University of California, Los Angeles
Department of Statistics

Statistics C173/C273 **Instructor: Nicolas Christou**

Computing the variogram using the geoR package in R

Spatial statistics computations can be done in R using the package geoR. Once you have installed the package you can load it in R by typing at the command line

```
> library(geoR)
```

To use the geoR package you need to convert your data to a **geodata** object. First you read your data as a data frame:

```
> a <- read.table("filename", header=T)
```

Now *a* is a data frame that contains your data in the form *x, y, var1, var2, etc.* However R does not know that these are spatial data. To convert the data frame into a geodata object you type

```
> b <- as.geodata(a)
```

This command will convert the first three columns *x, y, var1* to a geodata object b. If you want to create a geodata object with *x, y, var2* then you do the following:

```
b <- as.geodata(a[,c(1,2,4)])
```

It will use the first two columns of the data frame a (the *x, y* coordinates), plus column 4 which is *var2*.

Now *b* is what you will use to do geostatistical analysis. R recognizes that the first two columns are the coordinates, and the rest of the columns are your data. You can see what you have by typing:

```
> b$coords
```

and

```
> b$data
```

You can get the summary of the geodata as follows:

```
> summary(b)
Number of data points: 47

Coordinates summary
      x   y
min 100 100
max 900 600

Distance summary
     min      max
100.0000 943.3981

Data summary
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  28.00   35.00   37.00   36.34   38.50   44.00
```

For this first tutorial we will use the following data:

44		40	42	40	39	37	36	
42		43	42	39	39	41	40	38
37	37	37	35	38	37	37	33	34
35	38		35	37	36	36	35	
36	35	36	35	34	33	32	29	28
38	37	35		30		29	30	32

The distance between two points is 100 m (north-south or east-west).
The commands below will read the file, convert the data into geodata, and list the variables names:

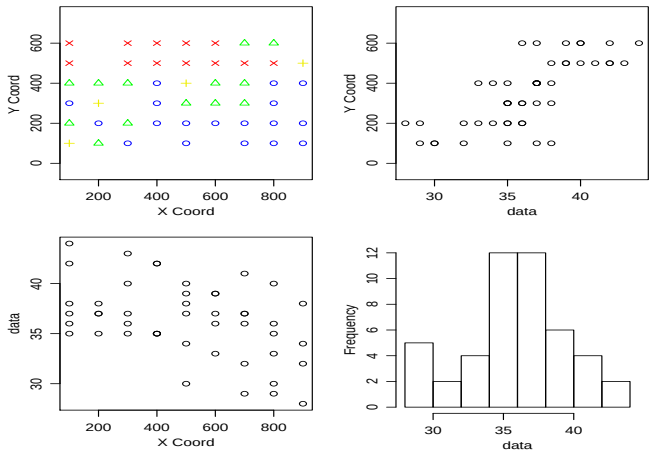
```
> a <- read.table("http://www.stat.ucla.edu/~nchristo/data_var.txt",
  header=TRUE)
> # Read data from a file (make sure you are at the correct working
  directory):
> a<-read.table("data_var.txt", header=T)
> b <- as.geodata(a)
> names(a)
[1] "x"    "y"    "var1"
> names(b)
$coords
[1] "x" "y"

$data
[1] "data"
```

A summary of the data can be obtained with

```
> summary(b)
```

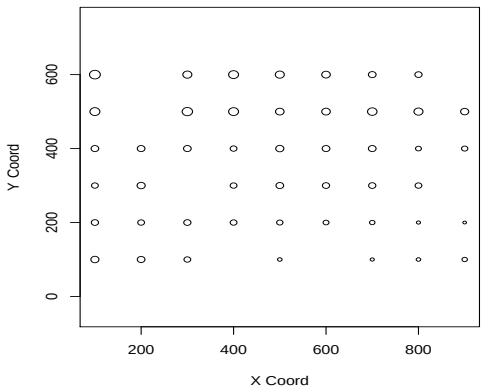
Some plots:
The `geoR` built-in plot function `plot` produces a 2 by 2 display showing the data points (spatial locations), the plot of the values against the x and y coordinates (separately), and the histogram of the observed values. Here it is:

```
> plot(b)
```



Another built-in plot function is `points` or `points.geodata` which gives the following:

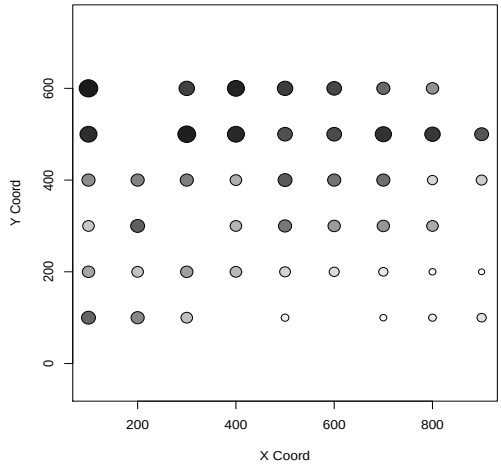
```
> points(b)
#or
> points.geodata(b)
```



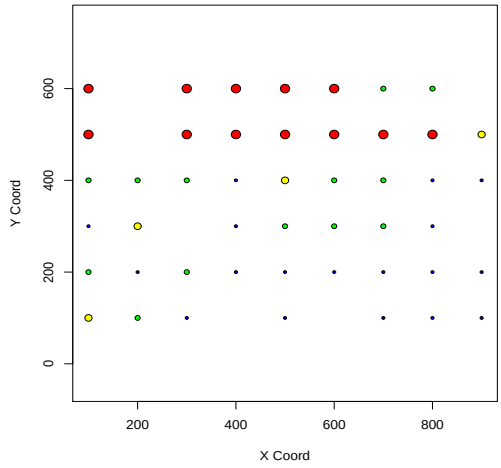
The plot above shows a bubble graph with the points having sizes proportional to the data values. By default the minimum value of the data set has size 0.5 and the maximum 1.5. You can change these using the `cex.min` and `cex.max` arguments. Other options here is to use the argument `pt.divide` which divides the points into categories based on quartiles or quintiles. Or you can give equal size to all the points. You can use for example:

```
> points(b, cex.min=1, cex.max=3, col="gray")

> points.geodata(b, pt.divide="quartile")
> points.geodata(b, pt.divide="quintile")
> points.geodata(b, pt.divide="equal")
```

The first command above will produce:



The second command above will produce:

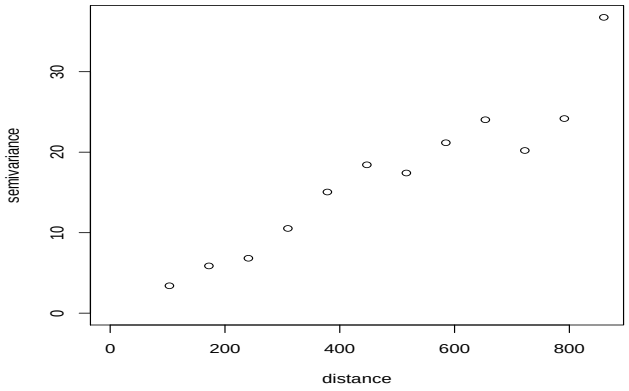


Computing the variogram:
The variogram can be computed using the function `variog`. There are many arguments within this function that we can use, but let's try something simple:

```
> variogram1 <- variog(b, max.dist=900)
```

And we get the following *omnidirectional* variogram (points are paired on all directions):

```
> plot(variogram1)
```



The object *variogram1* contains many components. You can see them below:

```
> names(variogram1)
 [1] "u"                "v"                "n"                "sd"
 [5] "bins.lim"         "ind.bin"          "var.mark"         "beta.ols"
 [9] "output.type"      "max.dist"         "estimator.type"   "n.data"
[13] "lambda"           "trend"            "pairs.min"        "nugget.tolerance"
[17] "direction"        "tolerance"        "uvec"             "call"
```

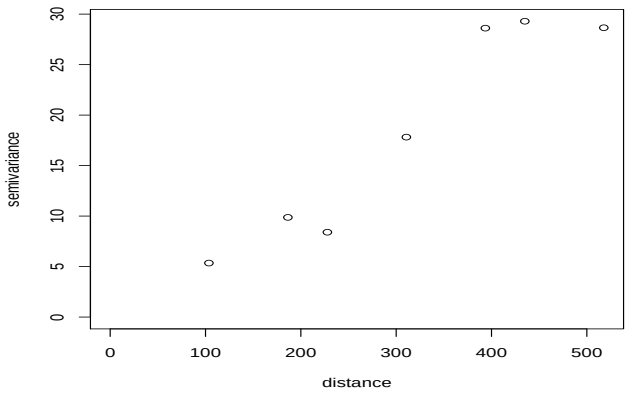
The most important components are *u, v, n*. The component u is a vector that contains the separation distances (h), v is a vector that contains the value of the sample semivariogram computed for each value in u, and n is the number of pairs for each value in u. Here is what we have for our example:

```
> variogram1$u
 [1] 103.2031 172.0052 240.8073 309.6094 378.4115 447.2136 516.0157 584.8178 653.6199
[10] 722.4220 791.2241 860.0261
> variogram1$v
 [1]  3.402778  5.869919  6.822430 10.525568 15.052941 18.431250 17.414179 21.171053
 [9] 24.030612 20.210000 24.176471 36.750000
> variogram1$n
 [1]  72 123 107 176 170  80 134  76  49  50  34   6
```

Suppose we want to compute a directional variogram on the north-south direction, with angular tolerance of 30^o. We use the following:

```
> variogram2 <- variog(b, dir=0, max.dist=900, tol=pi/6)
variog: computing variogram for direction = 0 degrees (0 radians)
        tolerance angle = 30 degrees (0.524 radians)
> plot(variogram2)
```

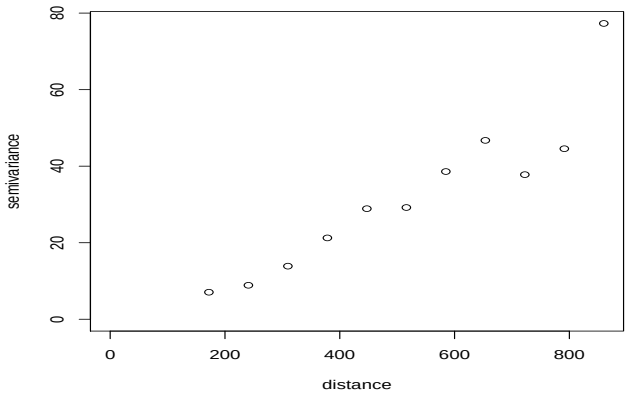
Note: The argument *dir* must be between $(0, \pi)$. The argument *tol* must be between $(0, \frac{\pi}{2})$. We get the following variogram:



Here is another one on the direction northwest-southeast, with angular tolerance 45^o.

```
> variogram3 <- variog(b, dir=3*pi/4, max.dist=900, tol=pi/4)
variog: computing variogram for direction = 135 degrees (2.356 radians)
        tolerance angle = 45 degrees (0.785 radians)
> plot(variogram3)
```

And we get the variogram below:

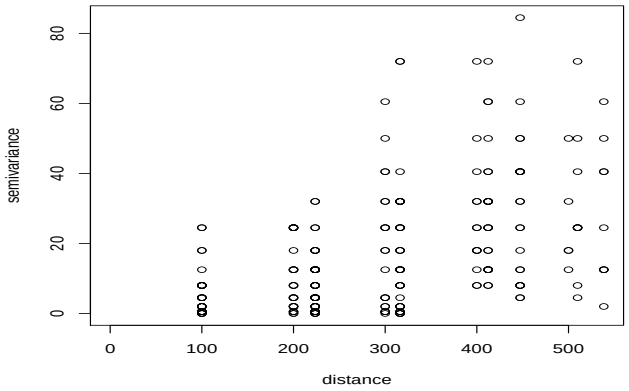


The previous variogram commands used the *option=bin* which is the default and therefore you do not have to type it. The default works as follows: The center of the bins are defined by *u=seq(0, max.dist, l=13)*. The interval spanned by each bin is given by the mid-points between the centers of the bins. You can also compute the *variogram cloud*. For both the *bin* or *cloud* option the variogram estimator can be either the classical (it is the default) or the robust (by Cressie and Hawkins (1980)). The robust estimator can be used with the argument *estimator.type="modulus"*.

Let's try the option *cloud*.

```
> variogram4 <- variog(b, dir=0, max.dist=900, tol=pi/6, option="cloud")
variog: computing variogram for direction = 0 degrees (0 radians)
        tolerance angle = 30 degrees (0.524 radians)
> plot(variogram4)
```

We get the following plot:



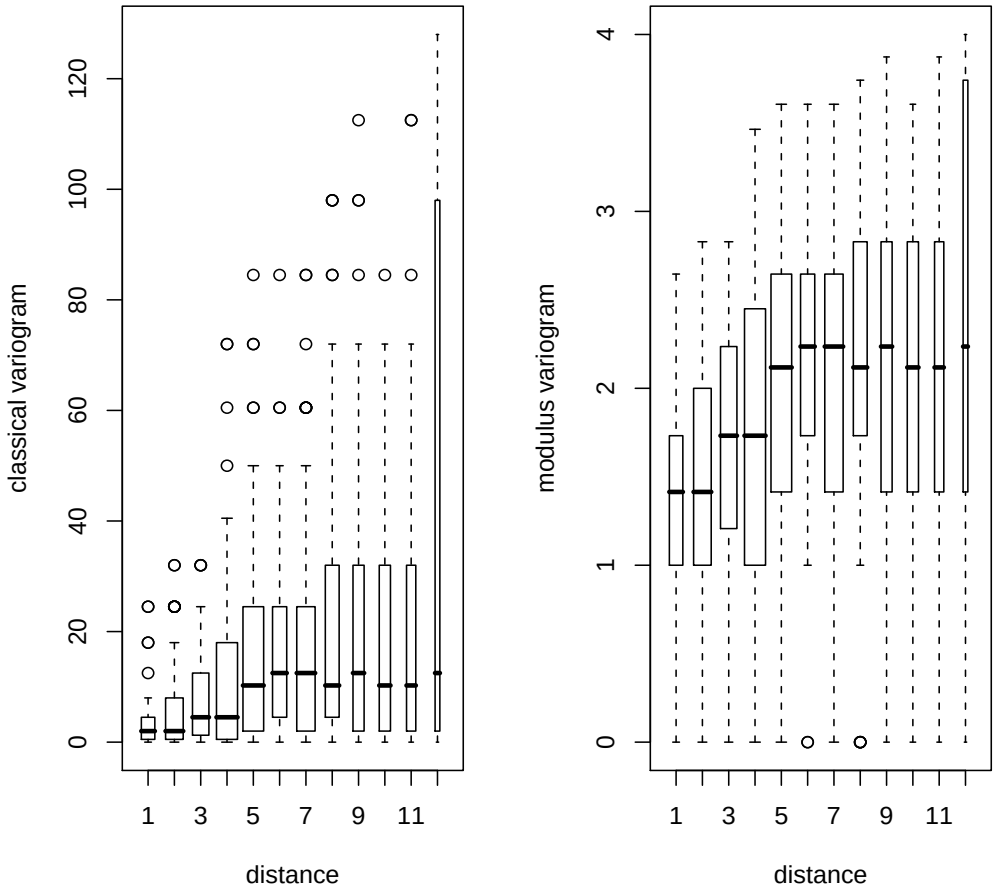
Using the robust estimator:

```
variogram_robust <- variog(b, dir=3*pi/4, max.dist=900,
  estimator.type="modulus", tol=pi/4)
```

Once the cloud variogram is constructed we can draw boxplots for the different bins to check the presence of outliers:

```
cloud1 <- variog(b, bin.cloud=T, max.dist=900)
cloud2 <- variog(b, bin.cloud=T, estimator.type="modulus", max.dist=900)

par(mfrow=c(1,2))
plot(cloud1, bin.cloud=T)
plot(cloud2, bin.cloud=T)
```

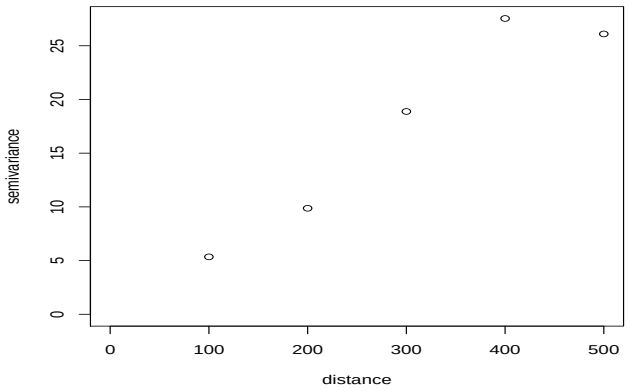
Here is the result of the previous commands:



On the bins:
The bins can be defined by the user with the argument *uvec* as shown below:

```
> variogram5 <- variog(b, dir=0, max.dist=500, tol=0, uvec=seq(0,500, 100))
variog: computing variogram for direction = 0 degrees (0 radians)
        tolerance angle = 0 degrees (0 radians)
```

And we get the following variogram:



Here are the values of u, v, n:

```
> variogram5$u
[1] 100 200 300 400 500
> variogram5$v
[1]  5.347222  9.870370 18.880952 27.538462 26.100000
> variogram5$n
[1] 36 27 21 13  5
```

Let's verify the last value of the semivariogram (26.1). We will have to look at the first figure (page 1) and pair all the data that are 500 ± 50 m apart on the north-south direction.

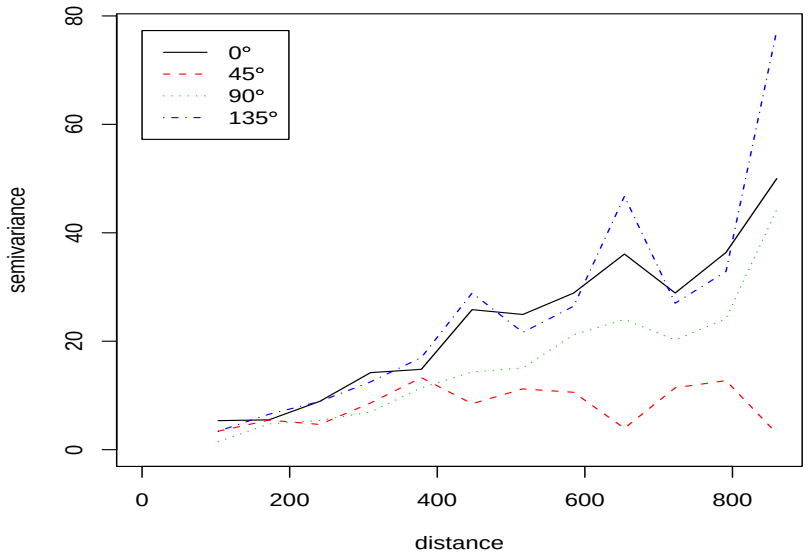
Note:

1. The vector `uvec` contains the center of the bins. The first separation distance h will be the distance between the centers of the first two bins. For example when $h = 100$ we will pair all the points that fall between 100 ± 50. The value 50 is the distance tolerance which is equal to $\frac{1}{2}h$. Similarly, if we look at the second element of the vector `uvec` which is 200 then we will pair points that fall between 200 ± 50 etc.
2. The argument `max.dist` is also very useful and should be used. Extending the variogram calculations over the full available range may not be necessarily helpful. This is because at large distances the variogram calculations become unstable.

Using the variog4 command:
We can construct four directional variograms on $0^o, 45^o, 90^o$, and 135^o as follows:

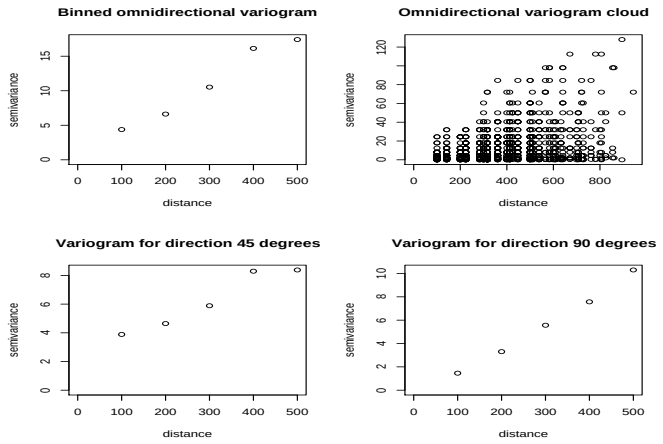
```
> variogram6 <- variog4(b, max.dist=900, tol=pi/3)
variog: computing variogram for direction = 0 degrees (0 radians)
        tolerance angle = 60 degrees (1.047 radians)
variog: computing variogram for direction = 45 degrees (0.785 radians)
        tolerance angle = 60 degrees (1.047 radians)
variog: computing variogram for direction = 90 degrees (1.571 radians)
        tolerance angle = 60 degrees (1.047 radians)
variog: computing variogram for direction = 135 degrees (2.356 radians)
        tolerance angle = 60 degrees (1.047 radians)
variog: computing omnidirectional variogram

> plot(variogram6)
```

We get a plot with the four variograms:



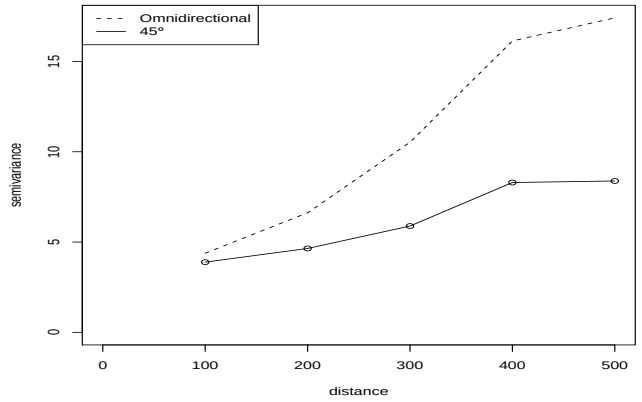
Try the following:

```
> var1 <- variog(b, uvec=seq(100,500,100))
> var2 <- variog(b,option="cloud")
> var3 <- variog(b, uvec=seq(100,500,100), dir=pi/4)
> var4 <- variog(b, uvec=seq(100,500,100), dir=pi/2)
> par(mfrow=c(2,2))
> plot(var1, main="Binned omnidirectional variogram")
> plot(var2, main="Omnidirectional variogram cloud")
> plot(var3, main="Variogram for direction 45 degrees")
> plot(var4,  main="Variogram for direction 90 degrees")
```



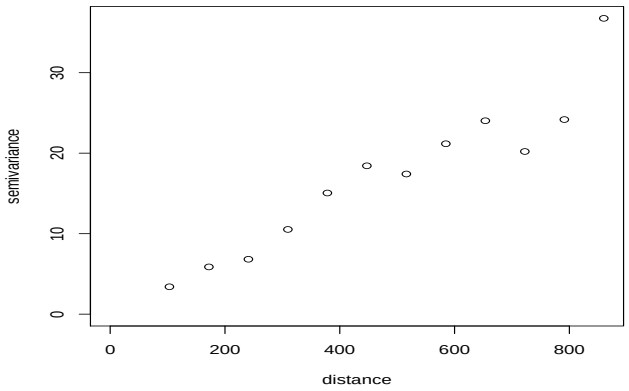
Also this:

```
> plot(var1, type="l", lty=2)
> lines(var3)
> legend("topleft", legend=c("Omnidirectional", expression(45*degree)),
  lty=c(2,1))
```



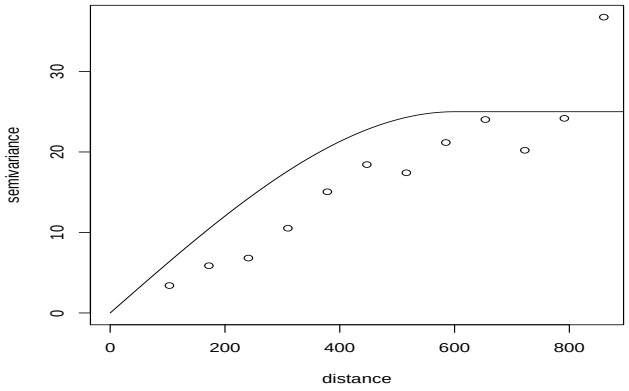
Fitting a variogram model over the empirical variogram:
If we know the variogram parameters we can use the function *lines.variomodel* to draw it over the empirical variogram. In practice of course we don't know the parameters. In the example here we will estimate the parameters *by eye*.

The graph below is the *omnidirectional* variogram from page 4.



Let's assume that we want to fit the spherical variogram over this empirical variogram. We can estimate by eye that the sill, range, and nugget are equal to 25, 600, and 0 respectively. Here is how the *lines.variomodel* works:

```
> variogram1 <- variog(b, max.dist=900)
variog: computing omnidirectional variogram
> plot(variogram1)
> lines.variomodel(cov.model="sph", cov.pars=c(25,600), nug=0, max.dist=900)
```



For exponential:

```
> lines.variomodel(cov.model="exp", cov.pars=c(25,600), nug=0, max.dist=900)
```