KrigeOrdinaryPart1

Kostas Goulias Geog 210B Winter 2018

3/4/2018

Ordinary Kriging Example from Isaacs and Srivastava 1989

The intro to kriging portion of this chapter and the noumerical example are on gauchopace

The problem

We have a set of points for which we have measurements in ppm at 7 stations.

We want to estimate the ppm at another location for which we know its xy coordinates.

We first enter the data we have:

```
test = data.frame (    SampleNo =c(0, 225, 437, 367,52, 259, 436, 366),
                       x = c(65, 61, 63, 64,68, 71, 73, 75),
                       y = c(137, 139, 140, 129, 128, 140, 141, 128),
                       v = c(999, 477,696,227,646,606,791,783))
```

x and y are the coordinates of the points. v is the measurement in ppm

```
test

##   SampleNo  x   y   v
## 1        0 65 137 999
## 2      225 61 139 477
## 3      437 63 140 696
## 4      367 64 129 227
## 5       52 68 128 646
## 6      259 71 140 606
## 7      436 73 141 791
## 8      366 75 128 783
```

In this table point with SampleNo = 0 is the point with the unknown value. In the Table above I eneterd this as 999

Our objective is to estimate the value at this location using the following formula:

$$v_0 = \sum_{i=1}^{7} w_j \, v_j$$

So this is a weighted sum of all the 7 values from my sample.

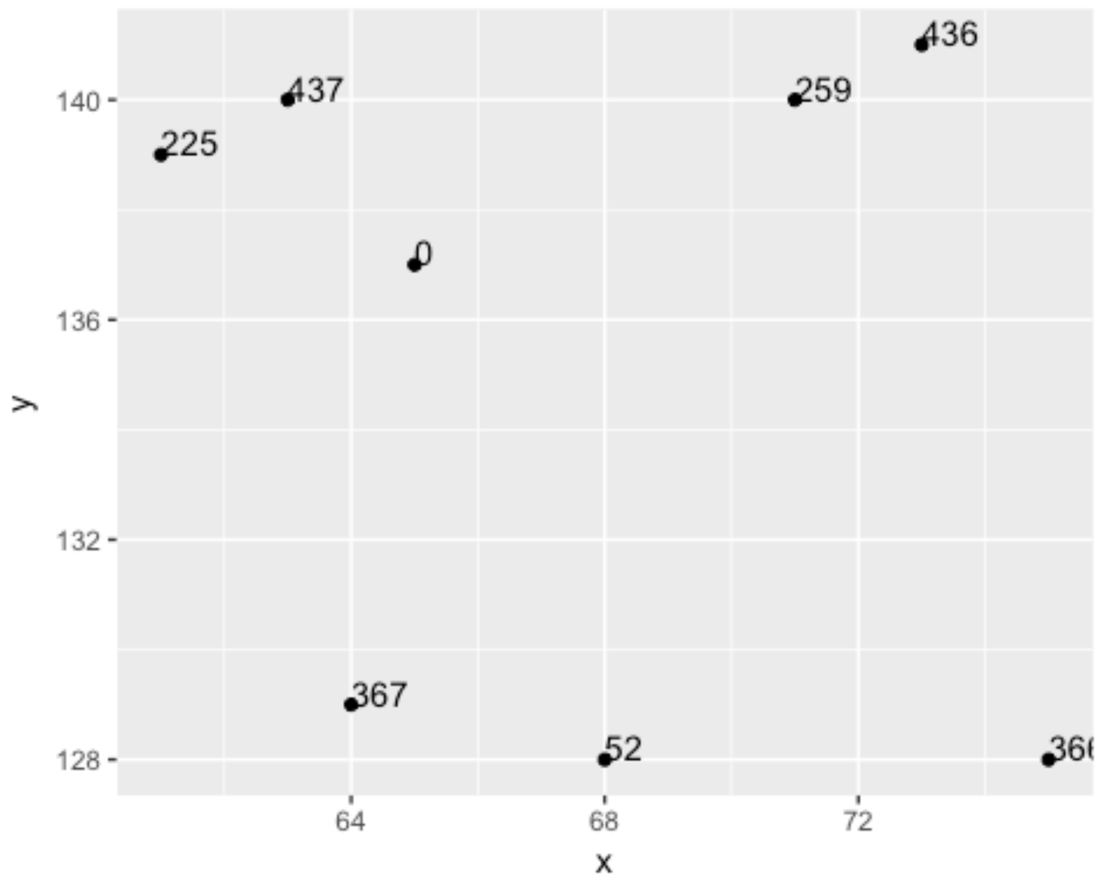
1. For theoretaical reasons (see the chapter on gauchospace) I know the weights need to sum to 1 to give me an unbiased prediction of the value in point 0.

2. I also assume that all the points come from a random process in which the overall mean is the same for the entire random field I observe.

3. I also know that all the random variables of this random field (the vs) are from the same distribution and therefore have the same variance.

The points with their SampleNo are in the follwoing figure.

```r
library(ggplot2)
ggplot(test, aes(x=x, y=y,  label=SampleNo))+
  geom_point() +geom_text(aes(label=SampleNo),hjust=0, vjust=0)
```



Compute the distances among all points (you might need to install package "distances")

Distance below is the Euclidean distance among points.

```
library(distances)

data_points <- data.frame(x = c(65, 61, 63, 64,68, 71, 73, 75),
                          y = c(137, 139, 140, 129, 128, 140, 141, 128))
# Euclidean distances
h <- distances(data_points)
h <-as.matrix(h)
h
```

```
##           1         2         3         4         5         6         7
## 1  0.000000  4.472136  3.605551  8.062258  9.486833  6.708204  8.944272
## 2  4.472136  0.000000  2.236068 10.440307 13.038405 10.049876 12.165525
## 3  3.605551  2.236068  0.000000 11.045361 13.000000  8.000000 10.049876
## 4  8.062258 10.440307 11.045361  0.000000  4.123106 13.038405 15.000000
## 5  9.486833 13.038405 13.000000  4.123106  0.000000 12.369317 13.928388
## 6  6.708204 10.049876  8.000000 13.038405 12.369317  0.000000  2.236068
## 7  8.944272 12.165525 10.049876 15.000000 13.928388  2.236068  0.000000
## 8 13.453624 17.804494 16.970563 11.045361  7.000000 12.649111 13.152946
##          8
## 1 13.45362
## 2 17.80449
## 3 16.97056
## 4 11.04536
## 5  7.00000
## 6 12.64911
## 7 13.15295
## 8  0.00000
```

Using the weighted sum of the 7 points we get a value for the unknown value at point 65E, 137N

The distance between this point and the other 7 points is the first row of the matrix above.

The solution

Our objective is to get a set of weights that will provide an unbiased estimate of the unknown value at this location and we also want this estimate to be with estimation of minimum variance

The weights will end up being produced by a formula that takes into account distance from the observed points but also clustering of the observed points.

One way to visualize what will happen is to write the weights as

w = Distance / Clustering

In this way, we can develop weights that are tailored to the observed values and the relative distance to all the observed points.

Distance in this sense is considered as statistical distance because it includes differences among the v values.

So, instead of using just Euclidean distances we also want to use something that combines the distance in space with the difference in values v observed at each location.

This information is contained in the covariance of values v among all the points we observed and the values of the unknown vs at locations that we predict.

But, we only know the location of the point to predict its v and made some assumptions about the random field from which the v values emerge.

Since we assumed all these observations are realizations from a random field with the same mean, any diffreneces between the values are due to the distance among the points.

This is a function that we can use to give us the estimate we need for point 0. So, all we need is a model of how the values v co-vary in space as a function of distance among points.

Let define a few basic functions used in Geostatistics

(Semi)Variogram:

$$\gamma_{ij} = 1/2E([V_i - V_j]^2)$$

This represents the squares of the differences among the values of the variable v. When we examine points that are close to each other this value should be small and when we examine points that are further apart this value should become larger. Think of a hill.

The covarinace should be the opposite if distance is a major factor. Points that are closer to each other should have higher covariance and points further apart should have lower covariance.

Isaacs and Srivastava (1989) on gauchspace in their example used the following functions as model of the covariance at different distance "bins" h:

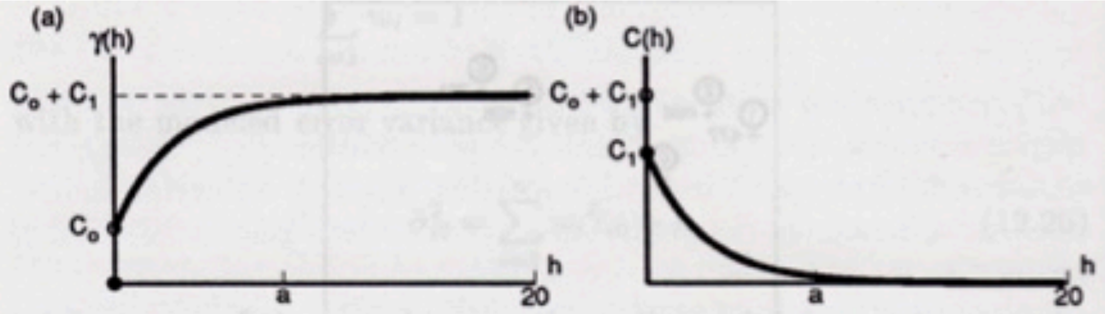


Figure 12.2 An example of an exponential variogram model (a) and an exponential covariance function (b).

our covariances from the following function:

$$\tilde{C}(\mathbf{h}) = \begin{cases} C_0 + C_1 & \text{if } |\mathbf{h}| = 0 \\ C_1 exp(\frac{-3|\mathbf{h}|}{a}) & \text{if } |\mathbf{h}| > 0 \end{cases} \tag{12.23}$$

Using Equation 12.17, this covariance function corresponds to the following variogram:

$$\tilde{\gamma}(\mathbf{h}) = \begin{cases} 0 & \text{if } |\mathbf{h}| = 0 \\ C_0 + C_1(1 - exp(\frac{-3|\mathbf{h}|}{a})) & \text{if } |\mathbf{h}| > 0 \end{cases} \tag{12.24}$$

Both of these functions, shown in Figure 12.2, can be described by the following parameters:

- C_0, commonly called the *nugget effect*, which provides a discontinuity at the origin.
- a, commonly called the *range*, which provides a distance beyond which the variogram or covariance value remains essentially constant.
- $C_0 + C_1$, commonly called the *sill* [3], which is the variogram value for very large distances, $\gamma(\infty)$. It is also the covariance value for $|\mathbf{h}| = 0$, and the variance of our random variables, $\tilde{\sigma}^2$.

One more set of assumptions about the paparemeters of the above:

$$C_0 = 0; a = 10; C_1 = 10$$

These assumptions mean that: The covariance at zero distance, which is the sill, is 10. this is also the variance of the random variables defining the random field.

Note that we have a model for the covariance at different distances and we can start computing the pieces that create our weights.

Get the observed locations, compute distances, and create the covraiance matrix. Then, using the equation of the covariance among values at different distances (Equation 12.23 above) compute the covariances for all points

```
observeddp <- data.frame (x = c( 61, 63, 64,68, 71, 73, 75),
                          y = c(139, 140, 129, 128, 140, 141, 128))
obsdist <- distances(observeddp)
obsdist <-as.matrix(obsdist)
ObsCh = as.matrix (10*exp(-0.3*obsdist))
```

When computing the weights in Ordinary Kriging we also need to impose the contsraint that the weights sum to 1.
This is done by adding a row and a column to the matrices we will use to estimate the weights.
The proof for this is on pages 286-287 of the Isaacs and Srivastava book.

Add one row and one column to the covariance

```
dummycol = as.matrix(c(1,1,1,1,1,1,1) )
ObsCh <- cbind(ObsCh, dummycol)

dummyrow = as.matrix(c(1,1,1,1,1,1,1,0))
dummyrow=t(dummyrow)
ObsCh <- rbind(ObsCh, dummyrow)
```

The matrix C(h) we need for the denominator of the weights computation is C and its inverse is CInv:

```
C=ObsCh
C

##            1           2          3          4          5          6
## 1 10.0000000  5.11288948  0.4362644  0.2001003  0.4904767  0.2600003
## 2  5.1128895 10.00000000  0.3638465  0.2024191  0.9071795  0.4904767
## 3  0.4362644  0.36384650 10.0000000  2.9027350  0.2001003  0.1110900
## 4  0.2001003  0.20241911  2.9027350 10.0000000  0.2445807  0.1532122
## 5  0.4904767  0.90717953  0.2001003  0.2445807 10.0000000  5.1128895
## 6  0.2600003  0.49047666  0.1110900  0.1532122  5.1128895 10.0000000
## 7  0.0478941  0.06150826  0.3638465  1.2245643  0.2248891  0.1933412
##    1.0000000  1.00000000  1.0000000  1.0000000  1.0000000  1.0000000
##             7  
## 1  0.04789410 1
## 2  0.06150826 1
## 3  0.36384650 1
## 4  1.22456428 1
## 5  0.22488905 1
## 6  0.19334119 1
## 7 10.00000000 1
##    1.00000000 0

CInv <- solve(C)
CInv

##              1            2            3            4            5
## 1  0.127013252 -0.076697116 -0.013056968 -0.009041150 -0.007680222
## 2 -0.076697116  0.129492450 -0.010253764 -0.008425832 -0.015243522
## 3 -0.013056968 -0.010253764  0.098176809 -0.041622890 -0.009545242
## 4 -0.009041150 -0.008425832 -0.041622890  0.101856313 -0.009099830
## 5 -0.007680222 -0.015243522 -0.009545242 -0.009099830  0.129933149
## 6 -0.008713079 -0.008232264 -0.010113049 -0.009256027 -0.076767527
## 7 -0.011824718 -0.010639952 -0.013584895 -0.024410585 -0.011596806
##    0.136115493  0.121176509  0.156455503  0.139307891  0.117511616
##              6           7           
## 1 -0.008713079 -0.01182472  0.1361155
## 2 -0.008232264 -0.01063995  0.1211765
## 3 -0.010113049 -0.01358490  0.1564555
## 4 -0.009256027 -0.02441059  0.1393079
## 5 -0.076767527 -0.01159681  0.1175116
## 6  0.126296973 -0.01321503  0.1409339
## 7 -0.013215027  0.08527198  0.1884991
##    0.140933893  0.18849910 -2.1801561
```

The vector D is the distance of the prediction point (the one we interpolate for) from the other 7 points plus one more element 1.00 at the end (similar reason as for the C matrix)

```
D <- as.matrix(c(2.61,3.39,0.89, 0.58, 1.34, 0.68, 0.18, 1.00))
D

##      [,1]
## [1,] 2.61
## [2,] 3.39
## [3,] 0.89
## [4,] 0.58
## [5,] 1.34
## [6,] 0.68
## [7,] 0.18
## [8,] 1.00
```

And the weights we need to perform the interpolation are:

```
w = CInv %*% D

w

##          [,1]
## 1  0.17240745
## 2  0.31802416
## 3  0.12873986
## 4  0.08629744
## 5  0.15183861
## 6  0.05655102
## 7  0.08614147
##   -0.90683187
```

The estimated value for point 0 and its variance are:

$$
\begin{aligned}
\hat{v}_0 &= \sum_{i=1}^{n} w_i v_i \\
&= (0.173)(477) + (0.318)(696) + (0.129)(227) + (0.086)(646) + \\
&\quad (0.151)(606) + (0.057)(791) + (0.086)(783) \\
&= 592.7 \text{ ppm}
\end{aligned}
$$

$$
\begin{aligned}
\tilde{\sigma}_R^2 &= \tilde{\sigma}^2 - \sum_{i=1}^{n} w_i \tilde{C}_{i0} + \mu \\
&= 10 - (0.173)(2.61) - (0.318)(3.39) - (0.129)(0.89) - \\
&\quad (0.086)(0.58) - (0.151)(1.34) - (0.057)(0.68) - \\
&\quad (0.086)(0.18) + 0.907 \\
&= 8.96 \text{ ppm}^2
\end{aligned}
$$

Let's talk about each of the items above.

w = D/C

C is a record of the statistical distance among all points. Points that are close to each other take high values and points that are further apart in terms of their values take smaller values See points 1 and 2, 5 and 6, and points 3 and 4. The C values for these pairs go to the denominator of the weights. So, closely located points weigh less in creating a weight

Compare the weights for different points and compare the values in each of the matrices D and C.

Points that are close to each other tend to be discounted by C (in essence discounting redundancies) but this is balanced by D. Points 1 and 2 have the top D values.