Geostatistics Athens Week project

SD Team

November 17, 2021

Contents

1 The Jura data set

The Jura data set comprises seven heavy metals measured in the top soil of the swiss Jura, along with consistently coded land use and rock type factors, as well as geographic coordinates.

Variable description :

- Xloc: X coordinate, local grid km
- Yloc: Y coordinate, local grid km

- Landuse: Land use: 1: Forest, 2: Pasture (Weide(land), Wiese, Grasland), 3: Meadow (Wiese, Flur, Matte, Anger), 4: Tillage (Ackerland, bestelltes Land)
- Rock: Rock Types: 1: Argovian, 2: Kimmeridgian, 3: Sequanian, 4: Portlandian, 5: Quaternary.
- Cd: mg cadmium kgˆ-1 topsoil
- Co: mg cobalt kgˆ-1 topsoil
- Cr: mg chromium kgˆ-1 topsoil
- Cu: mg copper kgˆ-1 topsoil
- Ni: mg nickel kgˆ-1 topsoil
- Pb: mg lead kgˆ-1 topsoil
- Zn: mg zinc kgˆ-1 topsoil

You are given three different files:

- jura_pred.csv: learning dataset
- jura_grid.csv: prediction grid (contains locations and covariables)
- jura_val_loc: validation locations and covariables

2 Exploratory analysis

```
rm(list=ls()) #Clean the working directory
```

2.1 Basic statistics

1. load the dataset from jura_pred.csv (on the cloud)

```
#Data set
jura = read.csv("../../jura/jura_pred.csv")
```

2. What is the class of the dataset?

```
class(jura)
```

```
## [1] "data.frame"
```

3. What is the number of observations? What is the number of variables?

```
str(jura) #e.g.
```

```
## 'data.frame':    259 obs. of  11 variables:
##  $ Xloc   : num  2.39 2.54 2.81 4.31 4.38 ...
##  $ Yloc   : num  3.08 1.97 3.35 1.93 1.08 ...
##  $ Landuse: int  3 2 2 3 3 3 3 3 3 3 ...
##  $ Rock   : int  3 2 3 2 5 5 5 1 1 3 ...
##  $ Cd     : num  1.74 1.33 1.61 2.15 1.56 ...
##  $ Co     : num  9.32 10 10.6 11.92 16.32 ...
##  $ Cr     : num  38.3 40.2 47 43.5 38.5 ...
##  $ Cu     : num  25.72 24.76 8.88 22.7 34.32 ...
##  $ Ni     : num  21.3 29.7 21.4 29.7 26.2 ...
##  $ Pb     : num  77.4 77.9 30.8 56.4 66.4 ...
##  $ Zn     : num  92.6 73.6 64.8 90 88.4 ...
```

4. Print the name of the variables.

```
names(jura)
```

```
##  [1] "Xloc"    "Yloc"    "Landuse" "Rock"    "Cd"      "Co"      "Cr"
##  [8] "Cu"      "Ni"      "Pb"      "Zn"
```

5. Compute the minimum and maximum value for each coordinate.

```
min(jura$Xloc);min(jura$Yloc)
```

```
## [1] 0.626
```

```
## [1] 0.58
```

```
apply(jura[,1:2],2,min)
```

```
##  Xloc  Yloc
## 0.626 0.580
```

```
range(jura$Xloc)
```

```
## [1] 0.626 4.920
```

```
max(jura$Xloc);max(jura$Yloc)
```

```
## [1] 4.92
```

```
## [1] 5.69
```

```
apply(jura[,1:2],2,max)
```

```
## Xloc Yloc
## 4.92 5.69
```

```
range(jura$Yloc)
```

```
## [1] 0.58 5.69
```

6. Compute basic statistics for the seven different heavy metals (mean, min, max, quartiles and standard deviation)

```
summary(jura[,5:11])
```

```
##        Cd               Co               Cr              Cu
##  Min.   :0.1350   Min.   : 1.552   Min.   : 8.72   Min.   :  3.96
##  1st Qu.:0.6375   1st Qu.: 6.520   1st Qu.:27.44   1st Qu.: 11.02
##  Median :1.0700   Median : 9.760   Median :34.84   Median : 17.60
##  Mean   :1.3091   Mean   : 9.303   Mean   :35.07   Mean   : 23.73
##  3rd Qu.:1.7150   3rd Qu.:11.980   3rd Qu.:42.22   3rd Qu.: 27.82
##  Max.   :5.1290   Max.   :17.720   Max.   :67.60   Max.   :166.40
##        Ni              Pb              Zn
##  Min.   : 4.20   Min.   : 18.96   Min.   : 25.20
##  1st Qu.:13.80   1st Qu.: 36.52   1st Qu.: 55.00
##  Median :20.56   Median : 46.40   Median : 73.56
##  Mean   :19.73   Mean   : 53.92   Mean   : 75.08
##  3rd Qu.:25.42   3rd Qu.: 60.40   3rd Qu.: 89.92
##  Max.   :53.20   Max.   :229.56   Max.   :219.32
```

```
apply(jura[,5:11],2,sd)
```

```
##         Cd         Co         Cr         Cu         Ni         Pb         Zn
##  0.9151877  3.5760463 10.9575090 20.7126345  8.2328582 29.7921582 29.0192912
```

7. Compute the mean of cobalt concentration for the four different landuses

```
mean(jura[jura$Landuse==1,6],na.rm=T)
```

```
## [1] 7.694545
```

```
mean(jura[jura$Landuse==2,6],na.rm=T)
```

```
## [1] 10.06429
```

```
mean(jura[jura$Landuse==3,6],na.rm=T)
```

```
## [1] 9.393867
```

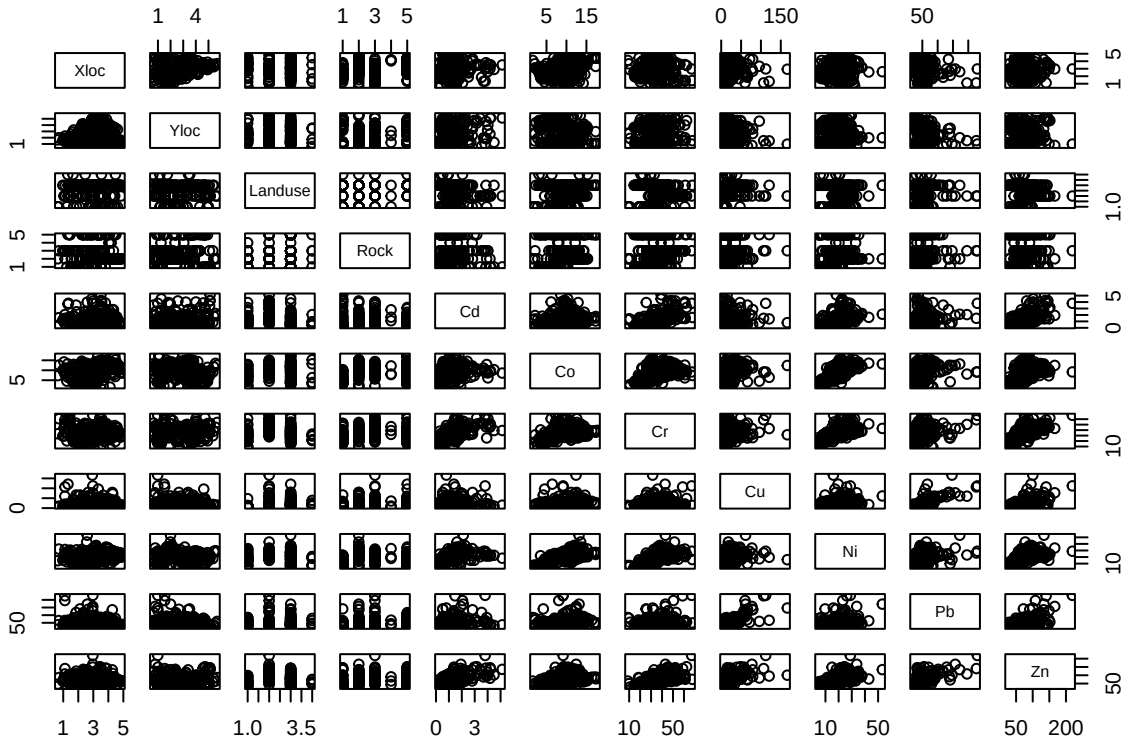
```
mean(jura[jura$Landuse==4,6],na.rm=T)
```

```
## [1] 8.372
```

2.1.1 Graphical Representations

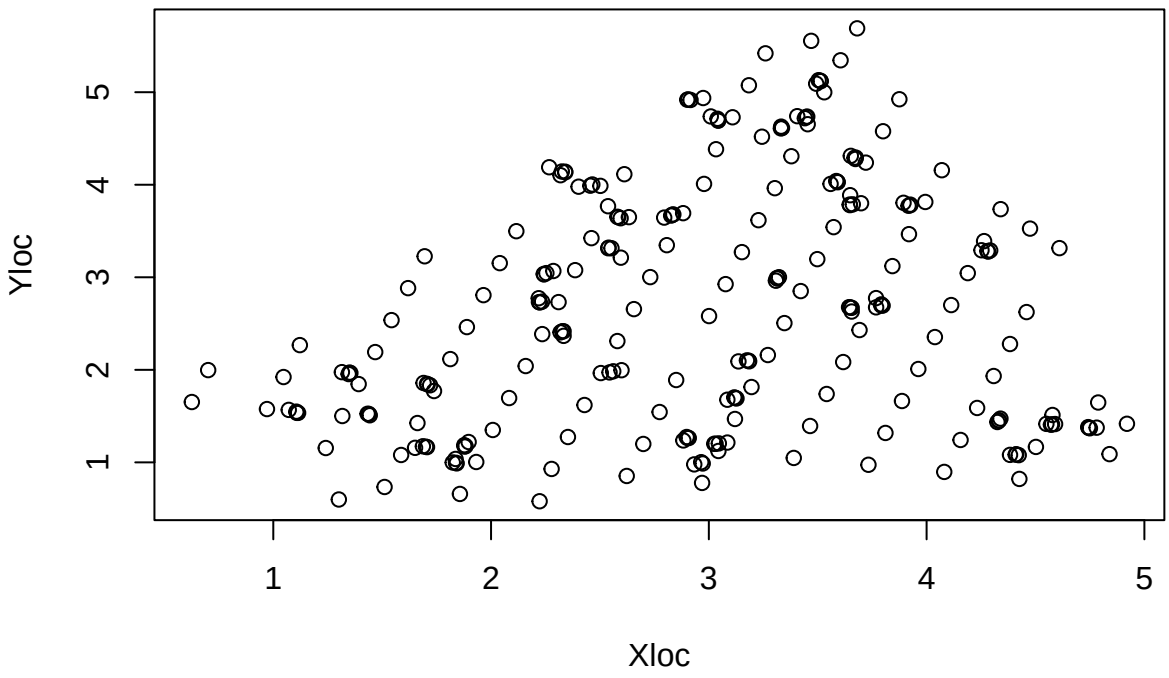
1. Apply the plot function to the whole dataset

```
plot(jura)
```



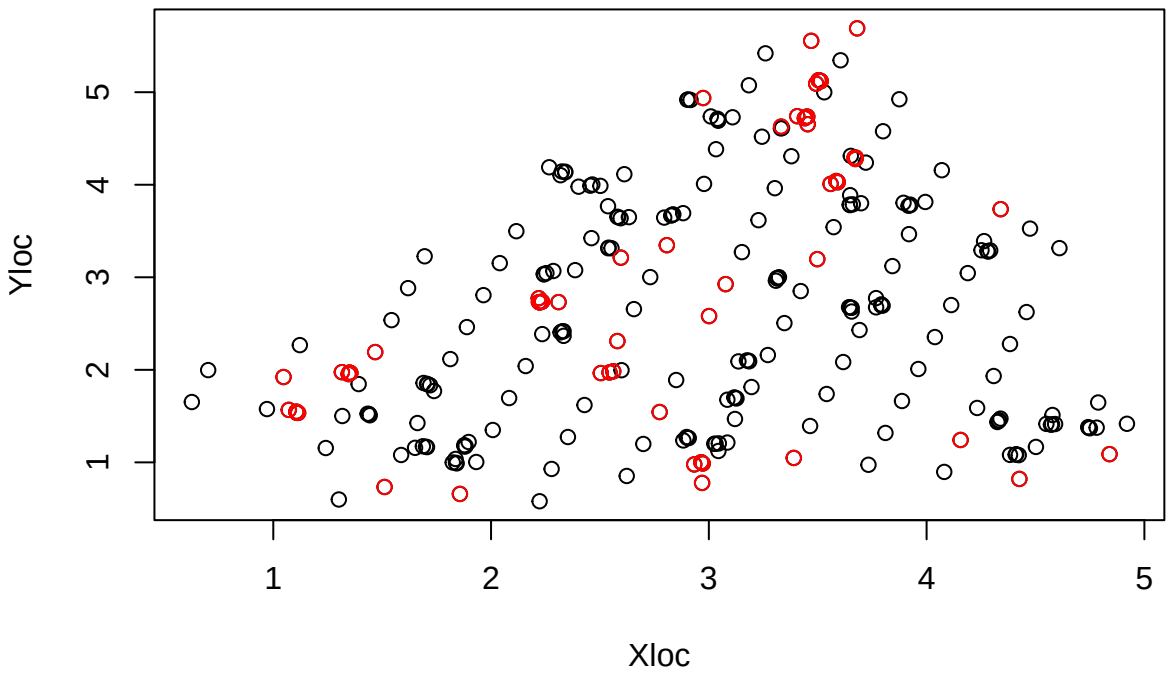
2. Plot the coordinates.

```
plot(jura[,1:2])
```



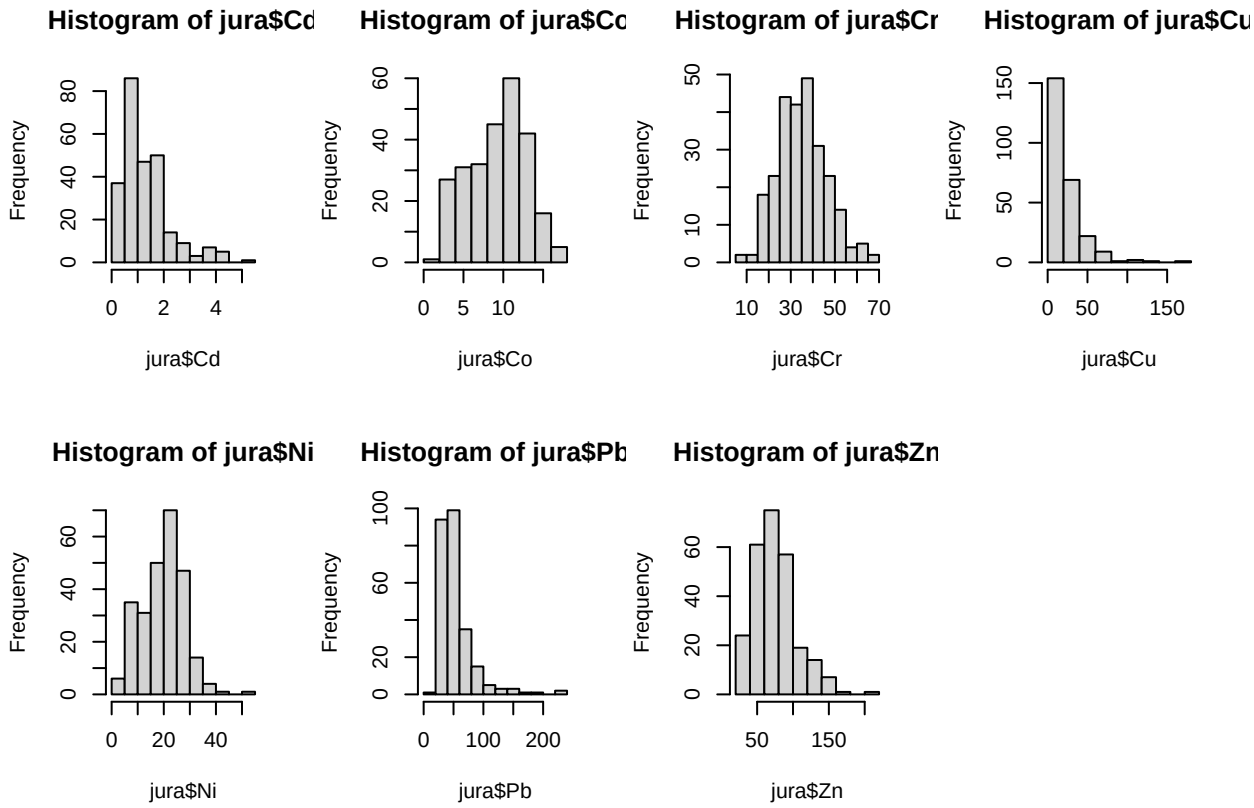
3. On the same plot, display the points with landuse 2 (pasture) in red.

```
plot(jura[,1:2])
points(jura[jura$Landuse==2,1:2],col='red')
```



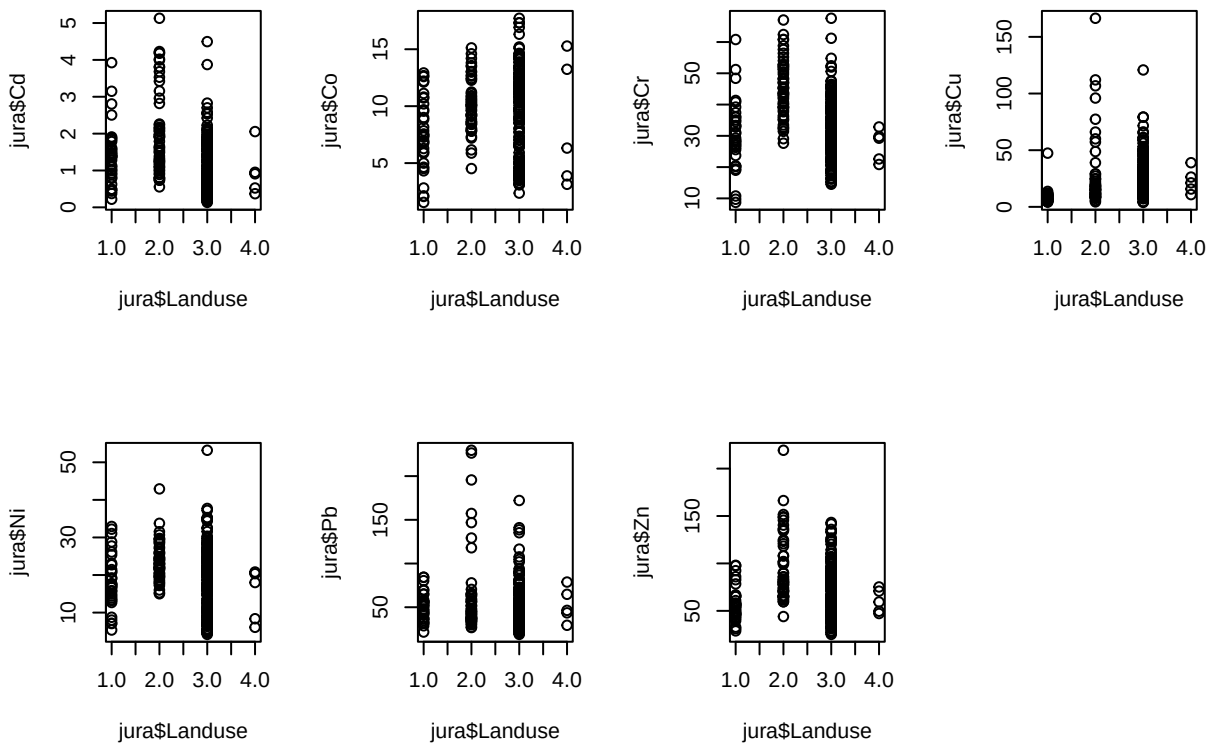
4. Plot the seven, heavy metal concentrations histograms.

```
par(mfrow=c(2,4)) # on a grid of 2x4 plots
hist(jura$Cd)
hist(jura$Co)
hist(jura$Cr)
hist(jura$Cu)
hist(jura$Ni)
hist(jura$Pb)
hist(jura$Zn)
```



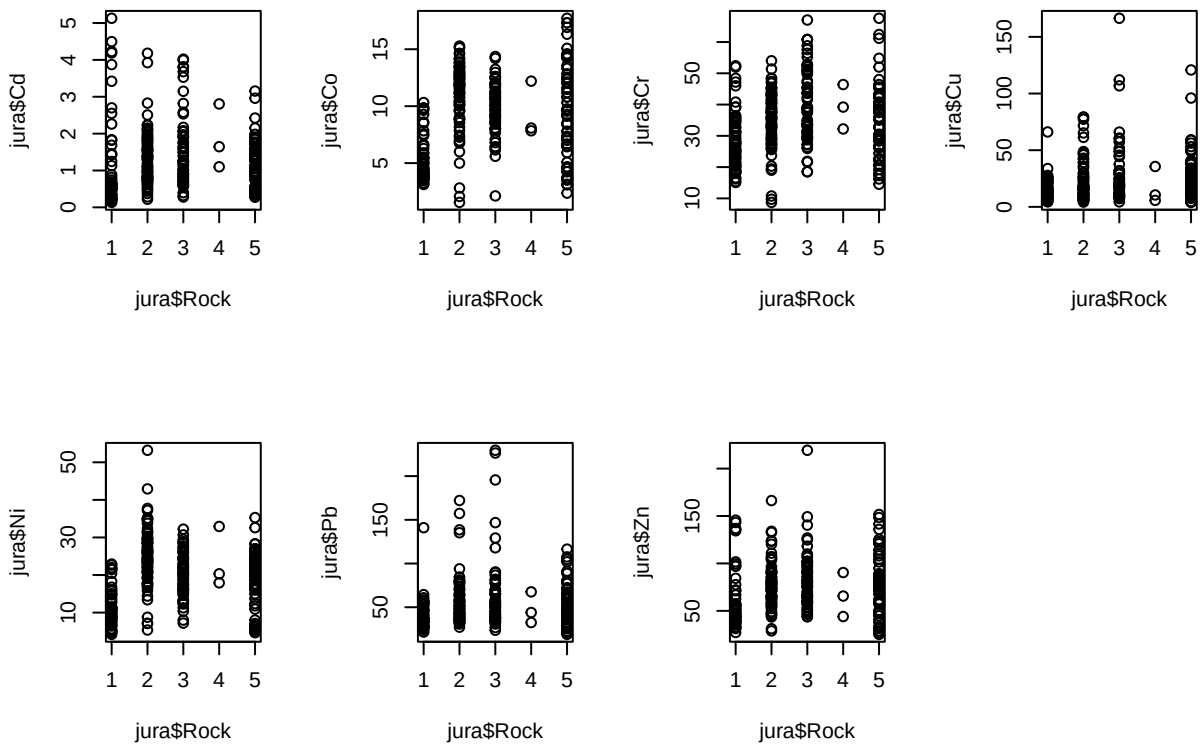
5. Plot the seven heavy metal concentrations as functions of the landuse

```
par(mfrow=c(2,4))
plot(jura$Landuse,jura$Cd)
plot(jura$Landuse,jura$Co)
plot(jura$Landuse,jura$Cr)
plot(jura$Landuse,jura$Cu)
plot(jura$Landuse,jura$Ni)
plot(jura$Landuse,jura$Pb)
plot(jura$Landuse,jura$Zn)
```



6. Plot the seven heavy metal concentrations as functions of the rocktype

```
par(mfrow=c(2,4))
plot(jura$Rock,jura$Cd)
plot(jura$Rock,jura$Co)
plot(jura$Rock,jura$Cr)
plot(jura$Rock,jura$Cu)
plot(jura$Rock,jura$Ni)
plot(jura$Rock,jura$Pb)
plot(jura$Rock,jura$Zn)
```



2.1.2 Some statistics

1. Transform the variables Landuse and Rock into factors (see ?as.factors).

```
jura$Landuse = as.factor(jura$Landuse)
jura$Rock = as.factor(jura$Rock)
```

2. Use the function aov to compute the analysis of variance of the cobalt concentrations with Landuse, Rock and their product as factors.

```
aov.co = aov(Co~Landuse+Rock+Landuse*Rock,data=jura)
summary(aov.co)
```

```
##               Df Sum Sq Mean Sq F value   Pr(>F)
## Landuse        3  123.5   41.18   5.719  0.00085 ***
## Rock           4 1154.1  288.53  40.078  < 2e-16 ***
## Landuse:Rock  10  286.7   28.67   3.982 4.72e-05 ***
## Residuals    241 1735.0    7.20
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

3. Do the same on the other concentrations (check the histograms prior to apply a transformation if necessary).

```
aov.cd = aov(Cd~Landuse+Rock+Landuse*Rock,data=jura)
summary(aov.cd)
```

```
##               Df Sum Sq Mean Sq F value   Pr(>F)
## Landuse        3  39.57  13.191  21.379 2.59e-12 ***
## Rock           4   0.84   0.209   0.339    0.851
## Landuse:Rock  10  26.99   2.699   4.375 1.20e-05 ***
## Residuals    241 148.69   0.617
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
aov.cr = aov(Cr~Landuse+Rock+Landuse*Rock,data=jura)
summary(aov.cr)
```

```
##               Df Sum Sq Mean Sq F value   Pr(>F)
## Landuse        3   7526  2508.5  32.317  < 2e-16 ***
## Rock           4   2044   510.9   6.582 4.83e-05 ***
## Landuse:Rock  10   2701   270.1   3.480 0.000268 ***
## Residuals    241  18707    77.6
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
aov.cu = aov(Cu~Landuse+Rock+Landuse*Rock,data=jura)
summary(aov.cu)
```

```
##               Df Sum Sq Mean Sq F value   Pr(>F)
## Landuse        3   8154  2717.9   6.985 0.000159 ***
## Rock           4   3774   943.5   2.425 0.048768 *
## Landuse:Rock  10   4982   498.2   1.280 0.242080
## Residuals    241  93776   389.1
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
aov.ni = aov(Ni~Landuse+Rock+Landuse*Rock,data=jura)
summary(aov.ni)
```

```
##               Df Sum Sq Mean Sq F value   Pr(>F)
## Landuse        3   1188   396.1  11.271 6.04e-07 ***
## Rock           4   5765  1441.4  41.015  < 2e-16 ***
## Landuse:Rock  10   2064   206.4   5.874 6.23e-08 ***
## Residuals    241   8469    35.1
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
aov.pb = aov(Pb~Landuse+Rock+Landuse*Rock,data=jura)
summary(aov.pb)
```

```
##               Df Sum Sq Mean Sq F value  Pr(>F)
## Landuse        3   4465  1488.2   1.749 0.15764
## Rock           4  11722  2930.5   3.444 0.00927 **
```

```
## Landuse:Rock  10   7736   773.6   0.909 0.52532
## Residuals    241 205071   850.9
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
aov.zn = aov(Zn~Landuse+Rock+Landuse*Rock,data=jura)
summary(aov.zn)
```

```
##               Df Sum Sq Mean Sq F value   Pr(>F)
## Landuse        3  31159   10386  16.840 5.79e-10 ***
## Rock           4  11417    2854   4.628  0.00129 **
## Landuse:Rock  10  26051    2605   4.224 2.03e-05 ***
## Residuals    241 148640     617
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

3 Utilities

Load RGeostats.

```
library(Rcpp)
library(RGeostats)
```

```
## Loading RGeostats - Version:13.0.1
```

```
constant.define("asp",1) #environment variable for the scale ratio between axes
```

Load the data set and the prediction grid

```
#Data set
jura_tot = read.csv("../../jura/jura_pred.csv")
#Prediction grid
grid = read.csv("../../jura/jura_grid.csv")
names(jura_tot)
```

```
##  [1] "Xloc"    "Yloc"    "Landuse" "Rock"    "Cd"      "Co"      "Cr"
##  [8] "Cu"      "Ni"      "Pb"      "Zn"
```

```
names(grid)
```

```
## [1] "Xloc"    "Yloc"    "Landuse" "Rock"
```

Coding of the factors

```
jura_tot$Landuse = as.factor(jura_tot$Landuse)
jura_tot$Rock = as.factor(jura_tot$Rock)

grid$Landuse = as.factor(grid$Landuse)
grid$Rock = as.factor(grid$Rock)
```

Change the names of the modalities to be consistent with their names on the grid

```
levels(jura_tot$Landuse)
```

```
## [1] "1" "2" "3" "4"
```

```
levels(grid$Landuse)
```

```
## [1] "Forest"  "Meadow"  "Pasture" "Tillage"
```

```
levels(jura_tot$Landuse)=c("Forest","Pasture","Meadow","Tillage")

levels(jura_tot$Rock)
```

```
## [1] "1" "2" "3" "4" "5"
```

```
levels(grid$Rock)
```

```
## [1] "Argovian"     "Kimmeridgian" "Portlandian"  "Quaternary"   "Sequanian"
```

```
levels(jura_tot$Rock)=c("Argovian","Kimmeridgian","Sequanian","Portlandian","Quaternary")
```

Separate the data set in two sets : the training set and the validation set. For the project, you should use the full data set for the training. You submit the prediction on Kaggle for a set of locations on which you only know the locations and the factors.

```
set.seed(1234) #Set the seed of the random generators

ntot = nrow(jura_tot)
ntrain = 200
nval = ntot - ntrain

indtrain = sample(ntot,ntrain)
indval = setdiff(1:ntot,indtrain)

jura = jura_tot[indtrain,]
val_loc = jura_tot[indval,1:4]

#val contains the values to predict. For the project, these values will be on Kaggle
#(for other locations) and you won't know them
#You will have the locations and covariables at the unknown locations by the following command :

#val_loc = read.csv("jura/jura_val_loc.csv")

val = cbind(1:nval,jura_tot[indval,]$Co)
```

RGeostats target grid and utilities for display

```
gridtemp = db.create(grid)
gridtemp = db.locate(gridtemp,c("Xloc","Yloc"),"x")
nx = c(length(unique(grid[,1])),length(unique(grid[,2])))
gridrg = db.grid.init(gridtemp,nodes=nx)
gridrg = migrate(gridtemp,gridrg,names=4:gridtemp$natt,radix="")
gridrg = db.rename(gridrg,2:3,c("Xloc","Yloc"))
gridrg = db.sel(gridrg,!is.na(gridrg[,"Landuse"])&!is.na(gridrg[,"Rock"]))

add.variable =function(var,grid,varname,gridtemp.=gridtemp)
{
  tt=db.add(gridtemp.,var)
  tt=db.rename(tt,tt$natt,varname)
  grid = migrate(tt,grid,names=tt$natt,radix="")
  grid
}
```

4 Interpolation exercise

Provide the maps of the cobalt concentration over the Swiss Jura obtained with several regression/interpolation methods, e.g.:

- anova
- linear regression on the coordinates
- local polynomial regression
- Nearest neighbour
- Inverse distance
- ...

1. Prediction by the mean

```
mean((val[,2]-mean(val[,2]))^2)
```

```
## [1] 12.61749
```

2. ANOVA

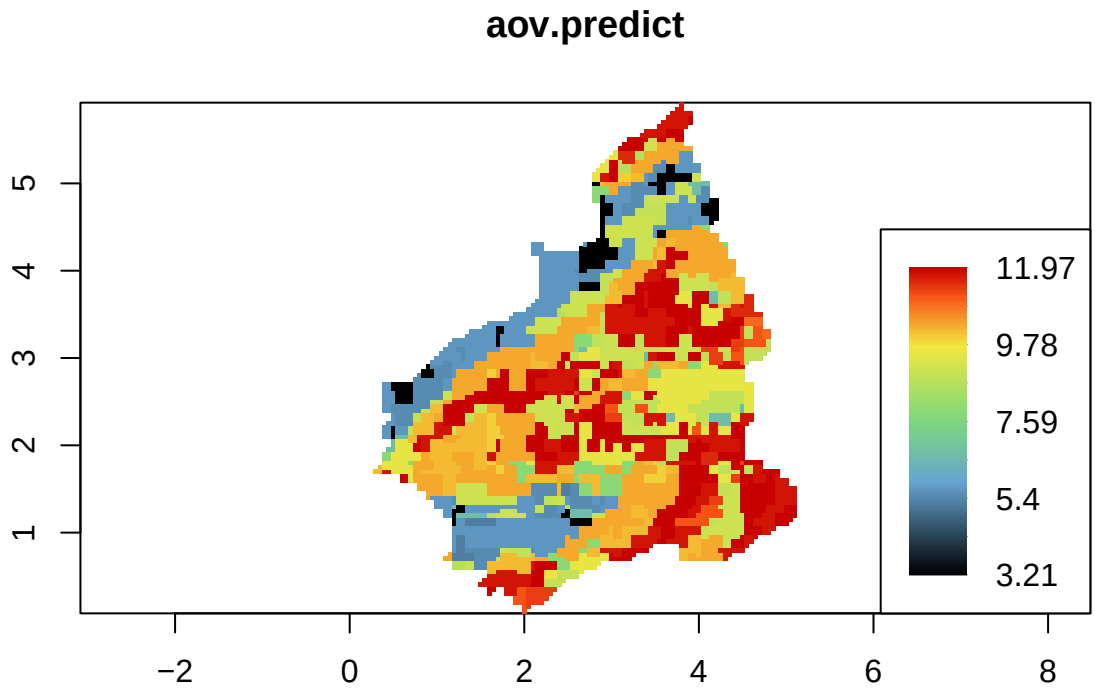
```
#Fit the anova model
aov.co = aov(Co~Landuse+Rock,data=jura)
summary(aov.co)
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## Landuse       3     74   24.68   3.201 0.0245 *
## Rock          4   1000  250.04  32.428 <2e-16 ***
## Residuals   192   1480    7.71
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#Prediction on the validation locations
res.aov.val=predict(aov.co,val_loc)

##
#Compute score
mean((res.aov.val-val[,2])^2)
```

```
## [1] 9.702636
```

```
#Prediction on the grid
res.aov=predict(aov.co,grid)

##
#Add to the RGeostats db and display
gridrg = add.variable(res.aov,gridrg,"aov.predict")
plot(gridrg,pos.legend=1)
```



3. Linear regression of functions of coordinates

```
#Fit the linear model

trend = lm(Co~Xloc+Yloc+Xloc*Yloc+I(Xloc^2)+I(Yloc^2),data=jura)
summary(trend)
```

```
#Prediction on the validation locations
res.trend.val=predict(trend,val_loc)

##
#Compute score
mean((res.trend.val-val[,2])^2)
##

#Prediction on the grid
res.trend=predict(trend,grid)

#Add to the RGeostats db and display
gridrg = add.variable(res.trend,gridrg,"trend.predict")
plot(gridrg,pos.legend=1)
```

4. N-Nearest neighbours

```
ns =  3 #number of samples to consider for the prediction
datrg = db.create(jura)
datrg = db.locate(datrg,2:3,"x")
datrg = db.locate(datrg,7,"z")
m = model.create(1)
neigh=neigh.create(radius=100,nmini=ns,nmaxi=ns)


###
valrg=db.create(val_loc)
valrg=db.locate(valrg,2:3,"x")
res.val = kriging(datrg,valrg,m,neigh)
mean((res.val[,6]-val[,2])^2,na.rm=T)
###


res = kriging(datrg,gridrg,m,neigh)
plot(res)
```

5. Inverse distance

```
degree = 4
datrg = db.create(jura)
datrg = db.locate(datrg,2:3,"x")
datrg = db.locate(datrg,7,"z")


###
valrg=db.create(val_loc)
valrg=db.locate(valrg,2:3,"x")
res.val = invdist(datrg,valrg,exponent = degree)
mean((res.val[,6]-val[,2])^2)

res = invdist(datrg,gridrg)
plot(res)
```

6. Improve the prediction by using other parameterizations for the previous methods.
7. Use other methods (Local Polynomial Regression, Random Forests, ...).

5 Univariate analysis

5.1 Variography

5.1.1 Experimental variogram (isotropic case)

1. Compute and plot the experimental variogram of the cobalt concentration (using `vario.calc()`). Try different values of lag and comment the results.

For starters it is important for us to outlie the importance of the variogram when dealing with structural analysis, we will also reinforce the fact that we are dealing with second order stationary random functions.

For a random function Z(x) the variogram is calculated as :

$\gamma(h) = \frac{1}{2}.Var[Z(x+h) - Z(x)]$

Hence the variogram shows how the difference between the random functions Z(x+h) and Z(x) evolve as a function of the lag h.

The variogram is a powerful tool because unlike the covariance , it does not require a knowledge of the mean which has to be estimated in the case of the covariance and hence could introduce a bias.

We will now proceed to some simulations on R studio , mainly plotting the variogram of the cobalt concentrations with different values of lags.

We will limit our study on lags up till 3. We will start our simulation with 10 lag points , that is a 0.3 lag between a random function and its translated version.

Create the db :

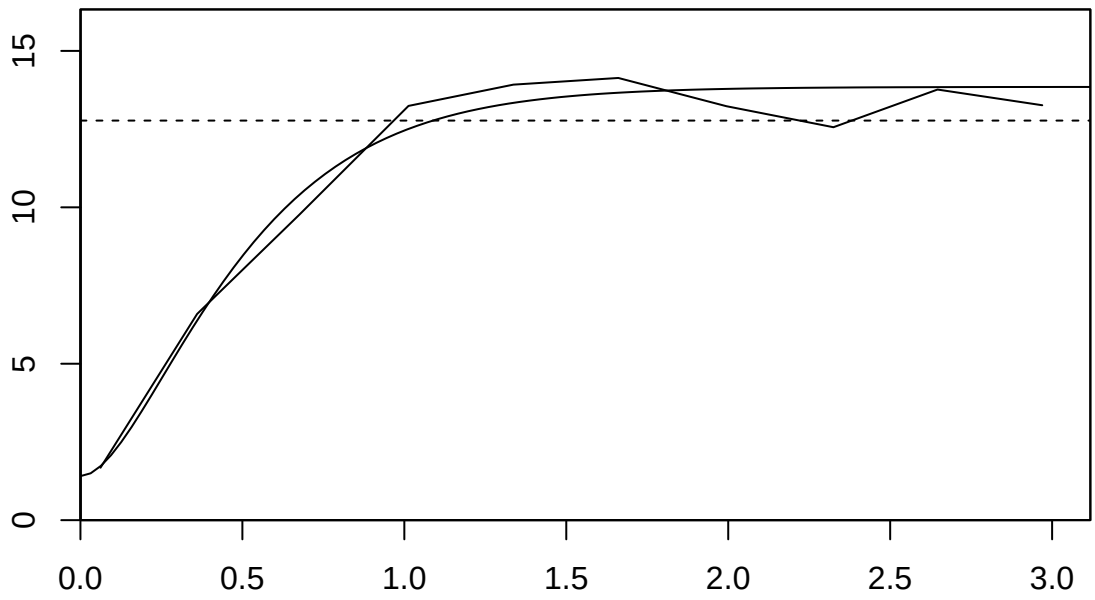
```
library(RGeostats)
jurarg = db.create(jura)
jurarg = db.locate(jurarg,c("Xloc","Yloc"),"x")
jurarg = db.locate(jurarg,"Co","z")
```

Compute the empirical variogram (with the function *vario.calc*)

```
v = vario.calc(jurarg,nlag=10)
```

Fit a model (with *model.auto*)

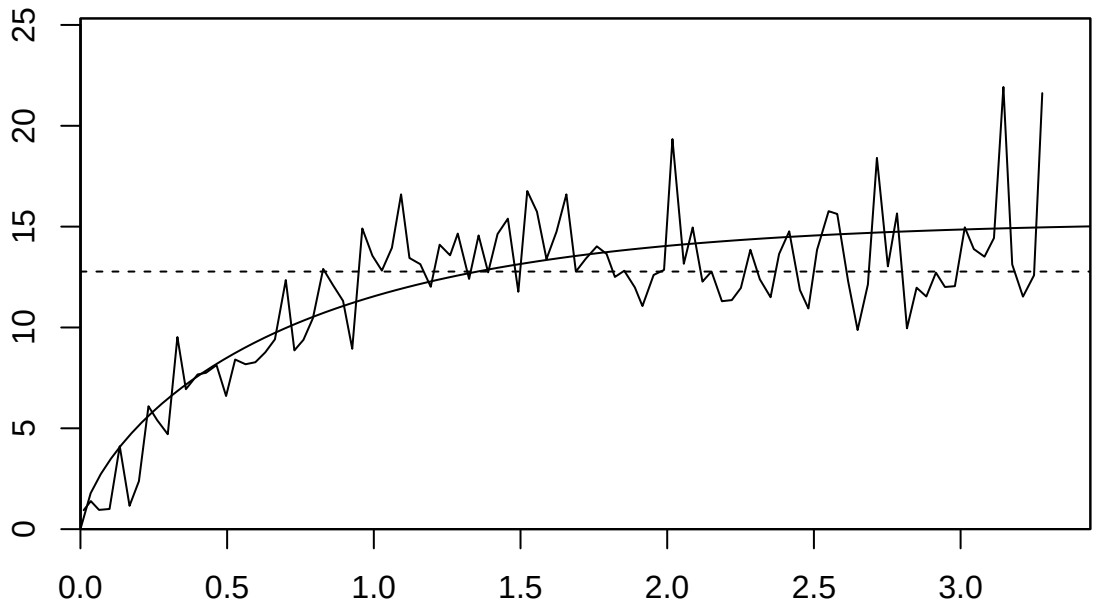
```
m = model.auto(v,struct = c(1,8))
```



It is intuitive to think that points close to each other (having a low lag) should have similar properties and hence assume close values , and this is further proved by the results of the simulation , when the lag is low (0.3 for the first point for example) the dissimilarity between the two points is low , the dissimilarities further increase when the lag increases. The solid smooth line is the fitted model generated using model.auto.

Let us now increase the value of the lag points to 100 instead of 10 , this will reduce the lag increment by 10 , we get the following results:

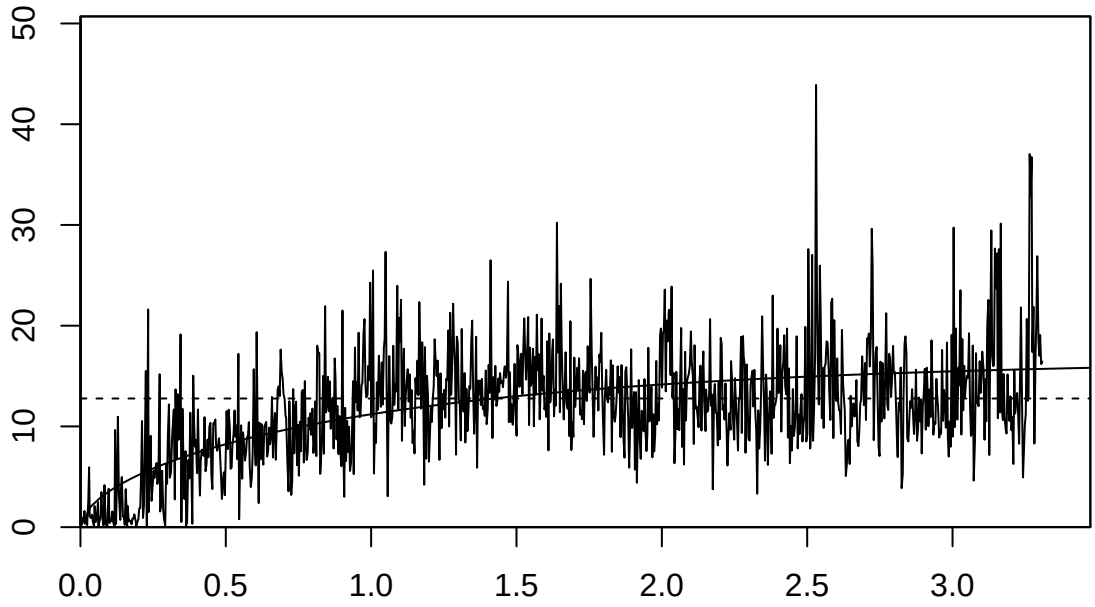
```
v = vario.calc(jurarg,nlag=100)
m = model.auto(v,struct = c(1,8))
```



Similarly we can also see at lower lag values the variogram produces smaller results indicated that close points have similar properties.

Let us now increase the value of the lag point to 1000 , we get the following results :

```
v = vario.calc(jurarg,nlag=1000)
m = model.auto(v,struct = c(1,8))
```

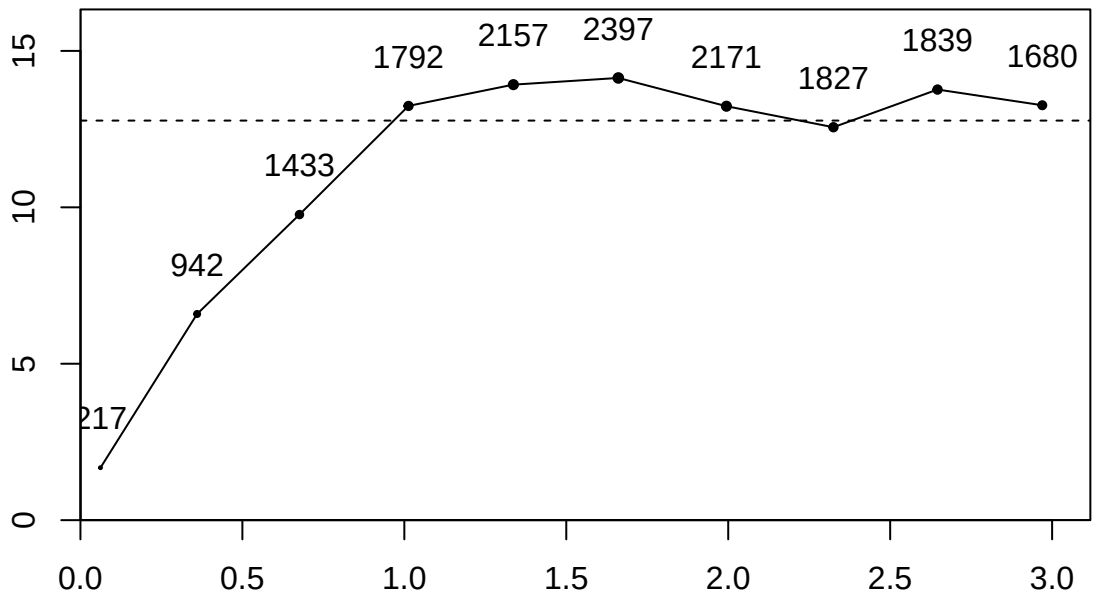


At lower lag levels the variogram produces very small values , at higher lag values the variogram produces higher values , however one can notice that beyond a certain point the variogram values do not change much that is Z(x+h) and Z(x) become uncorrelated from that point onward , we are lead to think that this variogram has a sill.

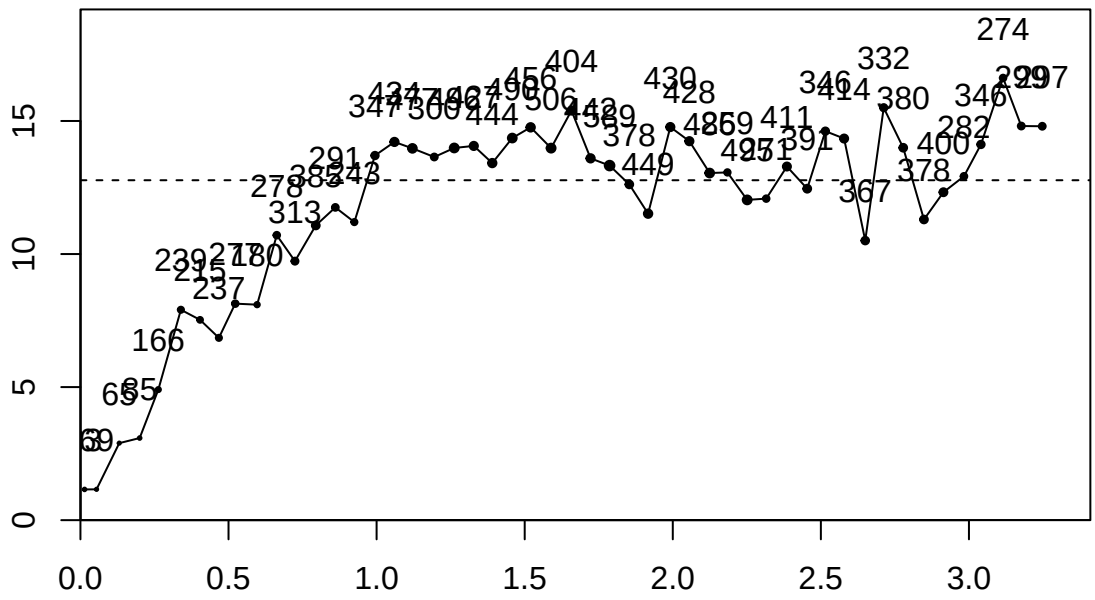
2. Print the number of pairs of points used to compute the variogram values (using the options `npairpt=TRUE,npairdw=TRUE` in `plot()`) for different values of lag and comment the results.

Now let us print the number of pairs of points used to represent these variograms , for a lag value of 10,50,100 respectively we get the following results :

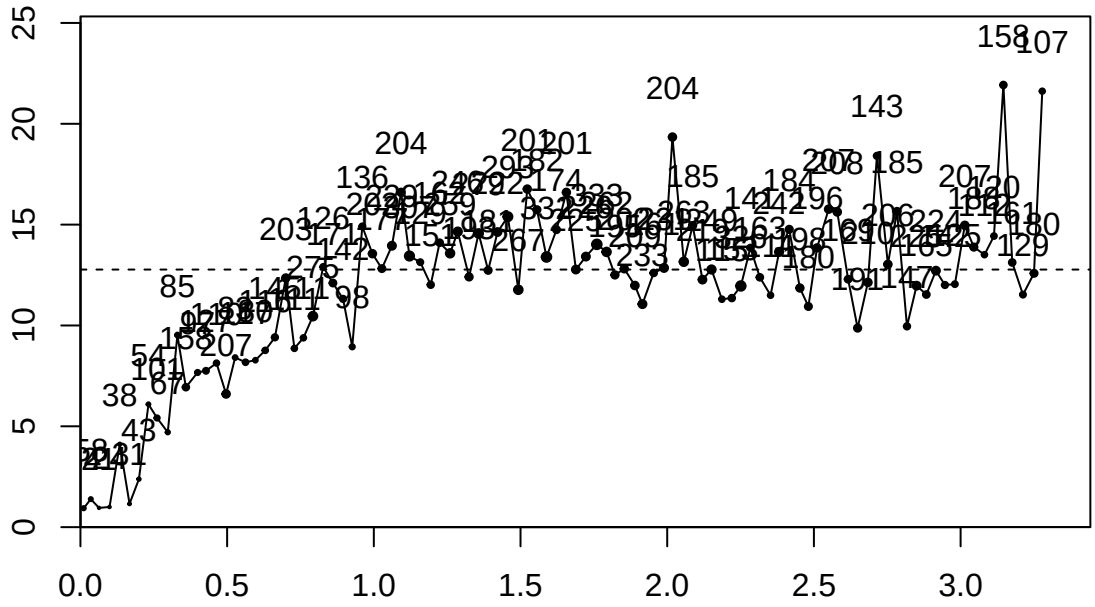
```
v = vario.calc(jurarg,nlag=10)
draw.vario(v,npairpt = T,npairdw = T)
```




```
v = vario.calc(jurarg,nlag=50)
draw.vario(v,npairpt = T,npairdw = T)
```




```
v = vario.calc(jurarg,nlag=100)
draw.vario(v,npairpt = T,npairdw = T)
```



As we can see from the simulation results when the lag h decreases (or respectively the number of lag points increase) the number of pairs decrease considerably , and this is obvious because when the lag h decreases there will be less and less points within that lag distance and hence the number of pairs of points within that lag h of each other should also decrease.

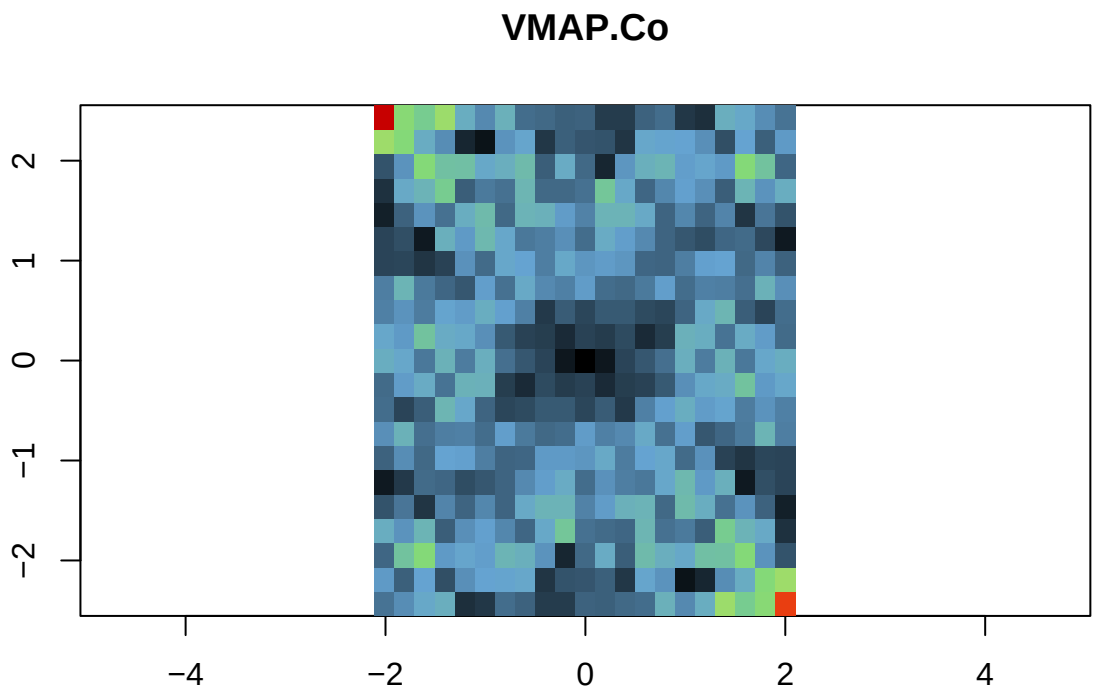
5.1.2 Experimental variogram (anisotropic case)

1. Compute and plot the variogram maps (using `vmap.calc()`) the Cobalt concentration order to check for anisotropies. Comment

So far we have only considered the variogram in the isotropic case in which the variogram does not vary with direction , it is only a function of the modulus of the lag h. We will now move on to the anisotropic case where not only the modulus of h is of importance but the direction of h is also important , to check for these anisotropies we will need to plot the variogram map of the cobalt concentration :

Compute a variogram map

```
plot(vmap.calc(jurarg))
```



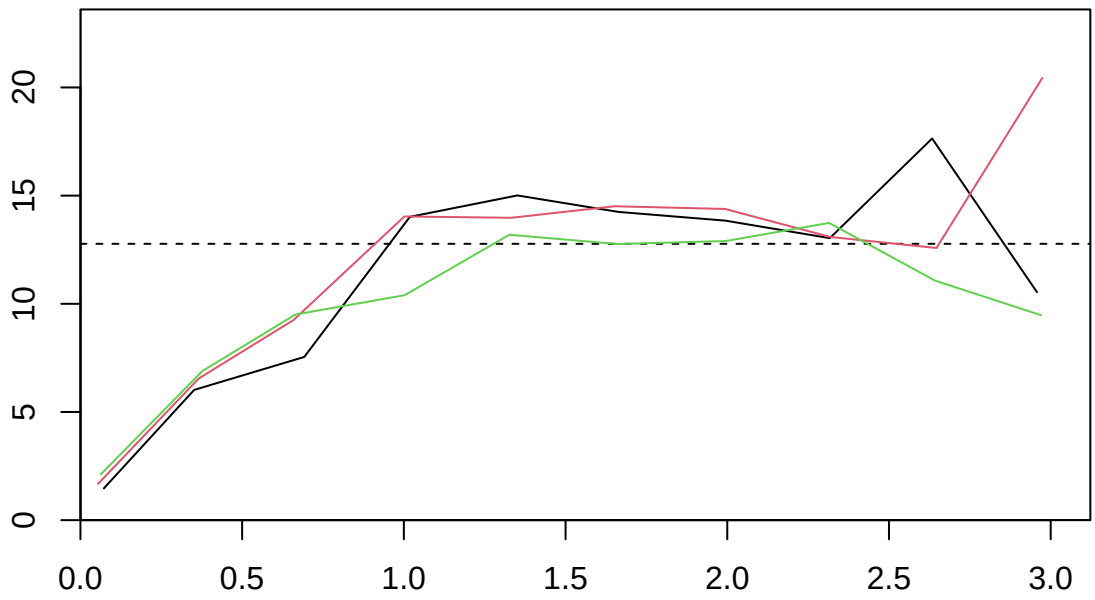
We can obviously see that the behavior of the lag in different directions is not the same as the colors in the variogram map tend to change with direction , for example in a -45 degrees direction the dissimilarities between Z(x+h) and Z(x) is way higher than in a 0 degrees direction and this will be shown in the upcoming variograms.

2. Compute and plot directional variograms (according to the anisotropy directions determined with the maps).

We will now plot the variograms for different directions namely 0 , 45 and -45 degrees:

Directional variograms

```
vdir = vario.calc(jurarg,nlag=10,dir=c(0,-45,45))
plot(vdir)
```

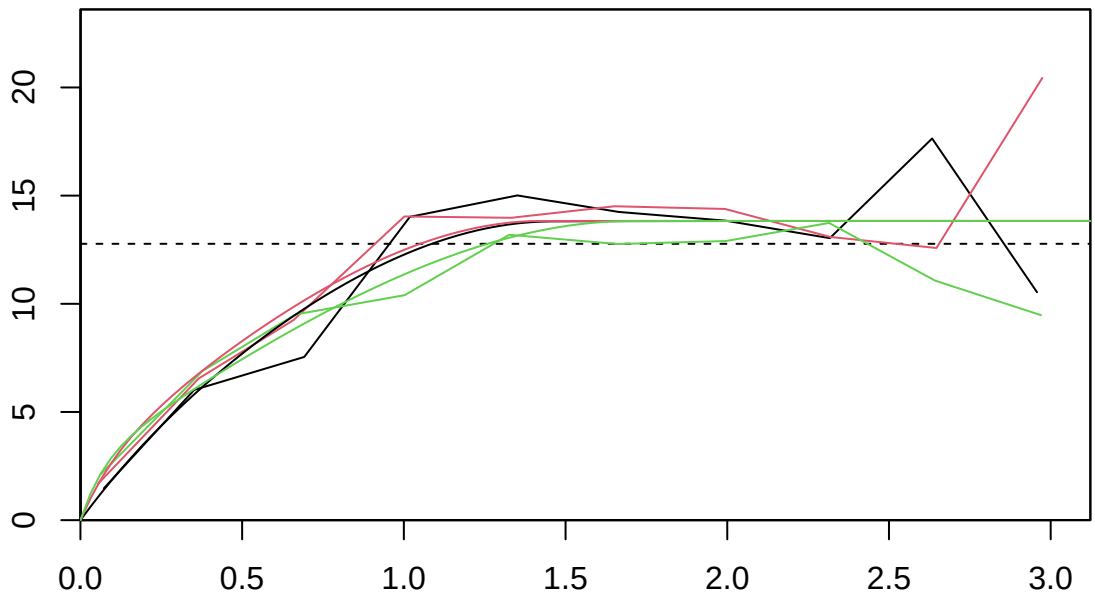


The black curve computes the variogram in the 0 degrees direction, the red curve computes it in the -45 degrees direction and the green curve in the 45 degrees direction , the behavior for these 3 curves is quite similar when the lag is small , however when the lag increases we can see that the difference between Z(x+h) and Z(x) is way higher for the red curve (+45 degrees direction) however this is not enough to say the data is anisotropic.

If we try to fit a model for these curves we obtain:

Fit a model

```
maniso = model.auto(vdir,struct=c(1,2,3,4))
```



The model fit suggests that the sill is constant however the range for which the sill is obtained is different for the three curves and hence varies with direction suggesting that there may be geometric anisotropies but no zonal anisotropies.

5.1.3 Model adjustement

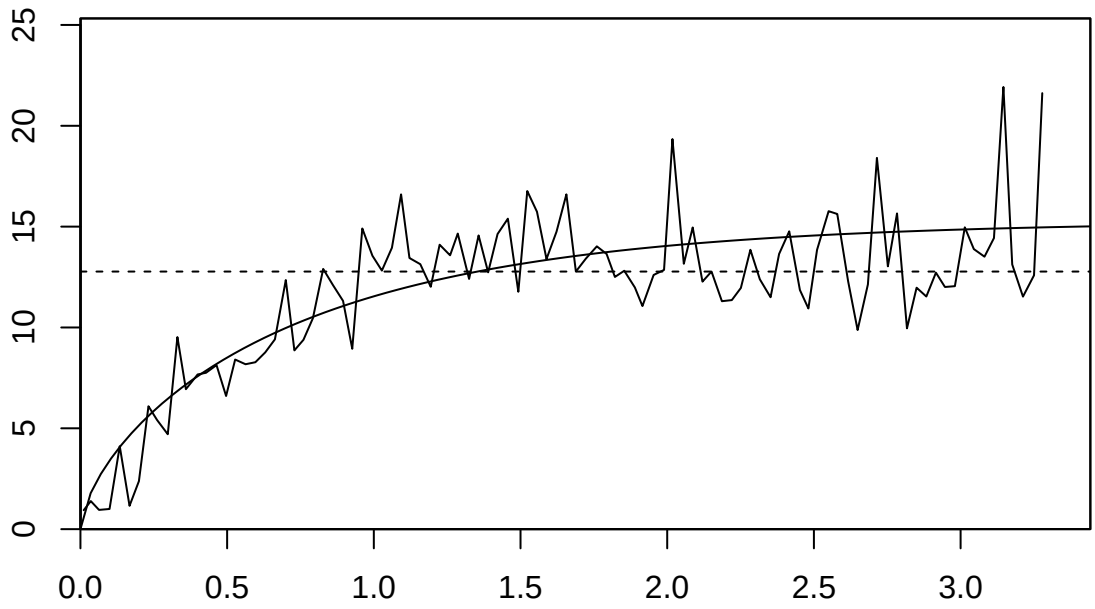
The function `melem.name()` gives the models available in RGeostats.

1. Adjust a model using the function `model.auto()` (isotropic and anisotropic cases) on experimental variograms and print the model caracteristics.
2. Try imposing different structures or combinations of structures.

Let us first try different models for the isotropic case

K-Bessel function with nugget effect

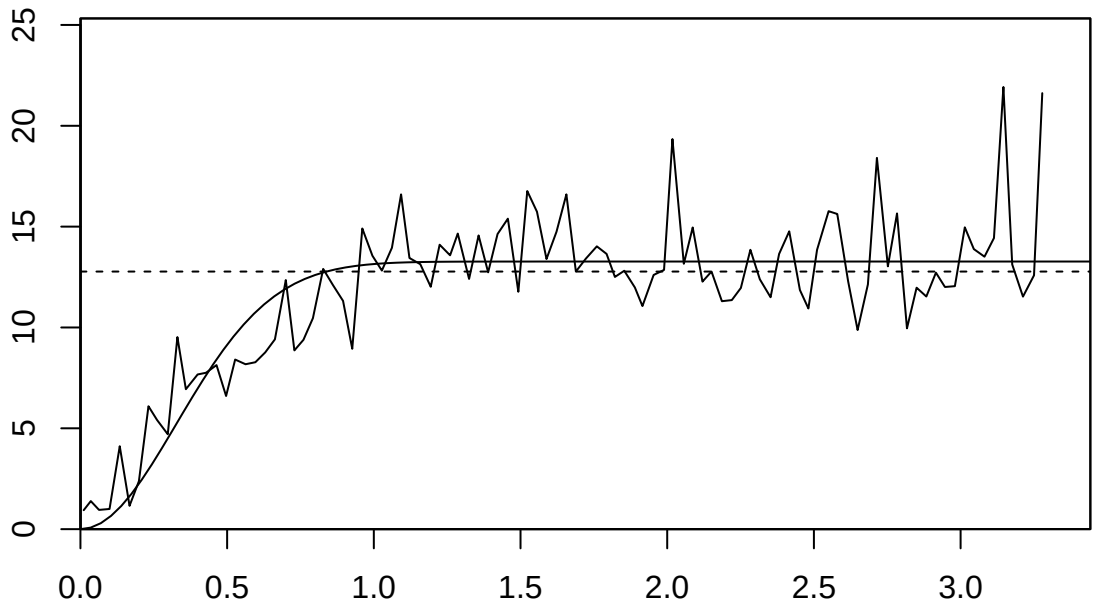
```
model.auto(v,struct=c(1,8))
```




```
##
## Model characteristics
## =====================
## Space dimension              = 2
## Number of variable(s)        = 1
## Number of basic structure(s) = 1
## Number of drift function(s)  = 1
## Number of drift equation(s)  = 1
##
## Covariance Part
## ---------------
## K-Bessel (Third Parameter = 0.312472)
## - Sill         =     15.278
## - Range        =      1.931
## - Theo. Range  =      0.997
## Total Sill     =     15.278
##
## Drift Part
## ----------
## Universality Condition
```

Gaussian function with nugget effect

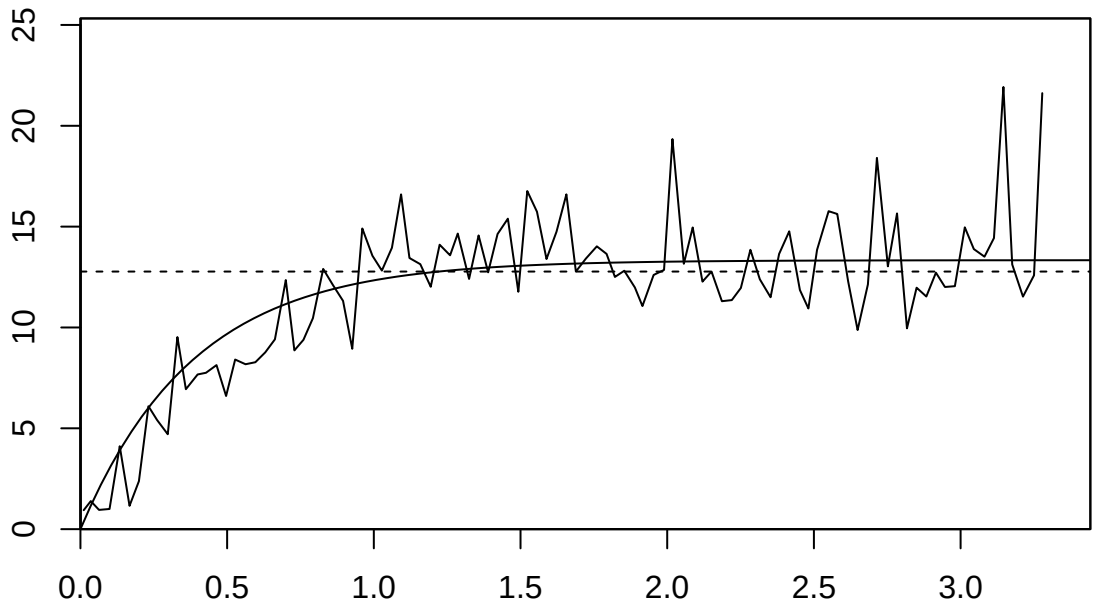
```
model.auto(v,struct=c(1,4))
```




```
##
## Model characteristics
## =====================
## Space dimension              = 2
## Number of variable(s)        = 1
## Number of basic structure(s) = 1
## Number of drift function(s)  = 1
## Number of drift equation(s)  = 1
##
## Covariance Part
## ---------------
## Gaussian
## - Sill         =     13.269
## - Range        =      0.801
## - Theo. Range  =      0.463
## Total Sill     =     13.269
##
## Drift Part
## ----------
## Universality Condition
```

Exponential function with nugget effect

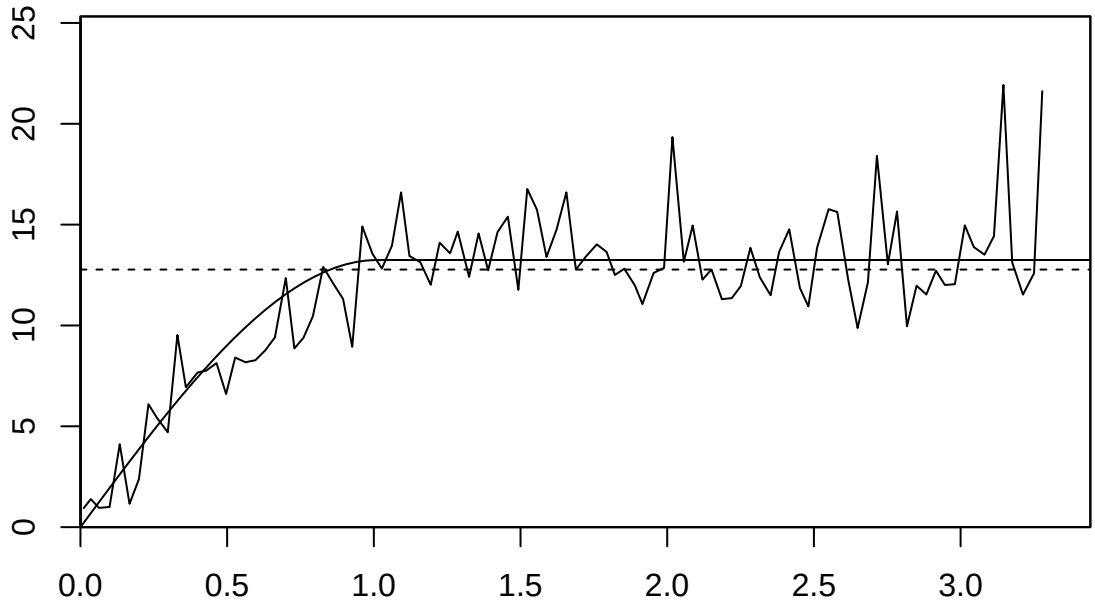
```
model.auto(v,struct=c(1,2))
```




```
##
## Model characteristics
## =====================
## Space dimension              = 2
## Number of variable(s)        = 1
## Number of basic structure(s) = 1
## Number of drift function(s)  = 1
## Number of drift equation(s)  = 1
##
## Covariance Part
## ---------------
## Exponential
## - Sill         =     13.336
## - Range        =      1.156
## - Theo. Range  =      0.386
## Total Sill     =     13.336
##
## Drift Part
## ----------
## Universality Condition
```

Exponential,spherical,K-Bessel function with nugget effect

```
model.auto(v,struct=c(1,2,3,8))
```

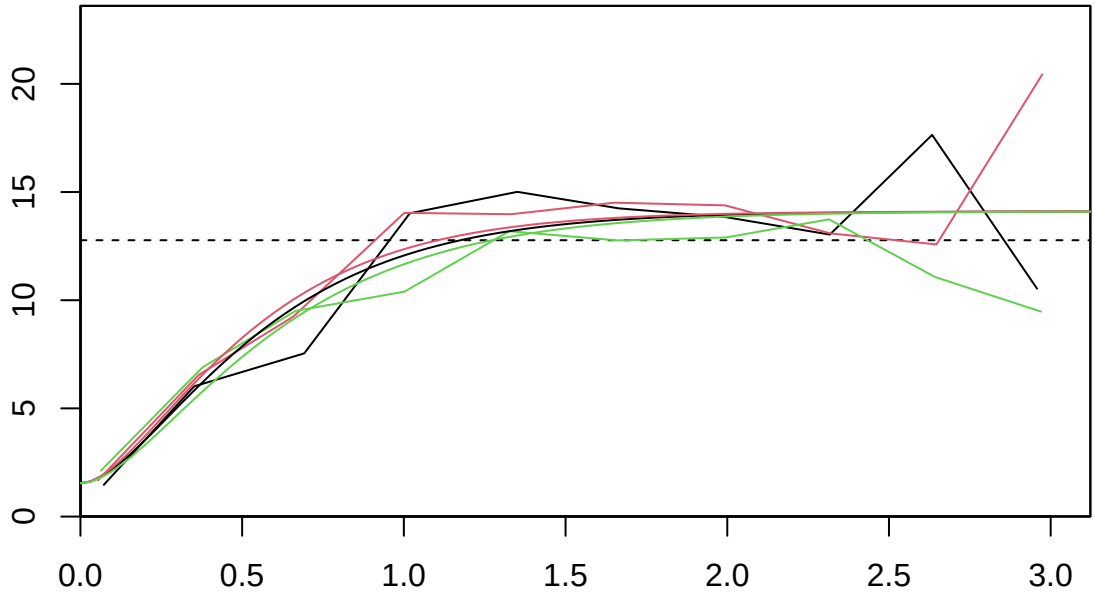



```
##
## Model characteristics
## =====================
## Space dimension              = 2
## Number of variable(s)        = 1
## Number of basic structure(s) = 1
## Number of drift function(s)  = 1
## Number of drift equation(s)  = 1
##
## Covariance Part
## ---------------
## Spherical
## - Sill         =     13.247
## - Range        =      1.012
## Total Sill     =     13.247
##
## Drift Part
## ----------
## Universality Condition
```

Let us try the same models for the anisotropic case

K-Bessel function with nugget effect

```
model.auto(vdir,struct=c(1,8))
```

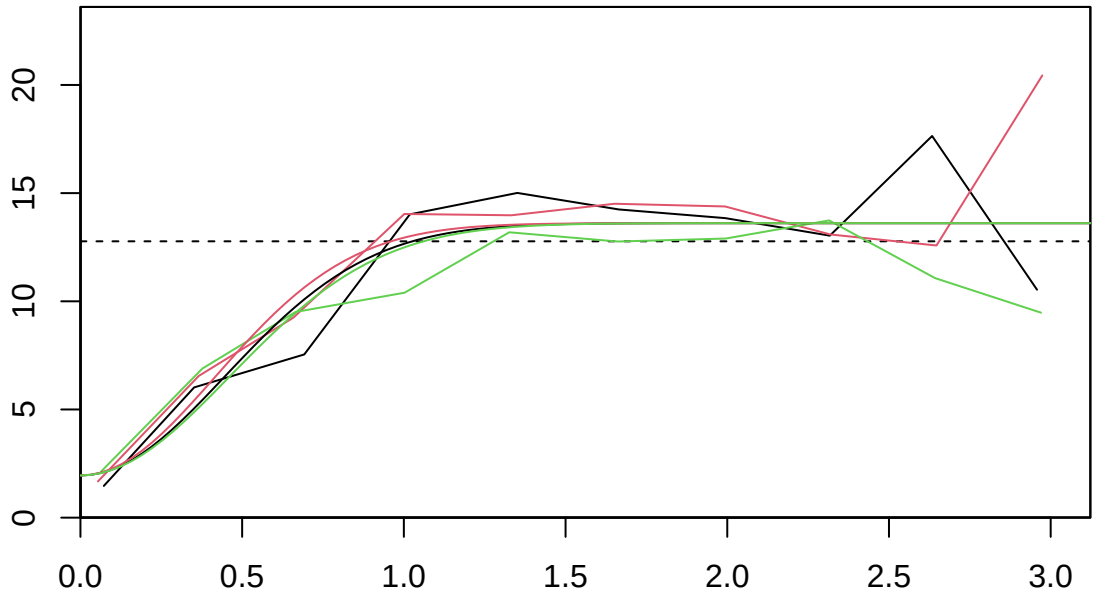



```
##
## Model characteristics
## =====================
## Space dimension              = 2
## Number of variable(s)        = 1
## Number of basic structure(s) = 2
## Number of drift function(s)  = 1
## Number of drift equation(s)  = 1
##
## Covariance Part
## ---------------
## Nugget Effect
## - Sill         =      1.537
## K-Bessel (Third Parameter = 1.24012)
## - Sill         =     12.572
## - Ranges       =      1.409     1.223
## - Theo. Ranges =      0.365     0.317
## - Angles       =   -133.423     0.000
## - Rotation Matrix
##                [,  1]    [,  2]
##      [  1,]    -0.687     0.726
##      [  2,]    -0.726    -0.687
## Total Sill     =     14.109
```

```
##
## Drift Part
## ----------
## Universality Condition
```

Gaussian function with nugget effect

```
model.auto(vdir,struct=c(1,4))
```




```
##
## Model characteristics
## =====================
## Space dimension              = 2
## Number of variable(s)        = 1
## Number of basic structure(s) = 2
## Number of drift function(s)  = 1
## Number of drift equation(s)  = 1
##
## Covariance Part
## ---------------
## Nugget Effect
## - Sill         =      1.950
## Gaussian
## - Sill         =     11.662
## - Ranges       =      1.136     1.021
```

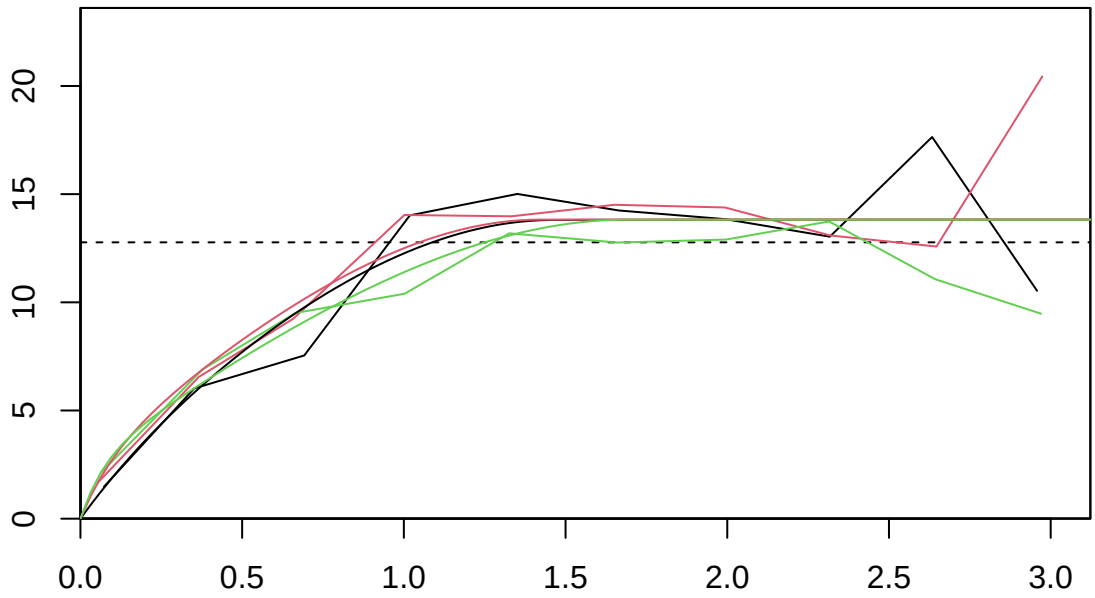
```
## - Theo. Ranges =      0.656     0.590
## - Angles       =     34.739     0.000
## - Rotation Matrix
##                [,  1]    [,  2]
##      [  1,]     0.822    -0.570
##      [  2,]     0.570     0.822
## Total Sill     =     13.612
##
## Drift Part
## ----------
## Universality Condition
```

Exponential function with nugget effect

```
model.auto(vdir,struct=c(1,2))
```

Exponential,spherical,K-Bessel function with nugget effect

```
model.auto(vdir,struct=c(1,2,3,8))
```



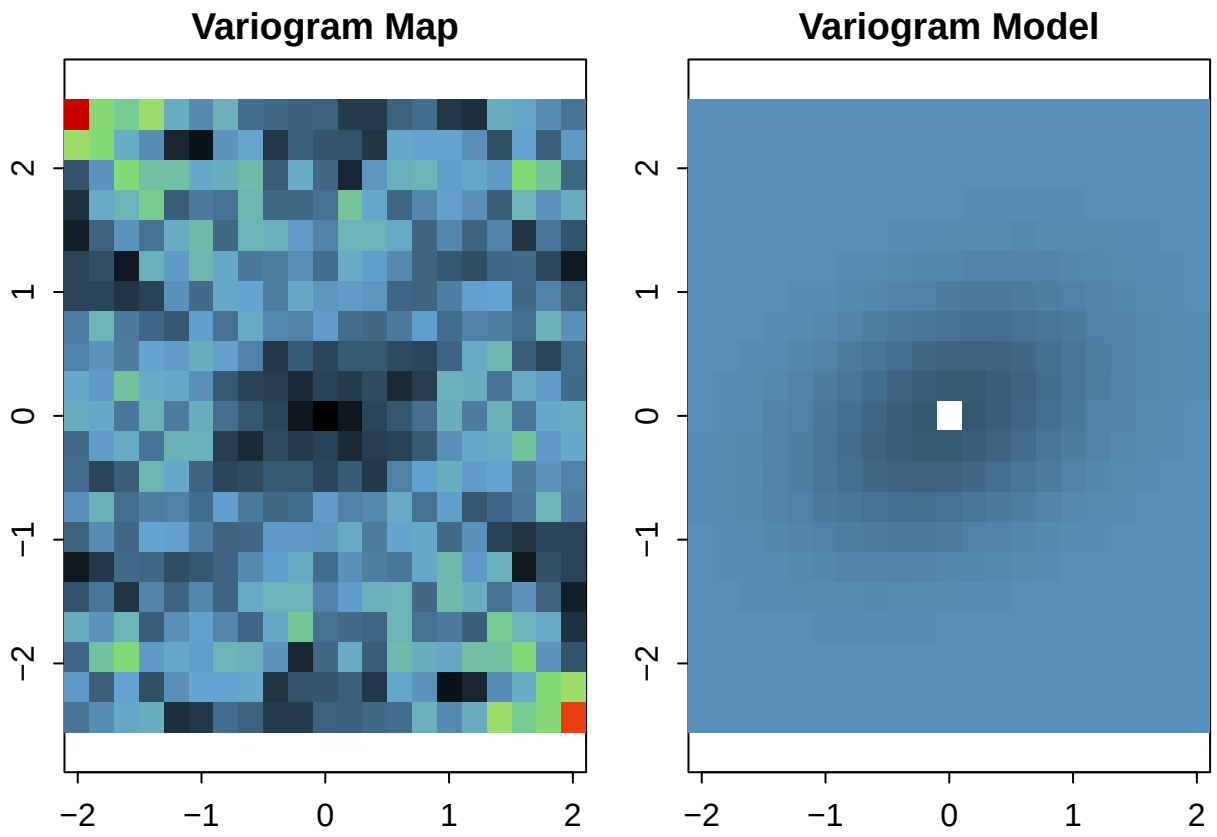

```
##
## Model characteristics
## =====================
## Space dimension              = 2
## Number of variable(s)        = 1
```

```
## Number of basic structure(s) = 2
## Number of drift function(s)  = 1
## Number of drift equation(s)  = 1
##
## Covariance Part
## ---------------
## Exponential
## - Sill         =      2.601
## - Ranges       =      1.686     0.182
## - Theo. Ranges =      0.563     0.061
## - Angles       =    -10.659     0.000
## - Rotation Matrix
##                [,  1]    [,  2]
##      [  1,]     0.983     0.185
##      [  2,]    -0.185     0.983
## Spherical
## - Sill         =     11.222
## - Ranges       =      1.399     1.728
## - Angles       =    151.512     0.000
## - Rotation Matrix
##                [,  1]    [,  2]
##      [  1,]    -0.879    -0.477
##      [  2,]     0.477    -0.879
## Total Sill     =     13.822
##
## Drift Part
## ----------
## Universality Condition
```

3. Compare the models adjusted on the experimental variogram and the variogram map (using `vmap.auto()`).

```
vmap.auto(vmap.calc(jurarg),struct=c(1,8))
```

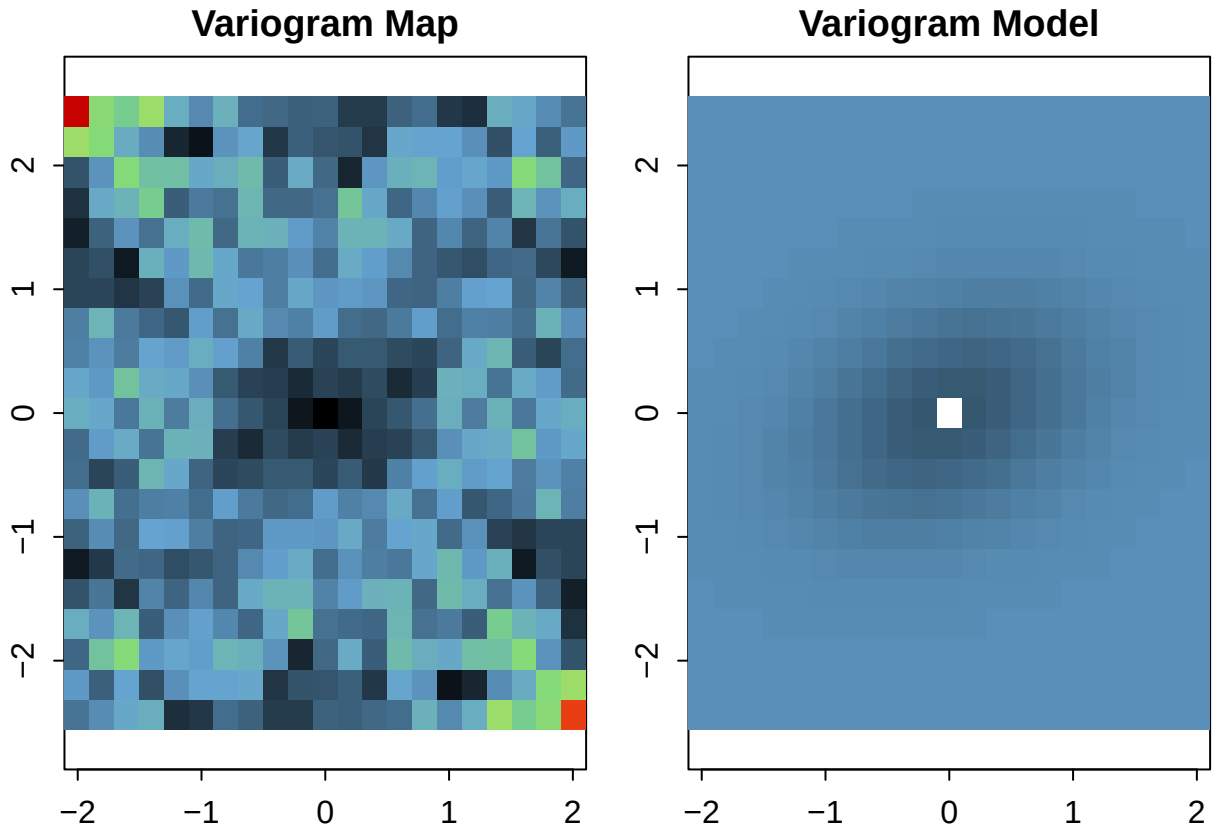
```
## Convergence not reached after 100 iterations (4 parameters)
```




```
##
## Model characteristics
## =====================
## Space dimension              = 2
## Number of variable(s)        = 1
## Number of basic structure(s) = 2
## Number of drift function(s)  = 1
## Number of drift equation(s)  = 1
##
## Covariance Part
## ---------------
## Nugget Effect
## - Sill         =      8.830
## K-Bessel (Third Parameter = 89.8687)
## - Sill         =      5.253
## - Ranges       =      2.209     1.503
## - Theo. Ranges =      0.067     0.046
## - Angles       =     33.209     0.000
## - Rotation Matrix
##                [,  1]    [,  2]
##      [  1,]     0.837    -0.548
##      [  2,]     0.548     0.837
## Total Sill     =     14.083
##
## Drift Part
## ----------
```

Universality Condition

```
vmap.auto(vmap.calc(jurarg),struct=c(1,4))
```

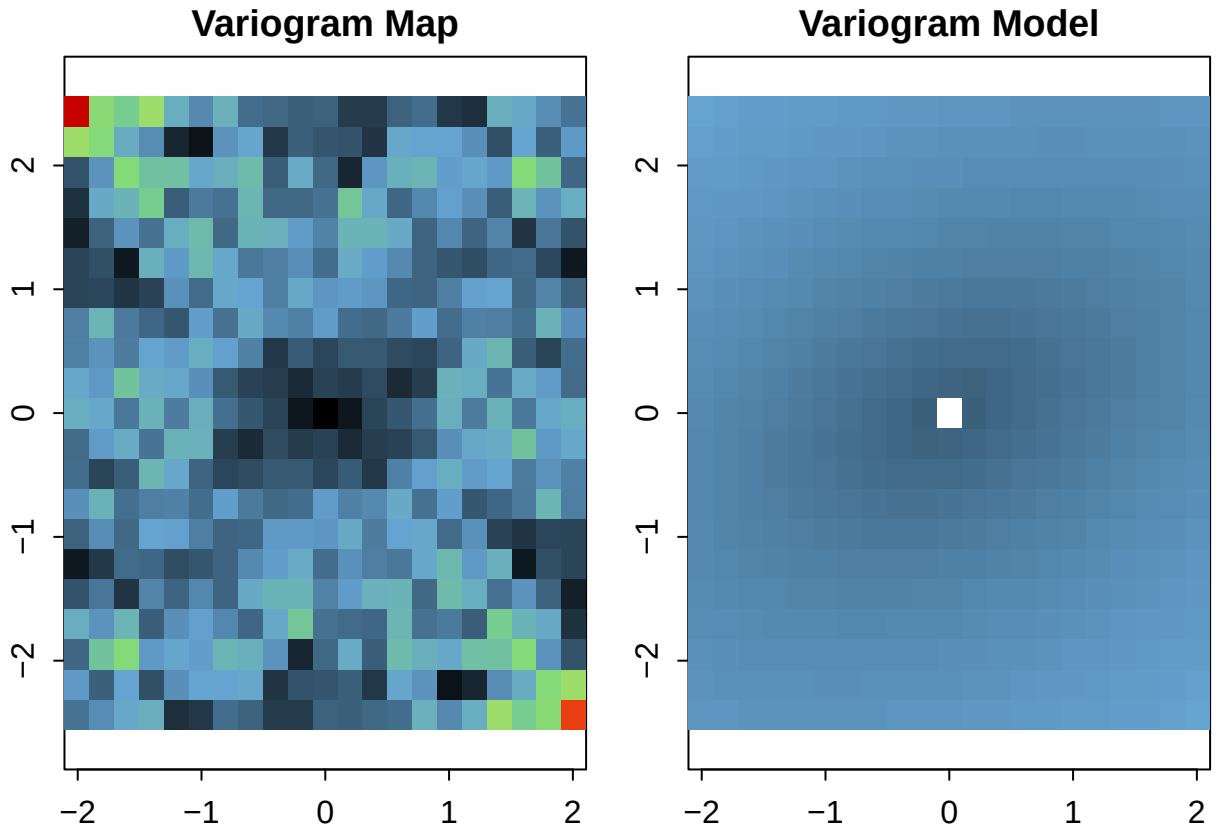



```
##
## Model characteristics
## =====================
## Space dimension              = 2
## Number of variable(s)        = 1
## Number of basic structure(s) = 2
## Number of drift function(s)  = 1
## Number of drift equation(s)  = 1
##
## Covariance Part
## ---------------
## Nugget Effect
## - Sill         =      8.783
## Gaussian
## - Sill         =      5.117
## - Ranges       =      1.384     1.976
## - Theo. Ranges =      0.800     1.141
## - Angles       =    117.681     0.000
## - Rotation Matrix
##                [,  1]    [,  2]
##      [  1,]    -0.465    -0.886
```

```
##      [  2,]     0.886    -0.465
## Total Sill     =     13.900
##
## Drift Part
## ----------
## Universality Condition
```

```
vmap.auto(vmap.calc(jurarg),struct=c(1,2))
```

```
## Convergence not reached after 100 iterations (3 parameters)
```

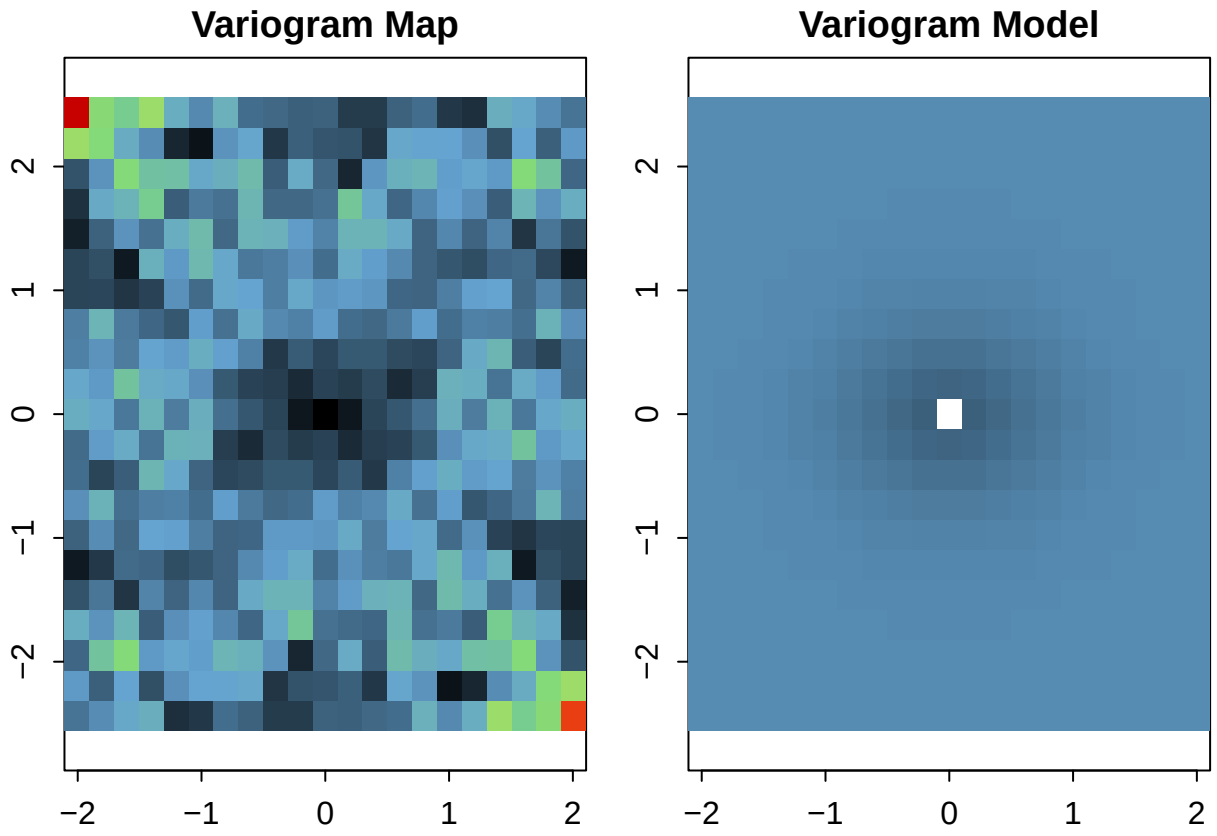



```
##
## Model characteristics
## =====================
## Space dimension              = 2
## Number of variable(s)        = 1
## Number of basic structure(s) = 2
## Number of drift function(s)  = 1
## Number of drift equation(s)  = 1
##
## Covariance Part
## ---------------
## Nugget Effect
## - Sill         =      9.197
```

```
## Exponential
## - Sill          =      8.660
## - Ranges        =      7.004    10.546
## - Theo. Ranges =      2.338     3.520
## - Angles        =    -59.190     0.000
## - Rotation Matrix
##                [,  1]    [,  2]
##      [  1,]     0.512     0.859
##      [  2,]    -0.859     0.512
## Total Sill      =     17.857
##
## Drift Part
## ----------
## Universality Condition
```

```
vmap.auto(vmap.calc(jurarg),struct=c(1,2,3,8))
```

```
## Convergence not reached after 100 iterations (10 parameters)
```




```
##
## Model characteristics
## =====================
## Space dimension              = 2
## Number of variable(s)        = 1
```

```
## Number of basic structure(s) = 4
## Number of drift function(s)  = 1
## Number of drift equation(s)  = 1
##
## Covariance Part
## ---------------
## Nugget Effect
## - Sill         =      8.693
## Exponential
## - Sill         =      2.040
## - Range        =      1.577
## - Theo. Range  =      0.526
## Spherical
## - Sill         =      1.320
## - Ranges       =      2.103     1.051
## K-Bessel (Third Parameter = 12.4364)
## - Sill         =      1.517
## - Range        =      1.577
## - Theo. Range  =      0.129
## Total Sill     =     13.570
##
## Drift Part
## ----------
## Universality Condition
```

5.2 Prediction

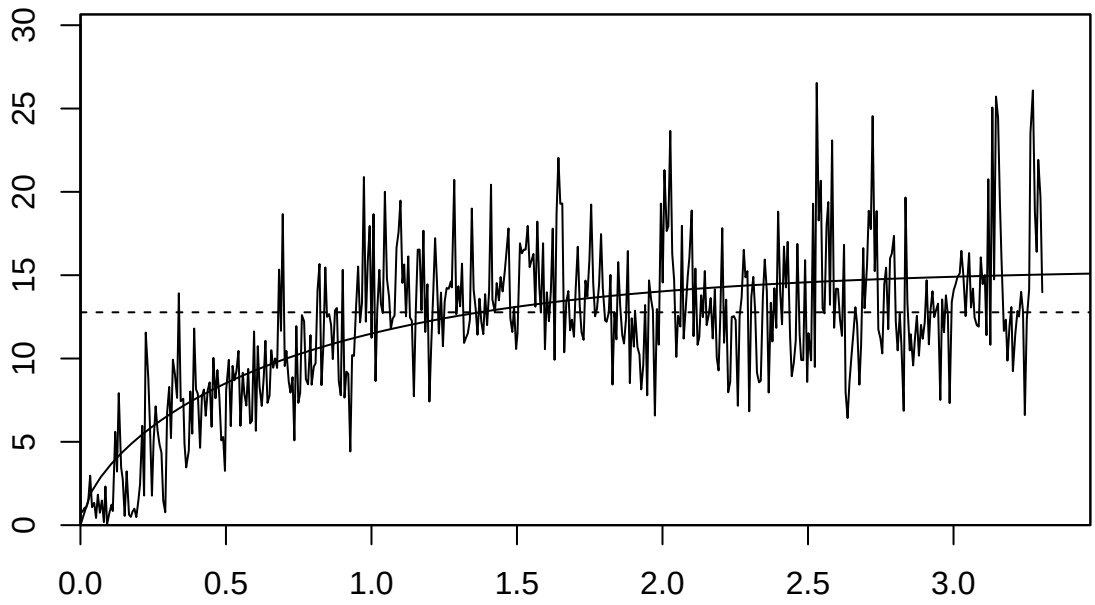
5.2.1 Ordinary Kriging

1. Compute and plot the ordinary kriging of the cobalt over the prediction grid. Plot the associated standard deviation map.
2. Try several variogram models (basic structures, anisotropy), and neighborood options. Compute the prediction scores. Comment the results

We will now move on to estimating the cobalt concentrations of the test results using the ordinary kriging method , let us state first that the ordinary kriging method is an interpolation method that assumes that the mean is unknown but constant, it uses the variogram as the basis.

We first have to create the target grid using the following code :

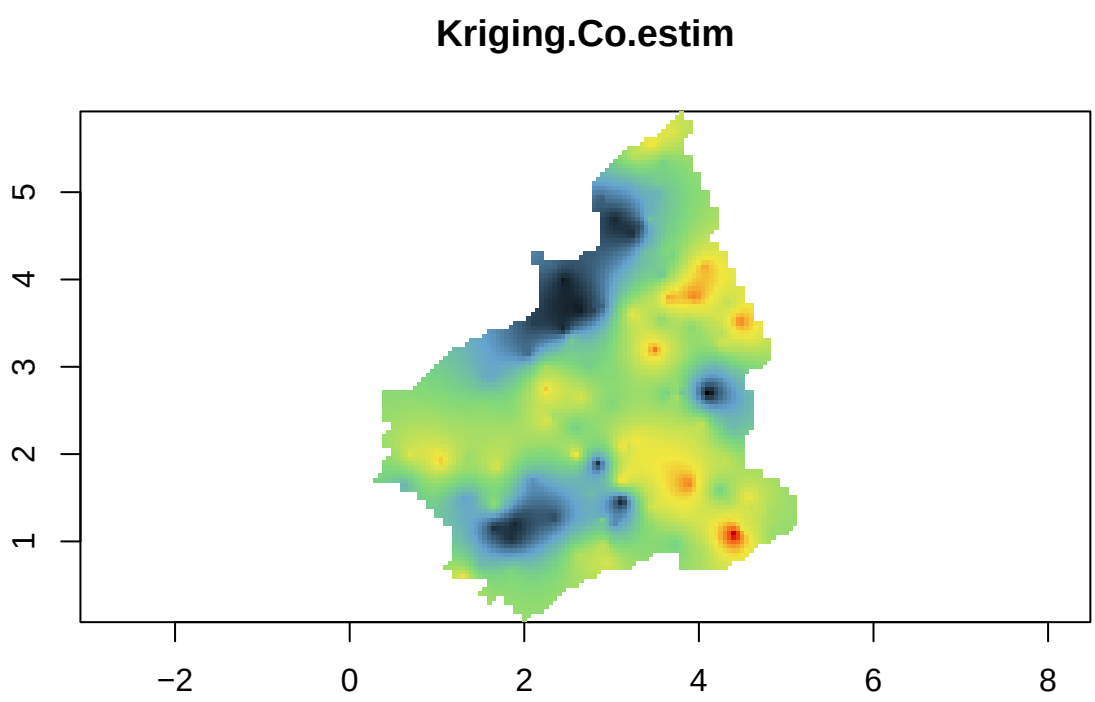
```
jurarg = db.create(jura)
jurarg = db.locate(jurarg,c("Xloc","Yloc"),"x")
jurarg = db.locate(jurarg,"Co","z")

v = vario.calc(jurarg,nlag=500)

m = model.auto(v,struct = c(1,8))
```




```
grid = read.csv("../../jura/jura_grid.csv")
gridtemp = db.create(grid)
gridtemp = db.locate(gridtemp,c("Xloc","Yloc"),"x")
nx = c(length(unique(grid[,1])),length(unique(grid[,2])))
gridrg = db.grid.init(gridtemp,nodes=nx)
gridrg = migrate(gridtemp,gridrg,names=4:gridtemp$natt,radix="")
gridrg = db.rename(gridrg,2:3,c("Xloc","Yloc"))
gridrg = db.sel(gridrg,!is.na(gridrg[,"Landuse"])&!is.na(gridrg[,"Rock"]))
```

Next we will have to apply the ordinary kriging on the grid we have just created and plot the result:

```
neigh = neigh.create(type = 0)
res=kriging(jurarg,gridrg,m,neigh)
plot(res)
```



This plot shows us the different possible estimated cobalt concentrations based on the Xloc on the horizontal axis and the Yloc on the vertical axis , the red color indicating high cobalt concentration , the blue color indicating lower cobalt concentration while the green and yellow colors are inbetween.

Next we can apply the ordinary kriging on the validation set using the following code:

```
neigh = neigh.create(nmini=10,nmaxi=30,radius=1)
val_locrg = db.create(val_loc)
val_locrg = db.locate(val_locrg,c("Xloc","Yloc"),"x")
res_val=kriging(jurarg,val_locrg,m,neigh)
mean((res_val[,"Kriging*estim"]-val[,2])^2)
```

```
## [1] 7.176295
```

The neighborhood we took into consideration here is a moving neighborhood with 10 minimum points in the neighborhood and 30 maximum points in the neighborhood.

We get a prediction score of : 7.176295

As an attempt to improve this prediction score we will try to use different combinations of basic structures, let us first try using the sector option in the neighborhood:

```
neigh = neigh.create(nmini=10,nmaxi=30,radius=3,flag.sector = T,nsect=12,nsmax=30)
val_locrg = db.create(val_loc)
val_locrg = db.locate(val_locrg,c("Xloc","Yloc"),"x")
res_val=kriging(jurarg,val_locrg,m,neigh)
mean((res_val[,"Kriging*estim"]-val[,2])^2)
```

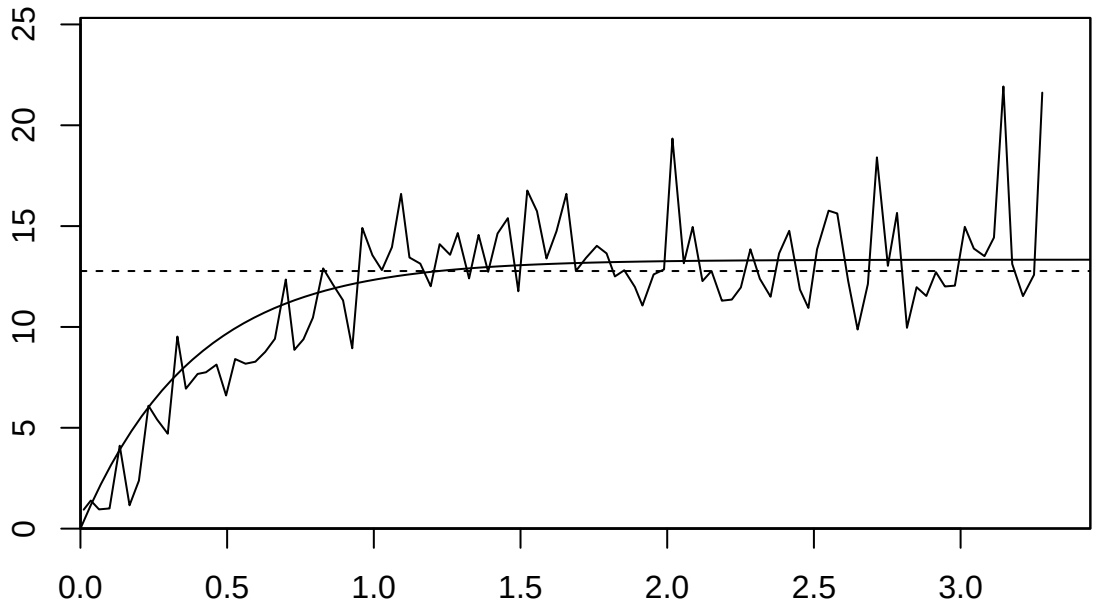
```
## [1] 7.146272
```

This gives us a prediction score of : 7.146272 which is an improvement of the previous score.

Let us attempt an anisotropic model as opposed to the previous isotropic model , for this purpose the radius can no longer be constant but has to be a vector equal to the dimension we are working with. In this case the dimension is equal to 2 since we are only working alongside 2 parameters the Xloc and Yloc. We will try an ellipsoid c(3,3.5) and rotating this ellipsoid a -45 degrees angle , the code is as follows :

Ordinary kriging on the validation set anisotropic case

```
v = vario.calc(jurarg,nlag=100)
m = model.auto(v,struct = c(1,2))
```




```
radvec1=c(3,3.5)
angle=-45
rotmat1 = matrix(
  c(cos(angle), -sin(angle), sin(angle), cos(angle)),
  nrow = 2,
  ncol = 2,
  byrow = TRUE
)
neigh=neigh.create(type=2,nmini=10,nmaxi=30,radius=radvec1,flag.sector=T,flag.aniso=T,nsect=10,nsmax=30
neigh = neigh.create(nmini=10,nmaxi=30,radius=1)
val_locrg = db.create(val_loc)
val_locrg = db.locate(val_locrg,c("Xloc","Yloc"),"x")
```

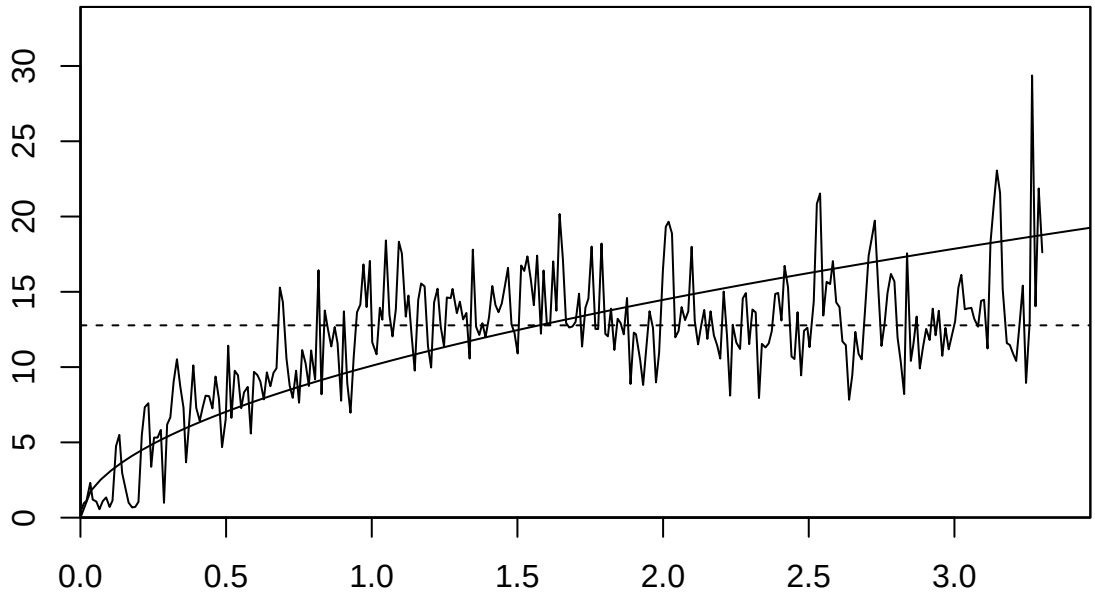
```
res_val=kriging(jurarg,val_locrg,m,neigh)
mean((res_val[,"Kriging*estim"]-val[,2])^2)
```

```
## [1] 7.942153
```

The prediction score we got was : 7.942153 which is worse than the isotropic case.

We will try right now changing the structure , the structure we have used so far c(1,8) takes into account the nugget effect and generates a K-bessel model , we will now attempt a power model incorporating the nugget effect c(1,13) :

```
v = vario.calc(jurarg,nlag=300)
m = model.auto(v,struct = c(1,13))
```



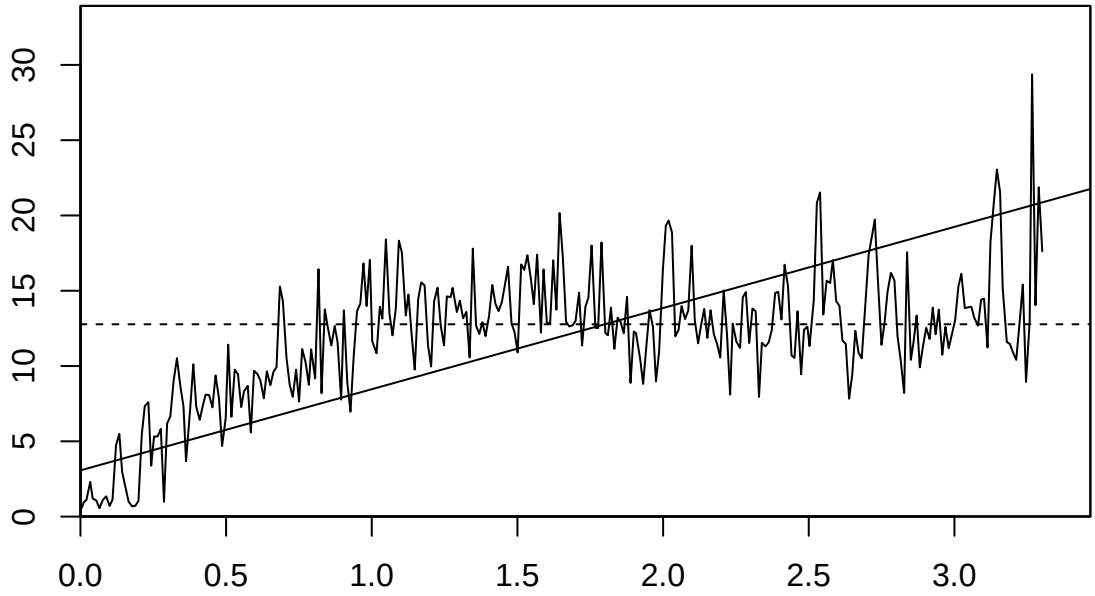

```
neigh = neigh.create(nmini=10,nmaxi=30,radius=2.5)
val_locrg = db.create(val_loc)
val_locrg = db.locate(val_locrg,c("Xloc","Yloc"),"x")
res_val=kriging(jurarg,val_locrg,m,neigh)
mean((res_val[,"Kriging*estim"]-val[,2])^2)
```

```
## [1] 7.011163
```

The prediction score we get is : 7.011163 which is an improvement to the previous score.

Let us now try the order 1 G-C model , using :

```
v = vario.calc(jurarg,nlag=300)
m = model.auto(v,struct = c(1,14))
```



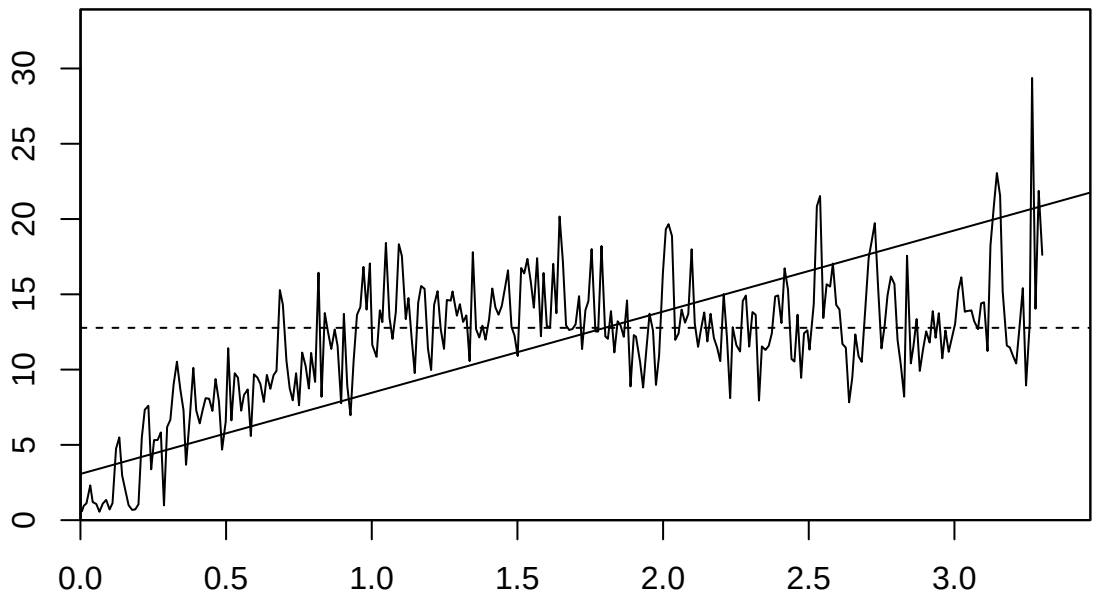

```
neigh = neigh.create(nmini=10,nmaxi=30,radius=2.5)
val_locrg = db.create(val_loc)
val_locrg = db.locate(val_locrg,c("Xloc","Yloc"),"x")
res_val=kriging(jurarg,val_locrg,m,neigh)
mean((res_val[,"Kriging*estim"]-val[,2])^2)
```

```
## [1] 6.783702
```

The prediction score we get is : 6.783702 which is a huge improvement , we could have also done a combination of models such as c(1,8,2,5).

We could also try changing the type of the neighborhood for example :

```
v = vario.calc(jurarg,nlag=300)
m = model.auto(v,struct = c(1,14))
```




```
neigh = neigh.create(type=0)
val_locrg = db.create(val_loc)
val_locrg = db.locate(val_locrg,c("Xloc","Yloc"),"x")
res_val=kriging(jurarg,val_locrg,m,neigh)
mean((res_val[,"Kriging*estim"]-val[,2])^2)
```

```
## [1] 6.838635
```

However this gives a prediction score of : 6.838635 which is worse than when we used the moving neighborhood.

.

5.2.2 Universal kriging

Use the indicators of the different levels of the factors (*Rock* and *Landuse*) as covariates to compute the universal kriging prediction.

Transformation of the Rock factor into indicators

```
indiccut = limits.create(mini=c(1,2,3),maxi=c(2,3,4))

jurarg_KU=db.indicator(jurarg,indiccut,name="Rock")
jurarg_KU = db.locate(jurarg_KU,"Indicator*","f")
jurarg_KU = db.locate(jurarg_KU,"Co","z")
```

```
gridrg_KU=db.indicator(gridrg,indiccut,name="Rock")
gridrg_KU = db.locate(gridrg_KU,"Indicator*","f")
gridrg_KU = db.locate(gridrg_KU,"Co","z")

val_locrg_KU=db.indicator(val_locrg,indiccut,name="Rock")
val_locrg_KU = db.locate(val_locrg_KU,"Indicator*","f")
val_locrg_KU = db.locate(val_locrg_KU,"Co","z")
```

Variogram of the residuals

```
drift = c("1","f1","f2","f3")
vres = vario.calc(jurarg_KU,nlag=10,uc=drift)
plot(v)
plot(vres,add=T,col=2)
mres=model.auto(vres,struct=c(1,2))
```

Universal kriging on the grid

```
neigh=neigh.create(type=0)
res_gridKU=kriging(jurarg_KU,gridrg_KU,mres,neigh,uc=drift)
plot(res_gridKU)
```

Plot the associated standard deviation map.

Universal kriging on the validation set and prediction score

```
res_valKU=kriging(jurarg_KU,val_locrg_KU,mres,neigh,uc=drift)
mean((res_valKU[,"Kriging*estim"]-val[,2])^2)
```

1. Add the *Landuse* predictor to the model.
2. Try several variogram models (basic structures, anisotropy), and neighborood options. Compare the prediction scores. Comment the results.

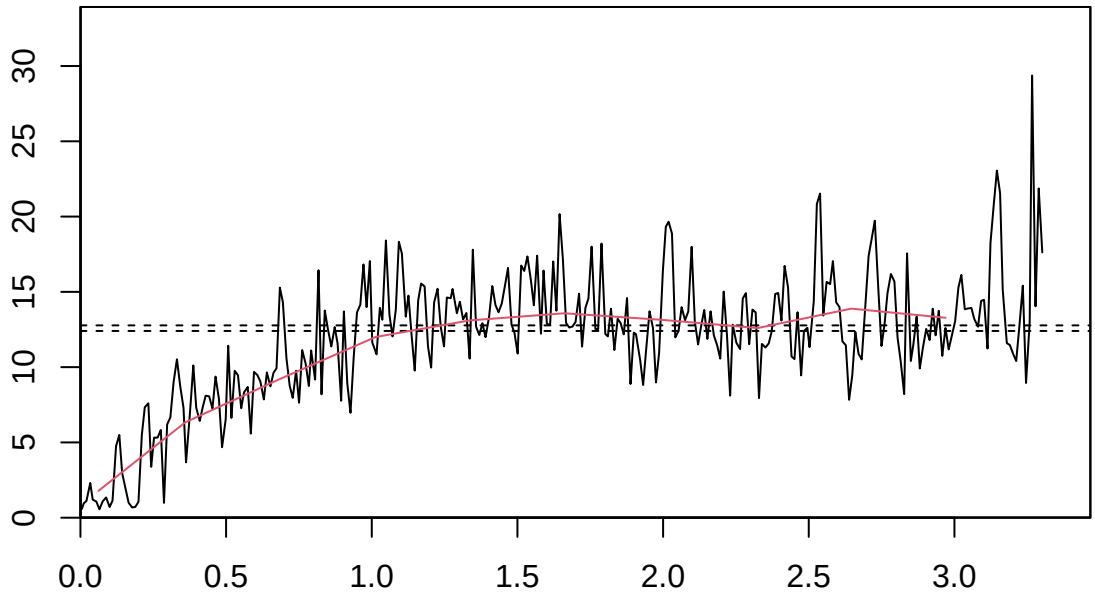
The difference between the universal kriging and the ordinary kriging lies in the mean , while the mean is still unknown in the universal kriging , it is no longer a constant and has a drift model that depends on the coordinates , the provided code used “Rock” as an indicator and gave a prediction score of 7.225692 , let us try adding the Landuse predictor to the model instead :

Transformation of the factors into indicators

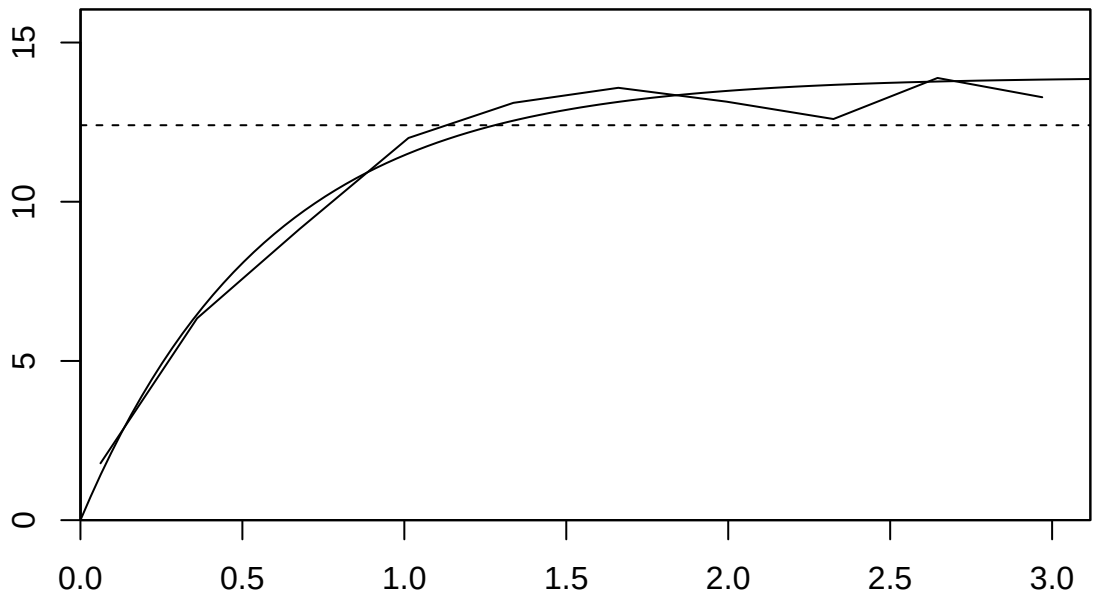
```
indiccut = limits.create(mini=c(1,2,3),maxi=c(2,3,4))

jurarg_KU=db.indicator(jurarg,indiccut,name="Landuse")
jurarg_KU = db.locate(jurarg_KU,"Indicator*","f")
jurarg_KU = db.locate(jurarg_KU,"Co","z")

gridrg_KU=db.indicator(gridrg,indiccut,name="Landuse")
gridrg_KU = db.locate(gridrg_KU,"Indicator*","f")
gridrg_KU = db.locate(gridrg_KU,"Co","z")

val_locrg_KU=db.indicator(val_locrg,indiccut,name="Landuse")
val_locrg_KU = db.locate(val_locrg_KU,"Indicator*","f")
val_locrg_KU = db.locate(val_locrg_KU,"Co","z")
```

Variogram of the residuals

```
drift = c("1","f1","f2","f3")
vres = vario.calc(jurarg_KU,nlag=10,uc=drift)
plot(v)
plot(vres,add=T,col=2)
```

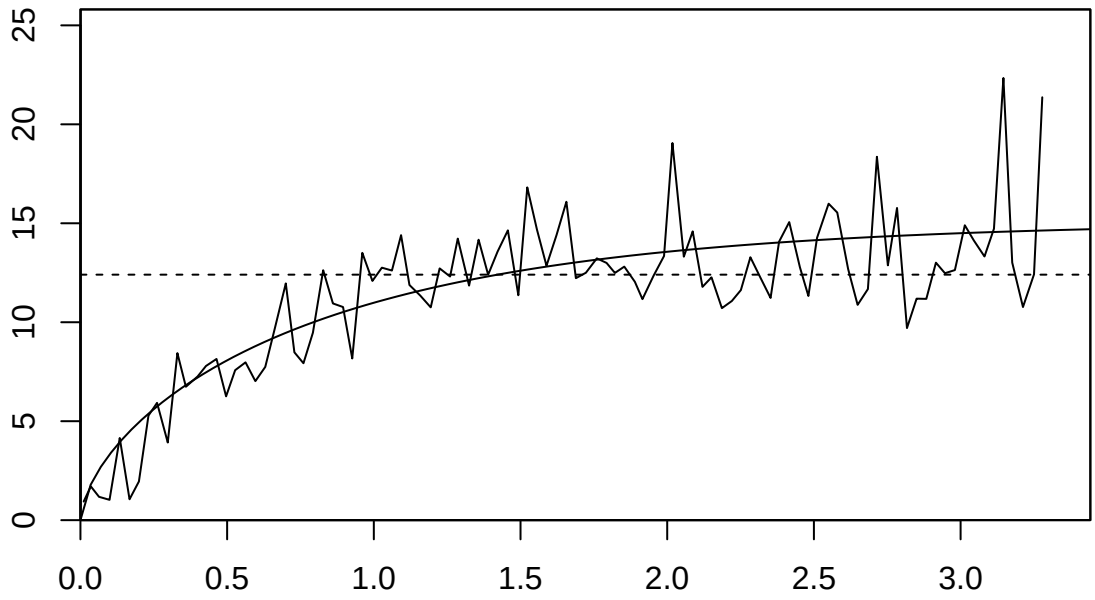



```
mres=model.auto(vres,struct=c(1,2))
```



Universal kriging on the validation set

```
drift = c("1","f1","f2","f3")
vres = vario.calc(jurarg_KU,nlag=100,uc=drift)
mres=model.auto(vres,struct=c(1,8))
```

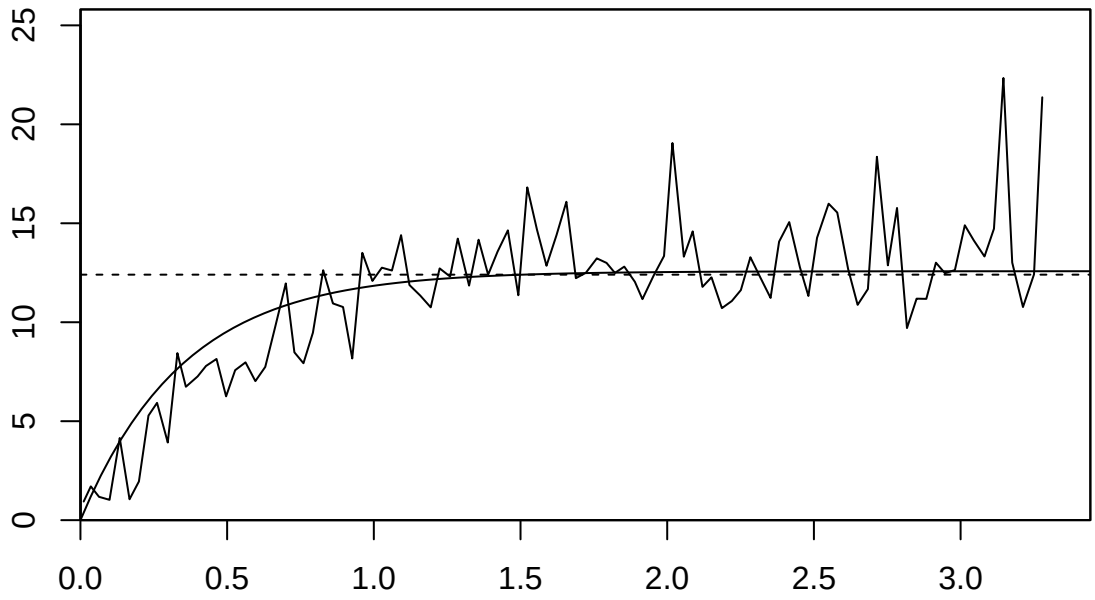



```
neigh = neigh.create(type=0)
res_valKU=kriging(jurarg_KU,val_locrg_KU,mres,neigh,uc=drift)
mean((res_valKU[,"Kriging*estim"]-val[,2])^2)
```

```
## [1] 6.382035
```

Let us try now changing the models , the model that was used c(1,8) takes into account the nugget effect while simulating a K-bessel model ,let us try an exponential model with c(1,2) instead :

```
mres=model.auto(vres,struct=c(1,2))
```



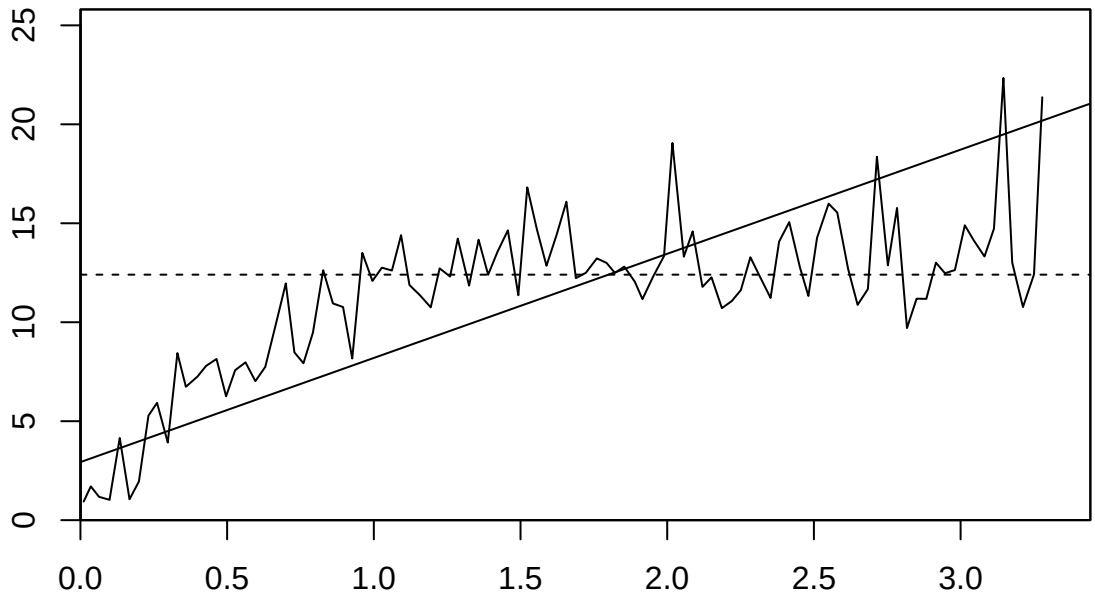

```
res_valKU=kriging(jurarg_KU,val_locrg_KU,mres,neigh,uc=drift)
mean((res_valKU[,"Kriging*estim"]-val[,2])^2)
```

```
## [1] 7.01927
```

The prediction score becomes 7.01927 which is considerably worse than when the K-Bessel model was used.

Let us try the "Order-1-Gc" model given by c(1,14) : We get a prediction score of 6.10542 which is considerably better than both cases.

```
mres=model.auto(vres,struct=c(1,14))
```




```
res_valKU=kriging(jurarg_KU,val_locrg_KU,mres,neigh,uc=drift)
mean((res_valKU[,"Kriging*estim"]-val[,2])^2)
```

```
## [1] 6.10542
```

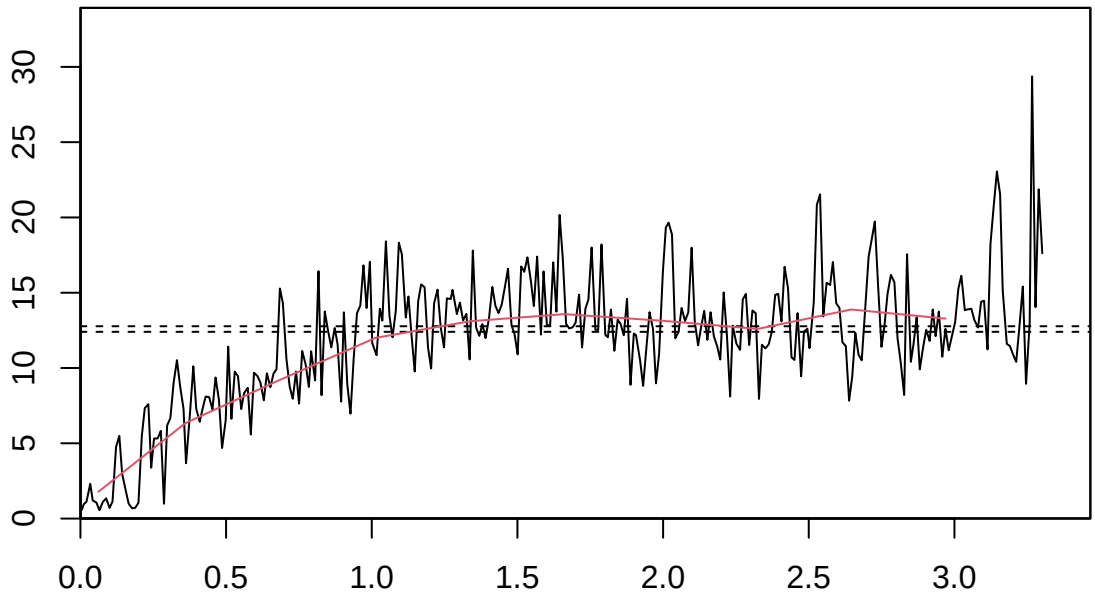
So far we had our neighborhood type set to 0 which is the unique neighborhood case Let us now try the case of the moving neighborhood , with sectors and anisotropies , the code becomes :

Transformation of the factors into indicators

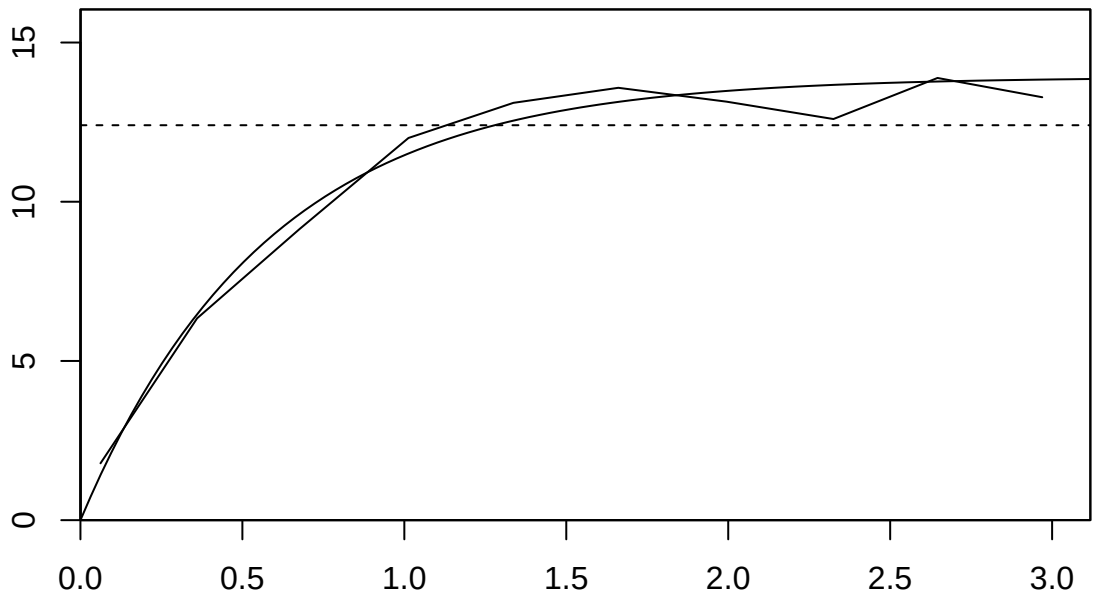
```
indiccut = limits.create(mini=c(1,2,3),maxi=c(2,3,4))

jurarg_KU=db.indicator(jurarg,indiccut,name="Landuse")
jurarg_KU = db.locate(jurarg_KU,"Indicator*","f")
jurarg_KU = db.locate(jurarg_KU,"Co","z")

gridrg_KU=db.indicator(gridrg,indiccut,name="Landuse")
gridrg_KU = db.locate(gridrg_KU,"Indicator*","f")
gridrg_KU = db.locate(gridrg_KU,"Co","z")

val_locrg_KU=db.indicator(val_locrg,indiccut,name="Landuse")
val_locrg_KU = db.locate(val_locrg_KU,"Indicator*","f")
val_locrg_KU = db.locate(val_locrg_KU,"Co","z")
```

Variogram of the residuals

```
drift = c("1","f1","f2","f3")
vres = vario.calc(jurarg_KU,nlag=10,uc=drift)
plot(v)
plot(vres,add=T,col=2)
```

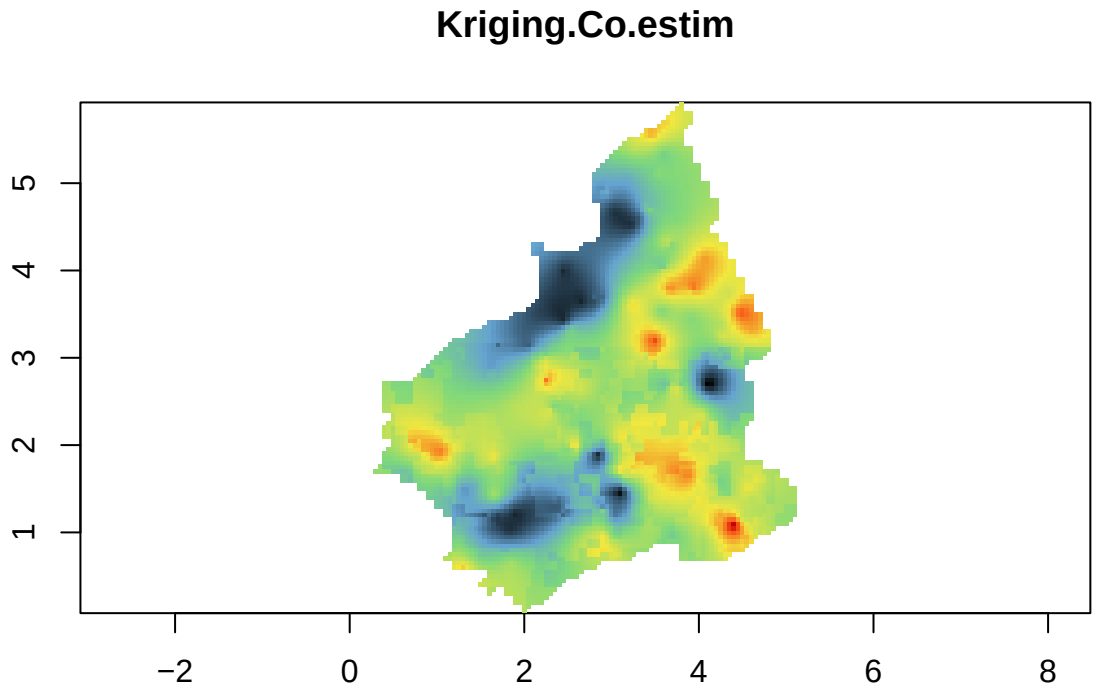



```
mres=model.auto(vres,struct=c(1,2))
```



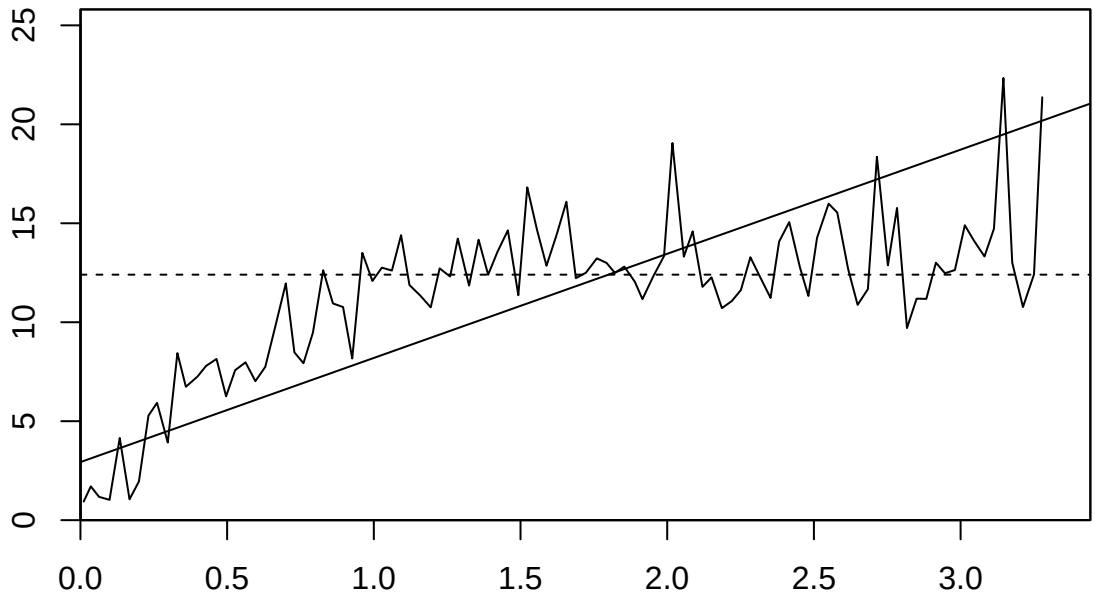
Universal kriging on the grid

```
neigh=neigh.create(type=0)
res_gridKU=kriging(jurarg_KU,gridrg_KU,mres,neigh,uc=drift)
plot(res_gridKU)
```



Universal kriging on the validation set

```
drift = c("1","f1","f2","f3")
vres = vario.calc(jurarg_KU,nlag=100,uc=drift)
mres=model.auto(vres,struct=c(1,14))
```




```
radvec=c(2.2,2.6)
neigh = neigh.create(type=2,nmini=5,nmaxi=5,radius=radvec,flag.sector = T,flag.aniso=T,nsect = 52,nsmax
res_valKU=kriging(jurarg_KU,val_locrg_KU,mres,neigh,uc=drift)
mean((res_valKU[,"Kriging*estim"]-val[,2])^2)
```

```
## [1] 5.796139
```

This prediction score is better than anything we have seen so far.

Optional:

You can do the same with the interaction. Hint: consider the product *Landuse* x *Rock* as a new factor. You will have to group some of the levels so as to have well balanced groups. See the functions *replicates* and *TukeyHSD*.

6 Multivariate analysis

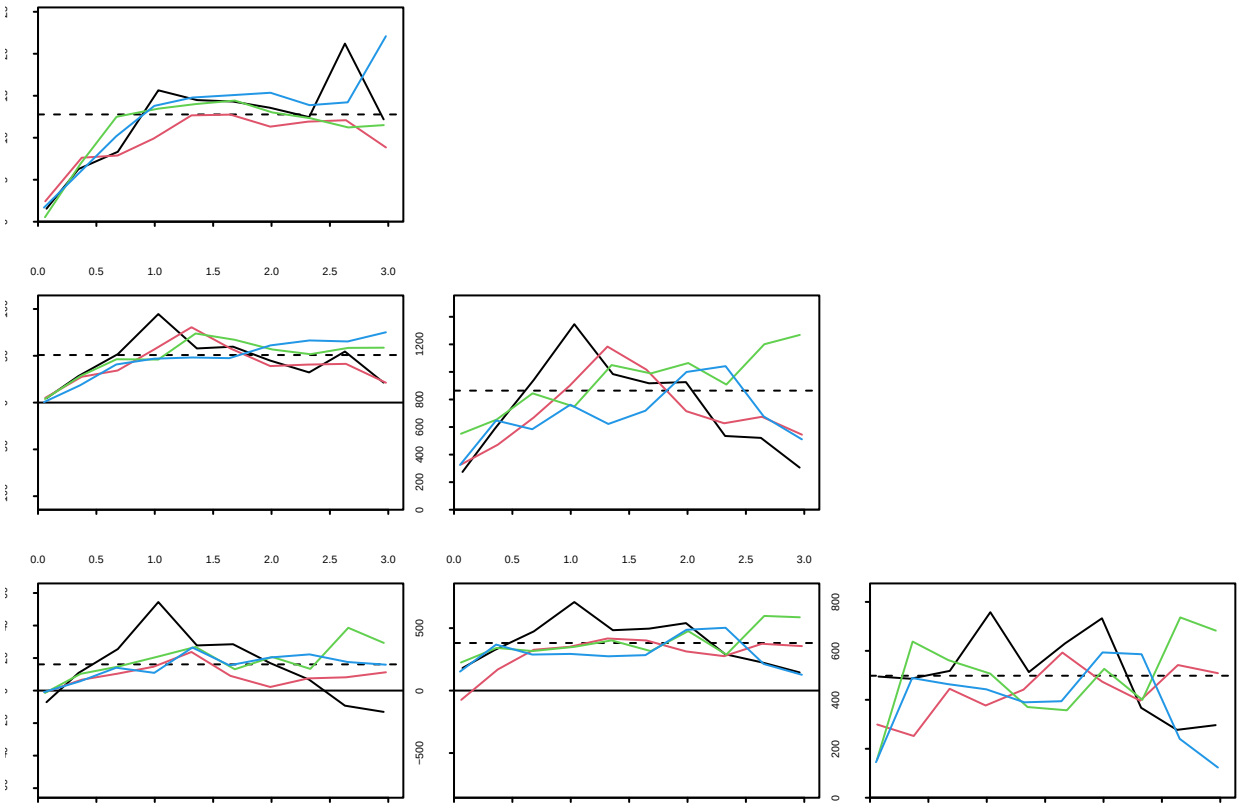
6.1 Variography

1. Compute the empirical directional variograms and covariograms of a carefully chosen (justify) set of variables. What would you conclude about anisotropy ?

2. Fit a model (with *model.auto*).

Let us compute the empirical directional variograms and covariograms of the variables, we will use cobalt zinc and copper as the regionalized varaibles just as follows , Xloc and Yloc have been previously set as the space dimension.

```
jurarg = db.locate(jurarg,c("Co","Zn","Cu"),"z")
vdir = vario.calc(jurarg,nlag=10,dir=c(-10,35,80,125))
plot(vdir)
```

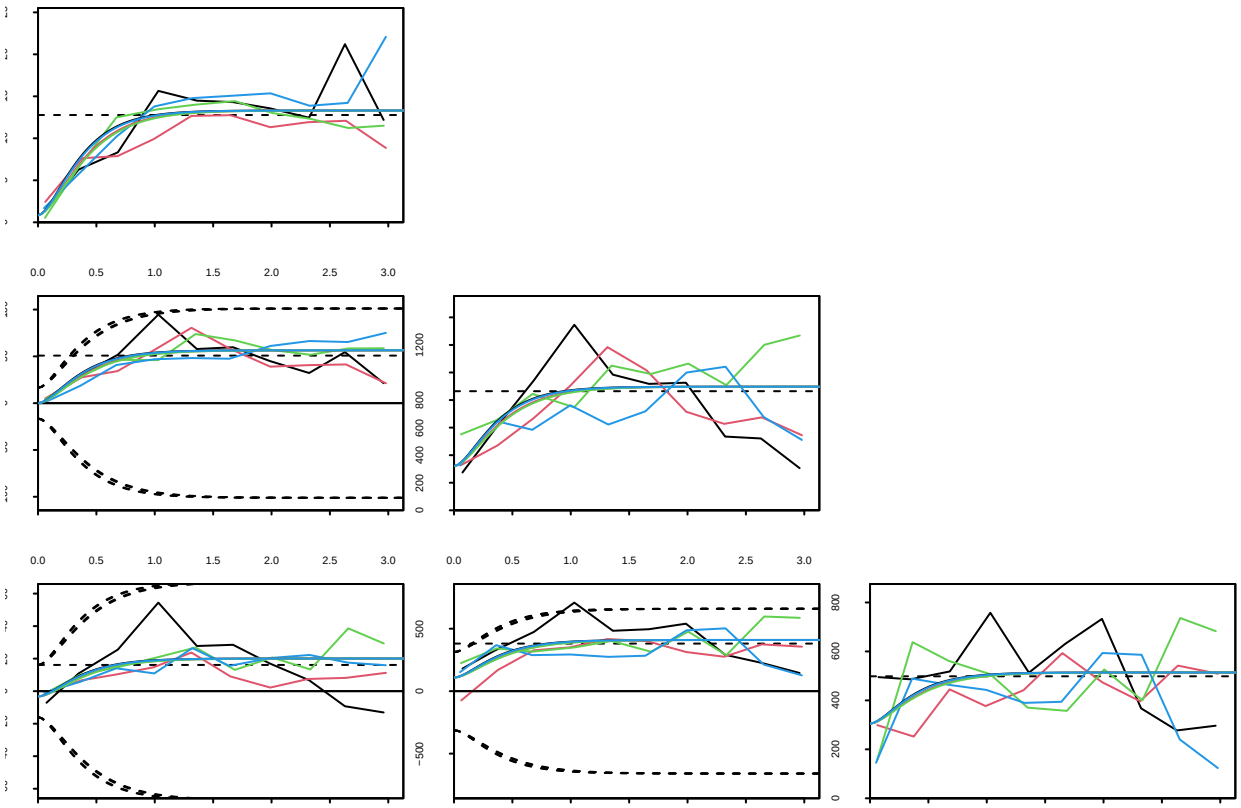


The plots on the diagonals are the specific variograms for Cobalt , Zinc and Copper , and the ones excluded from the diagonal are the cross variograms between each of the two metals , each colored curve in each plot represents one of the directions in our dir column vector defined above.

When first looking at the colored curves , the data doesn't seem to be anisotropic even though the values don't tend to be the same when the lag increases , this difference isn't that significant to say that the sill is different or the data is zonal anisotropic. The range also doesn't seem to vary with direction which leads us to think that the data isn't geometric anisotropic.

We will now fit a model to these directional variograms:

```
m = model.auto(vdir,struct=c(1,8))
```



The model that was fitted coincides for all the directional curves which reinforces our statement above.

6.2 Prediction

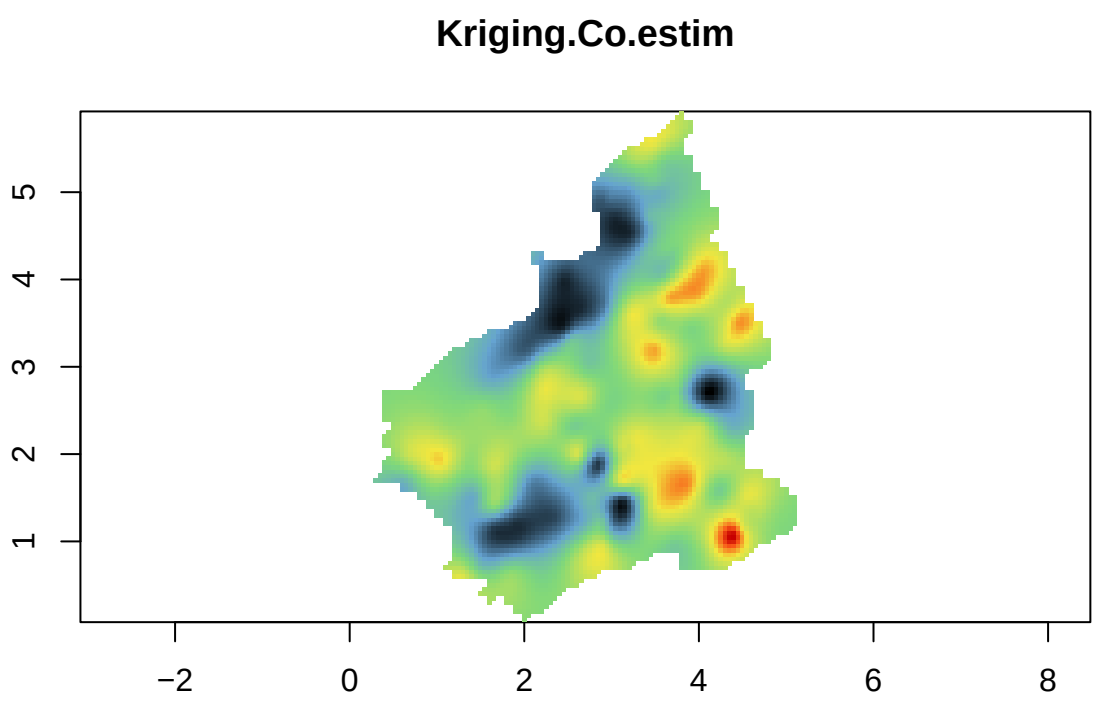
1. Interpolate *Co* on the grid using Ordinary Cokriging (function *kriging*) and plot the resulting map as well as the standard deviation map. Compute the prediction score.
2. You can also try to implement the universal cokriging (*optional*).

Let us now try to interpolate the Co concentrations using the ordinary cokriging , the code is as follows :

Creation of the target grid.

```
grid = read.csv("../../jura/jura_grid.csv")
gridtemp = db.create(grid)
gridtemp = db.locate(gridtemp,c("Xloc","Yloc"),"x")
nx = c(length(unique(grid[,1])),length(unique(grid[,2])))
gridrg = db.grid.init(gridtemp,nodes=nx)
gridrg = migrate(gridtemp,gridrg,names=4:gridtemp$natt,radix="")
gridrg = db.rename(gridrg,2:3,c("Xloc","Yloc"))
gridrg = db.sel(gridrg,!is.na(gridrg["Landuse"])&!is.na(gridrg["Rock"]))
```

```
neigh = neigh.create(type = 0)
res = kriging(jurarg,gridrg,m,neigh)
plot(res,sub="Cokriging of Co")
```



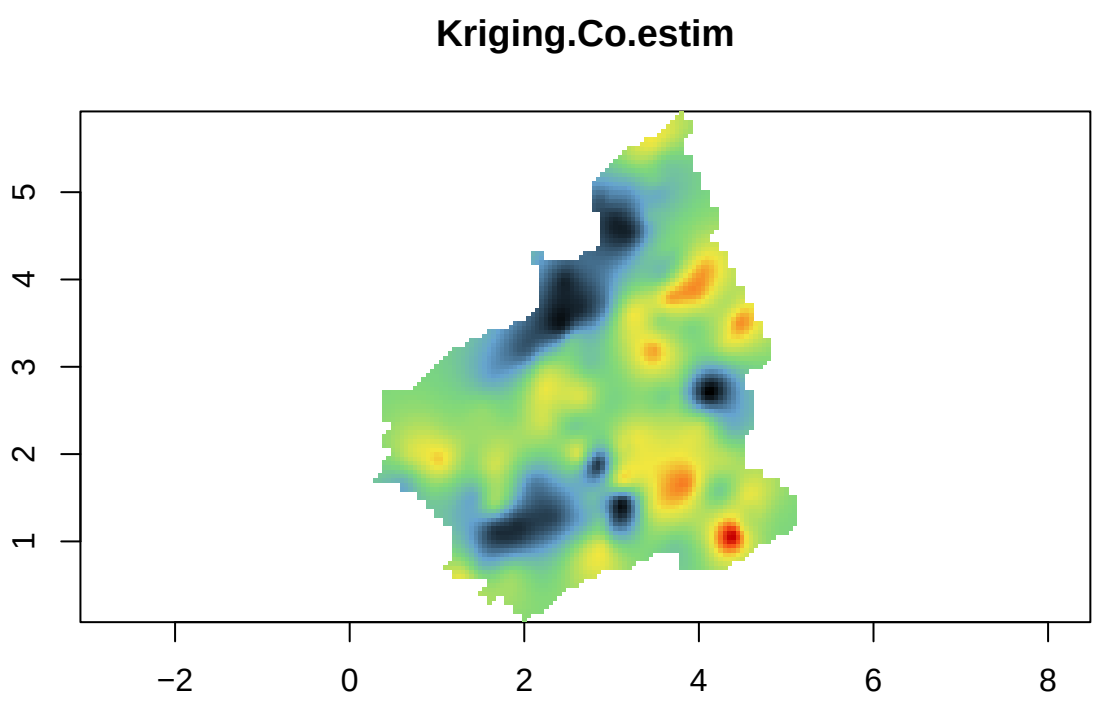
Now perform Ordinary Cokriging on the validation set.

```
val_locrg = db.create(val_loc)
val_locrg = db.locate(val_locrg,c("Xloc","Yloc"),"x")
res_val = kriging(jurarg,val_locrg,m,neigh)
mean((res_val[,"Kriging.Co.estim"]-val[,2])^2)  ## MSE for Cokriging
```

```
## [1] 7.299215
```

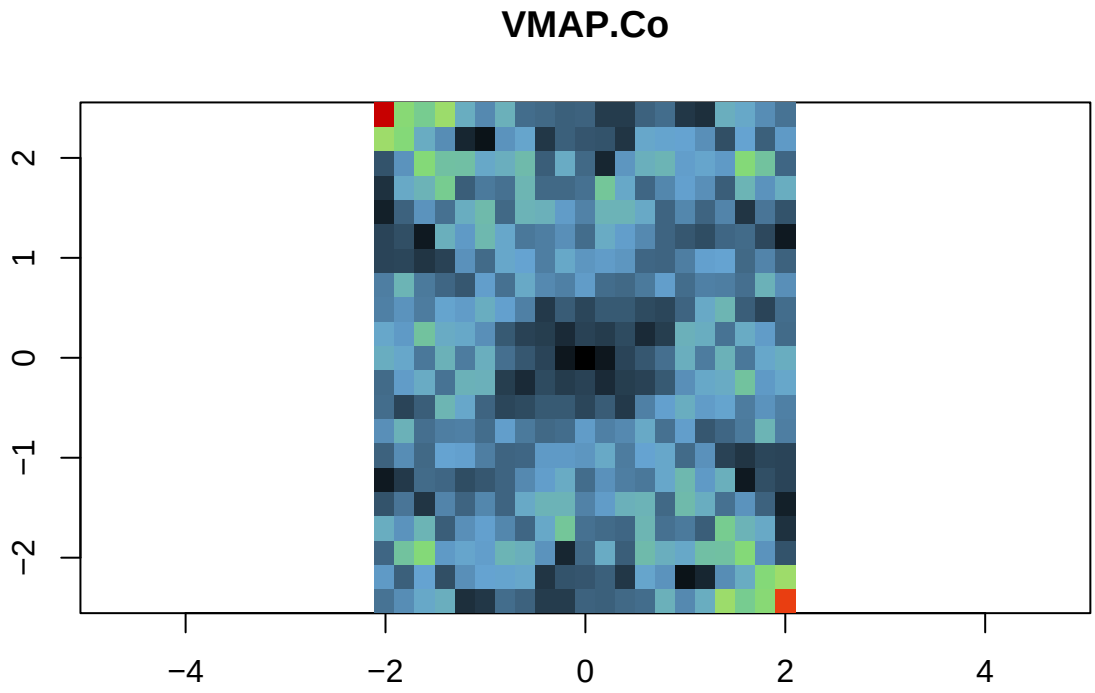
We could also attempt to plot all the vmaps and the standard deviation maps as follows :

```
neigh = neigh.create(type = 0)
res = kriging(jurarg,gridrg,m,neigh)
plot(res,sub="Cokriging of Co")
```

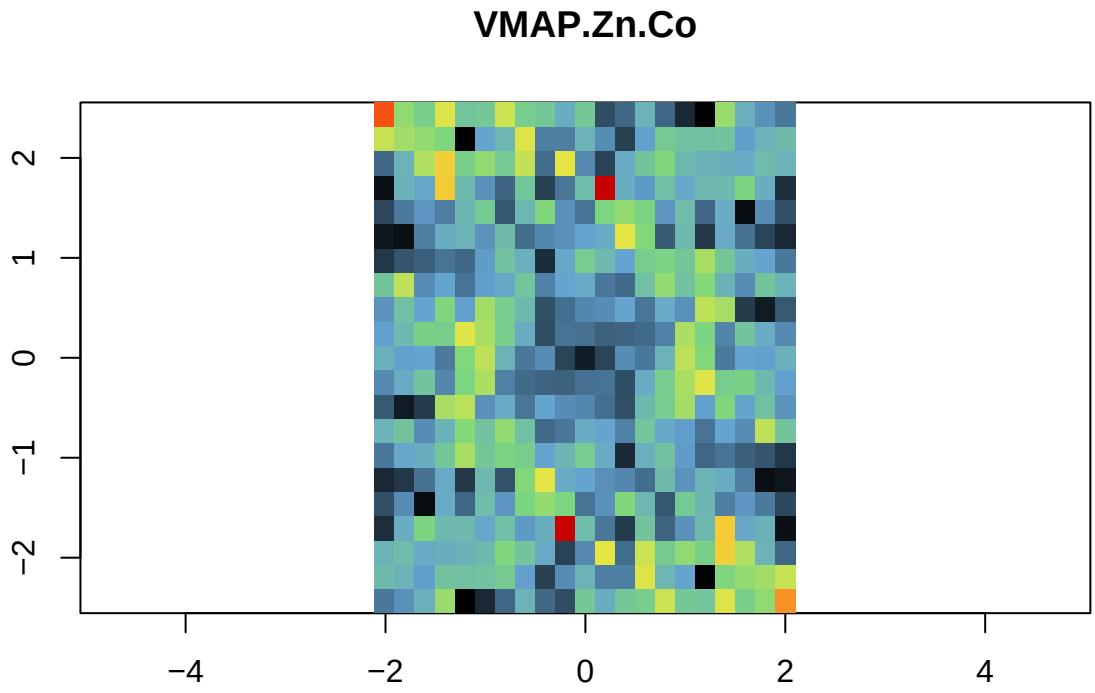


Cokriging of Co

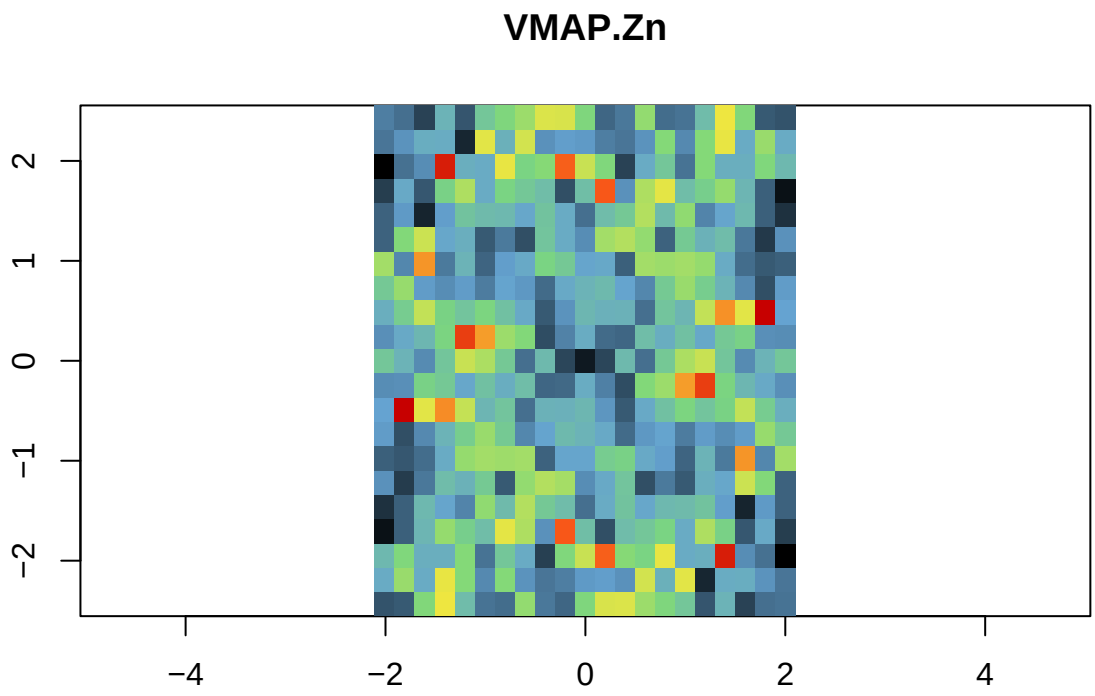
```
v=vmap.calc(jurarg)
std_co=sqrt(v$items[["VMAP.Co"]])
std_zn_co=sqrt(v$items[["VMAP.Zn.Co"]])
std_zn=sqrt(v$items[["VMAP.Zn"]])
std_cu_co=sqrt(v$items[["VMAP.Cu.Co"]])
std_cu_zn=sqrt(v$items[["VMAP.Cu.Zn"]])
std_cu=sqrt(v$items[["VMAP.Cu"]])
v1=db.add(v,std_co,std_zn_co,std_zn,std_cu_co,std_cu_zn,std_cu)
par(mfrow=c(2,3))
plot(v1,pos.x = 1,pos.y = 2,name="VMAP.Co")
```



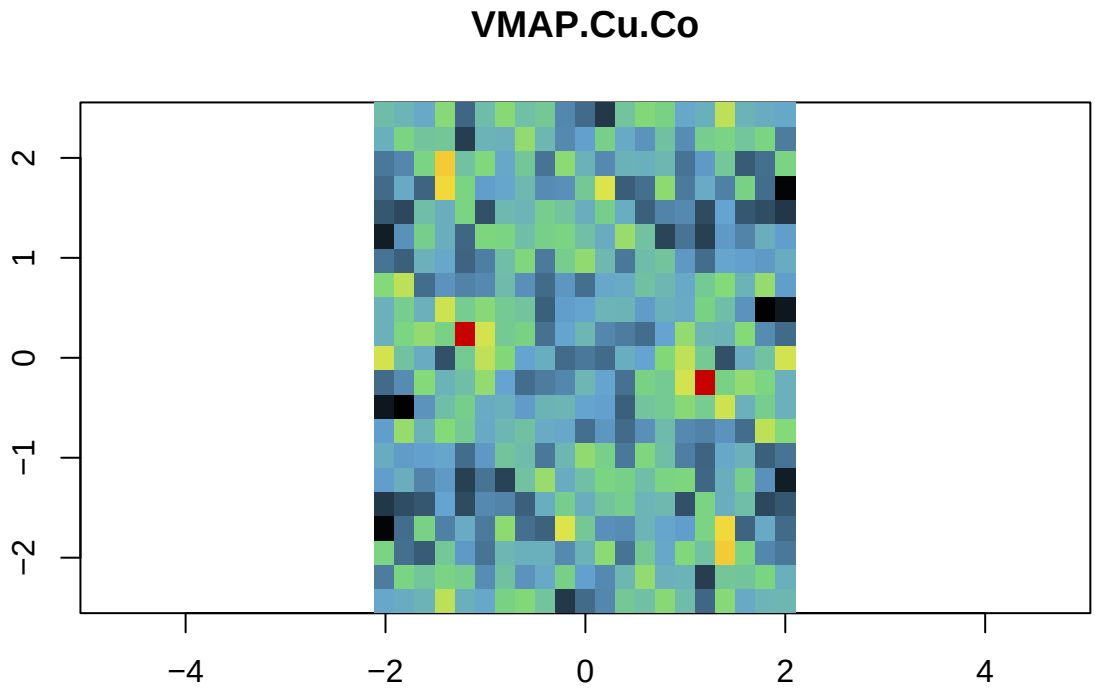

```
plot(v1,pos.x = 1,pos.y = 2,name="VMAP.Zn.Co")
```



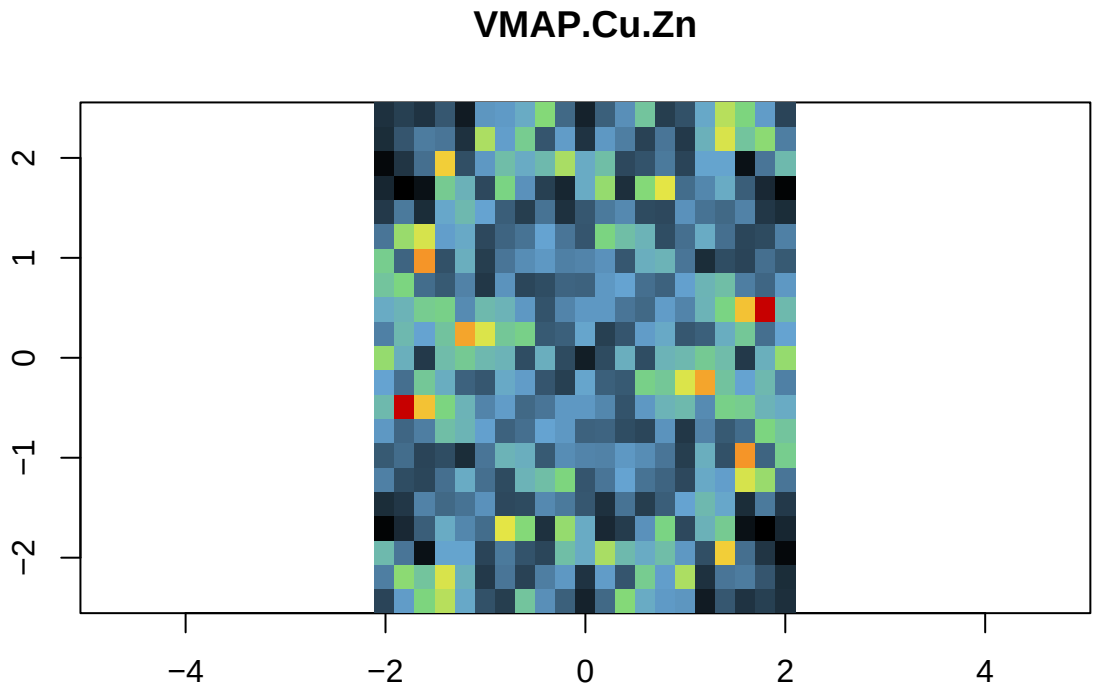

```
plot(v1,pos.x = 1,pos.y = 2,name="VMAP.Zn")
```



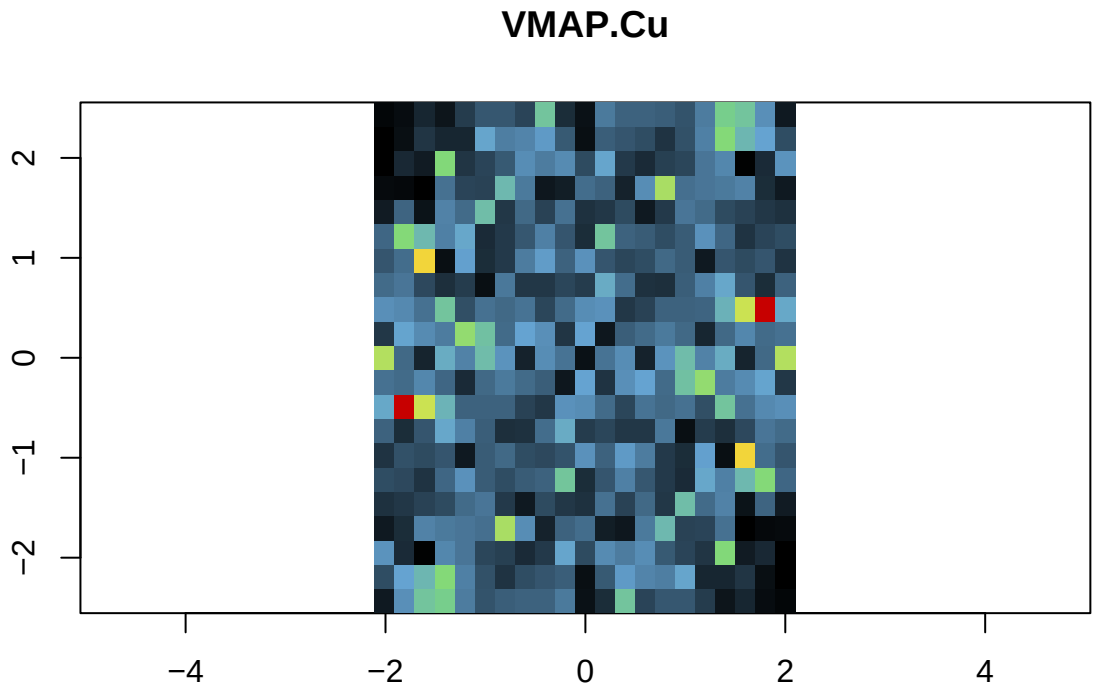

```
plot(v1,pos.x = 1,pos.y = 2,name="VMAP.Cu.Co")
```



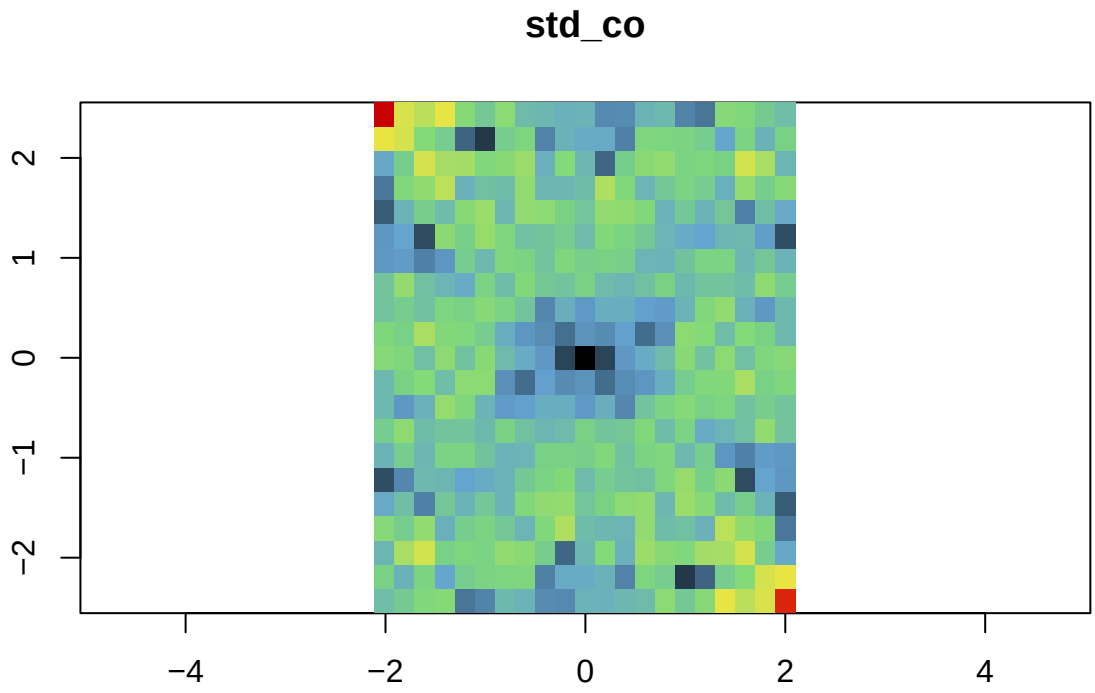

```
plot(v1,pos.x = 1,pos.y = 2,name="VMAP.Cu.Zn")
```



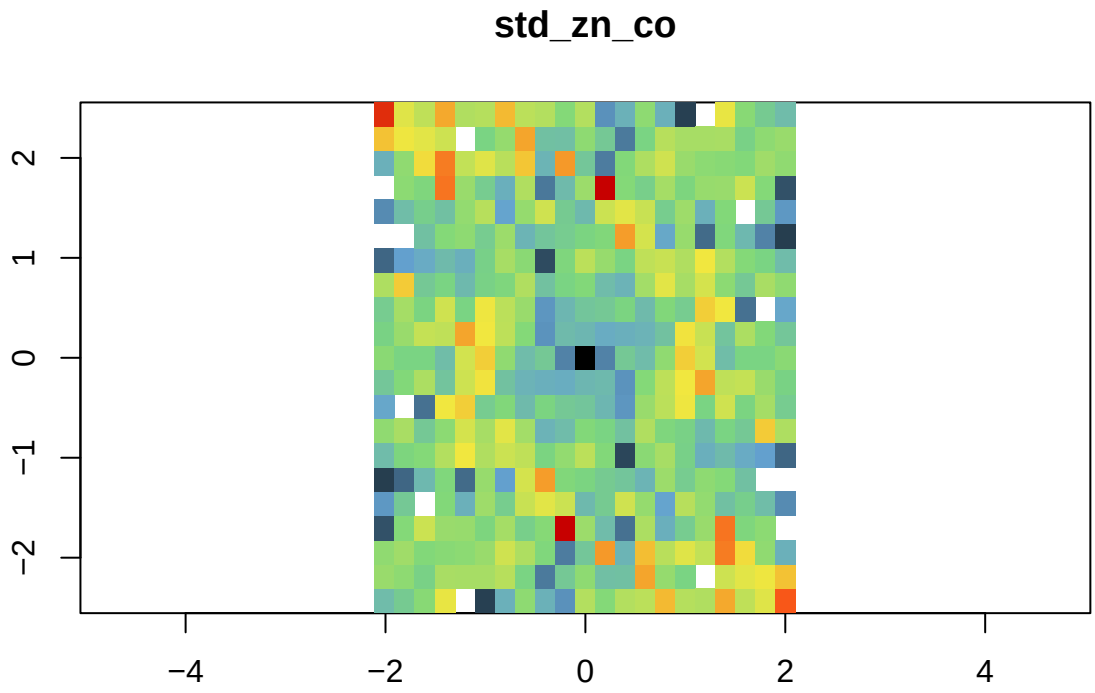

```
plot(v1,pos.x = 1,pos.y = 2,name="VMAP.Cu")
```



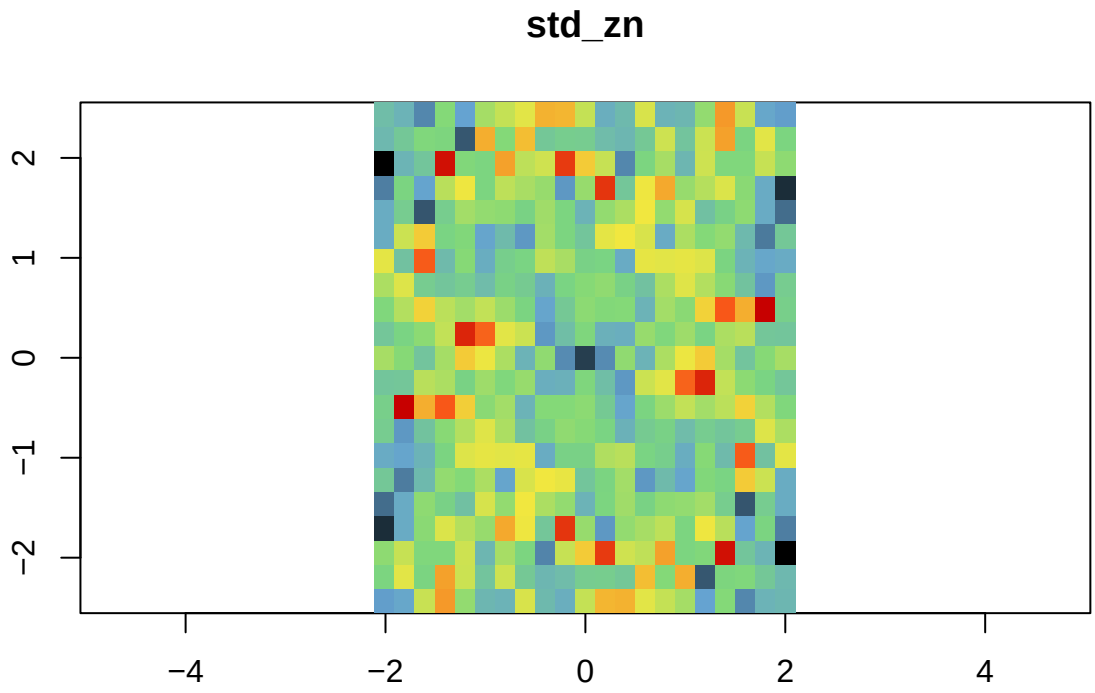

```
plot(v1,pos.x = 1,pos.y = 2,name="std_co")
```



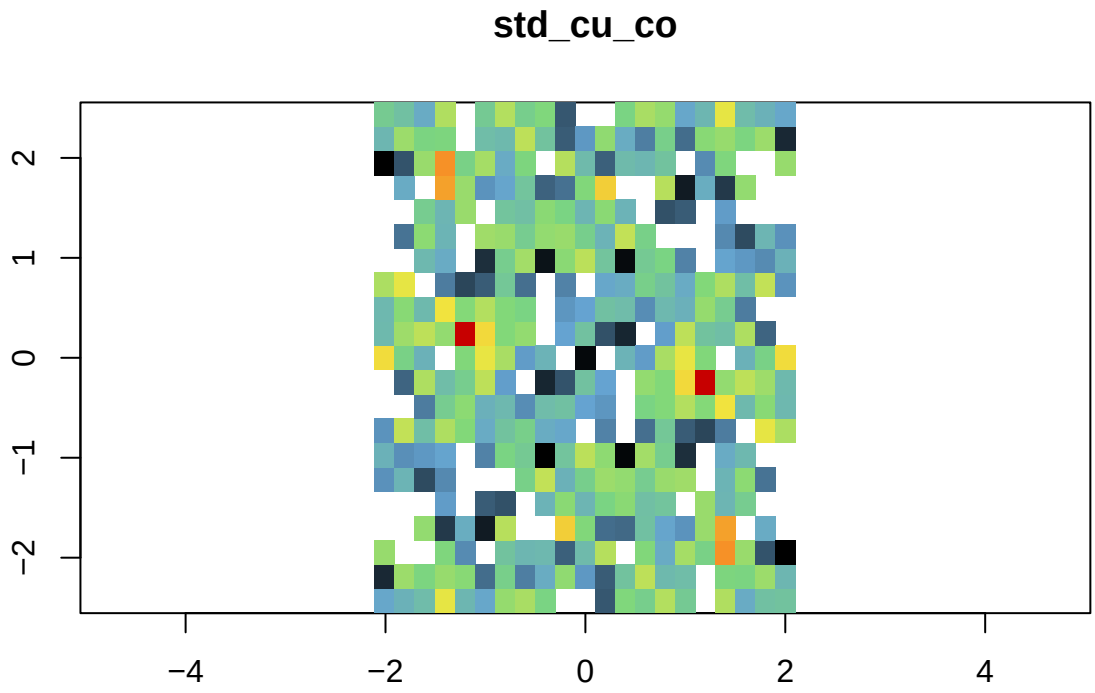

```
plot(v1,pos.x = 1,pos.y = 2,name="std_zn_co")
```



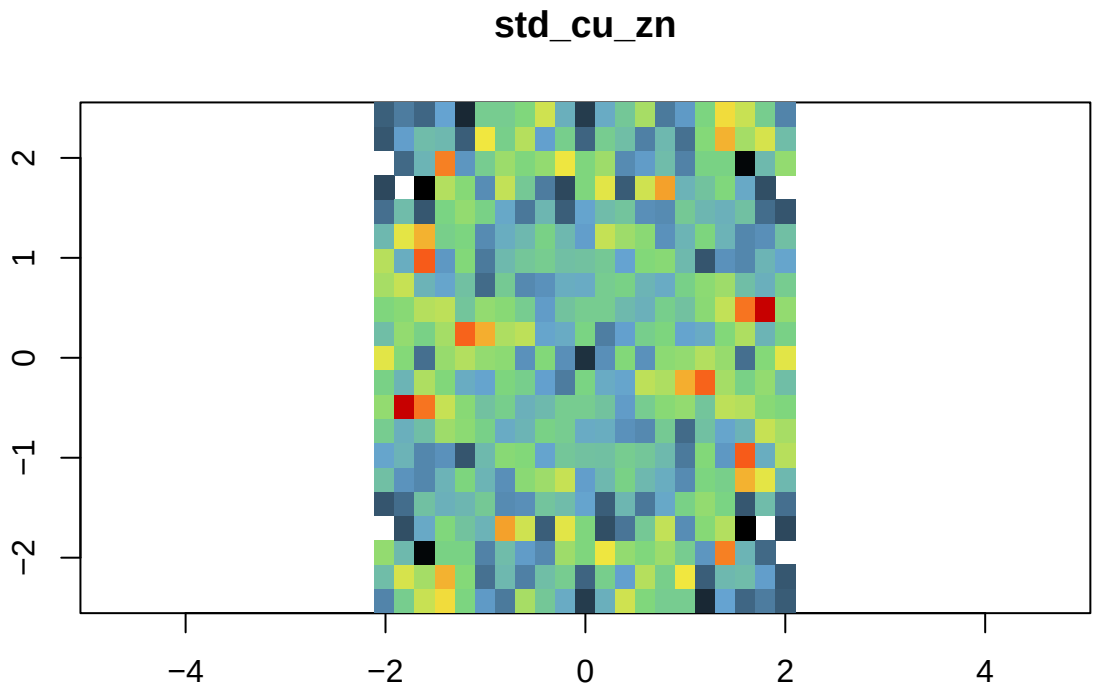

```
plot(v1,pos.x = 1,pos.y = 2,name="std_zn")
```



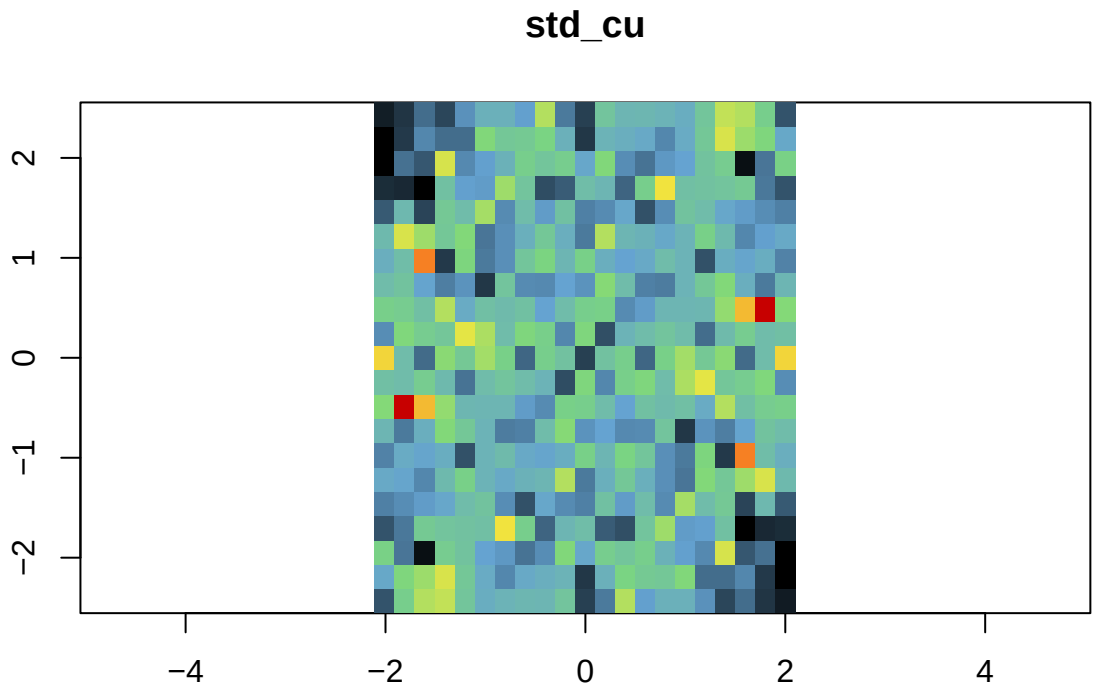

```
plot(v1,pos.x = 1,pos.y = 2,name="std_cu_co")
```




```
plot(v1,pos.x = 1,pos.y = 2,name="std_cu_zn")
```

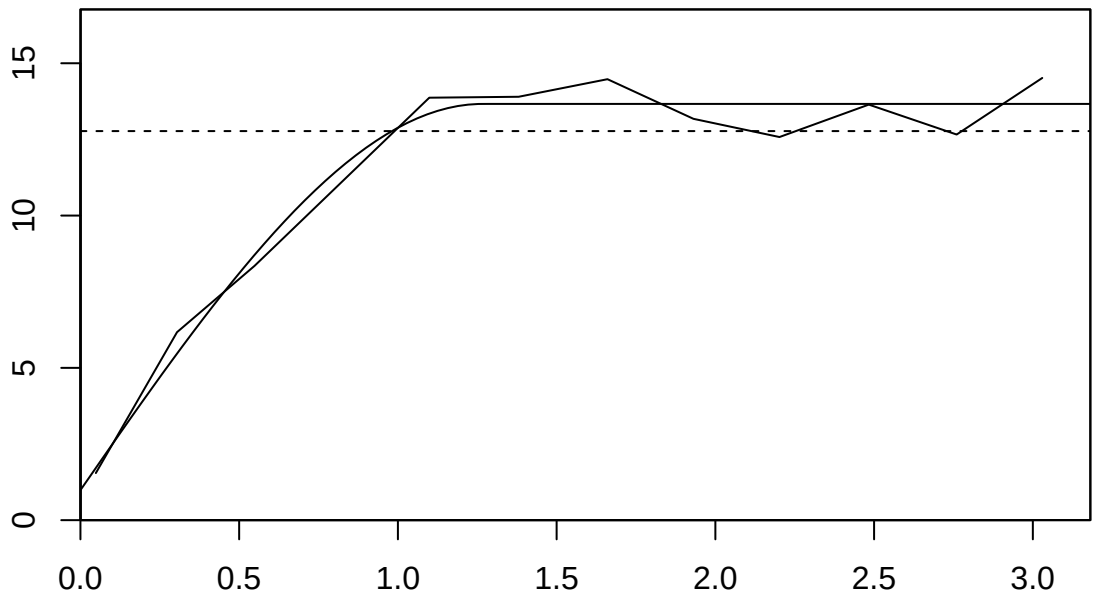



```
plot(v1,pos.x = 1,pos.y = 2,name="std_cu")
```

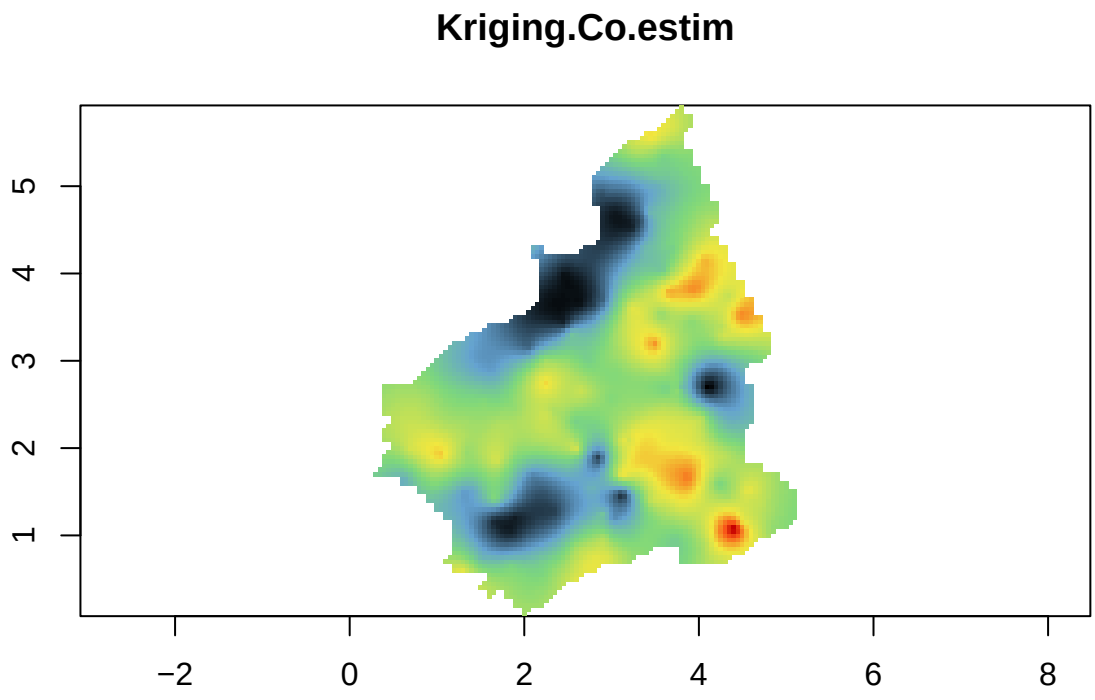


Now we can compare this with the ordinary kriging results by just removing Zn and Cu from our regionalized variables and computing the prediction score once again :

```
jurarg <- db.locate(jurarg, c("Zn","Cu"))# Remove Zn,Cu

# Variogram
v = vario.calc(jurarg,nlag=12)
m = model.auto(v,c(1,3,3))
```




```
# Ordinary Kriging
neigh = neigh.create(type = 0)
res=kriging(jurarg,gridrg,m,neigh)
plot(res,sub="Kriging of Co")
```



Kriging of Co

```
# Ordinary Kriging on val
res_val=kriging(jurarg,val_locrg,m,neigh)
mean((res_val[,"Kriging*estim"]-val[,2])^2)  ## MSE for Ordinary kriging
```

```
## [1] 6.939636
```

We get the following prediction score : 6.939636 which is better than the case where Zn and Cu were a part of the regionalized variables. We could also compare this to the universal kriging case where we got a score of 7.088469.

7 Maximum Likelihood estimation

1. Compute the maximum likelihood estimator of the parameters of (some of) your favorite univariate model(s) for the Cobalt concentration. In particular, to improve the prediction, add the explanatory variable *Landuse* to the model and estimate its parameters by maximum likelihood.
2. Compare the models with and without *Landuse* through a likelihood ratio test.
3. Compute the prediction map and the prediction at the validation locations for each model. Compute the prediction score.
4. Try the **SPBayes** package for the multivariate approach (*optional*)).

8 Conditional simulations

The information threshold for the concentration of cobalt in soils is *12 mg/kg*.

1. Generate 100 conditional simulations of the Cobalt concentrations over the swiss Jura according to your favorite model.

Example with the ordinary kriging (where mres is the fitted variogram of the cobalt)

```
res_simu = simtub(jurarg_KU,gridrg_KU,mres,neigh,nbsimu=100,nbtuba=1000)
```

2. Compute the mean surface of the area of exceedance as well as its associated centered 95% confidence interval.

```
res_simu[,"Simu.Co.S*"] = res_simu[,"Simu.Co.S*"]>12
res_simu[,"Simu.Co.S1"] <- rowMeans(res_simu[,"Simu.Co.S*"])
print("The mean surface of the area of exceedance is:")
```

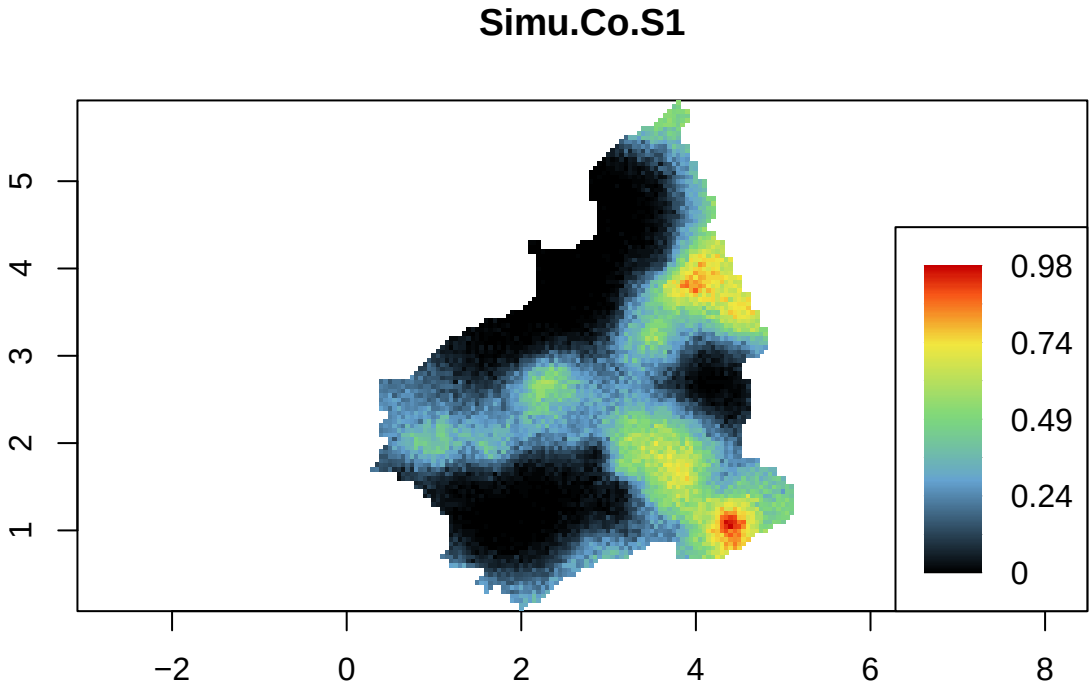
```
## [1] "The mean surface of the area of exceedance is:"
```

```
print(mean(rowMeans(res_simu[,"Simu.Co.S*"])))
```

```
## [1] 0.1234222
```

3. Compute and plot the exceedance probability map. Comment.

```
plot(res_simu,name="Simu.Co.S1",pos.legend=1)
```



9 Summary – Discussion