\log(\text{tcatch} + 1)$, the `log.catch` variable, we need to add back the *trend* to the kriging values:

```

> k.pred$pred <- predict(trend,sc.grid) + k.pred$fit
> k.pred$pred[!in.area] <- NA ## NA outside of AOI

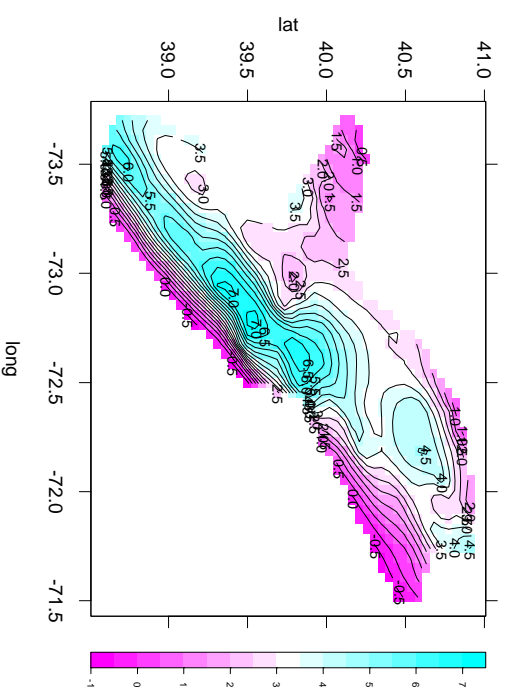
```

Then we plot:

```

> levelplot(pred ~ long*lat, data=k.pred,
+           contour=T, pretty=T)

```



Diagnostics**Cross-Validation**

The following function does cross-validation (CV) for a krige object (as discussed in class):

```
cv.krige <- function(obj, fit.data) {
  # obj is the output from krige()
  n <- nrow(fit.data)
  fit.all <- predict(obj, newdata=fit.data)$fit
  fit <- se <- numeric(n)
  for(i in 1:n) {
    cat("doing observation", i, "\n")
    new.krige <- update(obj, data=fit.data[-i,]) ## drop i
    pred <- predict(new.krige, newdata=fit.data[i,])
    fit[i] <- pred$fit[1]
    se[i] <- pred$se.fit[1]
  }
  res <- (fit.all - fit)/se ## stand. diff.
  data.frame(fit.all = fit.all, fit = fit,
             res = res, se = se)
}

(This function is available in /home/gardar/pub/.Data)
In our case:
> fit.data <- data.frame(log.catch, res, long, lat)
> k.fit.cv <- cv.krige(k.fit, fit.data)
```

Make a histogram and a Q-Q plot of the standardized difference (the column `res` in the output):

```
> histogram(k.fit.cv$res)
> gn <- qnorm(k.fit.cv$res, plot=F)
> plot(gn$x, gn$y, type="n", xlab="Norm", ylab="CV")
> text(gn$x, gn$y, 1:148)
> qqline(k.fit.cv$res)
```

Observation number 68 sticks out:

```
> scallppl[68,]
strata sample      lat      long      tcatch      prerec
68      6270      139 39.71667 -72.85      0      0
recruits
68      0
```

Observation 68 had zero scallops observed.

Where is observation 68: (DON'T DO IN LAB)

```
> plot(long, lat, type='n')
> points(long[68], lat[68], pch=16, col=2, cex=2)
> text(long, lat, tcatch, cex=0.8)
```

It is now not surprising that observation 68 (labeled with blue dot) has a high CV residual since its count is very different from the high counts of its near neighbors.

