Spatial Statistics

Computer lab – Session 1

Madlene Nussbaum Andreas Papritz (ehem. ETHZ)

07 Oct 2024

Table of contents

For the following exercises, use the R code of the lecture slides on the introduction to spatial data and the analysis of the Wolfcamp aquifer dataset as template.

1 Install required software

```
install.packages("spDataLarge", repos = "https://geocompr.r-universe.dev")

l.p <- c("Boruta", "caret", "data.table", "geoGAM", "georob", "ggplot2",
```

```
          "gstat", "mapview", "ranger","rgl", "sf", "sp", "terra", "tidyterra")
install.packages(l.p)
```

2 Spatial data in R – classes and plotting

In this section you will get familiar with the main R packages to handle spatial data, how to inspect the content and how to plot spatial data.

2.1 Spatial data types

The main spatial data types used for geostatistical tasks are point vector geometries and raster data.

> Task 1
>
> Familiarize yourself with spatial data types. Read through Pebesma and Bivand, 2023. Section 1, Getting started.

Coordinate reference systems (CRS) are often referred to by a identifying number (EPSG). An overview of worldwide CRS can be found here https://epsg.io.

2.2 Package `sf`

R package `sf` handles geographical vector data. Get familiar with `sf` objects by reading Pebesma and Bivand, 2023. Section 7.1 (up to and including *Subsetting*).

> Task 1
>
> Load the landslide dataset `lsl` from package `spDataLarge`. Create a spatial `sf` object using `st_as_sf()` and assign the correct CRS. See information on the dataset's help page.

> Task 2
>
> Extract the spatial coordinates as vectors (this is often needed to do non-spatial plotting or handing over to functions that cannot deal with `sf` objects).

Task 3

Create a simple plot of the landslide points (`true`: landslide initiation points, `false`: unaffected).

Solution Task 1

```
library(sf)
library(spDataLarge)
data(lsl, package = "spDataLarge")
v.lsl <- st_as_sf(lsl, coords = c("x", "y"), crs = 32717)
str(v.lsl)
```

```
Classes 'sf' and 'data.frame':  350 obs. of  7 variables:
 $ lslpts     : Factor w/ 2 levels "FALSE","TRUE": 1 1 1 1 1 1 1 1 1 1 ...
 $ slope      : num  33.8 39.4 37.5 31.5 44.1 ...
 $ cplan      : num  0.02318 -0.03864 -0.01333 0.04093 0.00969 ...
 $ cprof      : num  0.00319 -0.01719 0.00967 0.00589 0.00515 ...
 $ elev       : num  2423 2052 1958 1969 3008 ...
 $ log10_carea: num  2.78 4.15 3.64 2.27 3 ...
 $ geometry   :sfc_POINT of length 350; first list element:  'XY' num  713888 9558537
 - attr(*, "sf_column")= chr "geometry"
 - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA NA NA
  ..- attr(*, "names")= chr [1:6] "lslpts" "slope" "cplan" "cprof" ...
```

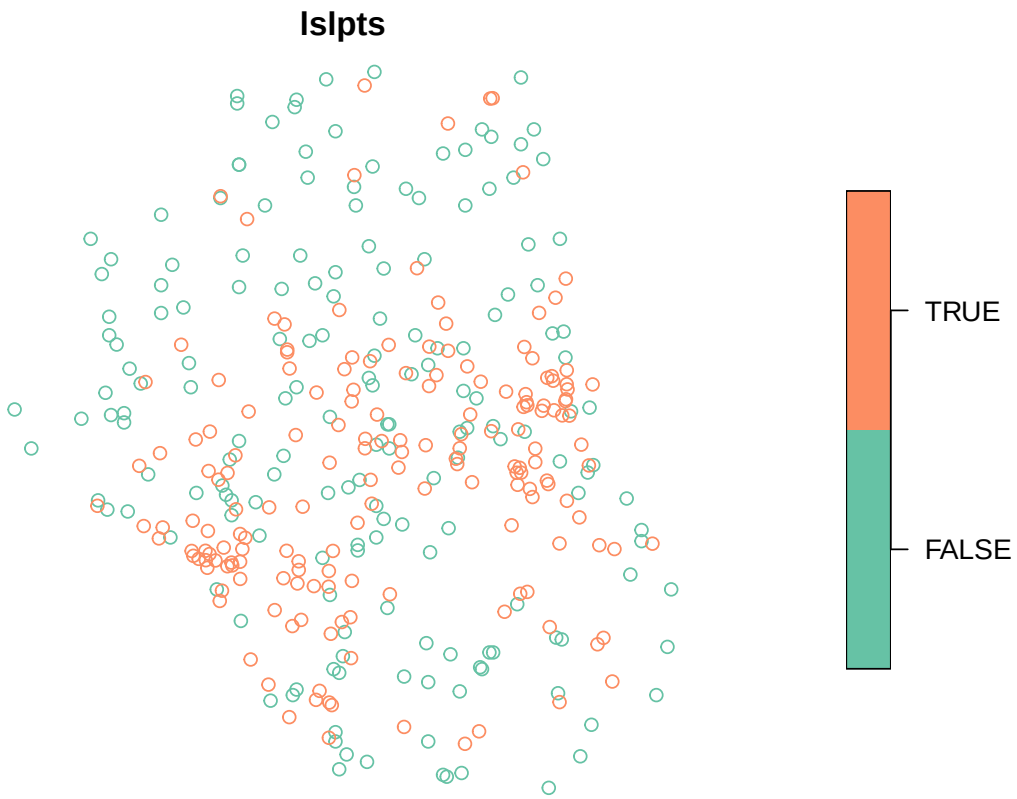
Solution Task 2

```
coords <- st_coordinates(v.lsl)
summary(coords[,1])
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 712198  713567  714111  714143  714810  715738
```

Solution Task 3

```
plot(v.lsl["lslpts"])
```



2.3 Package `terra`

R package **terra** handles geographical raster data. An alternative is the **stars** package which is yet less widespread. The predecessor of **terra** is the no longer supported **raster** package.

Read in the elevation data from **spDataLarge**.

```
library(terra)
ta <- rast(system.file("raster/ta.tif", package = "spDataLarge"))
```

> Task 1
>
> Inspect the content of the elevation raster. Access e.g. the first 10 pixel values of one layer by using `values()`.
> Optional: Set all values > 2000 m in the elevation raster to `NA`.

Task 2

Create a simple plot of the slope (first band).

Solution Task 1

The CRS is 32717. It's spatial resolution is 10 by 10 meters. The raster contains 5 bands corresponding to different elevation information.

```
ta
```

```
class       : SpatRaster
dimensions  : 415, 383, 5  (nrow, ncol, nlyr)
resolution  : 10, 10  (x, y)
extent      : 711962.7, 715792.7, 9556862, 9561012  (xmin, xmax, ymin, ymax)
coord. ref. : WGS 84 / UTM zone 17S (EPSG:32717)
source      : ta.tif
names       :    slope,      cplan,      cprof,     elev, log10_carea
min values  :  0.00000, -25.697536, -0.3194027, 1711.204,    2.000000
max values  : 76.17377,   4.267366,  0.1368342, 3164.165,    5.733915
```

```
values(ta)[1:10,4] # first 10 values of elevation
```

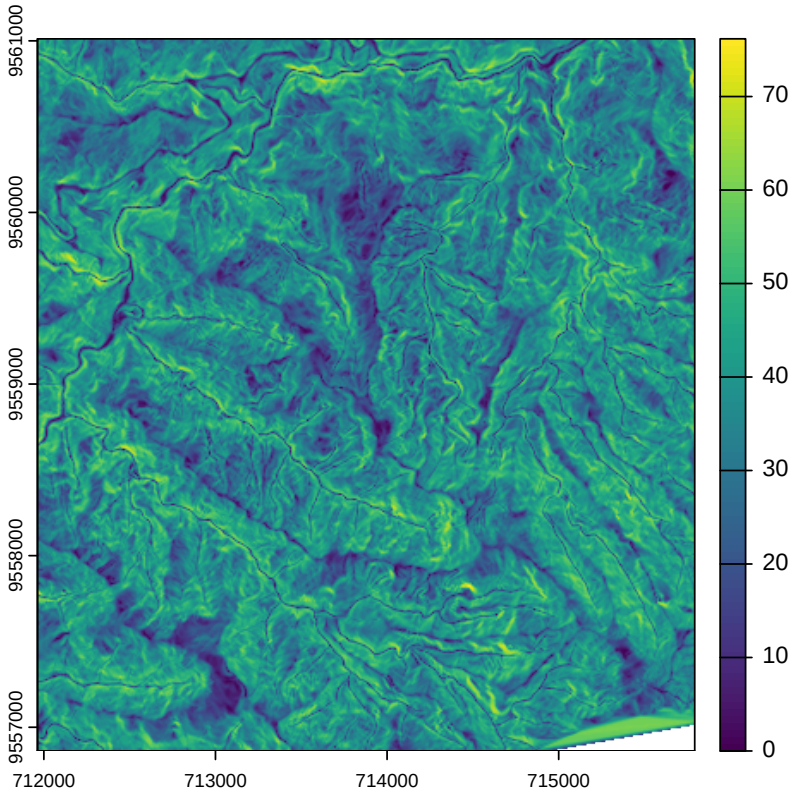
```
 [1] 2227.600 2225.095 2222.745 2222.446 2228.284 2231.895 2232.206 2225.988
 [9] 2213.598 2208.264
```

```
values(ta)[ values(ta)[,4] > 2000 , 4] <- NA
```

Using `values()` raster layers are accessed like a column, and pixel values like rows in a data.frame.

Solution Task 2

```
plot(ta["slope"])
```




```
# plot(ta[[1]]) # band no. also works
```

`SpatRaster` objects have a similar behavior to lists. Using a name requires single brackets while using list entry number requires double brackets to access the content.

2.4 Creating maps

Being able to display geographical data prevents erroneous analysis. Besides classical x-y-plotting, R offers a large variety of display options for spatial data. Above you already created maps using `plot` function. Besides, we will use `ggplot2` and `mapview` packages.

The R package `tmap` also offers many possibilities, if interested check out Lovelace et al. 2024, Chapter Making Maps with R.

Task 1

Using the elevation and landslide data loaded above: create a map showing the slope and overlay the landslide observations points (`points()`).

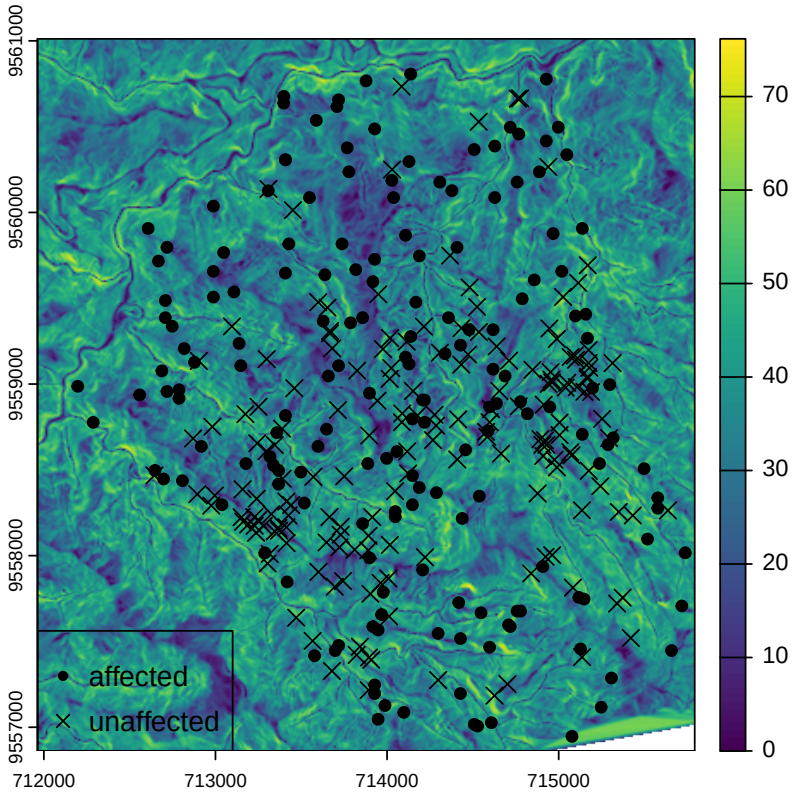
For mapping with `ggplot2` see option in chapter Pebesma and Bivand, 2023. Maps with ggplot2. For `terra` objects see here tidyterra. For extended examples to map with `ggplot2` see Wickham, et al., 2024.

Task 2

Plot the elevation and landslide data using `ggplot2`. To display terra objects with `ggplot2` use `geom_spatraster()` from the `tidyterra` package. `sf` objects can be plotted with `geom_sf()`.

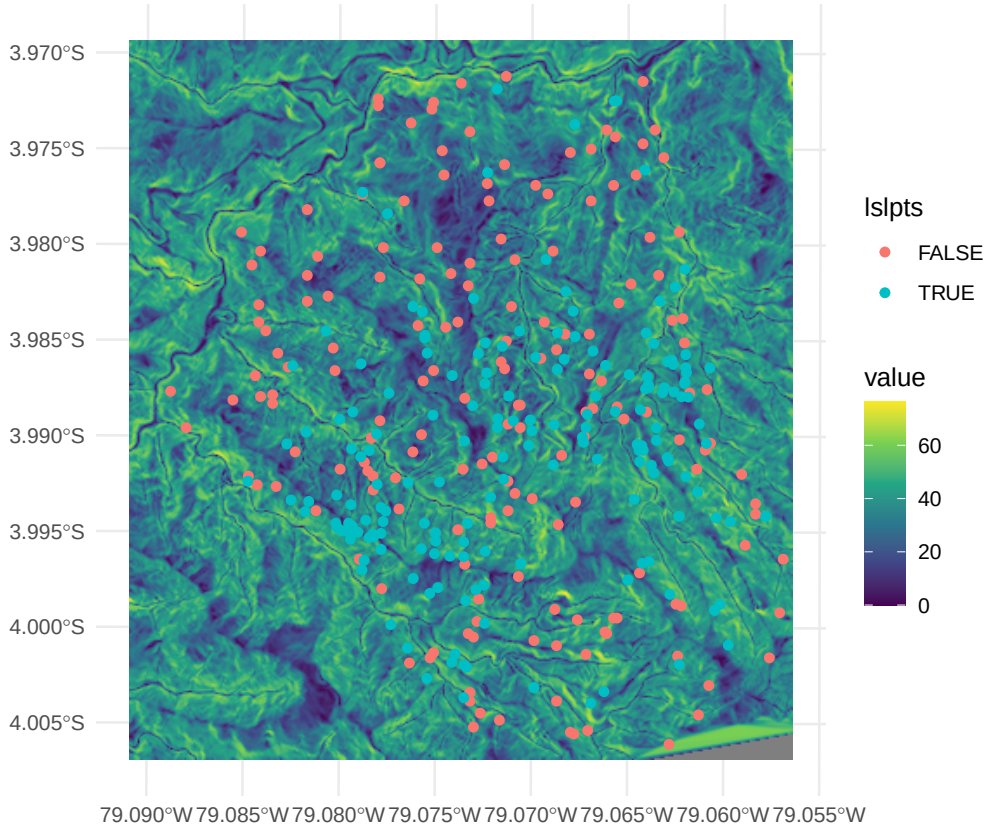
Solution Task 1

```
p <- plot(ta[[1]])
points(v.lsl[1], pch = c(20,4)[v.lsl$lslpts], cex = 1.3)
legend(p$ext["xmin"], p$ext["ymin"] + 700, pch = c(20,4),
       legend = c("affected", "unaffected"))
```



Solution Task 2

```
library(ggplot2)
library(tidyterra)
ggplot() +
    geom_spatraster(data = ta, aes(fill=slope)) +
    geom_sf(data = v.lsl, aes(col = lslpts)) +
    theme_minimal() + scale_fill_viridis_c()
```



2.5 Package `sp` and more maps

The R package `sp` has been replaced by the easier to use and more powerful `sf` package (published by the same authors). Geostatistical R packages, however, still depend on spatial `sp` objects and sometimes require `sp` objects as input.

How to transform `sf` to `sp` and vice versa see Pebesma and Bivand, 2023. Appendix A — Older R Spatial Packages.

> Task 1
>
> Load the dataset `meuse` from package `sp`. Create a spatial object and assign the correct CRS. See `example` on the dataset's help page.

Task 2

Plot the `meuse` observation locations using `plot`. Plot the topsoil zinc concentrations with `spplot`.

Task 3

Create an interactive map displaying the zinc concentration using the `mapview` package. Change the view to aerial imagery.

Task 4

Similarly, load `meuse.grid`. Create a `sp` raster object (`SpatialPixelsDataFrame`). Explore the content by printing and plotting.

Solution Task 1

```
library(sp)
data(meuse)
summary(meuse)
```

```
       x                y                cadmium          copper      
 Min.   :178605   Min.   :329714   Min.   : 0.200   Min.   : 14.00  
 1st Qu.:179371   1st Qu.:330762   1st Qu.: 0.800   1st Qu.: 23.00  
 Median :179991   Median :331633   Median : 2.100   Median : 31.00  
 Mean   :180005   Mean   :331635   Mean   : 3.246   Mean   : 40.32  
 3rd Qu.:180630   3rd Qu.:332463   3rd Qu.: 3.850   3rd Qu.: 49.50  
 Max.   :181390   Max.   :333611   Max.   :18.100   Max.   :128.00  

      lead             zinc             elev            dist        
 Min.   : 37.0   Min.   : 113.0   Min.   : 5.180   Min.   :0.00000  
 1st Qu.: 72.5   1st Qu.: 198.0   1st Qu.: 7.546   1st Qu.:0.07569  
 Median :123.0   Median : 326.0   Median : 8.180   Median :0.21184  
 Mean   :153.4   Mean   : 469.7   Mean   : 8.165   Mean   :0.24002  
 3rd Qu.:207.0   3rd Qu.: 674.5   3rd Qu.: 8.955   3rd Qu.:0.36407  
 Max.   :654.0   Max.   :1839.0   Max.   :10.520   Max.   :0.88039  

       om          ffreq  soil    lime        landuse       dist.m      
 Min.   : 1.000   1:84   1:97   0:111   W      :50   Min.   :  10.0  
 1st Qu.: 5.300   2:48   2:46   1: 44   Ah     :39   1st Qu.:  80.0  
```

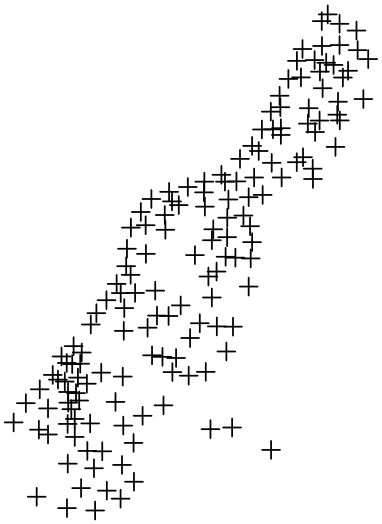
```
 Median : 6.900   3:23   3:12          Am     :22   Median : 270.0
 Mean   : 7.478                       Fw     :10   Mean   : 290.3
 3rd Qu.: 9.000                       Ab     : 8   3rd Qu.: 450.0
 Max.   :17.000                       (Other):25   Max.   :1000.0
 NA's   :2                            NA's   : 1
```

```
coordinates(meuse) <- ~x+y
proj4string(meuse) <- CRS("+init=epsg:28992")
str(meuse)
```

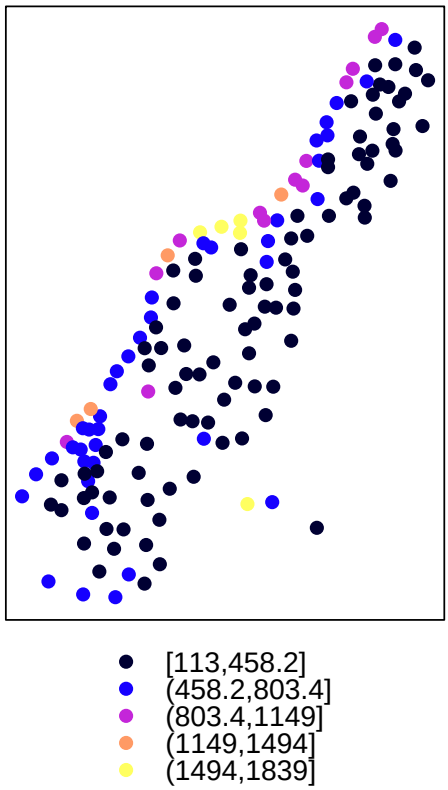
```
Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame': 155 obs. of  12 variables:
  .. ..$ cadmium: num [1:155] 11.7 8.6 6.5 2.6 2.8 3 3.2 2.8 2.4 1.6 ...
  .. ..$ copper : num [1:155] 85 81 68 81 48 61 31 29 37 24 ...
  .. ..$ lead   : num [1:155] 299 277 199 116 117 137 132 150 133 80 ...
  .. ..$ zinc   : num [1:155] 1022 1141 640 257 269 ...
  .. ..$ elev   : num [1:155] 7.91 6.98 7.8 7.66 7.48 ...
  .. ..$ dist   : num [1:155] 0.00136 0.01222 0.10303 0.19009 0.27709 ...
  .. ..$ om     : num [1:155] 13.6 14 13 8 8.7 7.8 9.2 9.5 10.6 6.3 ...
  .. ..$ ffreq  : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
  .. ..$ soil   : Factor w/ 3 levels "1","2","3": 1 1 1 2 2 2 2 1 1 2 ...
  .. ..$ lime   : Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 1 1 1 ...
  .. ..$ landuse: Factor w/ 15 levels "Aa","Ab","Ag",..: 4 4 4 11 4 11 4 2 2 15 ...
  .. ..$ dist.m : num [1:155] 50 30 150 270 380 470 240 120 240 420 ...
  ..@ coords.nrs : int [1:2] 1 2
  ..@ coords     : num [1:155, 1:2] 181072 181025 181165 181298 181307 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:155] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:2] "x" "y"
  ..@ bbox       : num [1:2, 1:2] 178605 329714 181390 333611
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "x" "y"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr "+proj=sterea +lat_0=52.1561605555556 +lon_0=5.38763888888889 +k=0
  .. .. ..$ comment: chr "PROJCRS[\"Amersfoort / RD New\",\n    BASEGEOGCRS[\"Amersfoort\",\n
```

Solution Task 2

```
plot(meuse)
```



```
spplot(meuse["zinc"])
```



Solution Task 3

Note: Change to aerial image view and inspect the situation. Since the `meuse` dataset has been sampled (1970) the river has been restructured. The samples therefore do not represent today's situation and are partly located inside the river.

```
library(mapview)
mapview(meuse, zcol = "zinc")
```

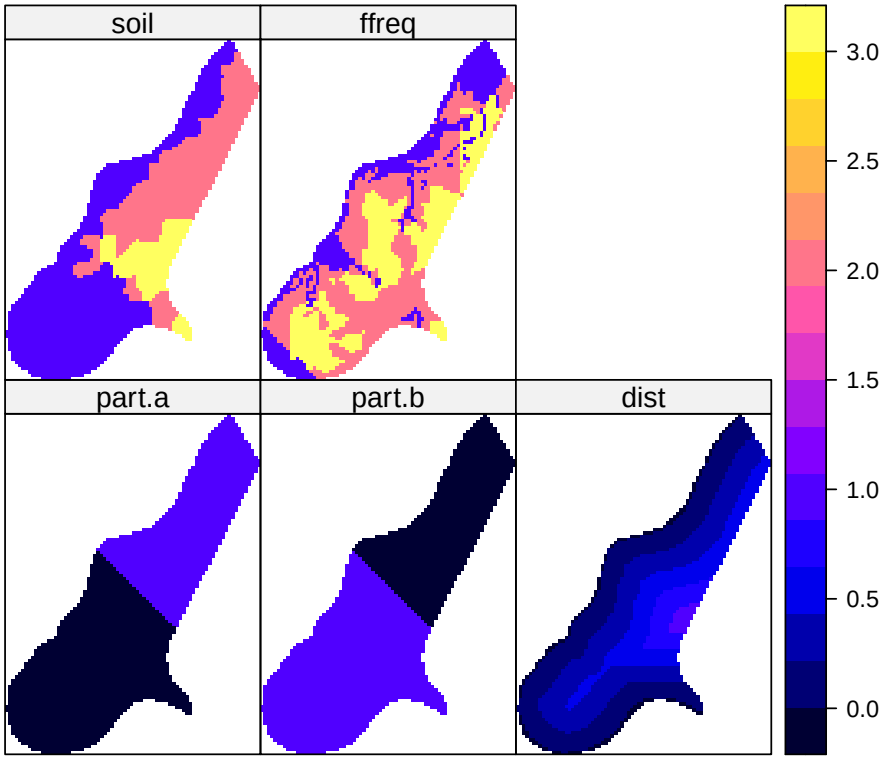
Interactive map, no output in PDF.

Solution Task 4

```
data(meuse.grid)
str(meuse.grid)
```

```
'data.frame':   3103 obs. of  7 variables:
 $ x     : num  181180 181140 181180 181220 181100 ...
 $ y     : num  333740 333700 333700 333700 333660 ...
 $ part.a: num  1 1 1 1 1 1 1 1 1 1 ...
 $ part.b: num  0 0 0 0 0 0 0 0 0 0 ...
 $ dist  : num  0 0 0.0122 0.0435 0 ...
 $ soil  : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
 $ ffreq : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
```

```
coordinates(meuse.grid) = ~x+y
proj4string(meuse.grid) <- CRS("+init=epsg:28992")
gridded(meuse.grid) = TRUE
spplot(meuse.grid)
```



3 First steps of geostatistical analysis

3.1 Exploratory analysis

We are now going to have a closer look into the elevation data contained in the package `georob`:

```
library(georob)
data(elevation)
```

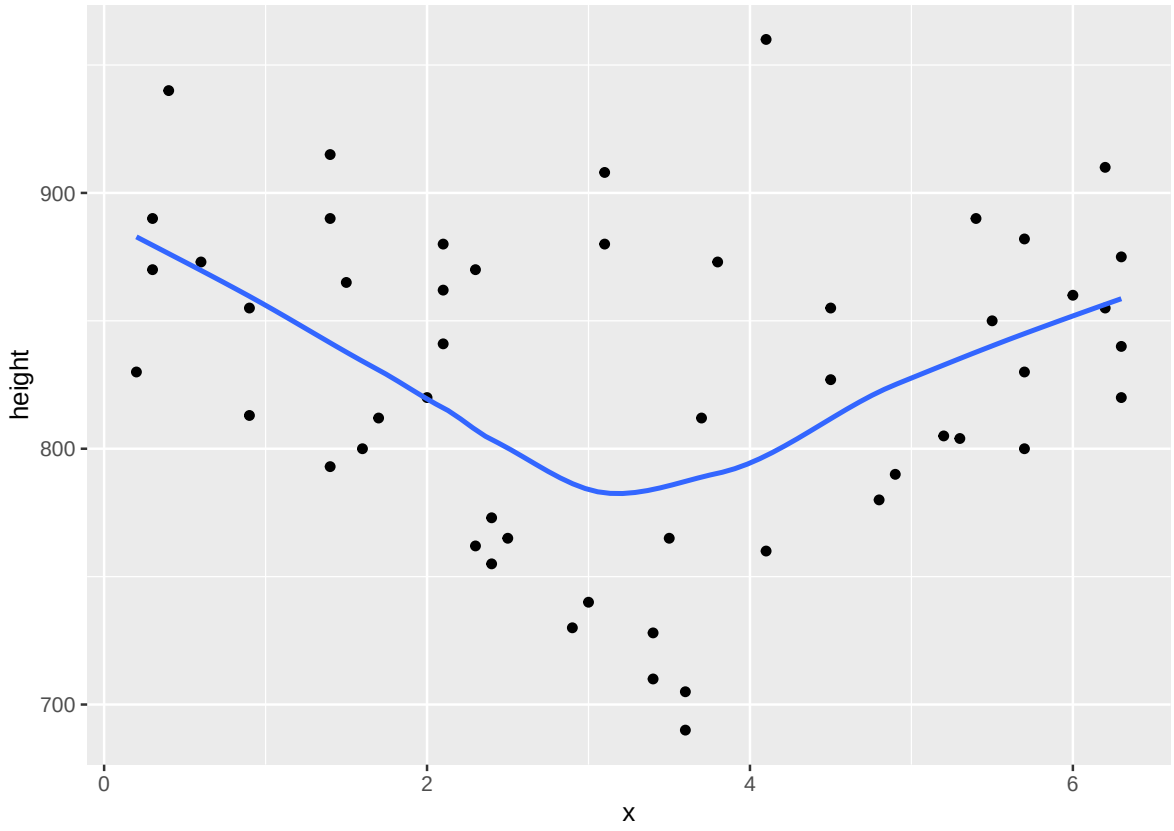
> Task 1
>
> Plot `height` against the `x`- and `y`-coordinates. Consider adding a smooth LOESS curve to the scatterplots.
> Create a "bubble plot" to explore the spatial distribution of the response variable. For the bubble plot, choose the size of the symbols such that their area depends linearly on `height`.

> Task 2
>
> Explore the spatial distributions of `height` further by the dynamic graphics of the package `rgl`.

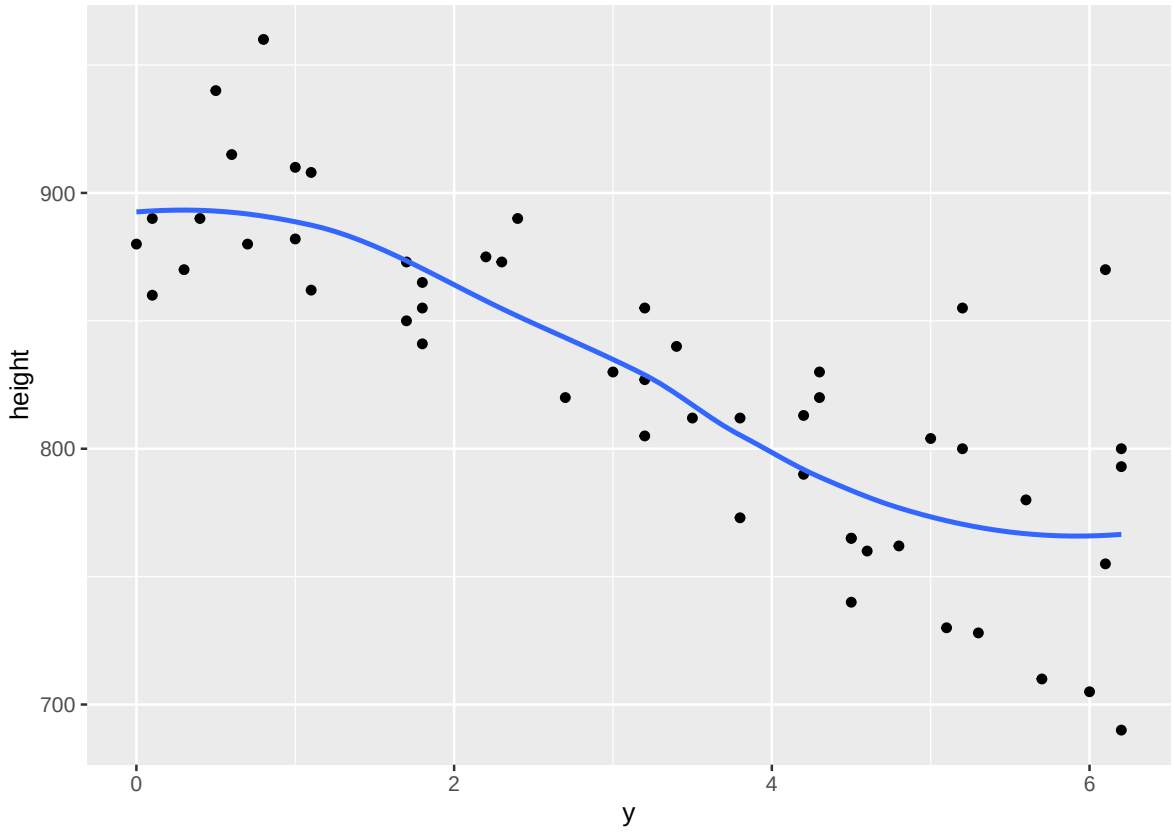
> Task 3
>
> Does `height` show some large-scale trend? If yes, how would you build a regression model to model this trend?

Solution Task 1

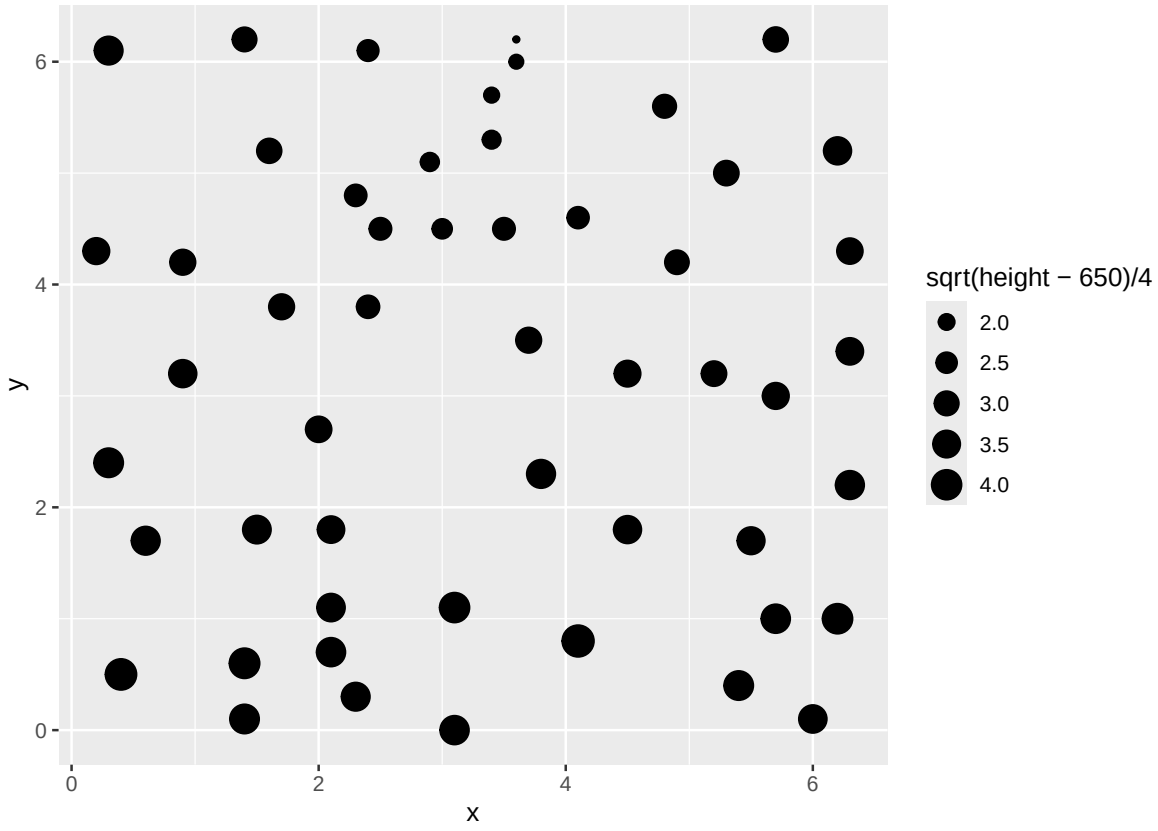
```
library(ggplot2)

ggplot(elevation, aes(x = x, y = height)) +
  geom_point() + geom_smooth(se=F)
```




```
ggplot(elevation, aes(x = y, y = height)) +
  geom_point() + geom_smooth(se=F)
```




```
ggplot(elevation, aes(x = x, y = y)) +
  geom_point(aes(size=sqrt(height-650)/4))
```



Solution Task 2

```
library(rgl)
plot3d(x=elevation$x, y=elevation$y, z=elevation$height/50,
  type="s", radius=0.1, col="orange", aspect="iso",
  xlab="x", ylab="y", zlab="height")
rglwidget()
```

Interactive 3D plot, not output in PDF.

Solution Task 3

The plots show a clear large-scale trend. `height` decreases clearly from S to N. Whereas `height` does not much depend on the `x`-coordinate along the southern border, it varies clearly with `x` in the northern part.

Hence, a linear function of `x` and `y` seems not adequate, and a second order polynomial in the coordinates is likely a suitable trend model for `height`.

3.2 Fitting a trend model

> Task 1
>
> Fit the trend model that you found in the previous problem now by ordinary least squares.

> Task 2
>
> Use customary residual diagnostics to see if the model can be improved.

> Task 3
>
> Create a bubble plot of the residuals to see if the residuals are show spatial autocorrelation.

Solution Task 1

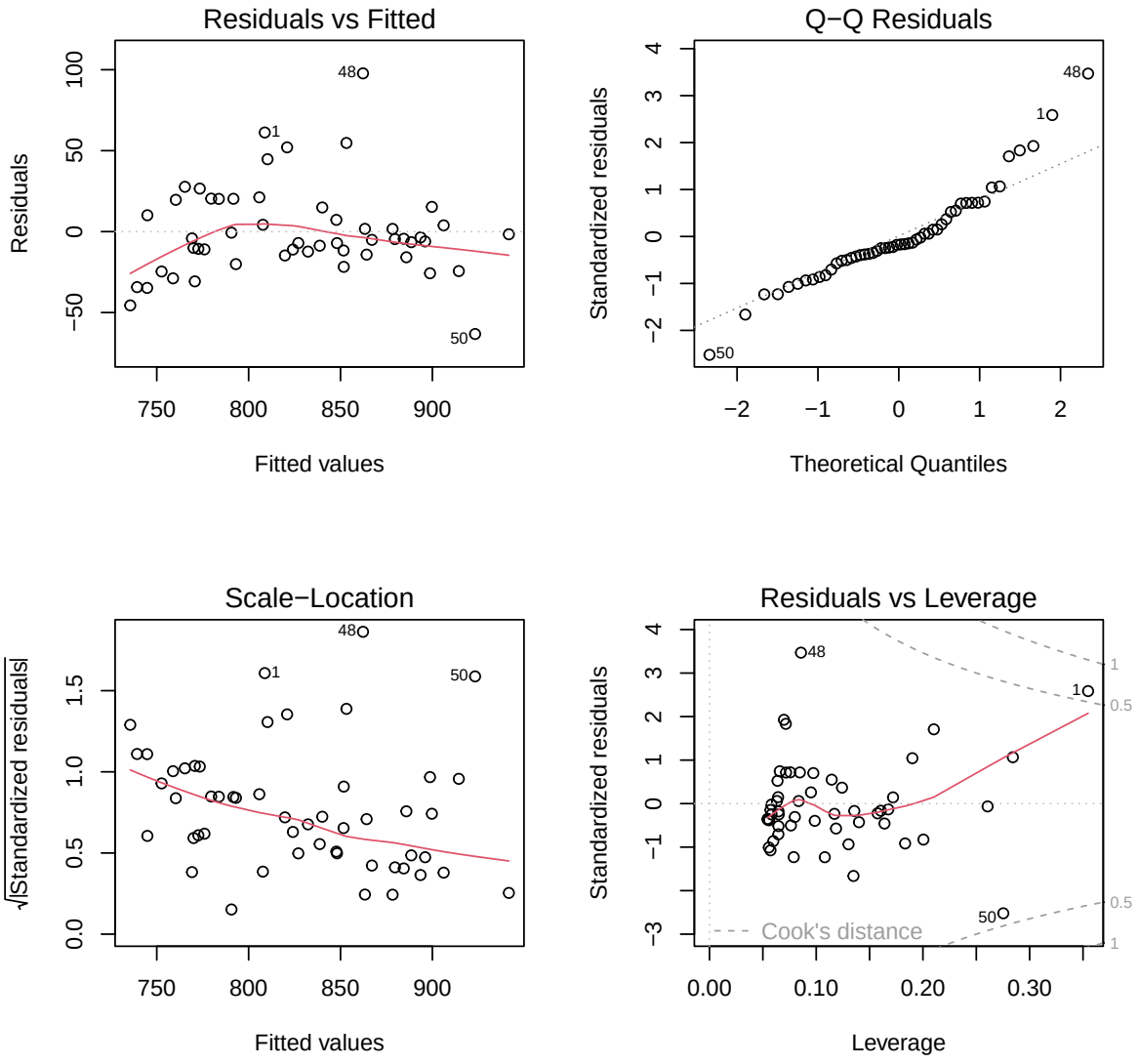
```
r.lm <- lm(height~x+y+I(x^2)+I(y^2)+x:y, elevation)
summary(r.lm)
```

```
Call:
lm(formula = height ~ x + y + I(x^2) + I(y^2) + x:y, data = elevation)

Residuals:
    Min      1Q  Median      3Q     Max
-63.251 -14.356  -4.869  15.003  97.755

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 976.3282    20.5668  47.471  < 2e-16 ***
x           -52.3832    10.0718  -5.201 4.46e-06 ***
y           -30.4004     9.1908  -3.308  0.00183 **
I(x^2)        7.3345     1.3118   5.591 1.18e-06 ***
I(y^2)        0.8681     1.2787   0.679  0.50060
x:y           0.3536     1.1653   0.303  0.76291
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 29.47 on 46 degrees of freedom
Multiple R-squared:  0.7962,    Adjusted R-squared:  0.774
F-statistic: 35.93 on 5 and 46 DF,  p-value: 8.42e-15
```

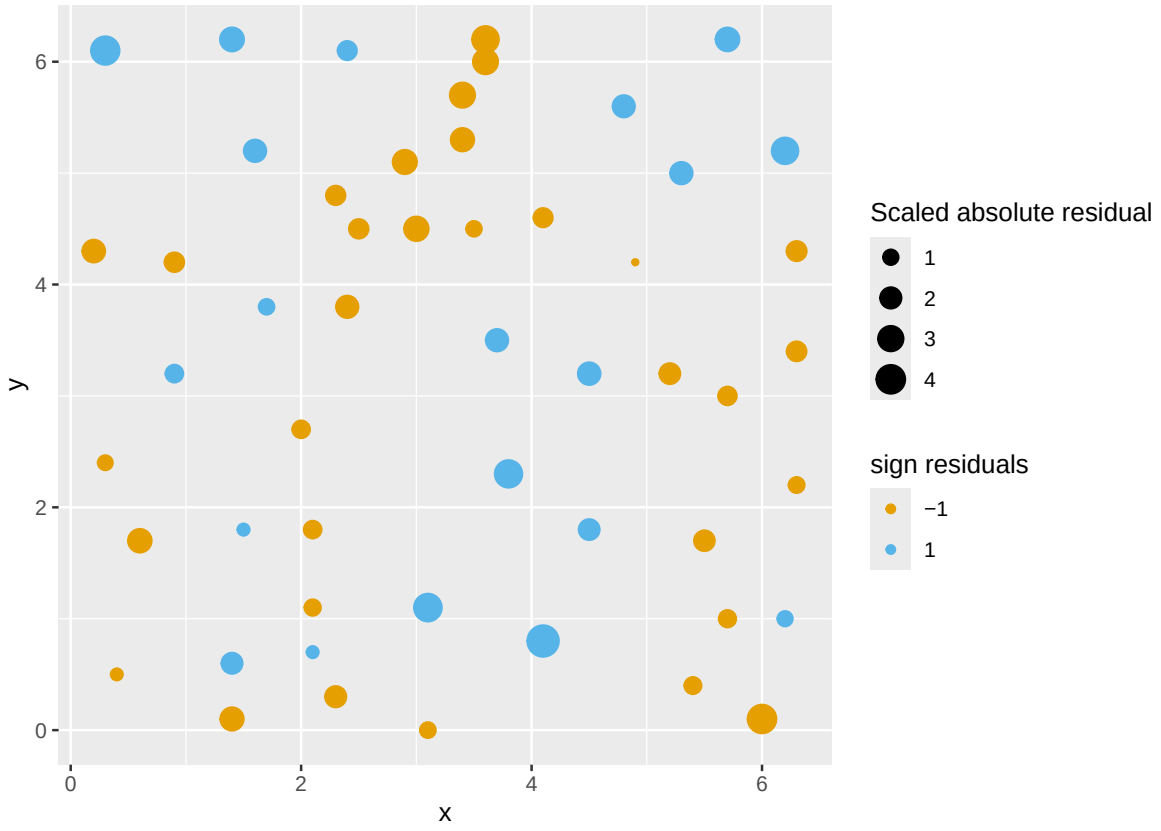
Solution Task 2

```r
par(mfrow=c(2, 2))
plot(r.lm)
```



Solution Task 3

```
ggplot(elevation, aes(x = x, y = y)) +
  geom_point(aes(size = sqrt(abs(residuals(r.lm)))/2,
      col = factor(sign(residuals(r.lm)), levels = -1:1))) +
  scale_color_manual(values = c('#E69F00', '#56B4E9', NA), name = "sign residuals") +
  labs(size = "Scaled absolute residual")
```




```
# with R base plot
# plot(y~x, elevation, cex=sqrt(abs(residuals(r.lm)))/2,
#  col=c("orange", NA, "blue")[sign(residuals(r.lm))+2])
```

Some observations are not fitted well by the quadratic trend surface, and stand out as outliers. But we ignore this for the time being. The bubble plots suggests that the residuals are auto-correlated.

3.3 Exploring and modelling auto-correlation

Task 1

Using the function `hscat()` of the package `gstat`, create lag-scatter plots of the regression residuals. To use `hscat()` you have to convert the dataframe `elevation` to a `SpatialPointsDataFrame` (package `sp`).

Task 2

Compute the sample variogram of the residuals and estimate the size of the nugget, total sill and range from the plot of the sample variogram.

Task 3

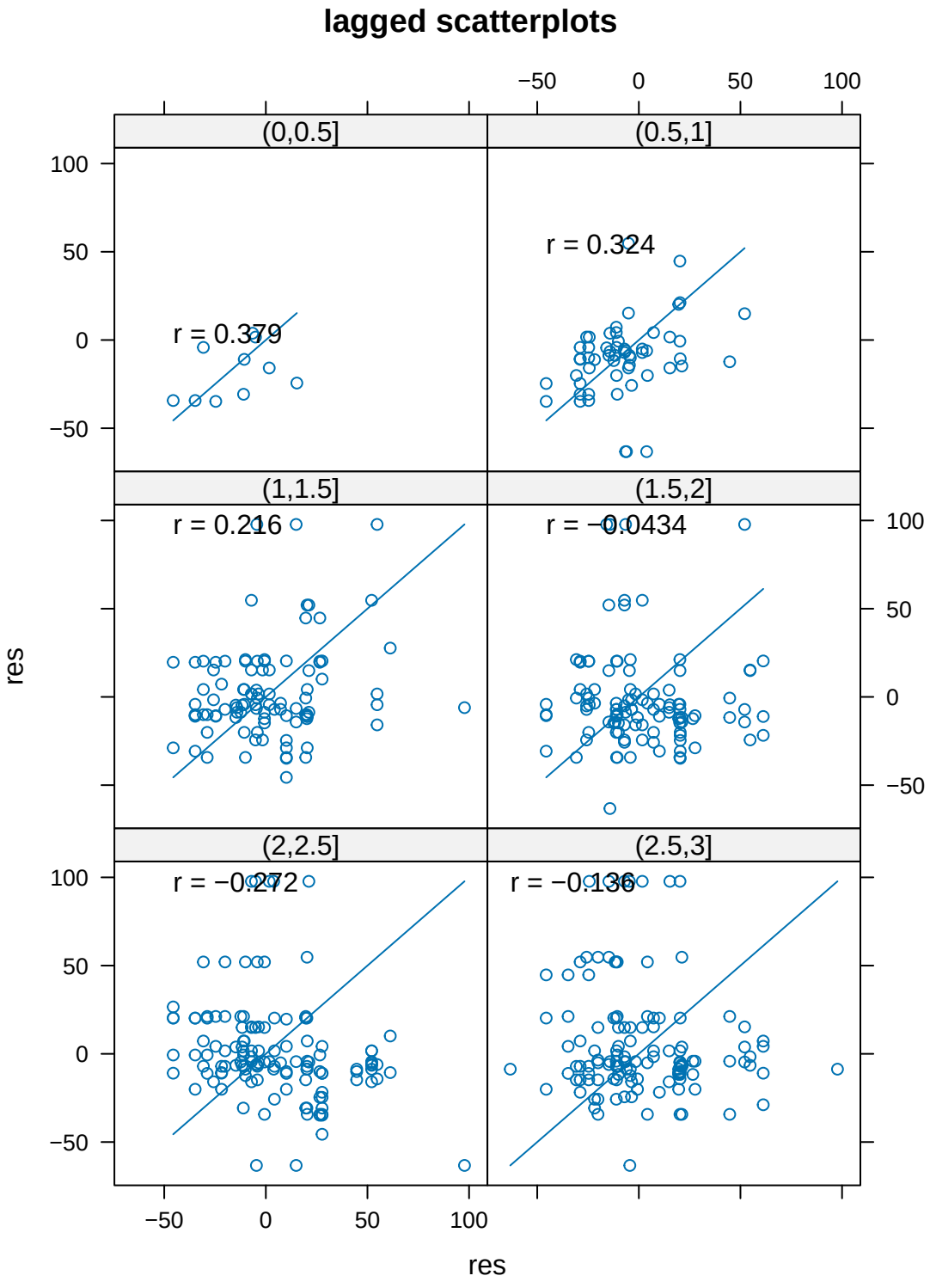
Fit a "spherical" variogram function to the sample variogram. Use your guess estimates of the variogram parameters as initial values for model fitting.
Do the fitted parameter values differ much from your guesses?

Solution Task 1

```
elevation <- st_as_sf(elevation, coords = c("x", "y"))
str(elevation)
```

```
Classes 'sf' and 'data.frame':  52 obs. of  2 variables:
 $ height  : num  870 793 755 690 800 800 730 728 710 780 ...
 $ geometry:sfc_POINT of length 52; first list element:  'XY' num  0.3 6.1
 - attr(*, "sf_column")= chr "geometry"
 - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA
  ..- attr(*, "names")= chr "height"
```

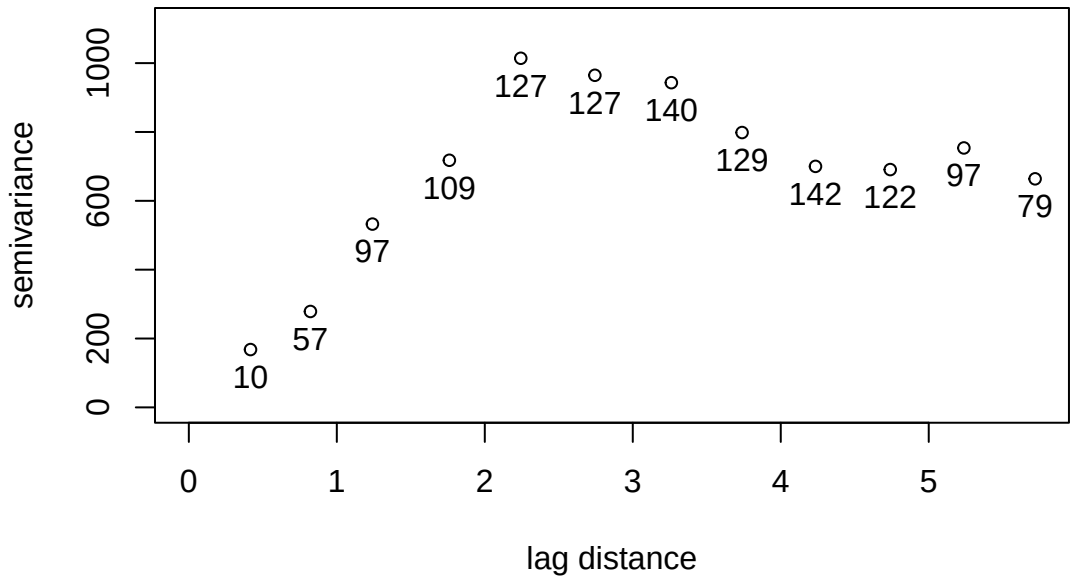
```
library(gstat)
elevation$res <- residuals(r.lm)
hscat(res~1, data=elevation, breaks=seq(0, 3, by=0.5))
```


lagged scatterplots
(0,0.5]
(0.5,1]
(1,1.5]
(1.5,2]
(2,2.5]
(2.5,3]
r = 0.379
r = 0.324
r = 0.216
r = −0.0434
r = −0.272
r = −0.136
res
res

Since the sample size is small the lag-scatterplots are not too informative. Still, we see decreasing with increasing lag size.

Solution Task 2

```
library(georob)
plot(r.v <- sample.variogram(elevation$res, locations = st_coordinates(elevation),
  estimator = "matheron", lag.dist.def = 0.5, max.lag = 6))
text(gamma~lag.dist, data = r.v, labels = npairs, pos = 1)
```



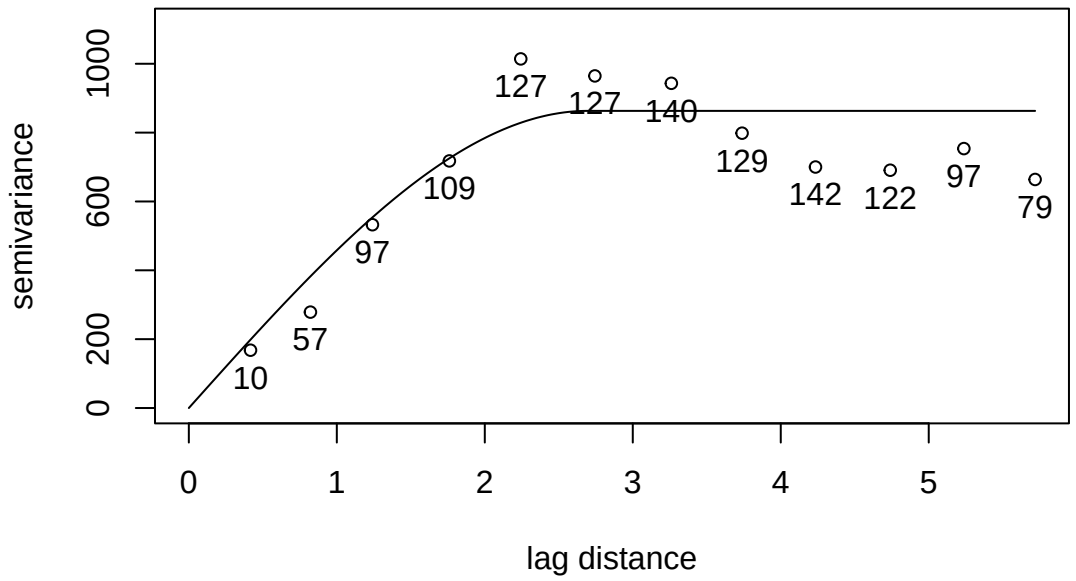
When computing the sample variogram, we have used the method-of-moments estimator (`estimator="matheron"`, `sample.variogram()` uses as default the robust `Qn` estimator). In the plot above, the points are labelled by the number of data pairs in each lag class. A rule of thumb says that only lag classes having at least 30 pairs and lag distances shorter than about half the maximum distance across the study domain should be considered when fitting a variogram model to a sample variogram. All except the shortest lag obey this rule. Clearly, the semivariance increases up to lag ≈ 2 (= range) and then levels off at a total sill of about 700; the nugget is close to zero.

Solution Task 3

```
r.v.sph <- fit.variogram.model(r.v, variogram.model="RMspheric",
  param=c(variance=700, nugget=0.01, scale=2), max.lag=5)
r.v.sph
```

```
Variogram:  RMspheric
        variance    snugget(fixed)            nugget             scale
       8.631e+02          0.000e+00         9.993e-03         2.696e+00
```

```
plot(r.v <- sample.variogram(elevation$res, locations= st_coordinates(elevation),
  estimator="matheron", lag.dist.def=0.5, max.lag=6))
text(gamma~lag.dist, data=r.v, labels=npairs, pos=1)
lines(r.v.sph)
```



The fitted variogram parameters are somewhat larger than the initial guesses.