Spatial Variogram

Peter Claussen

9/5/2017

Libraries

gstat : Spatial and Spatio-Temporal Geostatistical Modelling, Prediction and Simulation

```r
library(gstat)
```

```
## Warning: package 'gstat' was built under R version 3.3.2
```

geoR : Analysis of Geostatistical Data

```r
library(geoR)
```

```
## --------------------------------------------------------------
##  Analysis of Geostatistical Data
##  For an Introduction to geoR go to http://www.leg.ufpr.br/geoR
##  geoR version 1.7-5.2 (built on 2016-05-02) is now loaded
## --------------------------------------------------------------
```

ggplot2 : A system for 'declaratively' creating graphics, based on "The Grammar of Graphics"

```r
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.3.2
```

```r
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#0072B2", "#D55E00", "#F0E442","#CC79A7","#
```

Data

```r
load(file="sample.dat.Rda")
```

To extend our concept of autocorrelation to two dimensions, we introduce a new notation,

Variable	Univariate Equivalent	Comments
$\mathbf{u}$	i	set of spatial coordinates
$z(\mathbf{u})$	y_i	variable of interest, measured at a set of coordinates
$\mathbf{h}$	k	lag vector representing a spatial distance
$z(\mathbf{u}+\mathbf{h})$	y_{i+k}	lagged variable

$N(\mathbf{h})$ | n | number of pairs z_i, z_j seperated by lag $\mathbf{h}$

We also update the formula for correlation, covariance and autoregression,

Covariance

$$C(\mathbf{h}) = \frac{1}{N(\mathbf{h})} \Sigma_{i=1}^{N(\mathbf{h})} z(\mathbf{u}) z(\mathbf{u}+\mathbf{h}) - m_0 m_i$$

Correlation

$$\rho(\mathbf{h}) = \frac{C(\mathbf{h})}{\sqrt{\sigma_0^2 \sigma_{\mathbf{h}}^2}}$$

Semivariance

$$\gamma(\mathbf{h}) = \frac{1}{N(\mathbf{h})} \Sigma_{i=1}^{N(\mathbf{h})} [z(\mathbf{u}) - z(\mathbf{u}+\mathbf{h})]^2$$

with

$$m_0 = \frac{1}{N(\mathbf{h})} \Sigma_{i=1}^{N(\mathbf{h})} z(\mathbf{u})$$

$$m_i = \frac{1}{N(\mathbf{h})} \Sigma_{i=1}^{N(\mathbf{h})} z(\mathbf{u}+\mathbf{h})$$

$$\sigma_0^2 = \frac{1}{N(\mathbf{h})} \Sigma_{i=1}^{N(\mathbf{h})} [z(\mathbf{u}) - m_0]^2$$

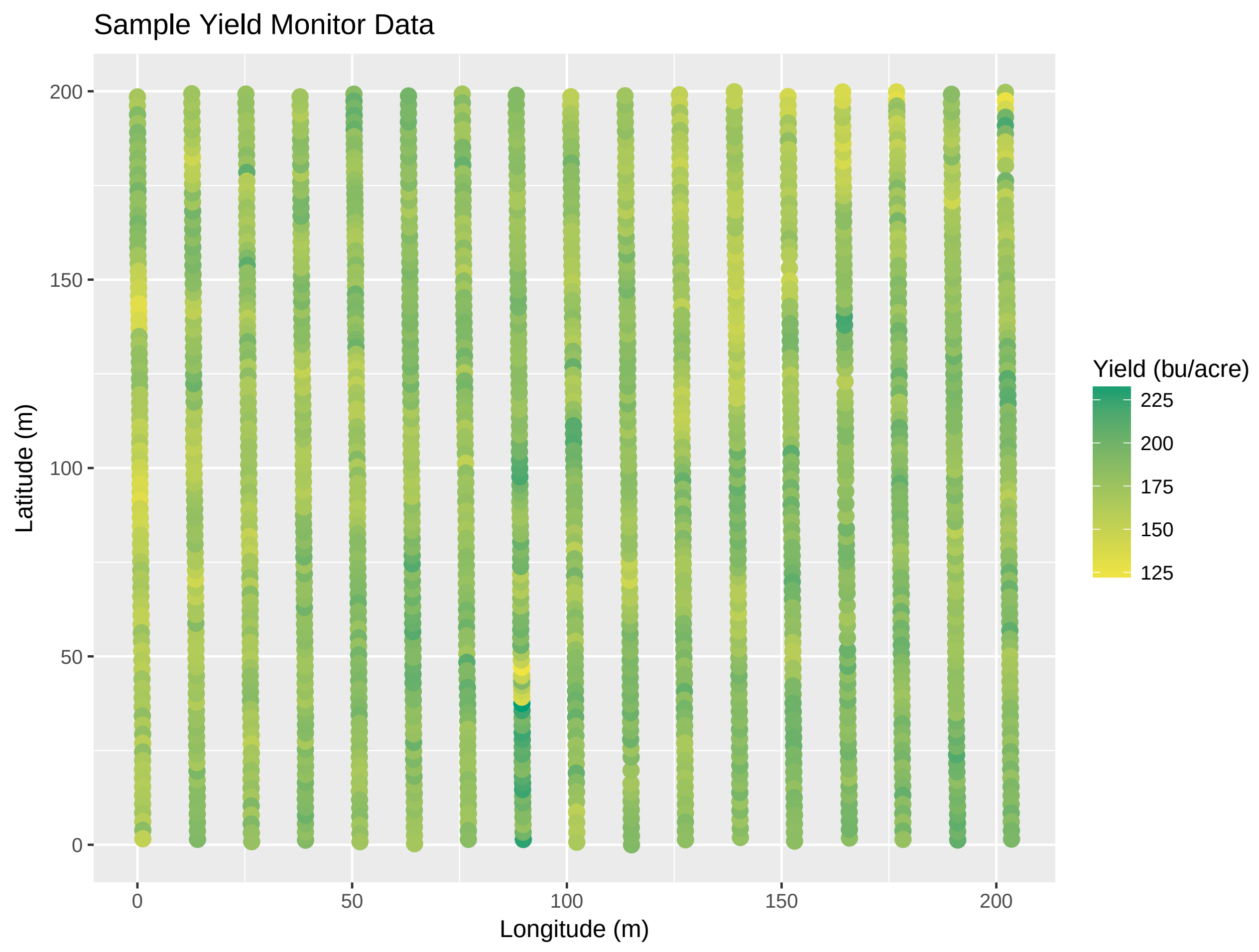
$$\sigma_i^2 = \frac{1}{N(\mathbf{h})} \Sigma_{i=1}^{N(\mathbf{h})} [z(\mathbf{u}+\mathbf{h})) - m_i]^2$$

A couple points to note. We think of $z(\mathbf{u}+\mathbf{h})$ as a ring of points around a specific point $z(\mathbf{u}$, but in most cases there will never be points at an exact distance h, so we usually use lag classes of $h \pm \delta$. Also note that we use $N(\mathbf{h})$ to compute σ^2, instead of $N(\mathbf{h}) - 1$.

I won't go into the implementation of semivariogram computations; it's more complicated and not worth our time at this point. Instead, we'll look at some sample data and use different R packages to produce variograms.

Semivariograms plots from Yield Monitor Data

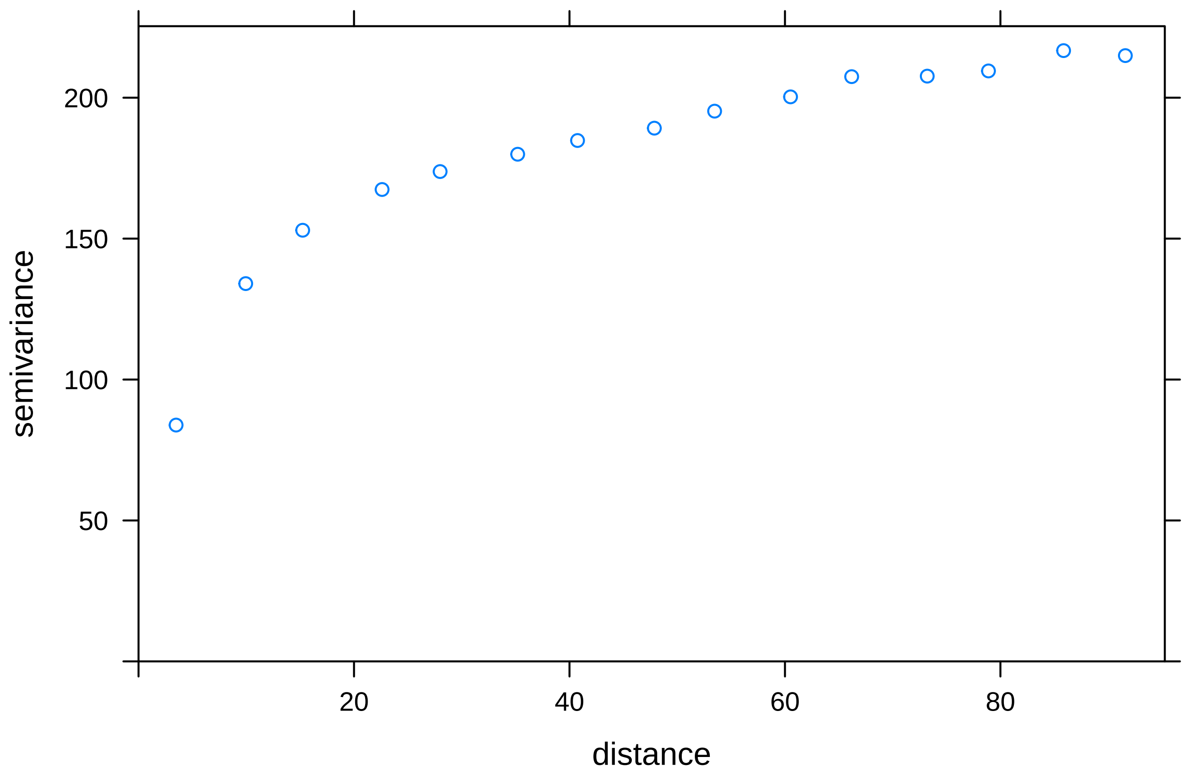
```r
ggplot(sample.dat, aes(LonM, LatM)) +
geom_point(aes(colour = Yield),size=3) +
scale_colour_gradient(low=cbPalette[7], high=cbPalette[4]) +
labs(colour = "Yield (bu/acre)", x="Longitude (m)", y="Latitude (m)", title = "Sample Yield Monitor Dat
```



gstat

The first library we consider is `gstat`. This has a simple interface to define spatial coordinates from the data.

```r
Yield.var <- variogram(Yield~1,
                        locations=~LonM+LatM,
                        data=sample.dat)
plot(Yield.var)
```



Once we've computed semivariance over lag distance, we want to fit a model and extract parameters. Most variogram models include three parameters - nugget, sill and range.

Components of a Variogram

nugget

The nugget represents the degree of correlation between two points, arbitrarily close together. If we could measure spatial values perfectly, the nugget would be 0 (two points in the same space being perfectly correlated). However, there will almost always be some inherent measurement error, and the nugget is an indication of that error. It also indicates how fine or coarse we can make our measurments in space.

sill

The sill is the maximum possible variance at points far apart, and represent the degree of variance when points are completely uncorrelated. We must be careful when measuring the sill; in part because as we measure larger lag classes, we have fewer points for computing variance.

range

The range is the distance at which points are effectively uncorrelated. This can be thought of the neighborhood for each point.

Variogram Models

There are several formula that can be used to estimate nugget, sill and range from a variogram. The most common are

Nugget

$$g(h) = \begin{cases} 0 & h = 0 \\ c & otherwise \end{cases}$$

### Spherical

$$g(h) = \begin{cases} c \times \left[1.5\left(\frac{h}{a}\right) - 0.5\left(\frac{h}{a}\right)^3\right] & h \leqslant a \\ c & otherwise \end{cases}$$

### Exponential

$$g(h) = c \times \left[1 - \exp\left(\frac{-3h}{a}\right)\right]$$

### Gaussian

$$g(h) = c \times \left[1 - \exp\left(\frac{-3h^2}{a^2}\right)\right]$$

### Power

$$g(h) = c \times h^\omega, 0 < \omega < 2$$

For semivariance calculated using `variogram`, we use the `fit.variogram` function. I don't know if there's any good method to automatically select starting parameters, so we need to specify parameters (for sill, range and nugget). I get these by inspection, and it frequently takes more than one try to get convergence, using different parameters each time. Parameters for `vgm` are given by:

```
vgm(psill = NA, model, range = NA, nugget, add.to, anis, kappa = 0.5, ..., covtable,
    Err = 0)
```

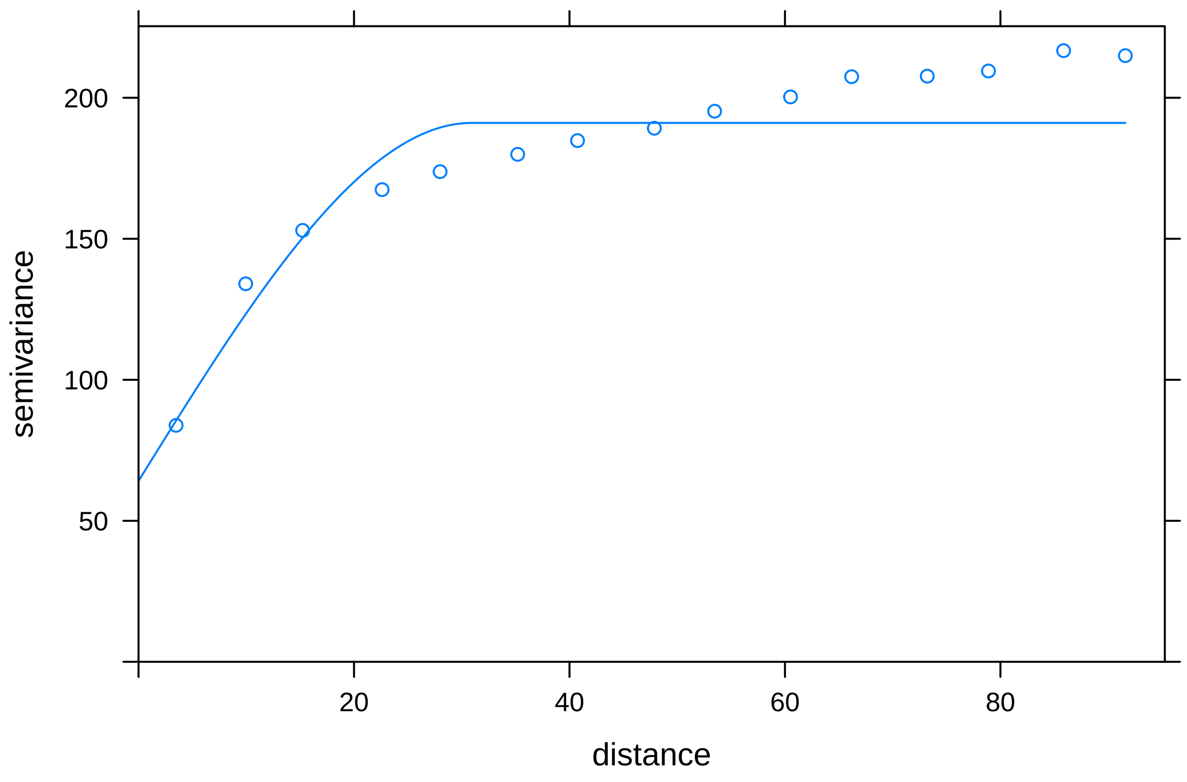
By inspection of the plot, we might guess a a nugget of 50, a sill of 200, and a range of 40m, so

```
print(Yield.vgm <- fit.variogram(Yield.var, vgm(200,"Sph",40,50)))
```

```
##   model     psill    range
## 1   Nug  64.12461  0.00000
## 2   Sph 126.97065 30.91816
```

We visualize the model by

```
plot(Yield.var, model=Yield.vgm)
```

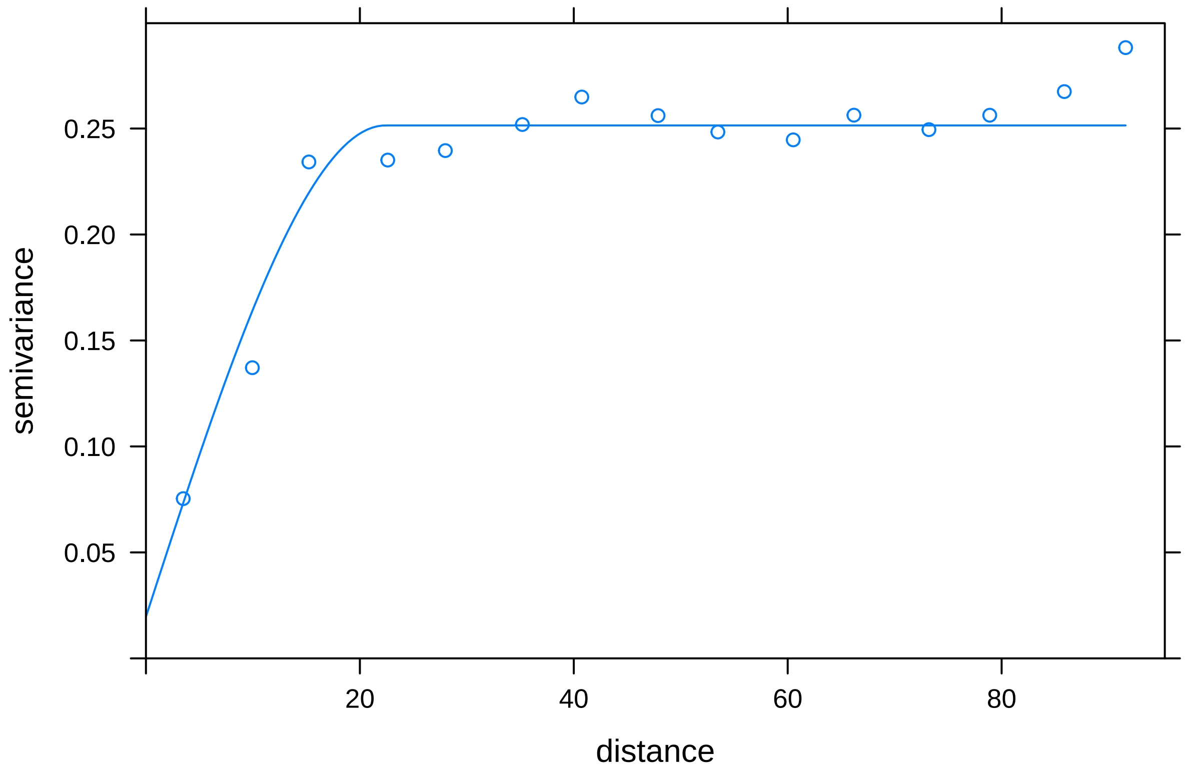


Similarly, for the other variables

```
Distance.var <- variogram(Distance~1,
                        locations=~LonM+LatM,
                        data=sample.dat)
print(Distance.vgm <- fit.variogram(Distance.var, vgm(0.06,"Sph",20,.02)))
```

```
##   model      psill    range
## 1   Nug 0.01963587  0.00000
## 2   Sph 0.23181980 22.39088
```

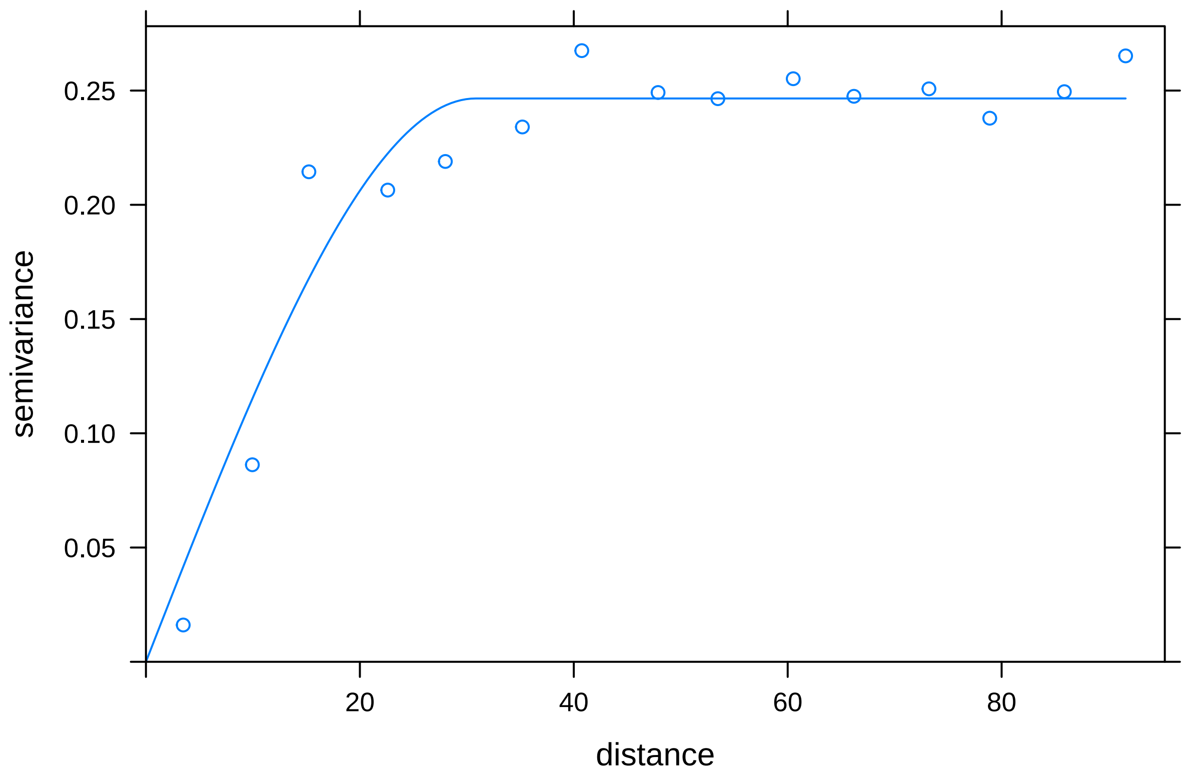
```
plot(Distance.var, model=Distance.vgm)
```



```
Moisture.var <- variogram(Moisture~1,
                          locations=~LonM+LatM,
                          data=sample.dat)
print(Moisture.vgm <- fit.variogram(Moisture.var, vgm(0.25,"Sph",20,0.1)))
```

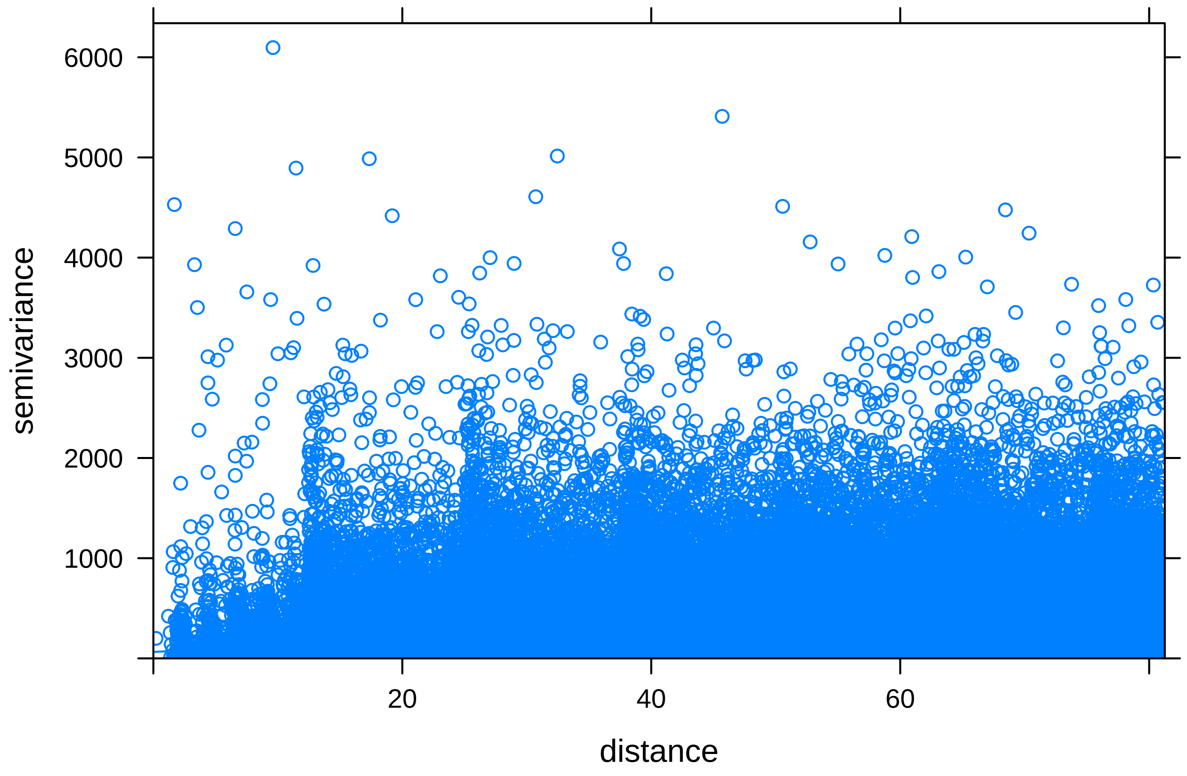
```
##   model     psill    range
## 1   Nug 0.0000000  0.00000
## 2   Sph 0.2465438 30.84941
```

```
plot(Moisture.var, model=Moisture.vgm)
```



It's worth noting that a variogram can be influence by how points are collected into lag groups. To examine a point-by-point correlation, we can produce a variogram cloud

```
Yield.cloud.var <- variogram(Yield~1,
                        locations=~LonM+LatM,
                        data=sample.dat[1:1000,],
                       cloud=T)
plot(Yield.cloud.var,model=Yield.vgm)
```



geoR

geoR requires geodata classes. Otherwise, the process is similar. However, specifying a geodata object can be cumbersome, and fitting variograms is less intuitive.

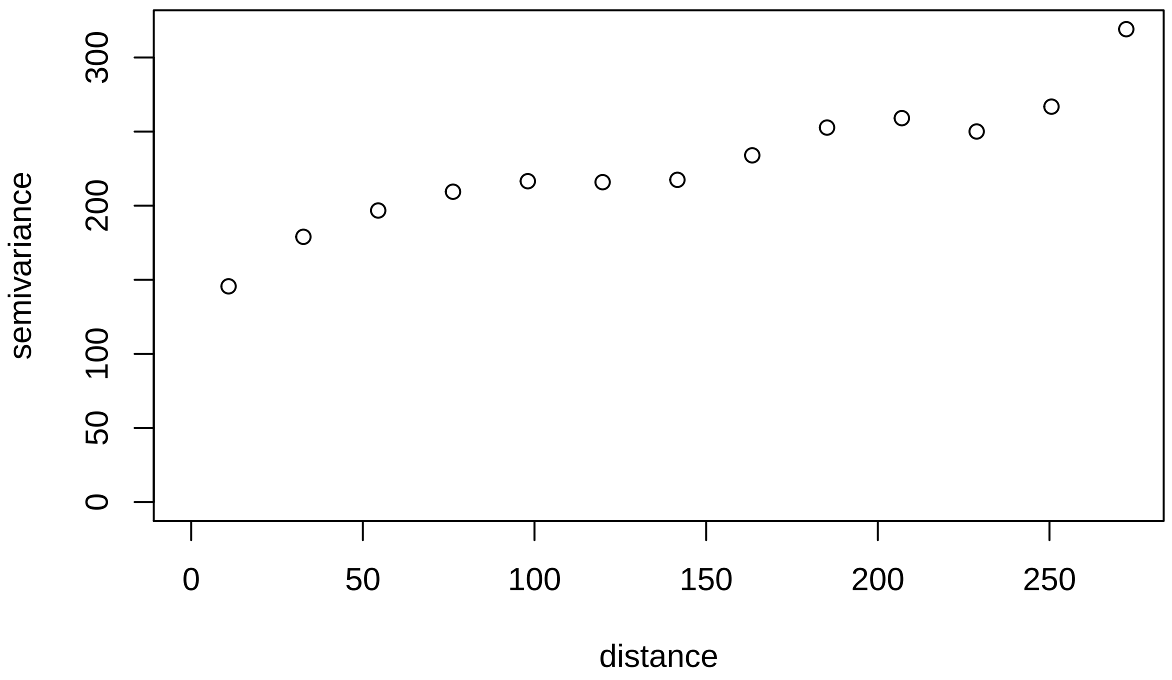
```
metric.col <- c(which(names(sample.dat)=="LatM"), which(names(sample.dat)=="LonM"))
coords.col <- c(which(names(sample.dat)=="Latitude"), which(names(sample.dat)=="Longitude"))
data.col = c(which(names(sample.dat)=="Yield"),which(names(sample.dat)=="Distance"),which(names(sample.
sample.gdat <- as.geodata(sample.dat, coords.col = metric.col, data.col = data.col)
str(sample.gdat)
```

```
## List of 2
##  $ LatM                              , LonM                              : num [1:1497, 1:2] 200 197
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:1497] "1274" "1275" "1276" "1277" ...
##   .. ..$ : chr [1:2] "LatM" "LonM"
##  $ Yield                             , Distance                          , Moisture
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:1497] "1274" "1275" "1276" "1277" ...
##   .. ..$ : chr [1:3] "Yield" "Distance" "Moisture"
##  - attr(*, "class")= chr "geodata"
```

```
Yield.gvar <- variog(sample.gdat,data=sample.gdat$data[,1])
```

```
## variog: computing omnidirectional variogram
```

```
plot(Yield.gvar)
```



variofit seems to work better if we give it a range of initial values, so we create a grid of values:

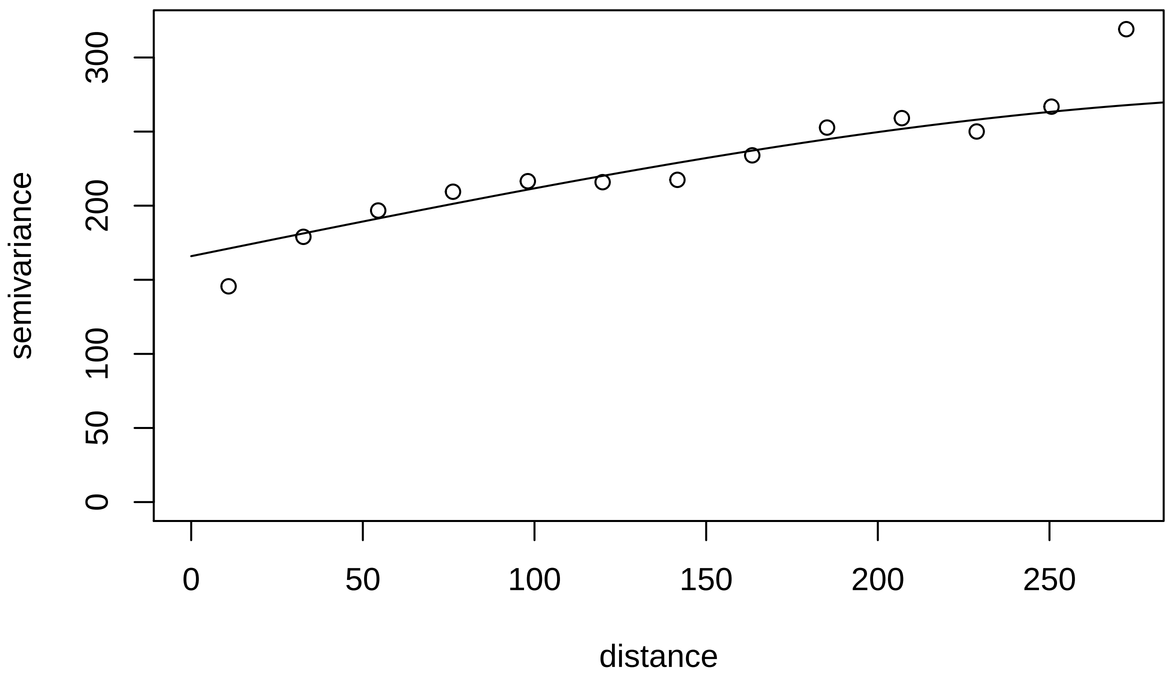
```
init.cov <- expand.grid(seq(10,200,l=10), seq(0,40,l=5))
```

```
print(Yield.fit <- variofit(Yield.gvar, cov.model="sph",
                        ini.cov.pars=init.cov,
                        fix.nugget=FALSE, nugget=50))
```

```
## variofit: covariance model used is spherical
## variofit: weights used: npairs
## variofit: minimisation function used: optim
## variofit: searching for best initial value ... selected values:
##                sigmasq  phi   tausq kappa
## initial.value "157.78" "40"  "50"  "0.5"
## status        "est"    "est" "est" "fix"
## loss value: 499605387.978827
## variofit: model parameters estimated by WLS (weighted least squares):
## covariance model is: spherical
## parameter estimates:
##    tausq  sigmasq      phi
## 165.9453 108.9368 347.2211
## Practical Range with cor=0.05 for asymptotic range: 347.2211
##
## variofit: minimised weighted sum of squares = 72336853
```

It also takes an extra step to add the fitted variogram.

```
plot(Yield.gvar)
lines(Yield.fit)
```

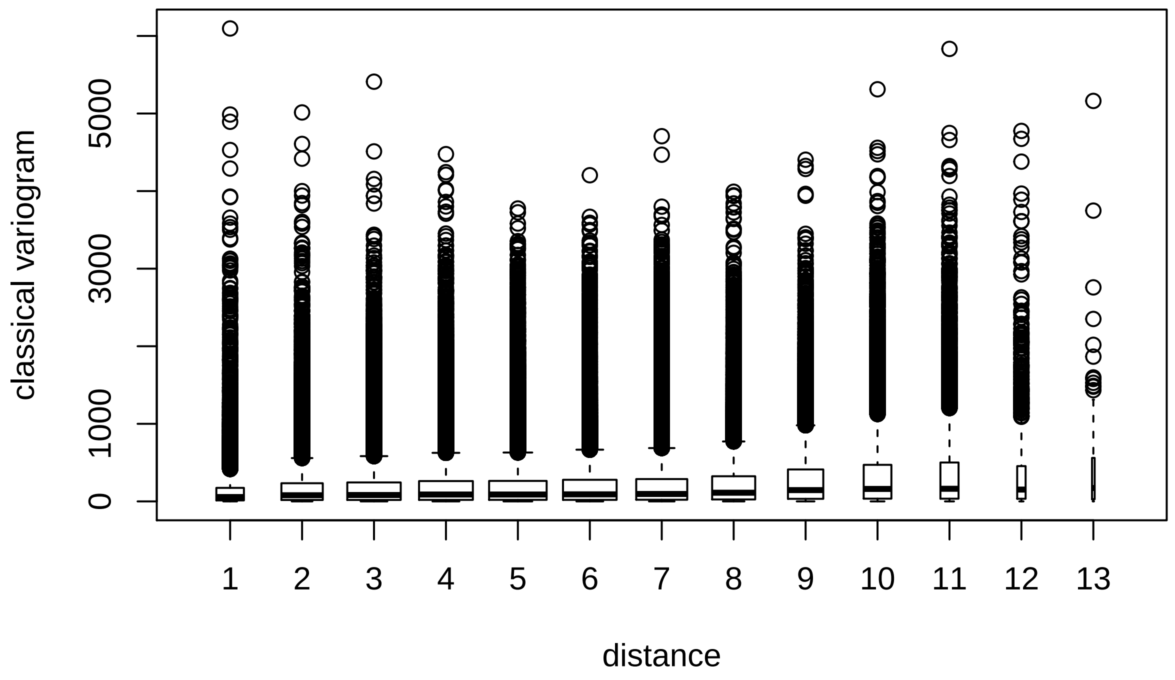


This function does produce a barplot for the variogram cloud. It does become noisy with the full data set, so we will illustrate with a subset.

```
subsample.gdat <- as.geodata(sample.dat[1:1000,], coords.col = metric.col, data.col = data.col)
Yield.cloud.gvar <- variog(subsample.gdat,data=subsample.gdat$data[,1],bin.cloud=TRUE)
```

```
## variog: computing omnidirectional variogram
```

```
plot(Yield.cloud.gvar,bin.cloud=TRUE)
```

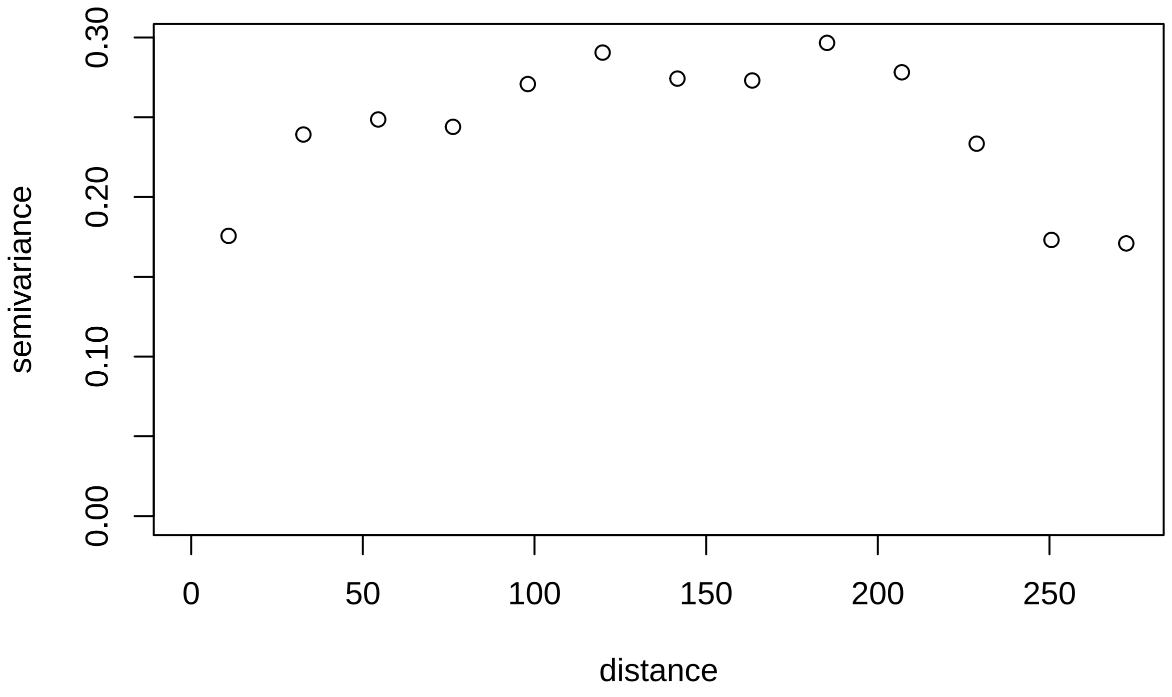


Sometimes it's easier to work with a one-shot data frame:

```r
Moisture.idx = which(names(sample.dat)=="Moisture")
Moisture.gdat <- as.geodata(sample.dat, coords.col = metric.col, data.col = Moisture.idx)
Moisture.gvar <- variog(Moisture.gdat)
```

```
## variog: computing omnidirectional variogram
```

```r
plot(Moisture.gvar)
```

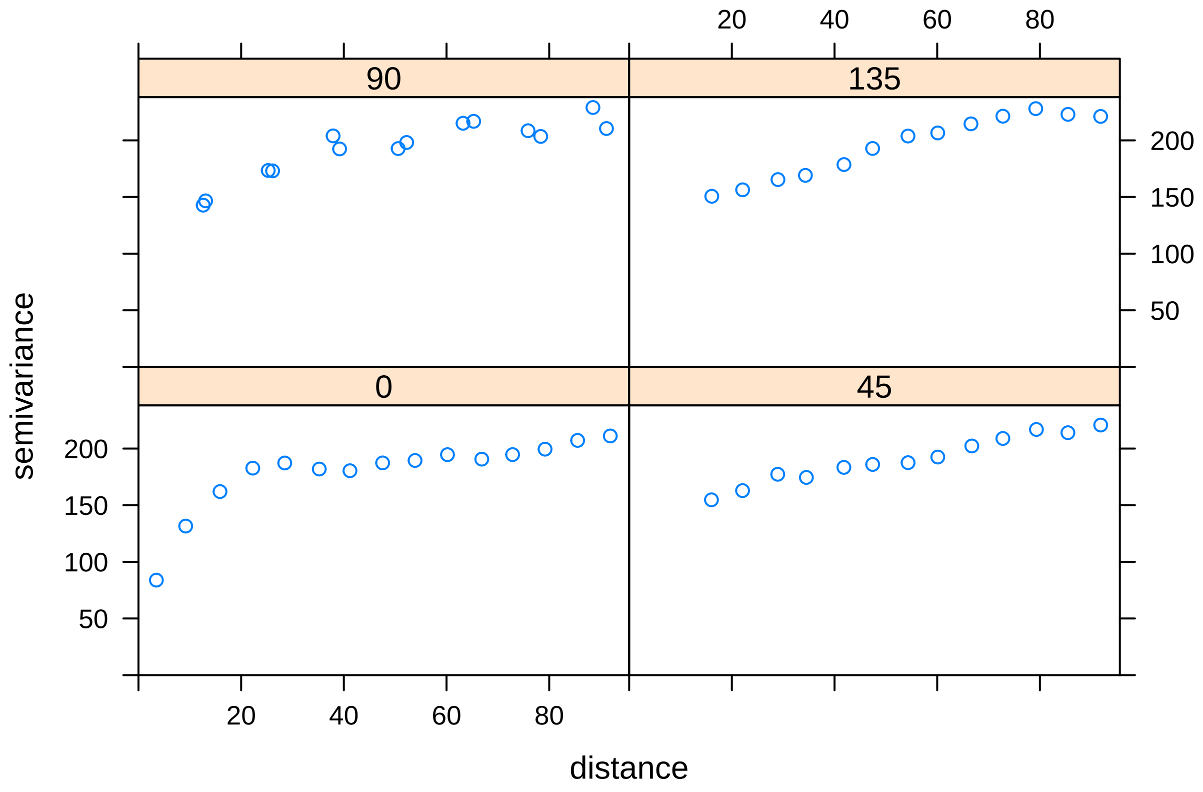


Isotropy

Isotropy implies that spatial correlation depends only on distance and not on direction. One way to determine if a set of spatial data a anisotropic is to compute variograms at different angles.

We can do this manually from `gstat` by

```r
Yield.ani.vgm <- variogram(Yield~1,
                           locations=~LonM+LatM,
                           data=sample.dat, alpha=c(0,45,90,135))
plot(Yield.ani.vgm)
```

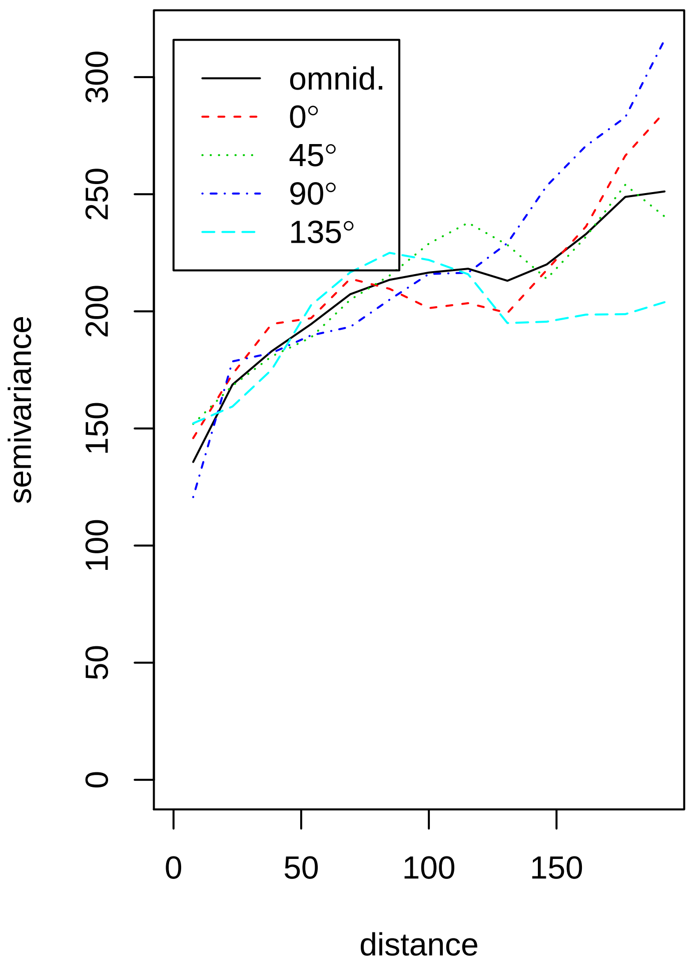


or we can use the `vario4` function from `geoR`:

```
Yield.idx = which(names(sample.dat)=="Yield")
Yield.gdat <- as.geodata(sample.dat, coords.col = metric.col, data.col = Yield.idx)
Yield.var4 <- variog4(Yield.gdat,max.dist=200)
```

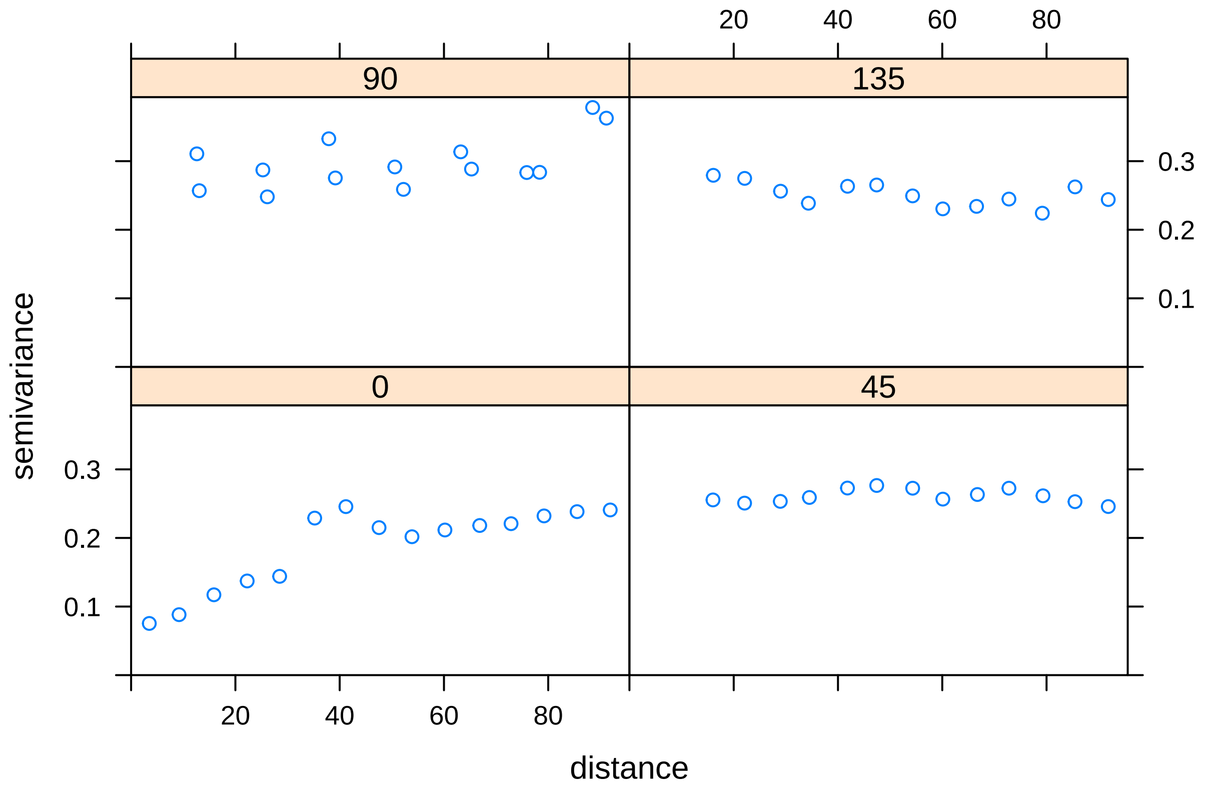
```
## variog: computing variogram for direction = 0 degrees (0 radians)
##         tolerance angle = 22.5 degrees (0.393 radians)
## variog: computing variogram for direction = 45 degrees (0.785 radians)
##         tolerance angle = 22.5 degrees (0.393 radians)
## variog: computing variogram for direction = 90 degrees (1.571 radians)
##         tolerance angle = 22.5 degrees (0.393 radians)
## variog: computing variogram for direction = 135 degrees (2.356 radians)
##         tolerance angle = 22.5 degrees (0.393 radians)
## variog: computing omnidirectional variogram
```

```
plot(Yield.var4, omnidirectional=TRUE)
```



We can see if this holds true for other variables as well.

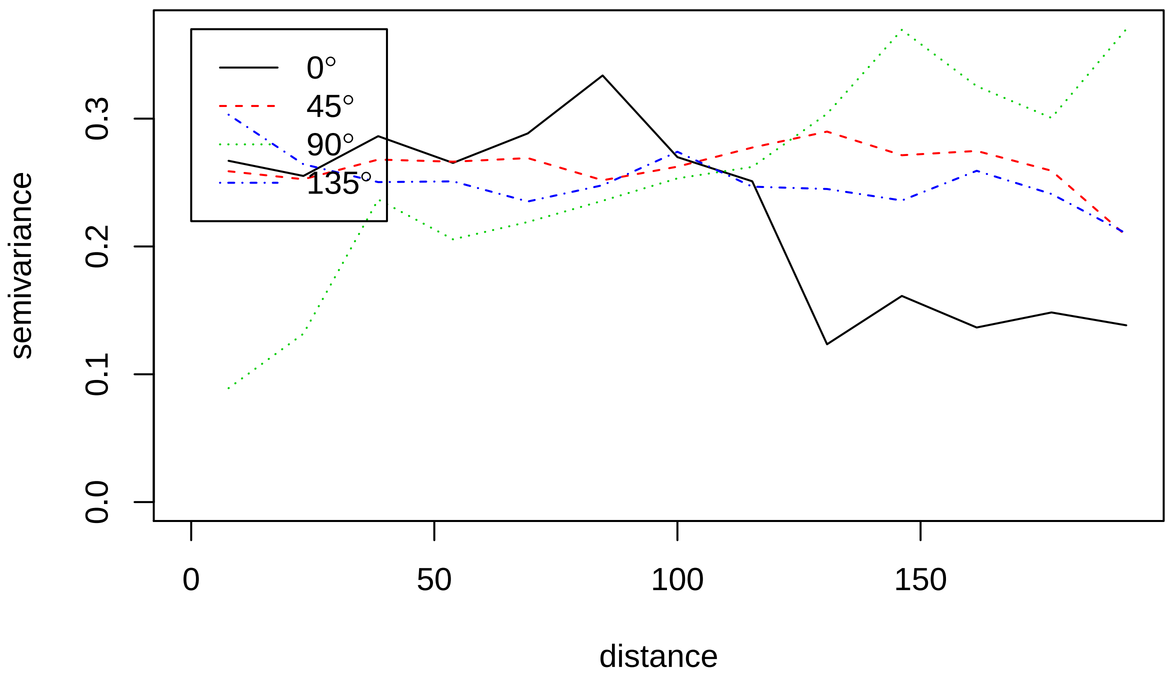
```
Distance.ani.vgm <- variogram(Distance~1,
                              locations=~LonM+LatM,
                              data=sample.dat, alpha=c(0,45,90,135))
plot(Distance.ani.vgm)
```



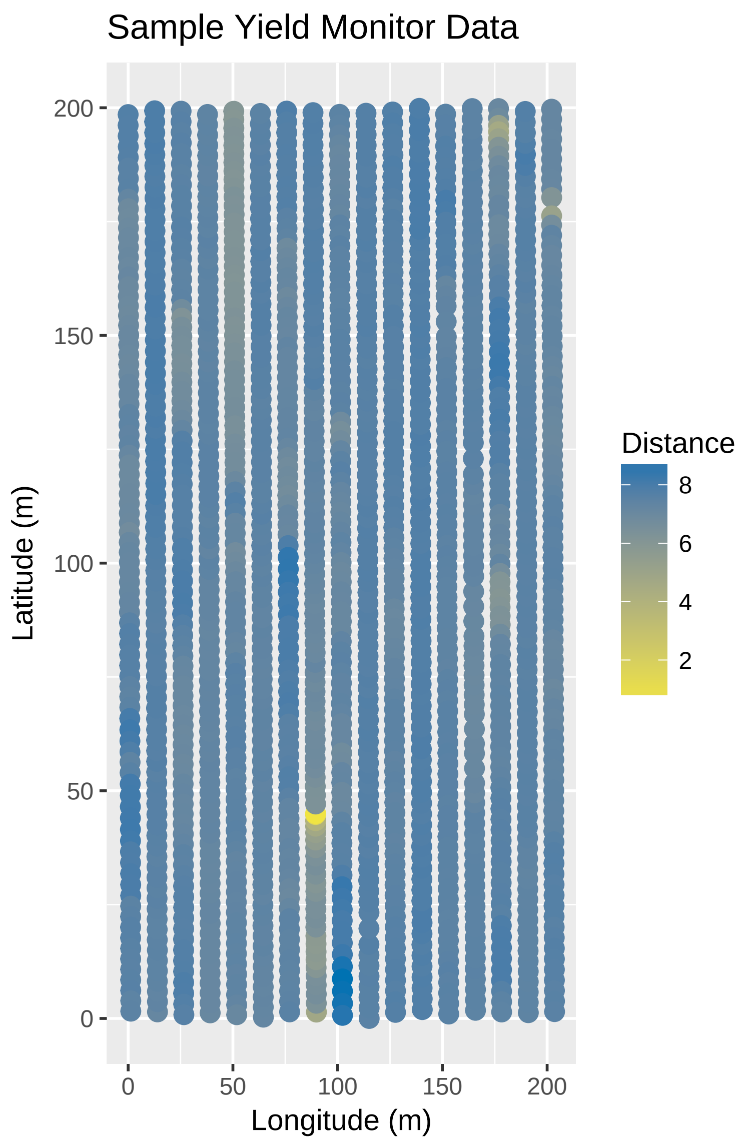
```r
Distance.idx = which(names(sample.dat)=="Distance")
Distance.gdat <- as.geodata(sample.dat, coords.col = metric.col, data.col = Distance.idx)
Distance.var4 <- variog4(Distance.gdat,max.dist=200)
```

```
## variog: computing variogram for direction = 0 degrees (0 radians)
##         tolerance angle = 22.5 degrees (0.393 radians)
## variog: computing variogram for direction = 45 degrees (0.785 radians)
##         tolerance angle = 22.5 degrees (0.393 radians)
## variog: computing variogram for direction = 90 degrees (1.571 radians)
##         tolerance angle = 22.5 degrees (0.393 radians)
## variog: computing variogram for direction = 135 degrees (2.356 radians)
##         tolerance angle = 22.5 degrees (0.393 radians)
## variog: computing omnidirectional variogram
```

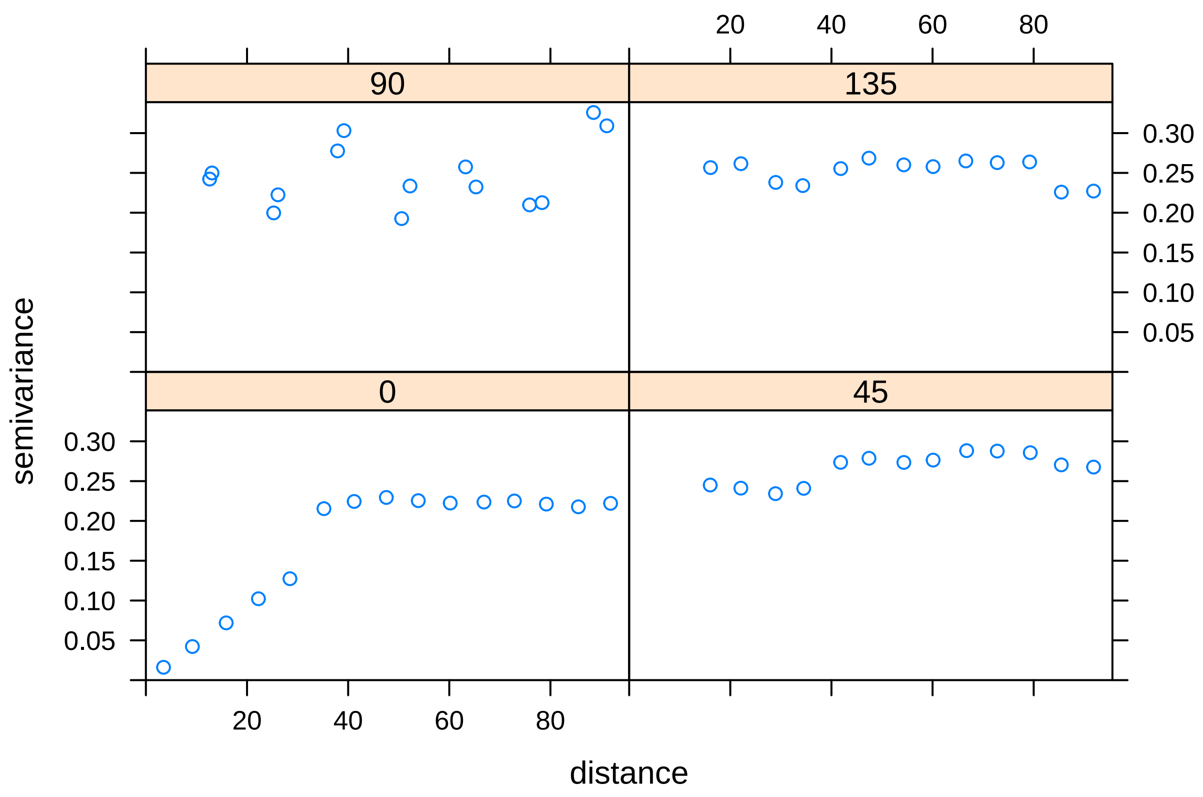
```r
plot(Distance.var4)
```



```r
ggplot(sample.dat, aes(LonM, LatM)) +
geom_point(aes(colour = Distance),size=3) +
scale_colour_gradient(low=cbPalette[7], high=cbPalette[5]) +
labs(colour = "Distance", x="Longitude (m)", y="Latitude (m)", title = "Sample Yield Monitor Data")
```



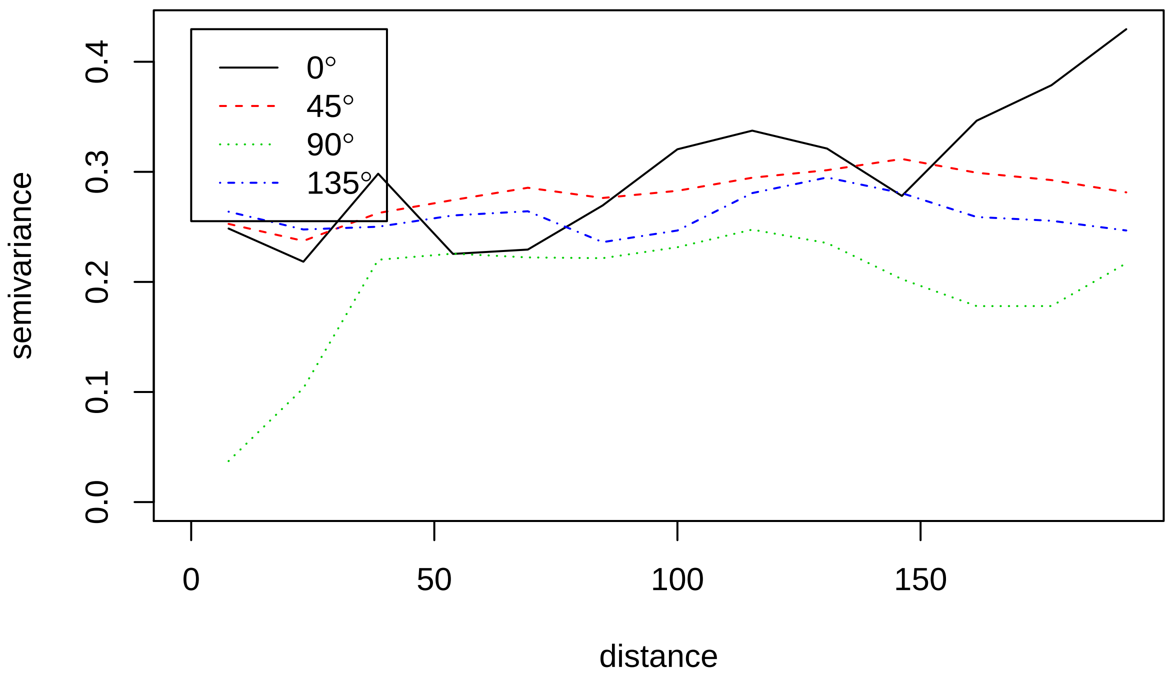
```
Moisture.ani.vgm <- variogram(Moisture~1,
                              locations=~LonM+LatM,
                              data=sample.dat, alpha=c(0,45,90,135))
plot(Moisture.ani.vgm)
```



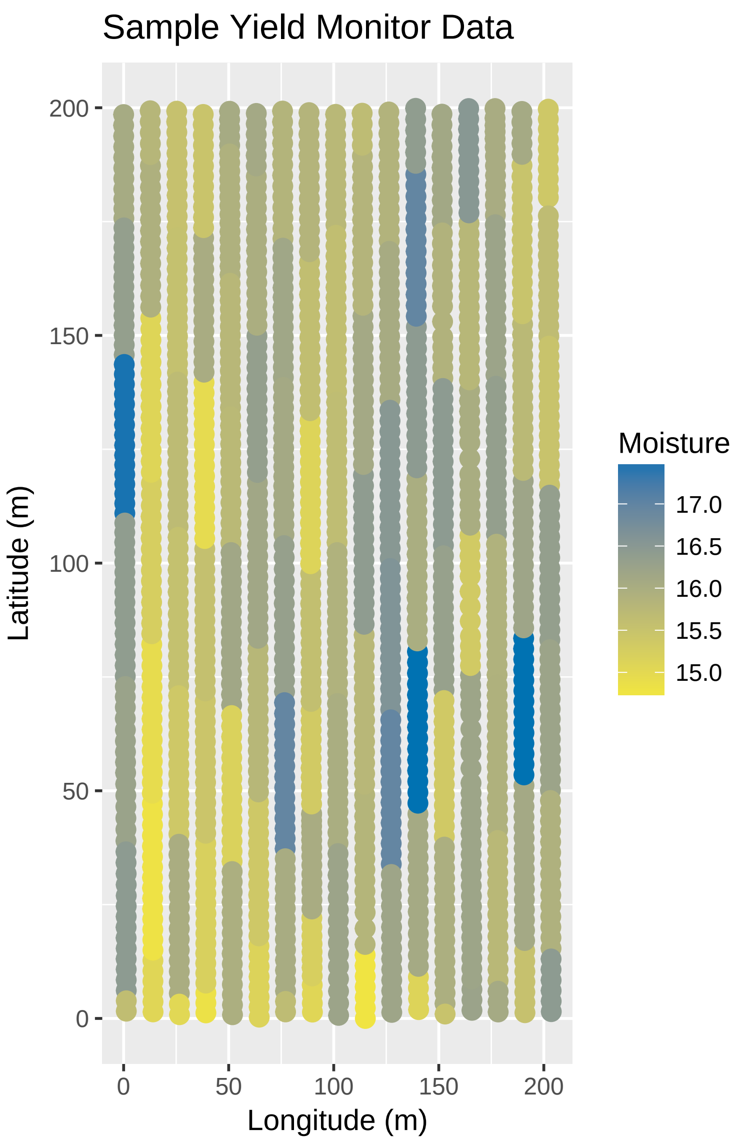
```
Moisture.var4 <- variog4(Moisture.gdat,max.dist=200)
```

```
## variog: computing variogram for direction = 0 degrees (0 radians)
##         tolerance angle = 22.5 degrees (0.393 radians)
## variog: computing variogram for direction = 45 degrees (0.785 radians)
##         tolerance angle = 22.5 degrees (0.393 radians)
## variog: computing variogram for direction = 90 degrees (1.571 radians)
##         tolerance angle = 22.5 degrees (0.393 radians)
## variog: computing variogram for direction = 135 degrees (2.356 radians)
##         tolerance angle = 22.5 degrees (0.393 radians)
## variog: computing omnidirectional variogram
```

```
plot(Moisture.var4)
```



```
ggplot(sample.dat, aes(LonM, LatM)) +
geom_point(aes(colour = Moisture),size=3) +
scale_colour_gradient(low=cbPalette[7], high=cbPalette[5]) +
labs(colour = "Moisture", x="Longitude (m)", y="Latitude (m)", title = "Sample Yield Monitor Data")
```


Sample Yield Monitor Data
Latitude (m)
200
150
100
50
0
0
50
100
150
200
Longitude (m)
Moisture
17.0
16.5
16.0
15.5
15.0