SISMID 2022 R Notes: Mapping for Point Data

Jon Wakefield
Departments of Biostatistics and Statistics
University of Washington

2022-07-15

Overview

In these notes we will consider mapping and modeling of point data in which the (nominal) exact locations are known.

In terms of modeling we will analyze continuous responses via mixed (geostatistical) models.

Continuous responses: example

We illustrate methods for continuous data using on Zinc levels in the Netherlands.

This data set gives locations and top soil heavy metal concentrations (in ppm), along with a number of soil and landscape variables, collected in a flood plain of the river Meuse, near the village Stein in the South of the Netherlands.

Heavy metal concentrations are bulk sampled from an area of approximately 28km $\times$ 39km.

The Meuse data are in a variety of packages. We will use the version in the `sp` library.

gstat for geostatistics

We start the analysis using functions from the `gstat` library.

```
library(gstat)
library(sp)
library(variosig)  # needed for MC envelopes
library(INLA)  # need for SPDE model
library(ggplot2)
library(ggpubr)
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  # convert meuse data to a SpatialPointsDataFrame
```

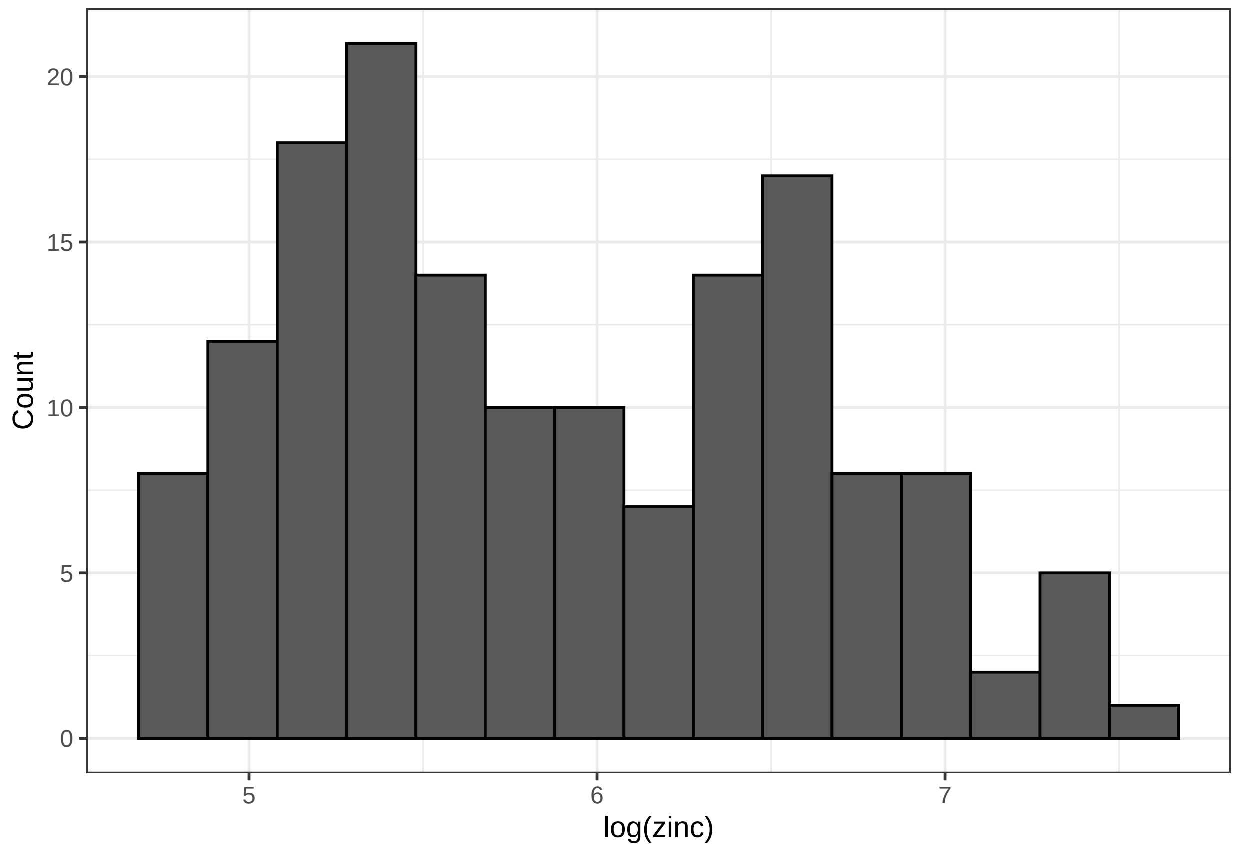
There are 155 observations (sampling locations)

Zinc data

Descriptive Plots

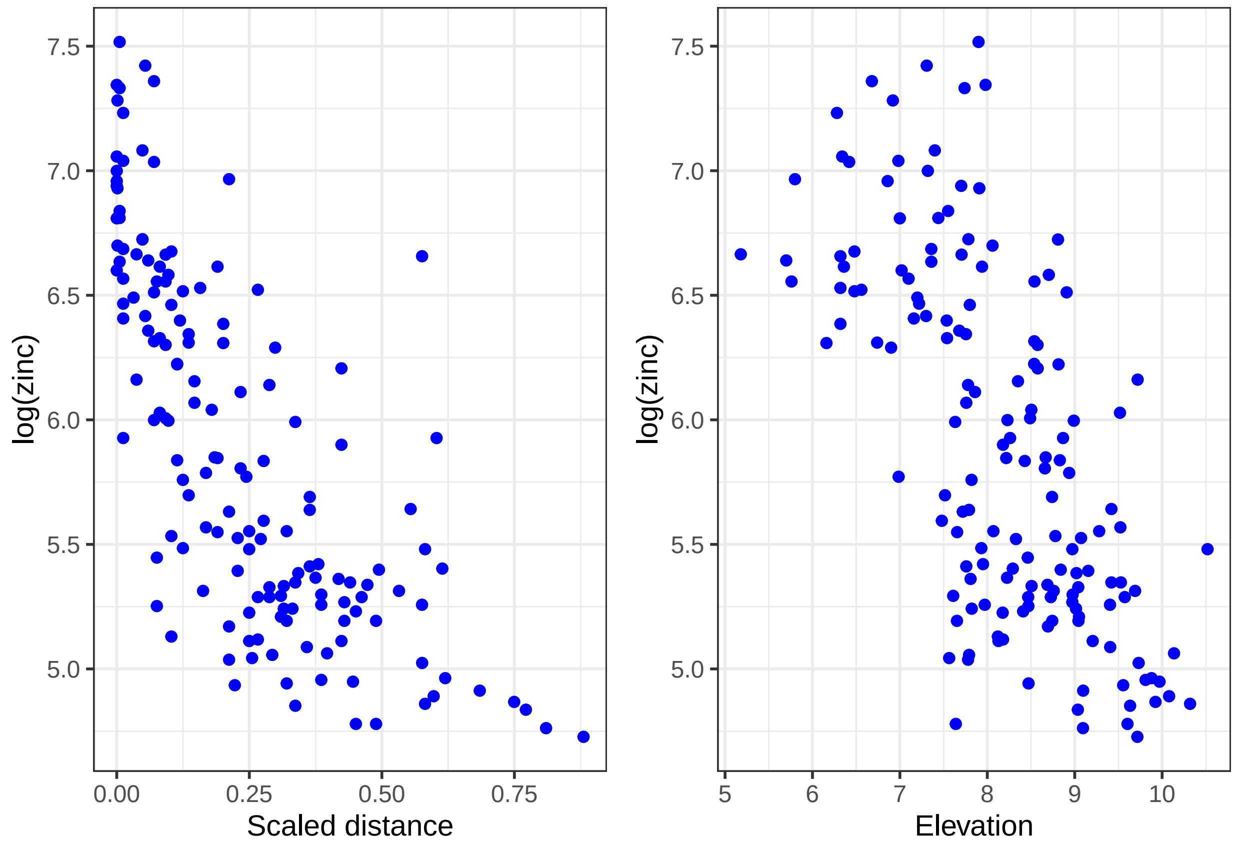
We work with log(zinc) as the distribution is more symmetric than on the original scale, and the variance more constant across levels of covariates.

```
meuse@data %>%
    ggplot(aes(log(zinc))) + geom_histogram(bins = 15, col = "black") + theme_bw() +
    ylab("Count")
```



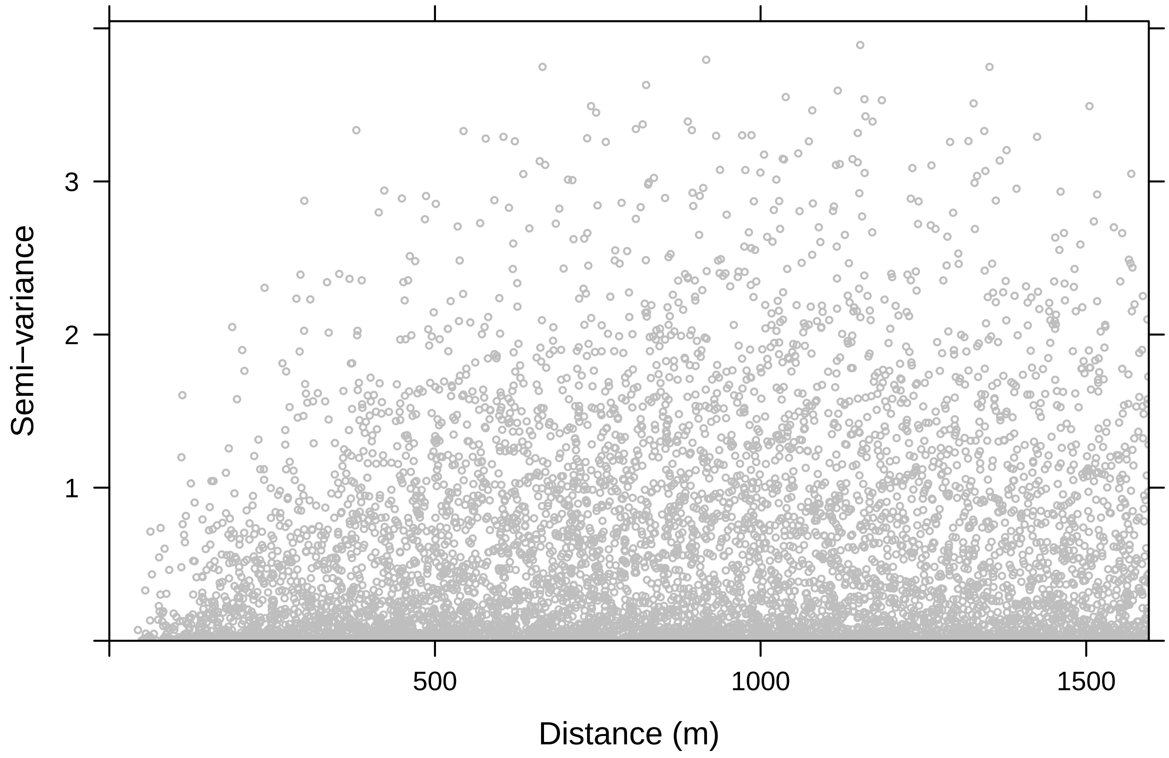
From the scatterplots of log(zinc) vs. elevation and scaled distance below, we see that log(zinc) appears to be negatively correlated with each variable.

```
dist_plot <- meuse@data %>%
    ggplot(aes(dist, log(zinc))) + geom_point(col = "blue") + xlab("Scaled distance") +
    theme_bw()

elev_plot <- meuse@data %>%
    ggplot(aes(elev, log(zinc))) + geom_point(col = "blue") + xlab("Elevation") +
    theme_bw()

ggarrange(dist_plot, elev_plot)
```



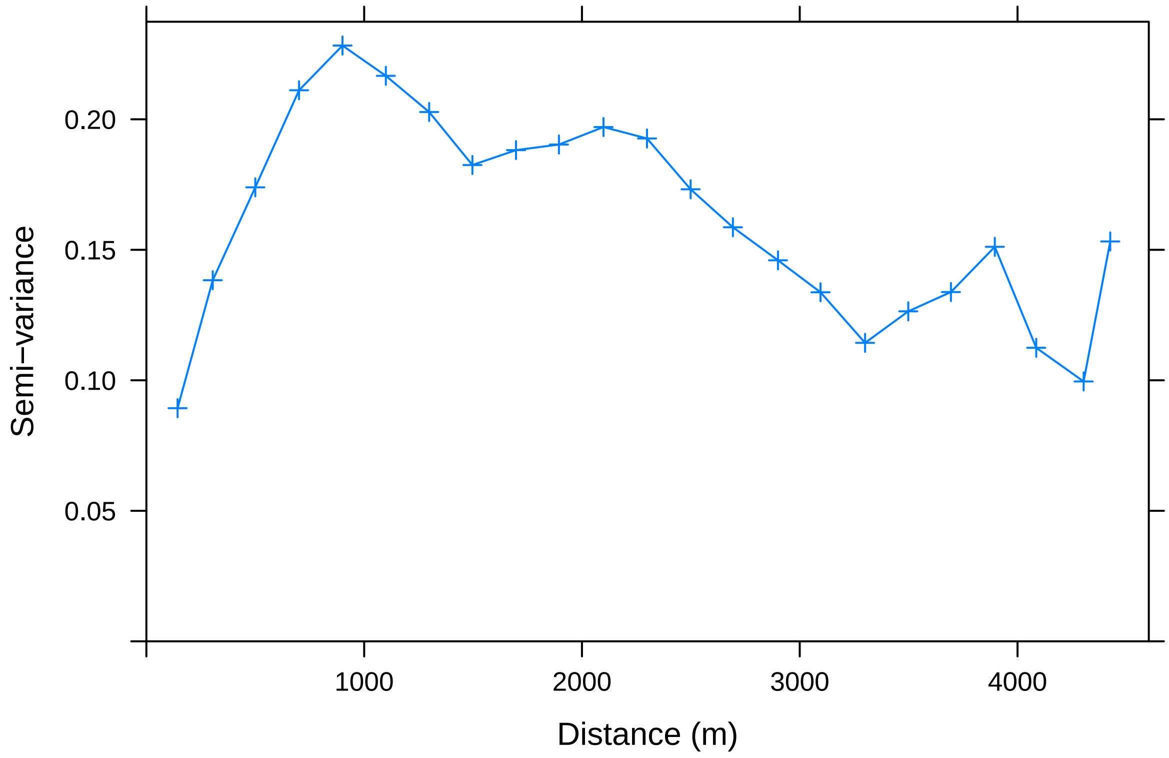
We now form a variogram cloud for log zinc, with just an intercept in the model.

```
cloudzinc <- variogram(log(zinc) ~ 1, data = meuse, cloud = TRUE)
plot(cloudzinc, ylab = "Semi-variance", xlab = "Distance (m)", col = "grey", cex = 0.4)
```



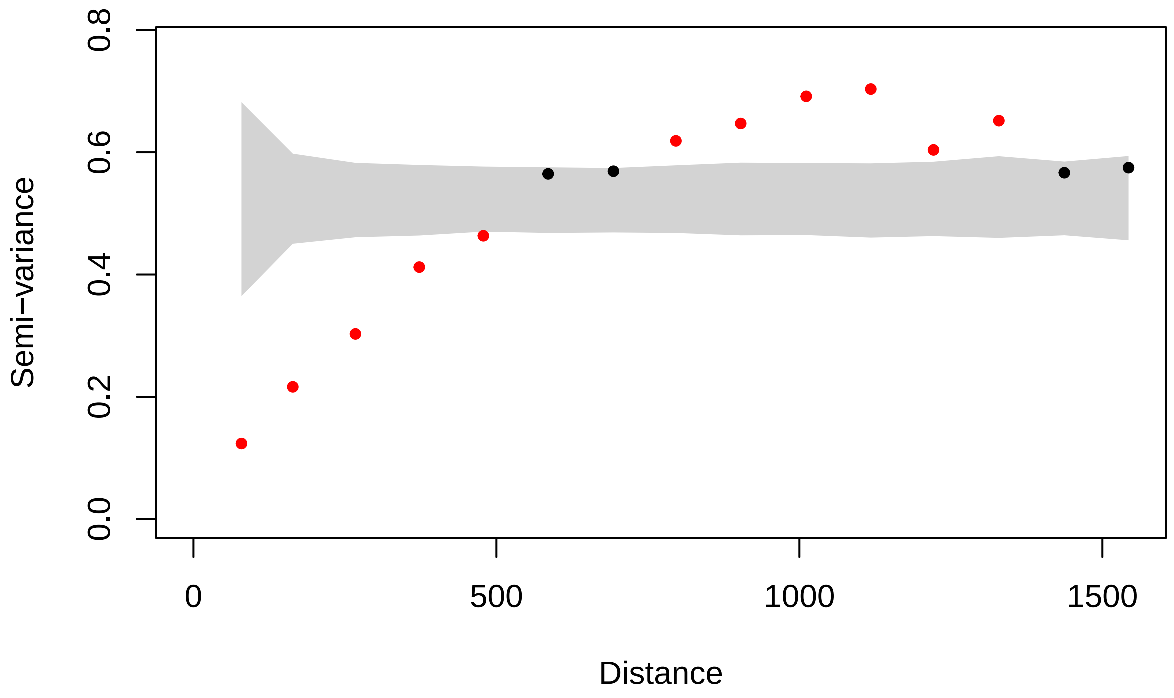
We now form a binned variogram for log zinc, with a linear trend in distance and elevation.

```
binzinc <- variogram(log(zinc) ~ elev + dist, data = meuse, cutoff = 5000, width = 200)
plot(binzinc, type = "b", pch = 3, xlab = "Distance (m)", ylab = "Semi-variance")
```



We now obtain Monte Carlo envelopes under the assumption of no spatial dependence for the variogram for log zinc with just an intercept in the model. There is clear spatial dependence present.

```
vario <- variogram(log(zinc) ~ 1, data = meuse)
geozinc.env <- envelope(vario, data = meuse, formula = log(zinc) ~ 1)
envplot(geozinc.env, ylab = "Semi-variance")
```



GAMs

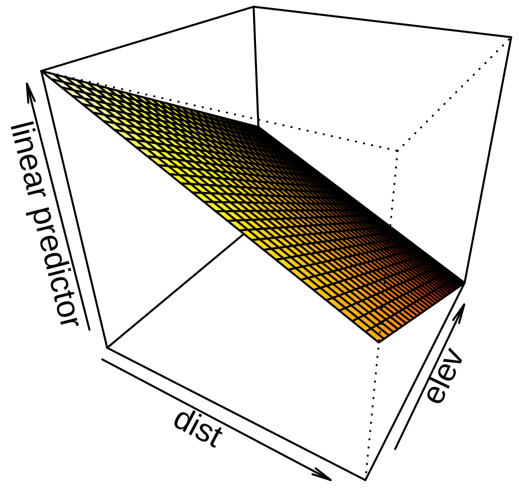
We now model the log(zinc) surface as linear in distance and elevation, and with the spatial surface modeled with a thin plate regression spline, with the smoothing parameter estimated using REML.

```
library(mgcv)
library(lattice)
library(latticeExtra)
library(RColorBrewer)


zinc.dat <- data.frame(x = meuse$x, y = meuse$y, lzinc = log(meuse$zinc),
    dist = meuse$dist, elev = meuse$elev)
gam.mod <- gam(lzinc ~ s(x, y, bs = "tp") + dist +
    elev, data = zinc.dat, method = "REML")
```

```
summary(gam.mod)
##
## Family: gaussian
## Link function: identity
##
## Formula:
## lzinc ~ s(x, y, bs = "tp") + dist + elev
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.52282    0.30738  27.727  < 2e-16 ***
## dist        -1.57105    0.62631  -2.508   0.0134 *
## elev        -0.27677    0.03361  -8.235 1.71e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##          edf Ref.df     F p-value
## s(x,y) 22.66  26.52 6.264  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.833   Deviance explained =   86%
## -REML = 67.057  Scale est. = 0.087094  n = 155
```

Below we display the fitted distance by elevation surface from the GAM output.

```
vis.gam(gam.mod, theta = 30, phi = 30)
```



We can also obtain predictions from the GAM, and plot the contour surface with observed points.

```
# create prediction grid
pred.grid <- expand.grid(seq(179000, 181000, l = 51),
    seq(330000, 332000, l = 51))
pred.dat <- data.frame(x = pred.grid[, 1], y = pred.grid[,
    2], dist = mean(meuse$dist), elev = mean(meuse$elev))
zinc.pred <- predict.gam(gam.mod, pred.dat, type = "terms")[,
    3]
# plot the smoother for log(zinc)
print(contourplot(zinc.pred ~ pred.grid[, 1] * pred.grid[,
    2], xlab = "", ylab = "", main = "", colorkey = T,
    scales = list(draw = F), pretty = T, region = T,
    par.settings = custom.theme(region = brewer.pal(9,
        "Greys"), bg = "grey80")))
# add the observed points
trellis.focus("panel", 1, 1, highlight = FALSE)
lpoints(zinc.dat[, 1], zinc.dat[, 2], pch = 19, col = "red",
    cex = 0.4)
## NULL
```

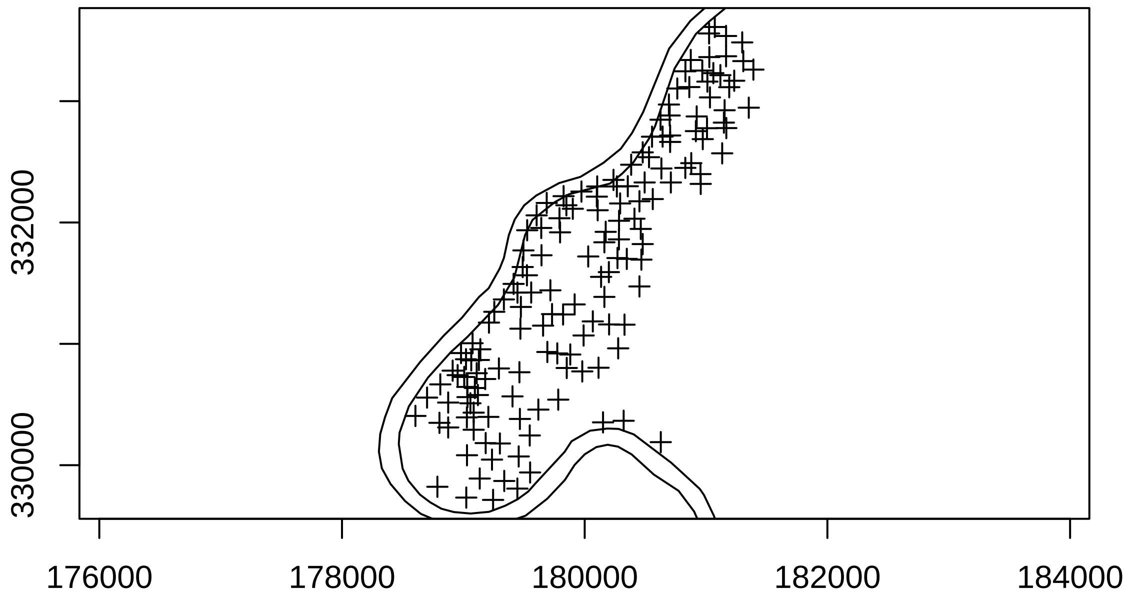
Analyzing zinc data using geostat functions

The sp package functions can make full use of the GIS capabilities of R more readily.

```
rm(list = ls())

pal <- function(n = 9) {
    brewer.pal(n, "Reds")
}

data(meuse)
coords <- SpatialPoints(meuse[, c("x", "y")])
meuse1 <- SpatialPointsDataFrame(coords, meuse)
data(meuse.riv)
river_polygon <- Polygons(list(Polygon(meuse.riv)),
    ID = "meuse")
rivers <- SpatialPolygons(list(river_polygon))
coordinates(meuse) = ~x + y
```

Below we plot the sampling locations from the zinc dataset.

```
plot(meuse1, axes = T)
plot(rivers, add = T)
```

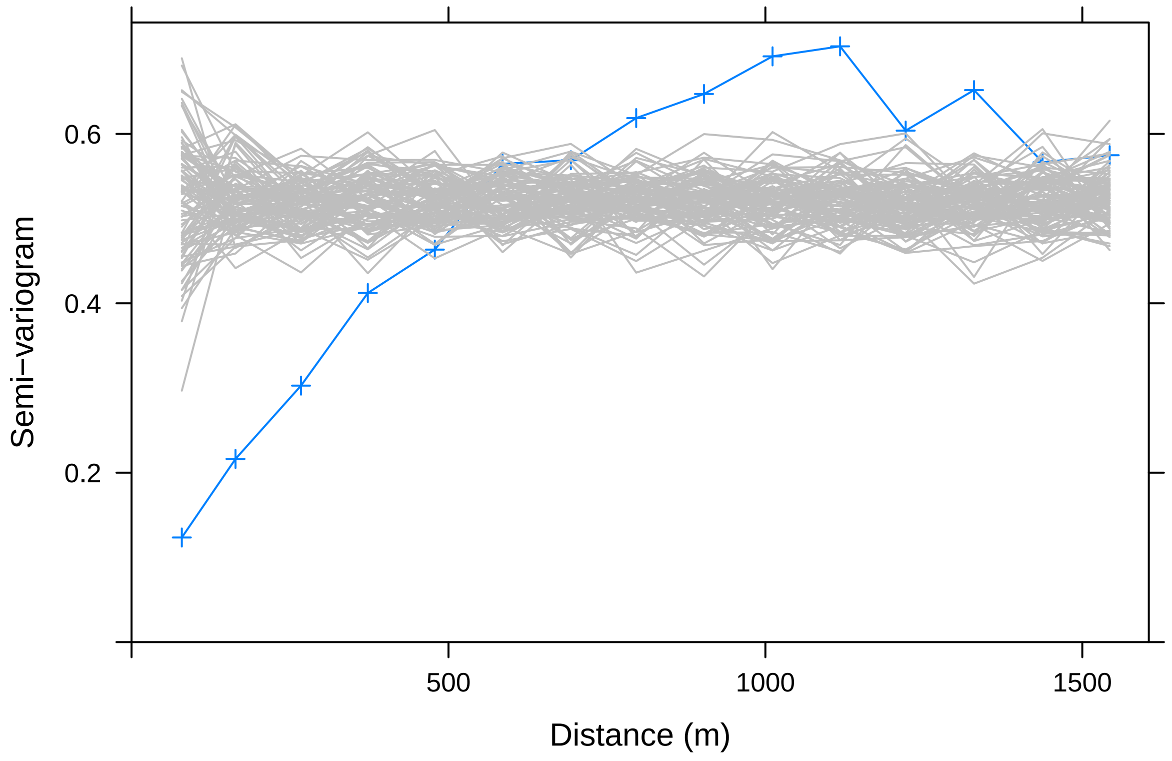


We can create additional variograms for the data including sample sizes.

```
cld <- variogram(log(zinc) ~ 1, meuse, cloud = TRUE)
svgm <- variogram(log(zinc) ~ 1, meuse)
d <- data.frame(gamma = c(cld$gamma, svgm$gamma),
    dist = c(cld$dist, svgm$dist),
    id = c(rep("cloud", nrow(cld)), rep("sample variogram", nrow(svgm)))
    )
xyplot(gamma ~ dist | id, d,
    scales = list(y = list(relation = "free",
      #ylim = list(NULL, c(-.005,0.7)))),
      limits = list(NULL, c(-.005,0.7)))),
    layout = c(1, 2), as.table = TRUE,
    panel = function(x,y, ...) {
        if (panel.number() == 2)
            ltext(x+10, y, svgm$np, adj = c(0,0.5),cex=.4) #$
        panel.xyplot(x,y,...)
    },
    xlim = c(0, 1590),
    cex = .5, pch = 3
)
```

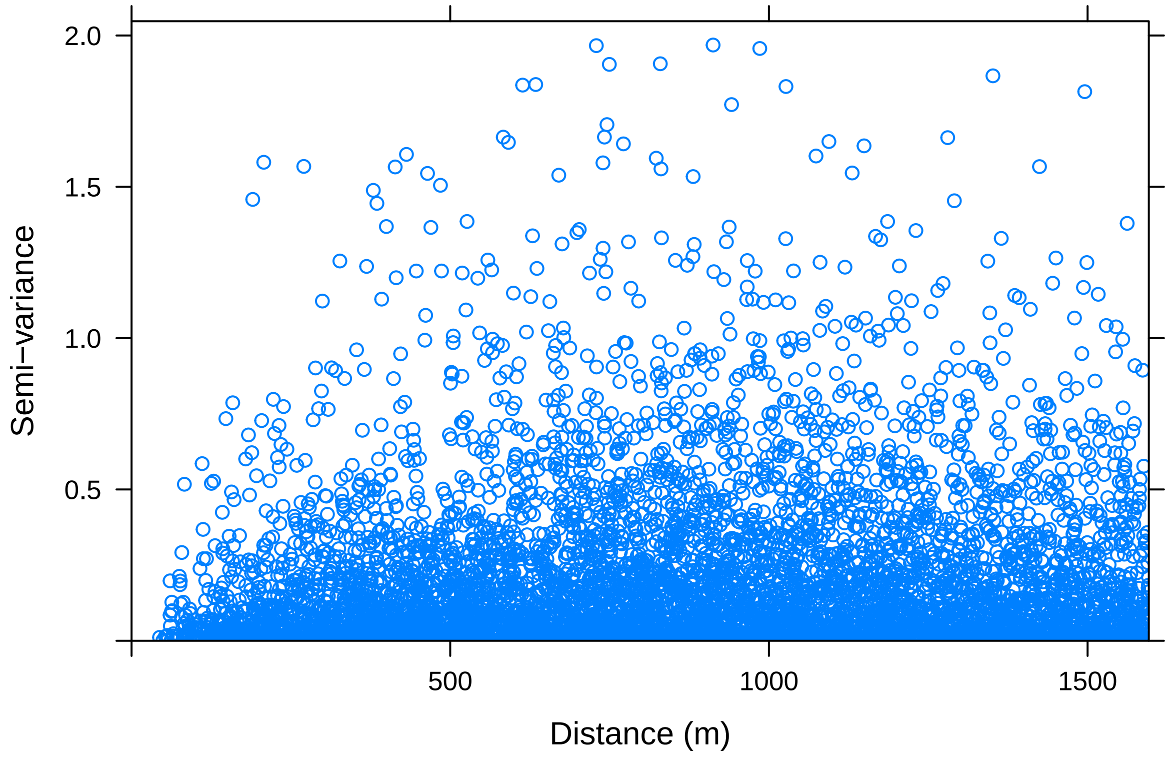
Rather than use the `variosig` package to obtain Monte Carlo envelopes, we can do this by hand. We simulate 100 datasets with random relabeling of points, and then form variograms for each.

```
v <- variogram(log(zinc) ~ 1, meuse)
plot(v, type = "b", pch = 3, xlab = "Distance (m)",
    ylab = "Semi-variance")
fn = function(n = 100) {
    for (i in 1:n) {
        meuse$random = sample(meuse$zinc)
        v = variogram(log(random) ~ 1, meuse)
        trellis.focus("panel", 1, 1, highlight = FALSE)
        llines(v$dist, v$gamma, col = "grey")
        trellis.unfocus()
    }
}
fn()
```



We now form a variogram cloud for log zinc, with a linear trend in distance and elevation.

```
cld2 <- variogram(log(zinc) ~ dist + elev, meuse, cloud = TRUE)
plot(cld2, ylab = "Semi-variance", xlab = "Distance (m)")
```

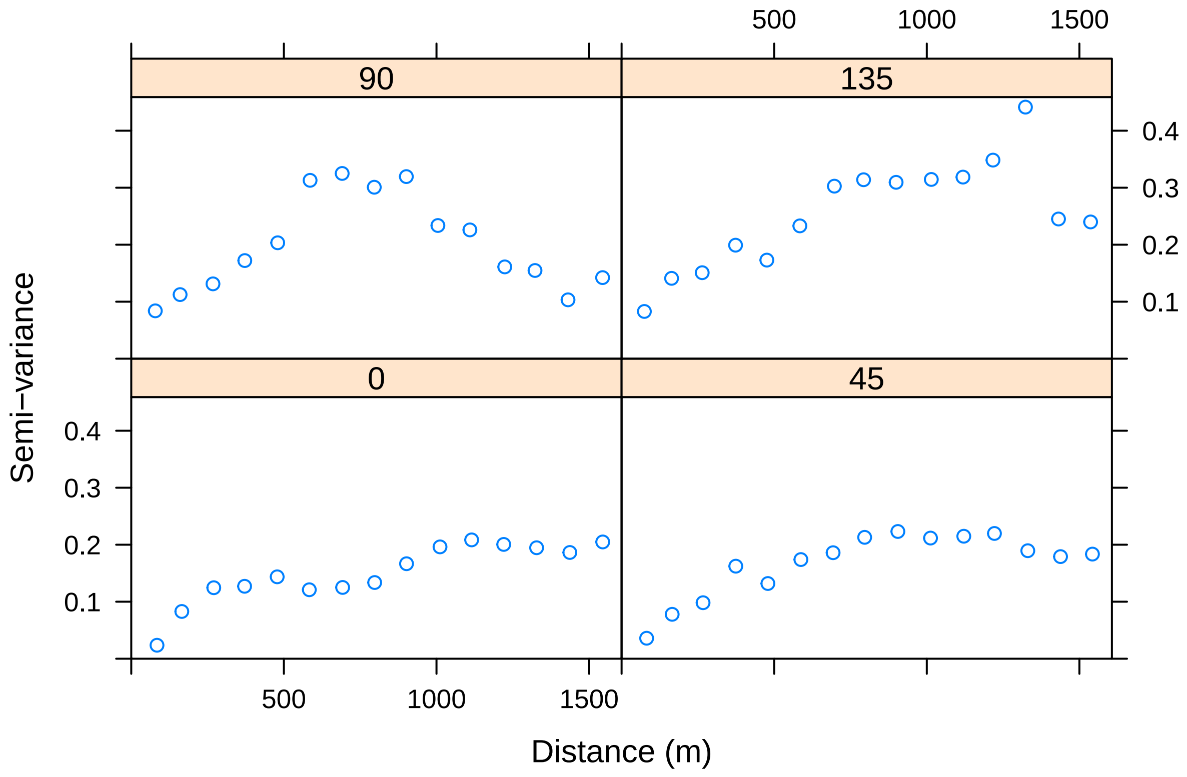


We form 4 variograms with data taken from different directions, with 0 and 90 corresponding to north and east, respectively.

Note that 0 is the same as 180.

```
dircld <- variogram(log(zinc) ~ dist + elev, meuse, alpha = c(0, 45, 90, 135))
```

```
plot(dircld, xlab = "Distance (m)", ylab = "Semi-variance")
```



We examine scatterplots of points within different distances of each other. This is another way of assessing whether spatial dependence exists.

```
hscat(log(zinc) ~ 1, meuse, c(0, 80, 120, 250, 500, 1000), cex = 0.1)
```

Other capabilities in gstat

See

- `fit.variogram` for estimation from the variogram
- `krige` (and associated functions) for Kriging,
- `vgm` generates variogram models

SPDE model

We illustrate kriging via SPDE using data on log(zinc) levels in the `meuse` dataset.

```
zincdf = data.frame(y = log(meuse$zinc), locx = meuse$x,
    locy = meuse$y, elev = meuse$elev)
```

Mesh construction The mesh is the discretization of the domain (study area). The domain is divided up into small triangles.

Can use the function `meshbuilder()` to learn about mesh construction.

The function `inla.mesh.2d()` requires at least 2 of the following 3 arguments to run

– `loc` or `loc.domain`: the function requires informations about the spatial domain given by spatial points or given by the domain extent.

– `max.edge`: the maximum edge length must be specified. If it is a two-dimensional vector then the first component is for the internal and the second for the part outside the boundary. Note that it uses the same scale unit as the coordinates.

Optional arguments:

– `offset`: specifies how much the domain will be extended in the outer and inner part. If negative it is interpreted as a factor relative to the approximate data diameter. If positive it is the extension distance on same scale unit to the coordinates provided.

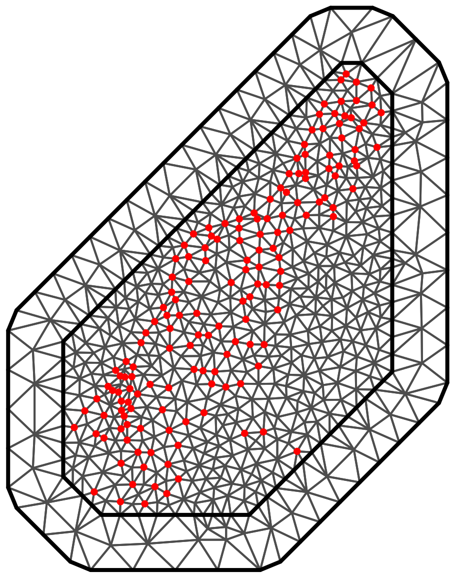
– `cutoff`: it specifies the minimum distance allowed between points. It means that if the distance between two points is less than the supplied value then they are replaced by a single vertex. It is very useful in case of clustered data points because it avoids building many small triangles arround clustered points.

– `min.angle`: it specifies the minimum internal angle of the triangles. This could be a two-dimensional vector with the same meaning as previously. We would like to have a mesh with triangles as regular as possible.

By specifying `loc` we obtain a mesh with observations lying at the vertices.

```
max.edge = 200
mesh <- inla.mesh.2d(loc = zincdf[, c("locx", "locy")],
    offset = c(100, 500), max.edge = c(max.edge, max.edge *
        3))
plot(mesh, asp = 1, main = "")
points(zincdf[, c("locx", "locy")], col = "red", cex = 0.4)
```

We visualize the mesh below.



To connect the measurement locations to the mesh representation, the A-matrix is needed. We create the A-matrix below.

The observed data lie on the vertices.

```
A = inla.spde.make.A(mesh = mesh, loc = data.matrix(zincdf[,
    c("locx", "locy")]))
dim(A)
## [1] 155 683
table(as.numeric(A))
##
##      0      1
## 105710    155
table(rowSums(A > 0))  # 155 values of 1
##
##   1
## 155
# Every point is at a mesh vertex, so each line
# on the projector matrix has exactly one
# non-zero mesh element A[1,]
```

We now create *the stack*. The stack is a complicated way of supplying the data (and covariates and effects) to INLA. For more complex spatial models, the stack is incredibly helpful, as the alternative is worse (you would have to construct the total model A matrix by hand). The stack allows different matrices to be combined (in more complex problems).

```
Xcov = data.frame(intercept = 1, elev = zincdf$elev)
# - expands the factor covariates
Xcov = as.matrix(Xcov)
colnames(Xcov)
## [1] "intercept" "elev"
```

See `?inla.stack` for lots of examples of the flexibility.

```
stack <- inla.stack(tag='est',
                    # - Name (nametag) of the stack
                    # - Here: est for estimating
                    data=list(y=zincdf$y),
                    effects=list(
                    # - The Model Components
                    s=1:mesh$n,
                    Xcov=Xcov),
                    # - The second is all fixed effects
                    A = list(A, 1)
                    # - First projector matrix is for 's'
                    # - second is for 'fixed effects'
                    )
```

The name `s` is arbitrary, but it must correspond to the letter we use in the `formula` (later).

We specify PC priors for the spatial SD and the spatial range.

```
prior.median.sd = 0.07
prior.median.range = 2000
# diff(range(mesh$loc[, 1]))/2 for range and
# sd(df$y)/10 for sd These are somewhat
# arbitrary, in general, thought is required!
spde = inla.spde2.pcmatern(mesh, alpha = 2, prior.range = c(prior.median.range,
    0.5), prior.sigma = c(prior.median.sd, 0.5), constr = T)
```

Now we specify the model – the intercept is in `Xcov` so we use `-1` in the formula.

```
formula = y ~ -1 + Xcov + f(s, model = spde)
prior.median.gaus.sd = 1  # Prior for measurement error
family = "gaussian"
control.family = list(hyper = list(prec = list(prior = "pc.prec",
    fixed = FALSE, param = c(prior.median.gaus.sd,
        0.5))))
```

We finally fit the SPDE model below.

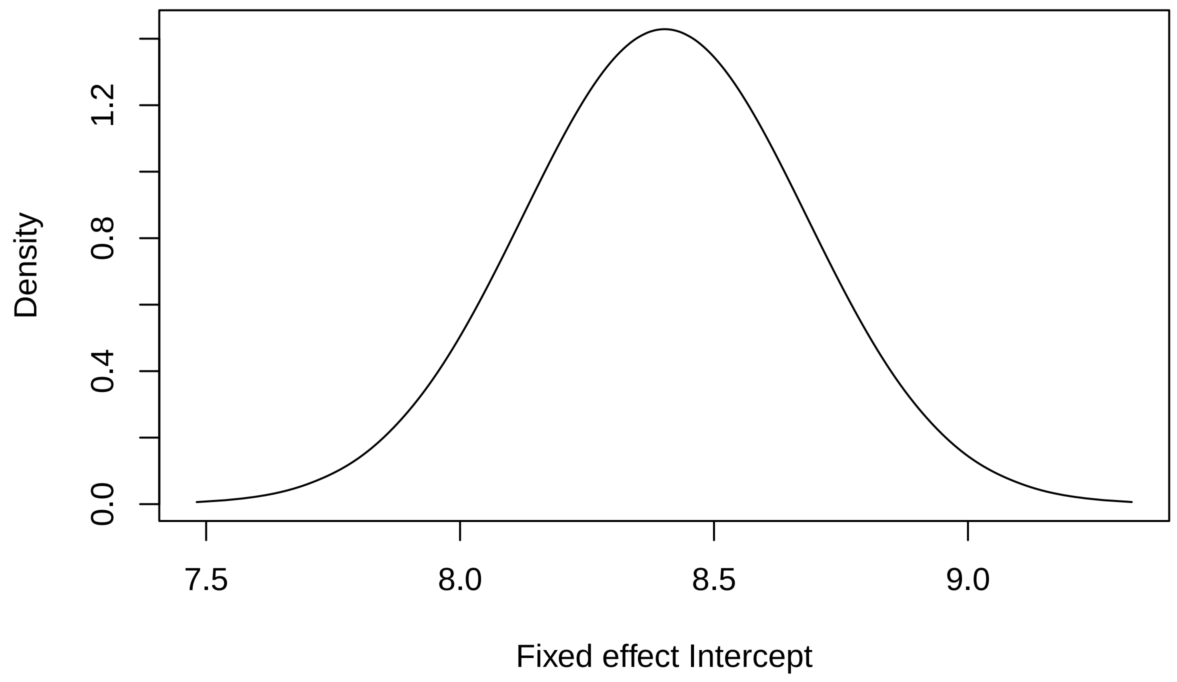
```
res <- inla(formula, data=inla.stack.data(stack,spde=spde),
            control.predictor=list(A = inla.stack.A(stack), compute=T),
            # compute=T to get posterior for fitted values
            family = family,
            control.family = control.family,
            #control.compute = list(config=T, dic=T, cpo=T, waic=T),
            # if Model comparisons wanted
            control.inla = list(int.strategy='eb'),
            # - faster computation
            #control.inla = list(int.strategy='grid'),
            # - More accurate integration over hyper-parameters
            verbose=F)
```

See `?inla.spde2.result` for extracting results.

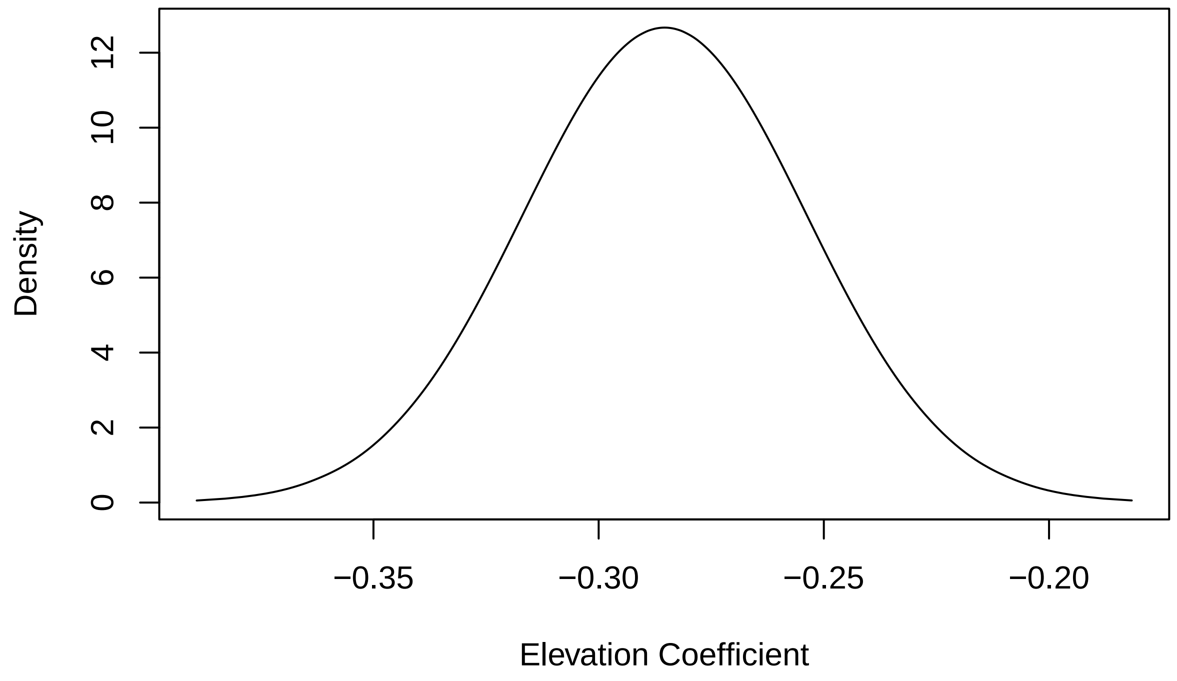
```
summary(res)
##
## Call:
##    c("inla(formula = formula, family = family, data =
##    inla.stack.data(stack, ", " spde = spde), verbose = F,
##    control.predictor = list(A = inla.stack.A(stack), ", " compute = T),
##    control.family = control.family, control.inla = list(int.strategy =
##    \"eb\"))" )
## Time used:
##     Pre = 3.27, Running = 1.14, Post = 0.269, Total = 4.67
## Fixed effects:
##         mean    sd 0.025quant 0.5quant 0.975quant   mode kld
## Xcov1  8.403 0.279      7.855    8.403      8.951  8.403   0
## Xcov2 -0.285 0.031     -0.347   -0.285     -0.224 -0.285   0
##
## Random effects:
##   Name     Model
##     s SPDE2 model
##
## Model hyperparameters:
##                                             mean      sd 0.025quant 0.5quant
## Precision for the Gaussian observations   24.988   7.740     13.472   23.744
## Range for s                             1217.587 323.211    695.449 1181.074
## Stdev for s                                0.666   0.116      0.468    0.656
##                                         0.975quant     mode
## Precision for the Gaussian observations     43.574   21.467
## Range for s                               1957.032 1111.596
## Stdev for s                                  0.921    0.637
##
## Marginal log-Likelihood:  -88.86
## Posterior summaries for the linear predictor and the fitted values are computed
## (Posterior marginals needs also 'control.compute=list(return.marginals.predictor=TRUE)')
```

```
tmp = inla.tmarginal(function(x) x, res$marginals.fixed[[1]])
plot(tmp, type = "l", xlab = "Fixed effect Intercept",
    ylab = "Density")
```

Visual summarization

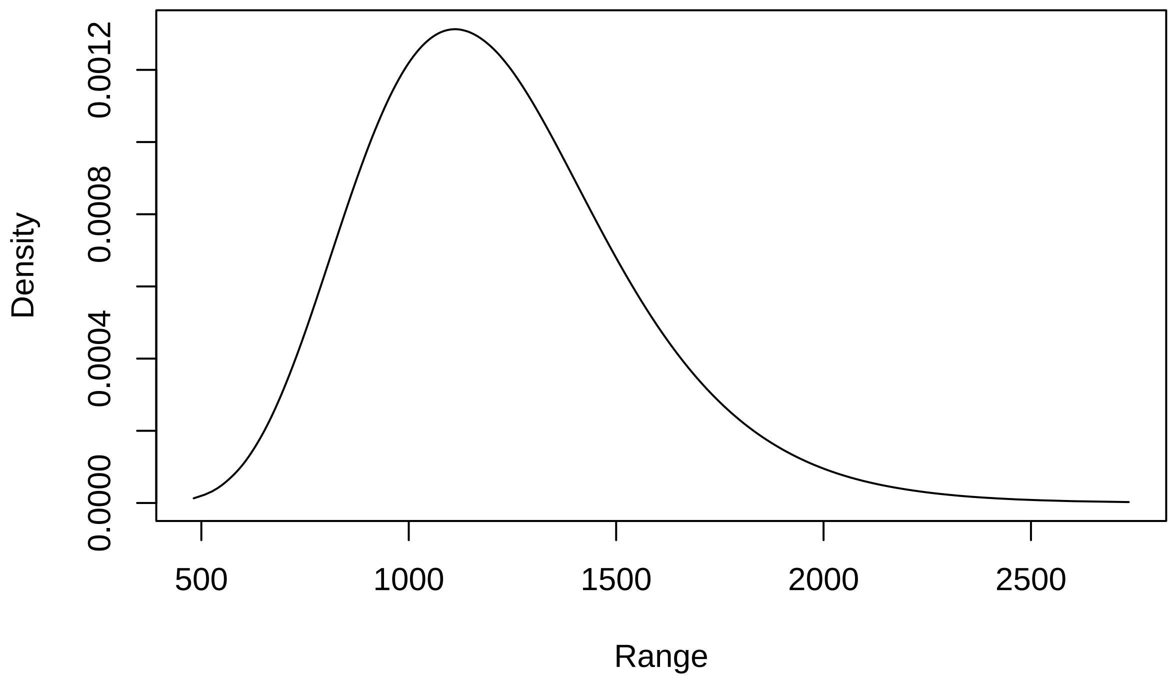


```
tmp = inla.tmarginal(function(x) x, res$marginals.fixed[[2]])
plot(tmp, type = "l", xlab = "Elevation Coefficient",
    ylab = "Density")
```



We plot summaries of the marginal posteriors for hyperparameters below.

```
range = inla.tmarginal(function(x) x, res$marginals.hyperpar[[2]])
plot(range, type = "l", xlab = "Range", ylab = "Density")
```

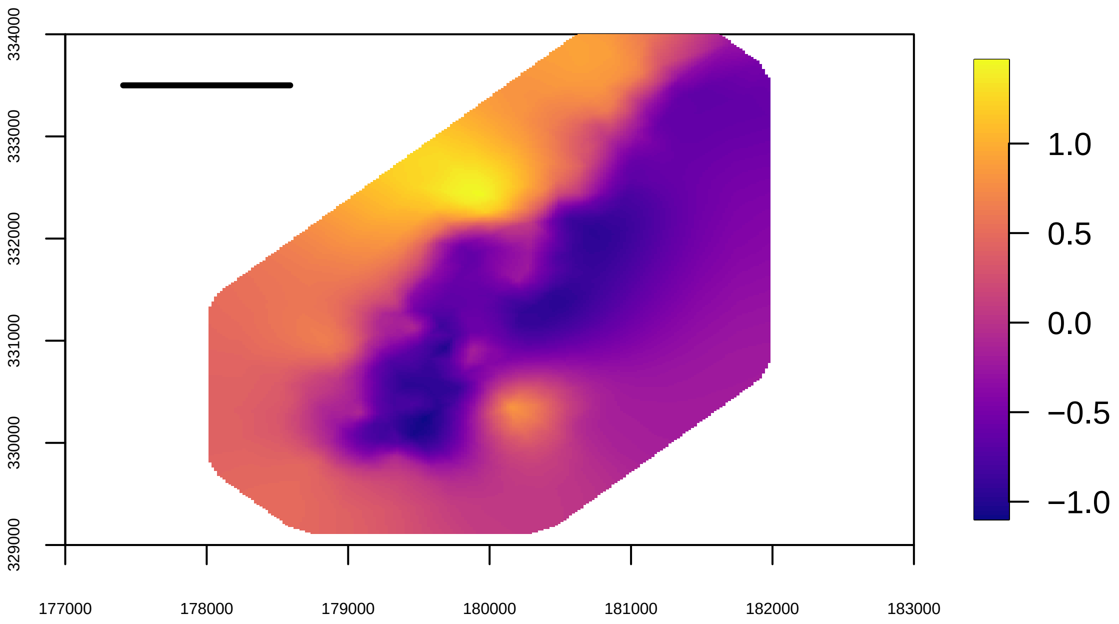


We define a function for plotting spatial fields for this application.

```
library(fields)  # needed for image.plot() function
local.plot.field = function(field, mesh, xlim = c(177000,
    183000), ylim = c(329000, 334000), ...) {
    stopifnot(length(field) == mesh$n)
    # - error when using the wrong mesh
    proj = inla.mesh.projector(mesh, xlim = xlim, ylim = ylim,
        dims = c(300, 300))
    # Project from the mesh onto a 300x300
    # plotting grid using whatever is fed to the
    # function. For example, it could be the
    # posterior mean, or draw from the posterior,
    # or fitted values
    field.proj = inla.mesh.project(proj, field)
    # Do the projection by taking a convex
    # combination (with up to 3 elements) from
    # the values on the vertices
    image.plot(list(x = proj$x, y = proj$y, z = field.proj),
        xlim = xlim, ylim = ylim, col = plasma(101),
        ...)
}
```

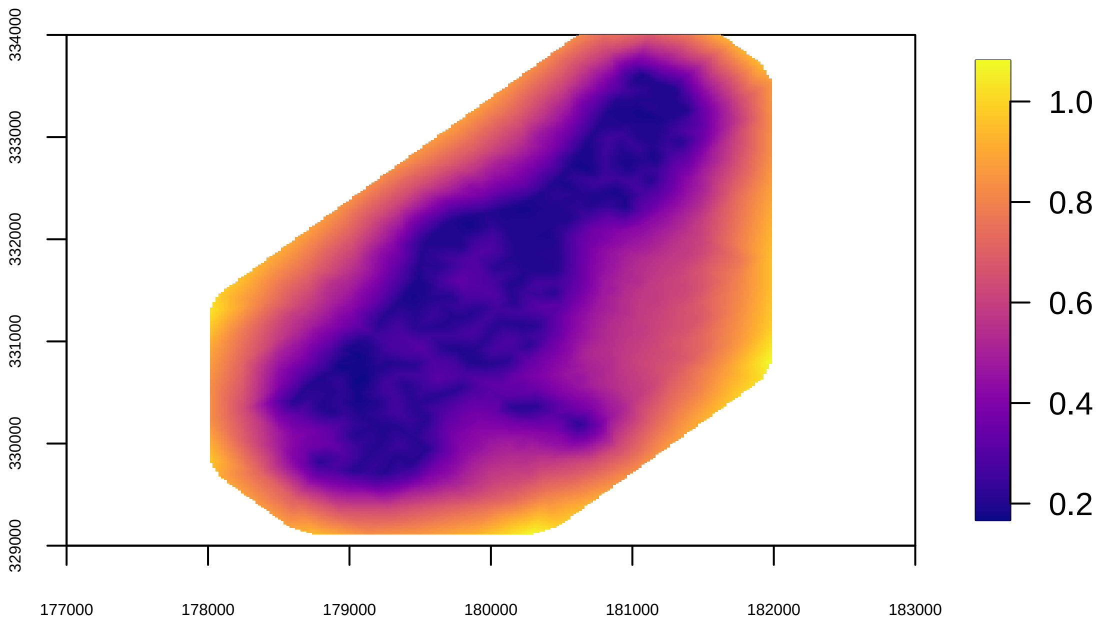
We now plot the predictive mean of the spatial field.

```
local.plot.field(res$summary.random[["s"]][["mean"]],
    mesh, cex.axis = 0.5)
lines(178000 + c(-0.5, 0.5) * (res$summary.hyperpar[2,
    "0.5quant"]), c(333500, 333500), lwd = 3)  # add on the estimated range
```



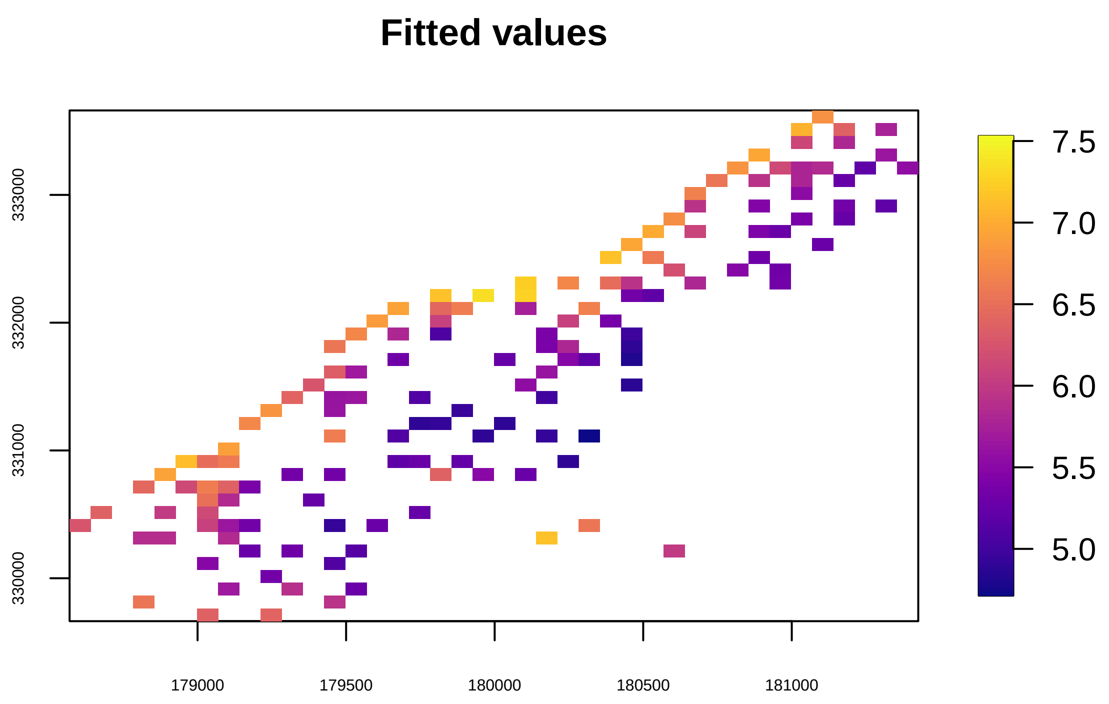
We now plot the predictive standard deviation of the spatial field.

```r
local.plot.field(res$summary.random$s$sd, mesh, cex.axis = 0.5)
```



And finally, we plot the fitted values.

```r
quilt.plot(x = zincdf$locx, y = zincdf$locy, z = res$summary.fitted.values$mean[1:nrow(zincdf)],
    nx = 40, ny = 40, col = plasma(101), main = "Fitted values",
    zlim = range(zincdf$y), cex.axis = 0.5)
```



Fitted values
333000
332000
331000
330000
179000
179500
180000
180500
181000
7.5
7.0
6.5
6.0
5.5
5.0