2023 554 R Notes on Mapping for Point Data

Jon Wakefield
Departments of Biostatistics and Statistics
University of Washington

2023-03-05

Overview

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

We will look at modeling a spatially-indexed continuous response via:

- Conventional Kriging via MLE and variants
- A generalized additive model (GAM)
- A Bayesian approach using stochastic partial differential equations (SPDE)

Continuous Response: Motivating 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. The version in the geoR library are not a spatial object, but can be used with likelihood and Bayes methods.

`geoR` for geostatistics

```
library(geoR)
library(sp)
library(tidyverse)
library(ggpubr)
library(viridis)
data("meuse")
```

We start the analysis using functions from the `geoR` library, for which a `geodata` data type is required. There are 155 observations (sampling locations)

```
zmat <- matrix(cbind(meuse$x, meuse$y, log(meuse$zinc)), ncol = 3, nrow = 155, byrow = F)
geozinc <- as.geodata(zmat, coords.col = c(1, 2), data.col = c(3))
```

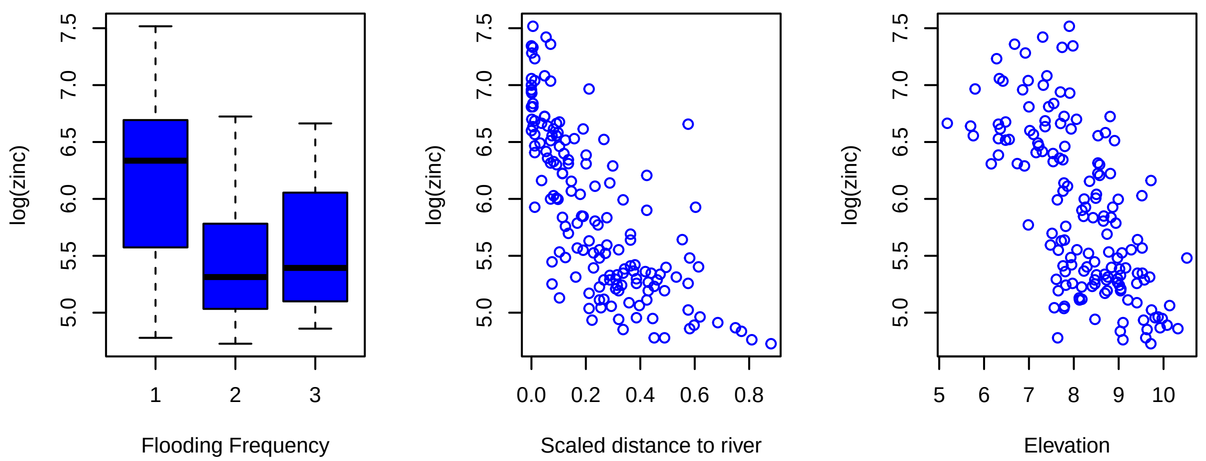
Exploratory analysis

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.

It's often a good idea to do some exploratory data analysis (EDA), so let's see how log(zinc) varies by three possible covariates:

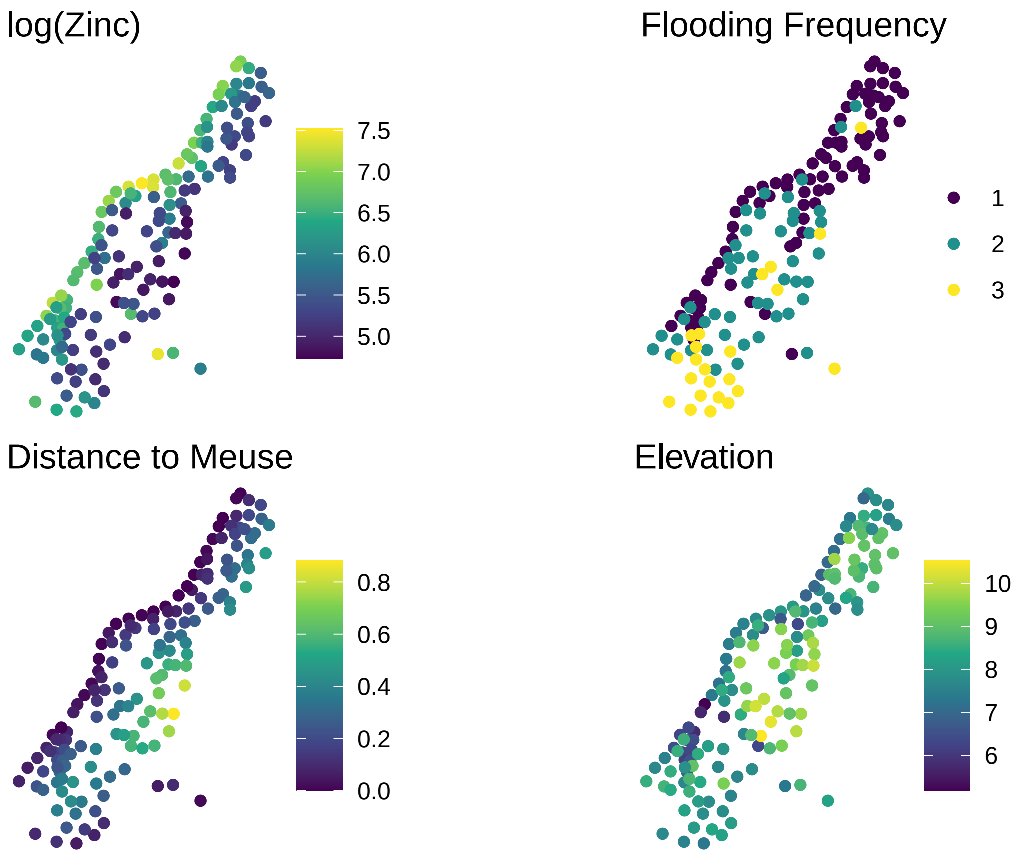
- Flooding frequency (`ffreq`); 1 = once in two years; 2 = once in ten years; 3 = one in 50 years
- Distance to the Meuse river (`dist`); normalized to $[0, 1]$
- Elevation (`elev`); relative to the local river bed, in meters.

```
par(mfrow = c(1, 3))
plot(log(meuse$zinc) ~ meuse$ffreq, ylab = "log(zinc)", xlab = "Flooding Frequency",
    col = "blue")
plot(log(meuse$zinc) ~ meuse$dist, ylab = "log(zinc)", xlab = "Scaled distance to river",
    col = "blue")
plot(log(meuse$zinc) ~ meuse$elev, ylab = "log(zinc)", xlab = "Elevation", col = "blue")
```



Also map these covariates.

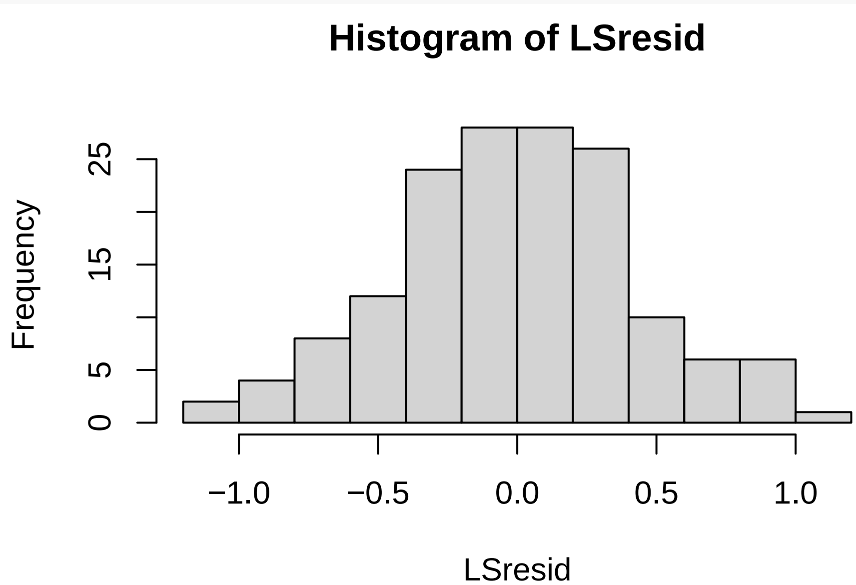
```
m.sf <- sf::st_as_sf(meuse, coords = c("x", "y"))
m.sf$logzinc <- log(m.sf$zinc)

a <- ggplot() + geom_sf(data = m.sf[, "logzinc"], aes(color = logzinc)) + theme_void() +
    scale_color_viridis_c() + labs(title = "log(Zinc)", color = NULL)
b <- ggplot() + geom_sf(data = m.sf[, "ffreq"], aes(color = ffreq)) + theme_void() +
    scale_color_viridis(discrete = T) + labs(title = "Flooding Frequency", color = NULL)
c <- ggplot() + geom_sf(data = m.sf[, "dist"], aes(color = dist)) + theme_void() +
    scale_color_viridis_c() + labs(title = "Distance to Meuse", color = NULL)
d <- ggplot() + geom_sf(data = m.sf[, "elev"], aes(color = elev)) + theme_void() +
    scale_color_viridis_c() + labs(title = "Elevation", color = NULL)
ggpubr::ggarrange(a, b, c, d, nrow = 2, ncol = 2)
```



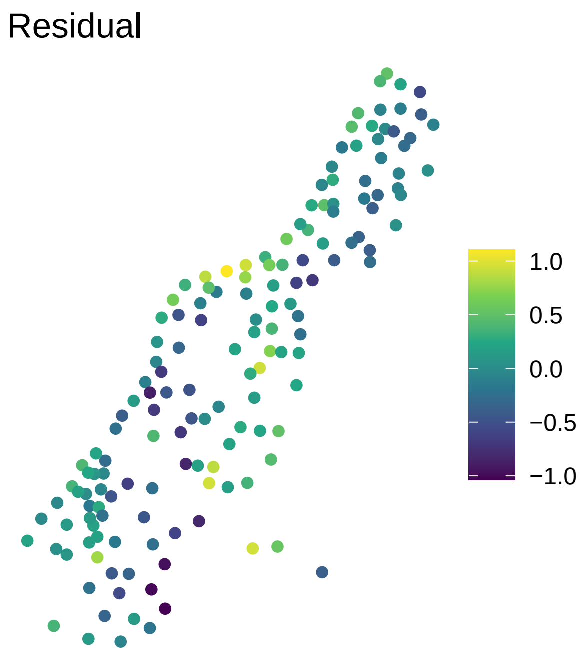
log(zinc) Variogram

First we will be assuming a spatial model on the residuals with elevation and distance in the model. Fit this initial linear model, and view the residuals by histogram and map.

```
LSmod <- lm(log(meuse$zinc) ~ meuse$dist + meuse$elev)
LSresid <- residuals(LSmod)
hist(LSresid)
```



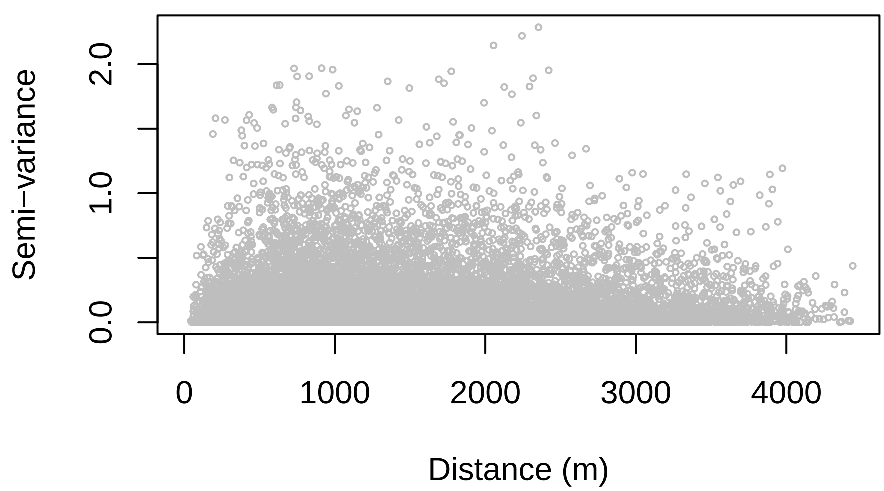
```
m.sf$resid <- LSresid
ggplot() + geom_sf(data = m.sf[, "resid"], aes(color = resid)) + theme_void() + scale_color_viridis_c()
    labs(title = "Residual", color = NULL)
```



Cloud

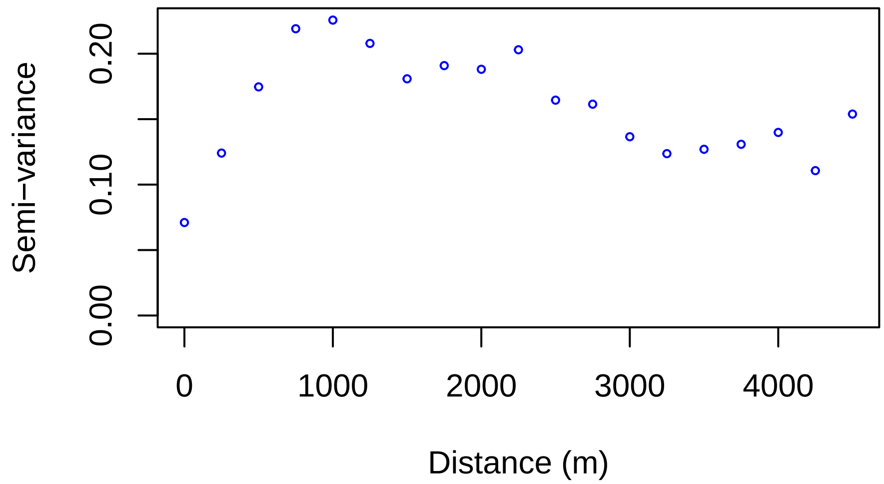
Variogram cloud for log zinc,

```
cloudzinc <- variog(geozinc, option = "cloud", trend = ~meuse$dist + meuse$elev)
## variog: computing omnidirectional variogram
plot(cloudzinc, ylab = "Semi-variance", xlab = "Distance (m)", col = "grey", cex = 0.4)
```



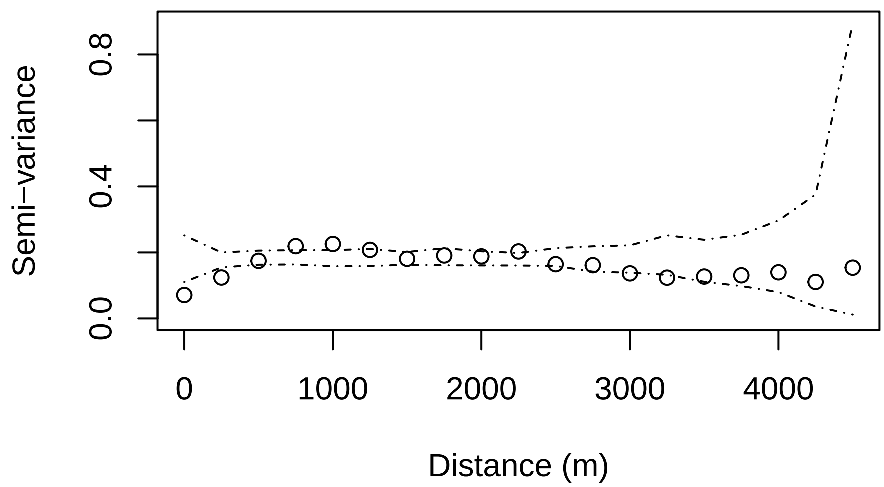
Binned variogram

```
binzinc <- variog(geozinc, uvec = seq(0, 5000, 250), trend = ~meuse$dist + meuse$elev)
## variog: computing omnidirectional variogram
plot(binzinc, ylab = "Semi-variance", xlab = "Distance (m)", cex = 0.5, col = "blue")
```



Monte Carlo envelopes under no spatial dependence - it is clear there is dependence here.

```
geozinc.env <- variog.mc.env(geozinc, obj = binzinc)
## variog.env: generating 99 simulations by permutating data values
## variog.env: computing the empirical variogram for the 99 simulations
## variog.env: computing the envelops
plot(binzinc, env = geozinc.env, xlab = "Distance (m)", ylab = "Semi-variance")
```



Parameter estimation from the variogram

We now estimate the parameters of the exponential covariance model which in `geoR` is parameterized as

$$\tau^2 + \sigma^2 \exp(-d/\phi),$$

where d is the distance between the points, σ^2 is the partial sill and τ^2 is the nugget.

The effective range is the distance at which the correlation is 0.05, and if we have a rough estimate of this $\tilde{d}$ (from the binned variogram, for example) then we can solve for an initial estimate $\tilde{\phi} = -\tilde{d}/log(0.05)$.

Maximum likelihood for log(zinc)

We suppress the output from the call.

```
mlfit <- likfit(geozinc, ini = c(0.2, 224), trend = ~meuse$dist + meuse$elev)
```

We examine the results, specifically point estimates and standard errors.

```
mlfit$parameters.summary
##            status   values
## beta0   estimated   8.6162
## beta1   estimated  -2.1072
## beta2   estimated  -0.2690
## tausq   estimated   0.0010
## sigmasq estimated   0.2065
## phi     estimated 241.1982
## kappa       fixed   0.5000
## psiA        fixed   0.0000
## psiR        fixed   1.0000
## lambda      fixed   1.0000
for (i in 1:3) {
    cat(cbind(mlfit$beta[i], sqrt(mlfit$beta.var[i, i])), "\n")
}
## 8.616183 0.2557962
## -2.107206 0.3414008
## -0.2689692 0.02989409
mlfit$practicalRange
## [1] 722.5653
```

Note that the standard errors change from the least squares fit.

Restricted maximum likelihood for log(zinc)

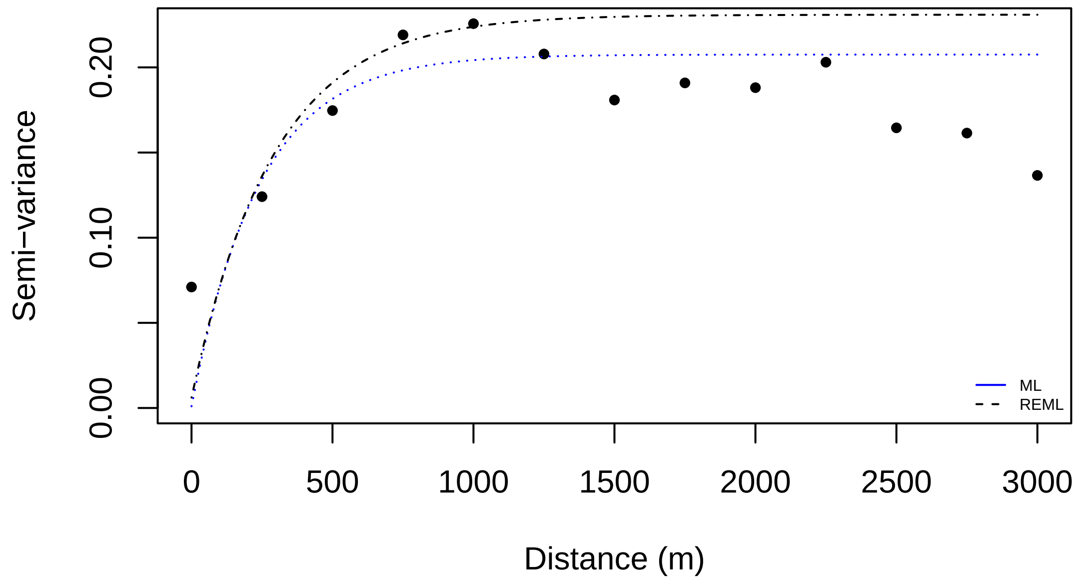
```
remlfit <- likfit(geozinc, ini = c(0.55, 224), lik.method = "RML", trend = ~meuse$dist +
    meuse$elev)
```

The results: slight differences from ML.

```
remlfit$parameters.summary
##            status   values
## beta0   estimated   8.6396
## beta1   estimated  -2.1215
## beta2   estimated  -0.2701
## tausq   estimated   0.0061
## sigmasq estimated   0.2248
## phi     estimated 289.1468
## kappa       fixed   0.5000
## psiA        fixed   0.0000
## psiR        fixed   1.0000
## lambda      fixed   1.0000
remlfit$practicalRange
## [1] 866.2064
```

Comparison of estimates

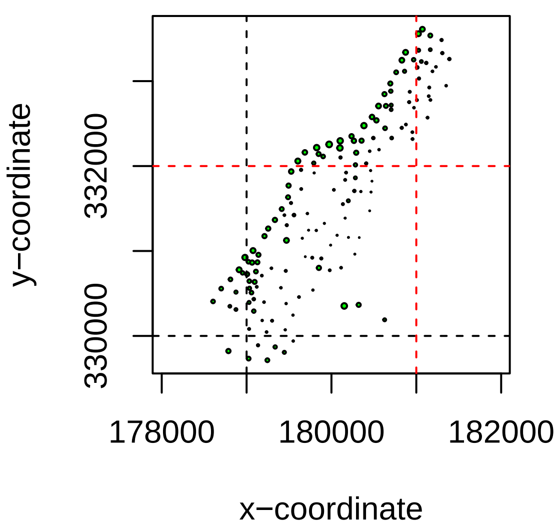
```
plot(binzinc, max.dist = 3000, xlab = "Distance (m)", ylab = "Semi-variance", pch = 19,
    cex = 0.6)
lines(mlfit, max.dist = 3000, lty = 3, col = "blue")
lines(remlfit, max.dist = 3000, lty = 4, col = "black")
legend("bottomright", legend = c("ML", "REML"), lty = c(1, 2), bty = "n", col = c("blue",
    "black"), cex = 0.5)
```



Prediction for log(zinc) by Kriging

First we plot the data along with the region within which we shall carry out prediction.

```
points(geozinc, pt.divide = "data.proportional", cex.min = 0.05,
    cex.max = 0.4, xlab = "x-coordinate", ylab = "y-coordinate",
    col = "green")
# See points.geodata description for explanation
# of this function
abline(h = 330000, lty = 2)
abline(h = 332000, lty = 2, col = "red")
abline(v = 179000, lty = 2)
abline(v = 181000, lty = 2, col = "red")
```

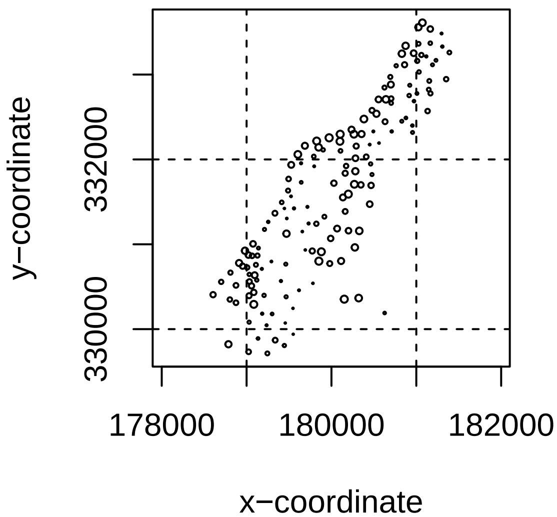


We now fit a linear model in distance and elevation to log(zinc). We then form a `geodata` object with the residuals as the response.

```
lmfit <- lm(geozinc$data ~ meuse$dist + meuse$elev)
lmfit
```

```
##
## Call:
## lm(formula = geozinc$data ~ meuse$dist + meuse$elev)
##
## Coefficients:
## (Intercept)   meuse$dist   meuse$elev
##      8.4845      -1.9600      -0.2607
detrend <- as.geodata(cbind(geozinc$coords, lmfit$residuals))
```

```
points(detrend, pt.divide = "rank.prop", xlab = "x-coordinate",
    ylab = "y-coordinate", cex.min = 0.1, cex.max = 0.5)
abline(h = 330000, lty = 2)
abline(h = 332000, lty = 2)
abline(v = 179000, lty = 2)
abline(v = 181000, lty = 2)
```



Next carry out MLE on the detrended data.

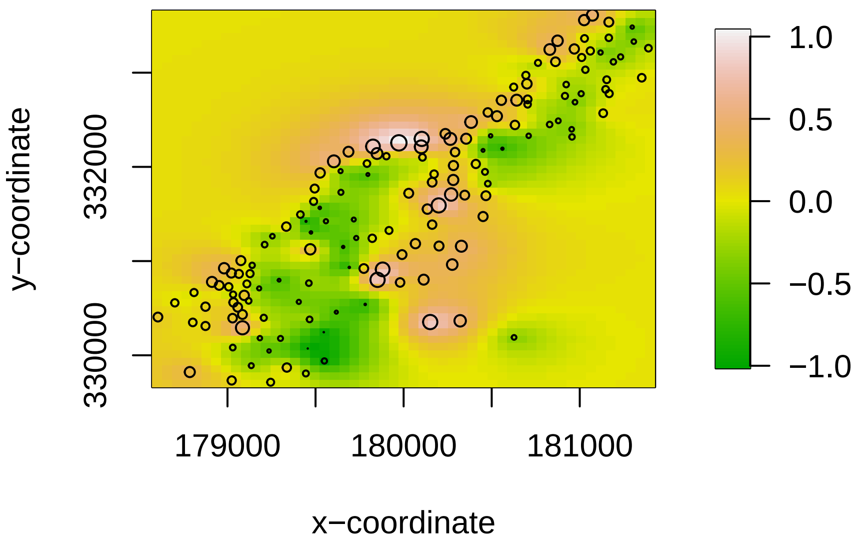
```
mlfit2 <- likfit(detrend, ini = c(0.2, 224))
## ---------------------------------------------------------------
## likfit: likelihood maximisation using the function optim.
## likfit: Use control() to pass additional
##          arguments for the maximisation function.
##         For further details see documentation for optim.
## likfit: It is highly advisable to run this function several
##         times with different initial values for the parameters.
## likfit: WARNING: This step can be time demanding!
## ---------------------------------------------------------------
## likfit: end of numerical maximisation.
mlfit2
## likfit: estimated model parameters:
##       beta      tausq    sigmasq        phi
## "  0.0306" "  0.0000" "  0.2076" "238.0914"
## Practical Range with cor=0.05 for asymptotic range: 713.2582
##
## likfit: maximised log-likelihood = -54.82
```

Finally, we can obtain spatial predictions on a grid, using the parameter estimates from the ML fit to the residuals. Ordinary Kriging is used for this prediction.

```
pred.grid <- expand.grid(seq(178600, 181400,
    l = 51), seq(329700, 333620, l = 51))
kc <- krige.conv(detrend, loc = pred.grid,
    krige = krige.control(obj.m = mlfit2))
## krige.conv: model with constant mean
## krige.conv: Kriging performed using global neighbourhood
```

To view the results produce an image plot of the predictions, with the data superimposed.

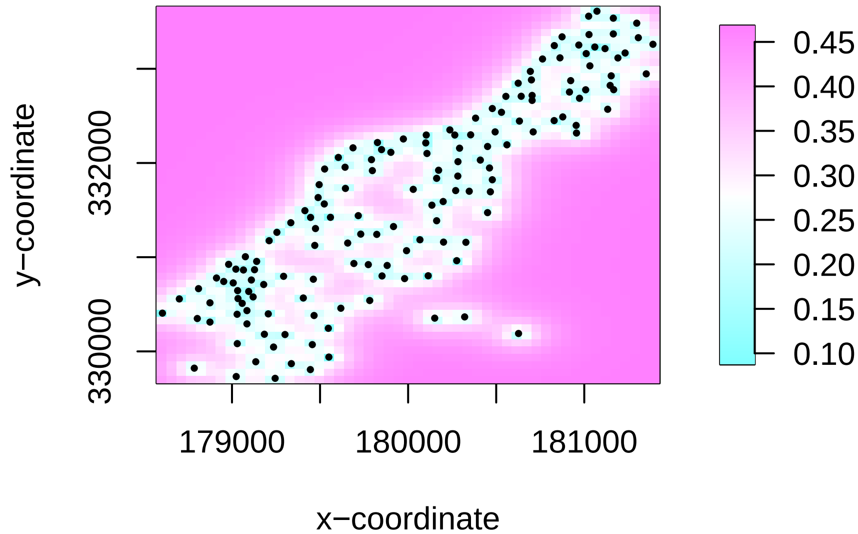
```
library(fields)
image.plot(x = pred.grid[["Var1"]][1:51], y = unique(pred.grid[["Var2"]]),
    z = matrix(kc$predict, nrow = 51, ncol = 51), col = terrain.colors(100),
    xlab = "x-coordinate", ylab = "y-coordinate")
symbols(detrend$coords[, 1], detrend$coords[, 2], circles = (detrend$data -
    min(detrend$data))/1, add = T, inches = 0.04)
```



Standard deviations of prediction for log(zinc)

We now plot the Kriging standard deviations of the predictions.

```
image.plot(x = pred.grid[["Var1"]][1:51], y = unique(pred.grid[["Var2"]]),
    z = matrix(sqrt(kc$krige.var), nrow = 51, ncol = 51),
    col = cm.colors(100), xlab = "x-coordinate", ylab = "y-coordinate")
points(detrend$coords[, 1], detrend$coords[, 2], cex = 0.5,
    pch = 16)
```



The standard deviation is smallest close to the datapoints, as expected.

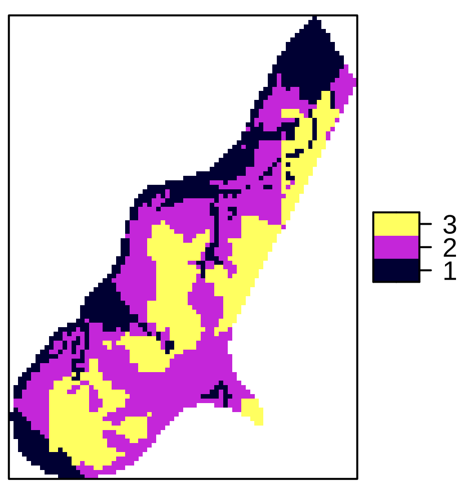
Another example, using categorical flooding frequency covariate

Now, we'll repeat, but use categorical flooding frequency and distance to the Meuse River as covariates. These two covariates are available for a grid, in the `meuse.grid` data object. This means we can predict log(Zinc) for a grid, using the sum of the linear model fit and the interpolated spatial field.

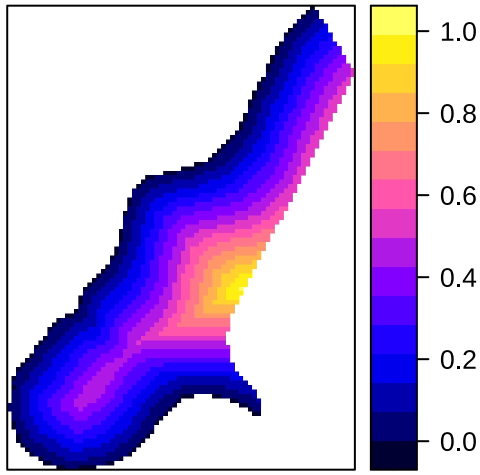
First take a look at the grid covariates.

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

```
spplot(meuse.grid["ffreq"])
```



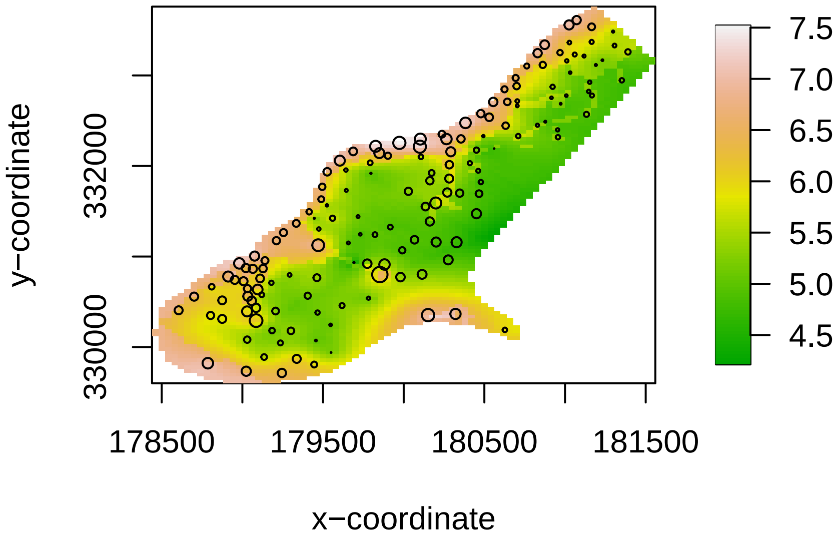
```
spplot(meuse.grid["dist"])
```



Now proceed with Kriging for log(Zinc).

```
# (label objects associated with this version with a '3')
lmdata <- data.frame(logzinc = geozinc$data, dist = meuse$dist, ffreq = meuse$ffreq)
lmfit3 <- lm(logzinc ~ dist + ffreq, data = lmdata)
detrend3 <- as.geodata(cbind(geozinc$coords, lmfit3$residuals))
mlfit3 <- likfit(detrend3, ini = c(0.2, 224))
## ---------------------------------------------------------------
## likfit: likelihood maximisation using the function optim.
## likfit: Use control() to pass additional
##          arguments for the maximisation function.
##         For further details see documentation for optim.
## likfit: It is highly advisable to run this function several
##         times with different initial values for the parameters.
## likfit: WARNING: This step can be time demanding!
## ---------------------------------------------------------------
## likfit: end of numerical maximisation.

# get meuse.grid and expand to all combinations
data(meuse.grid)
pred.grid3 <- expand.grid(sort(unique(meuse.grid$x)), sort(unique(meuse.grid$y)))

# kriging
kc3 <- krige.conv(detrend3, loc = pred.grid3, krige = krige.control(obj.m = mlfit3))
## krige.conv: model with constant mean
## krige.conv: Kriging performed using global neighbourhood

# merge linear predictions onto grid
pred_lm <- meuse.grid[, c("x", "y", "dist", "ffreq")]
pred_lm$pred_lm <- predict(lmfit3, newdata = pred_lm)
pred_resid <- data.frame(pred_resid = kc3$predict, x = pred.grid3$Var1, y = pred.grid3$Var2)
pred_logzinc <- merge(pred_resid, pred_lm, by = c("x", "y"), all = T)
pred_logzinc$pred_logzinc <- pred_logzinc$pred_lm + pred_logzinc$pred_resid
pred_logzinc <- pred_logzinc %>%
    arrange(y, x)
```

```r
# plot prediction
image.plot(x = unique(pred.grid3[["Var1"]]), y = unique(pred.grid3[["Var2"]]),
    z = matrix(pred_logzinc$pred_logzinc, nrow = 78,
        ncol = 104), col = terrain.colors(100), xlab = "x-coordinate",
    ylab = "y-coordinate")
symbols(detrend3$coords[, 1], detrend3$coords[, 2],
    circles = (detrend3$data - min(detrend3$data))/1,
    add = T, inches = 0.04)
```



GAM model for log(zinc)

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.

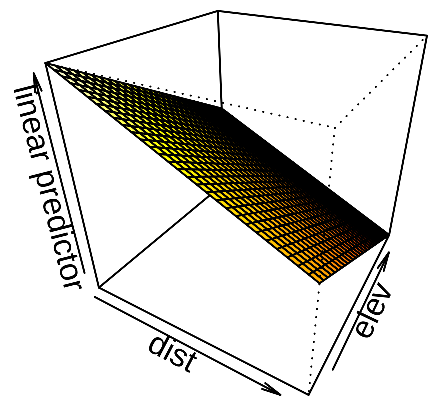
```r
library(mgcv)
library(lattice)
library(latticeExtra)
library(RColorBrewer)
zinc.dat <- data.frame(x = meuse$x, y = meuse$y, lzinc = log(meuse$zinc),
    dist = meuse$dist, elev = meuse$elev)
gam.mod <- gam(lzinc ~ s(x, y, bs = "tp") + dist +
    elev, data = zinc.dat, method = "REML")
```

```r
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
```

GAM output: The fitted distance by elevation surface

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

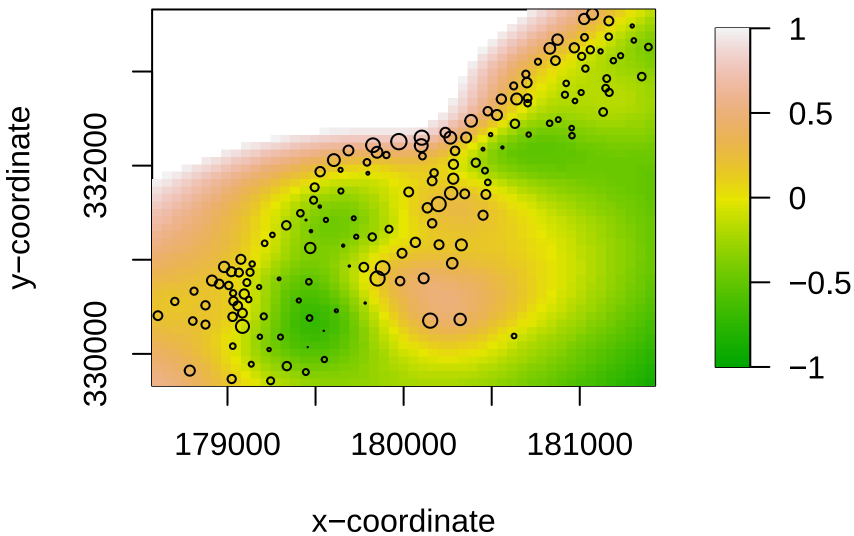


GAM prediction

```
pred.grid.gam <- expand.grid(seq(178600, 181400, l = 51),
    seq(329700, 333620, l = 51))

pred.dat.gam <- data.frame(x = pred.grid.gam[, 1],
    y = pred.grid.gam[, 2], dist = mean(meuse$dist),
    elev = mean(meuse$elev))
zinc.pred.gam <- predict.gam(gam.mod, pred.dat.gam,
    type = "terms")[, 3]

zinc.pred.gam.sd <- predict.gam(gam.mod, se.fit = T,
    pred.dat.gam, type = "terms")[[2]][, 3]
```

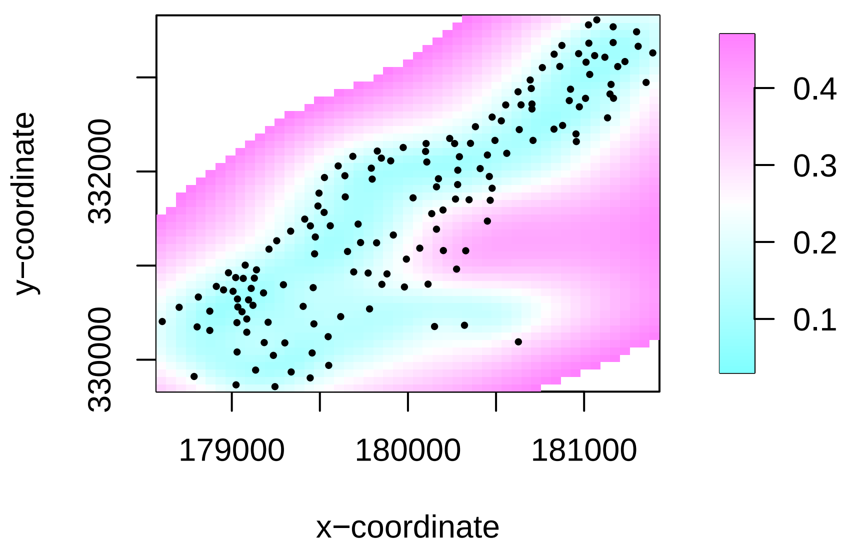
```
image.plot(x = pred.grid.gam[["Var1"]][1:51], y = unique(pred.grid.gam[["Var2"]]),
    z = matrix(zinc.pred.gam, nrow = 51, ncol = 51),
    col = terrain.colors(100), xlab = "x-coordinate",
    ylab = "y-coordinate", breaks = seq(-1, 1, , 101),
    axis.args = list(at = c(-1, -0.5, 0, 0.5, 1), labels = c("-1",
        "-0.5", "0", "0.5", "1")), legend.cex = 0.5)

symbols(detrend$coords[, 1], detrend$coords[, 2], circles = (detrend$data -
```

```
    min(detrend$data))/1, add = T, inches = 0.04)
```



Standard deviations of prediction from GAM

```
image.plot(x = pred.grid[["Var1"]][1:51], y = unique(pred.grid[["Var2"]]),
    z = matrix(zinc.pred.gam.sd, nrow = 51, ncol = 51),
    col = cm.colors(100), xlab = "x-coordinate", ylab = "y-coordinate",
    breaks = seq(0.03, 0.47, , 101), axis.args = list(at = c(0.1,
        0.2, 0.3, 0.4), labels = c("0.1", "0.2", "0.3",
        "0.4")), legend.cex = 0.5)
points(detrend$coords[, 1], detrend$coords[, 2], cex = 0.5,
    pch = 16)
```



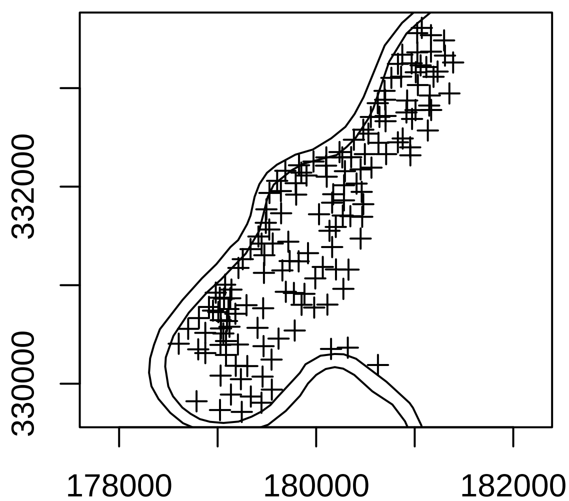
Meuse analysis using geostat functions

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

```
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
```

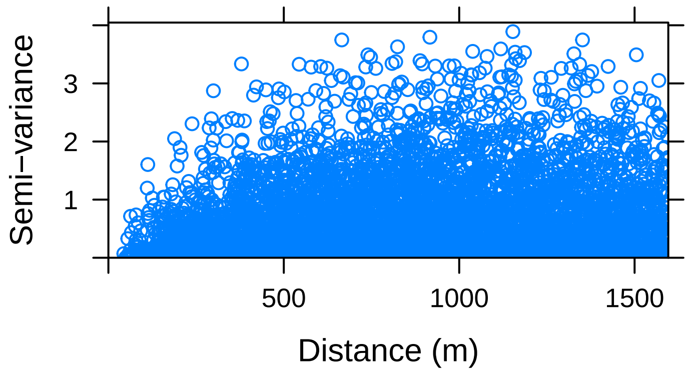
Zinc: Sampling locations

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



log(zinc): Variogram cloud, no trend removed

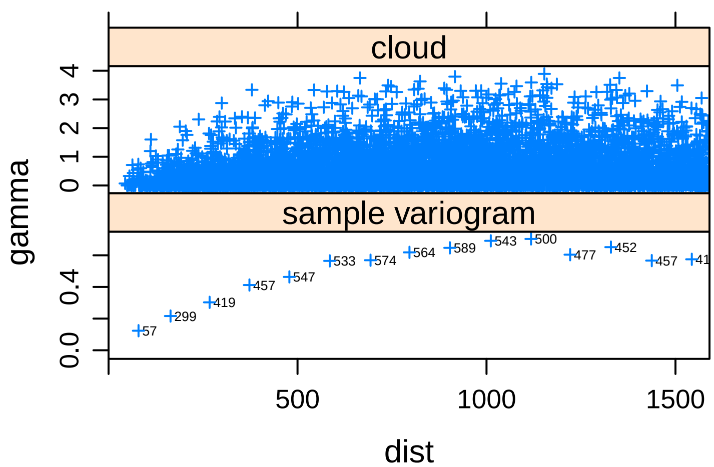
```
library(gstat)
cld <- variogram(log(zinc) ~ 1, meuse, cloud = TRUE)
plot(cld, ylab = "Semi-variance", xlab = "Distance (m)")
```



More variograms, with 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
)
```

More variograms



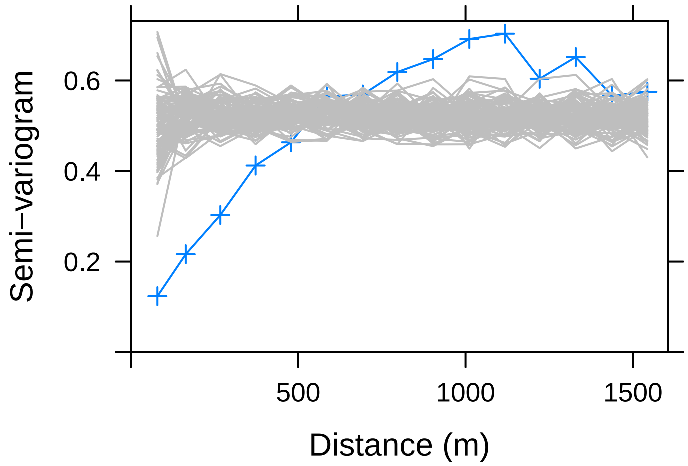
Monte Carlo simulations of semi-variogram

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()
    }
```

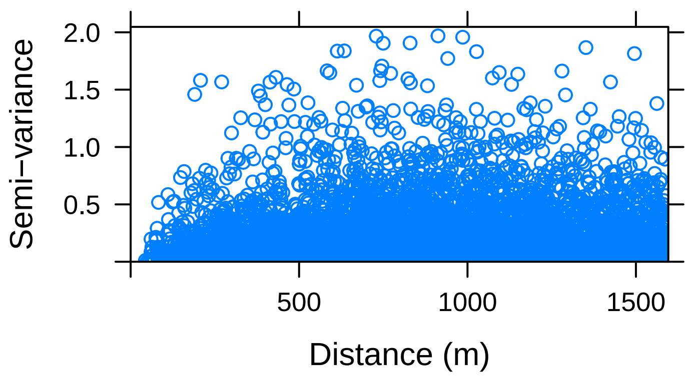
```
}
fn()
```

Monte Carlo simulations of semi-variogram



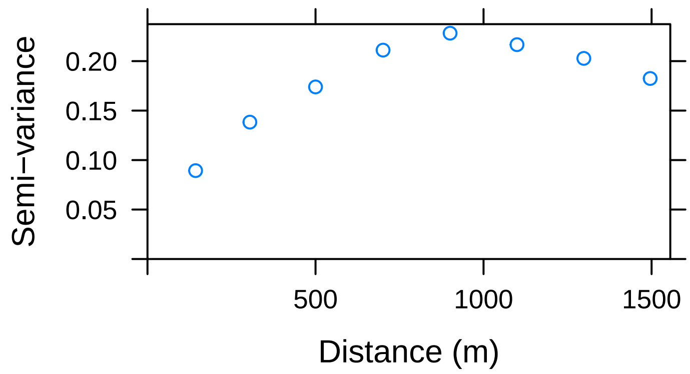
log(zinc): Variogram cloud, detrended

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



log(zinc): Binned variogram, detrended

```
gstatbin <- variogram(log(zinc) ~ dist + elev, meuse, width = 200)
plot(gstatbin, ylab = "Semi-variance", xlab = "Distance (m)")
```



Zinc: Directional variogram with linear trend removed

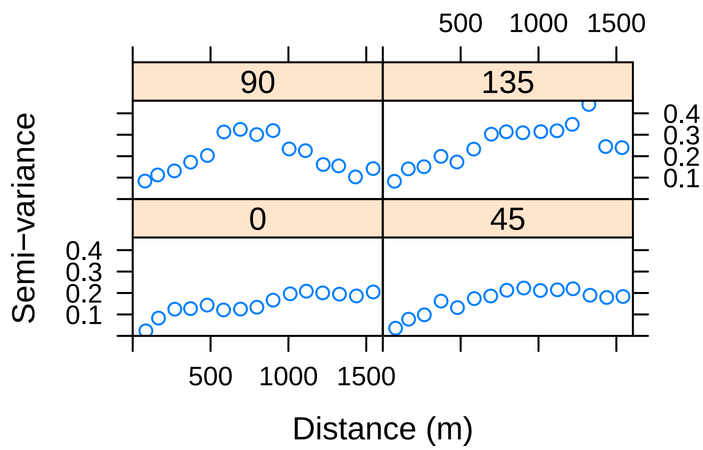
We form 4 variograms with data taken from different directions, with 0 and 90 corresponding to north and east, respectively.

Note that 0 is the same as 180.

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

Zinc: Directional variogram with linear trend removed

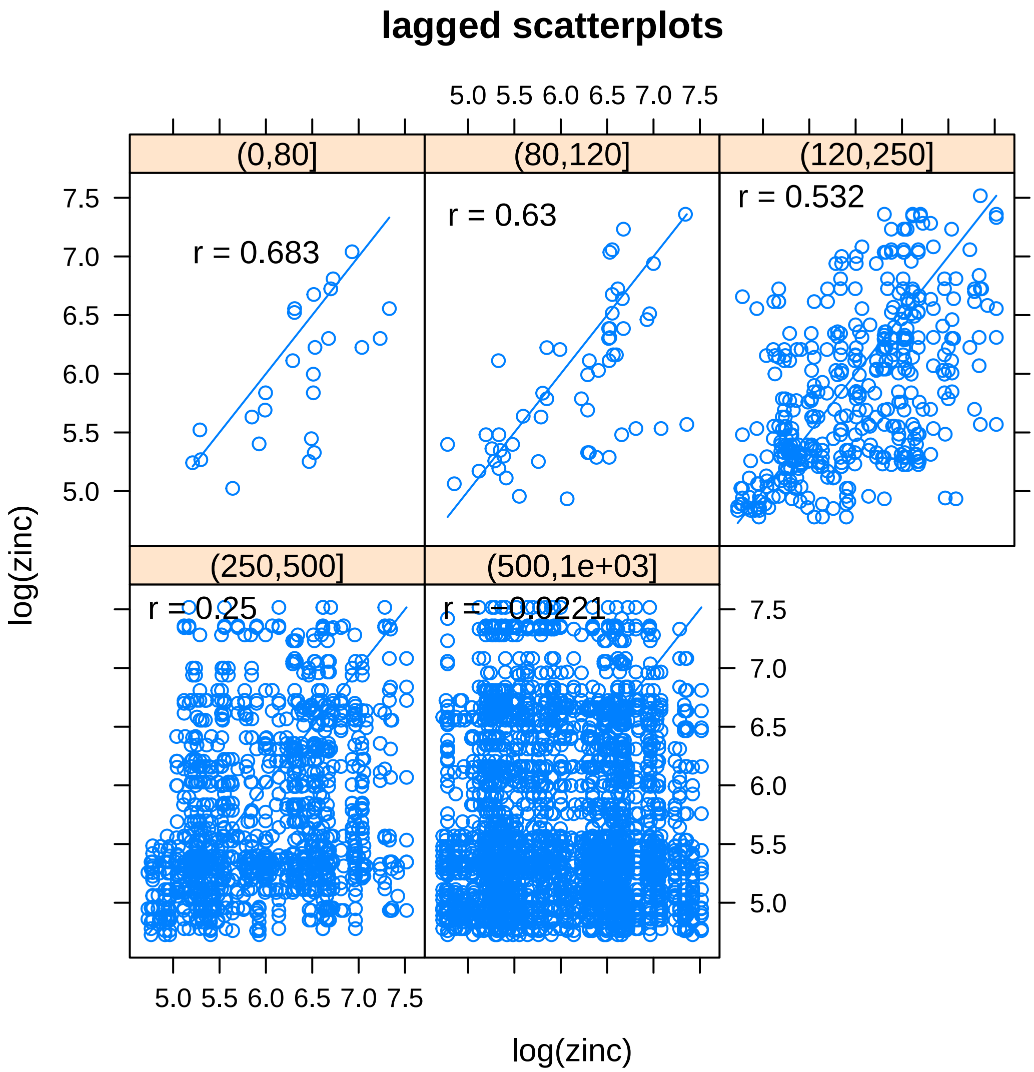
```
plot(dircld, xlab = "Distance (m)", ylab = "Semi-variance")
```



Zinc: Lagged scatterplots

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

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

```
library(INLA)
zincdf = data.frame(y = log(meuse$zinc), locx = meuse$x,
    locy = meuse$y, dist = meuse$dist, 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:

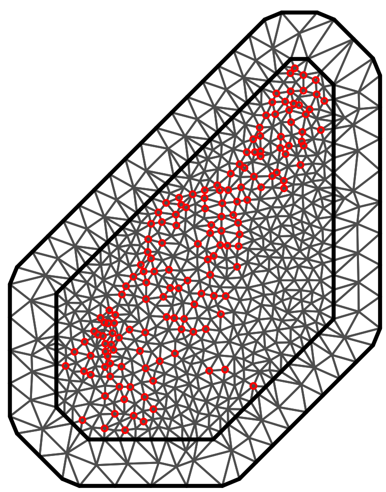
– `offset`: specifies how much the domain will be extended in the outer and inner part. If negative it is interpreted as a factor relative to the approximate data diameter. If positive it is the extension distance on same scale unit to the coordinates provided.

– `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.

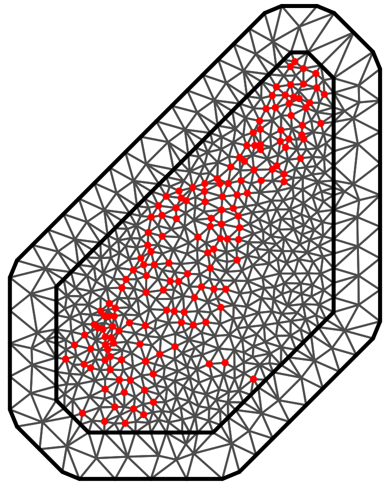
By specifying `loc` we obtain a mesh with observations lying at the vertices.

```
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.

```
plot(mesh, asp = 1, main = "")
points(zincdf[, c("locx", "locy")], col = "red", cex = 0.5, pch = 16)
```



```
# axis(1); axis(2)
```

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] 155 695
# table(as.numeric(A)) table(rowSums(A > 0)) #
# 155 values of 1 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, dist = zincdf$dist,
    elev = zincdf$elev)
# - expands the factor covariates
Xcov = as.matrix(Xcov)
colnames(Xcov)
## [1] "intercept" "dist"      "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'
```

```r
                    # - 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.

```r
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.

```r
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.

```r
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)
```

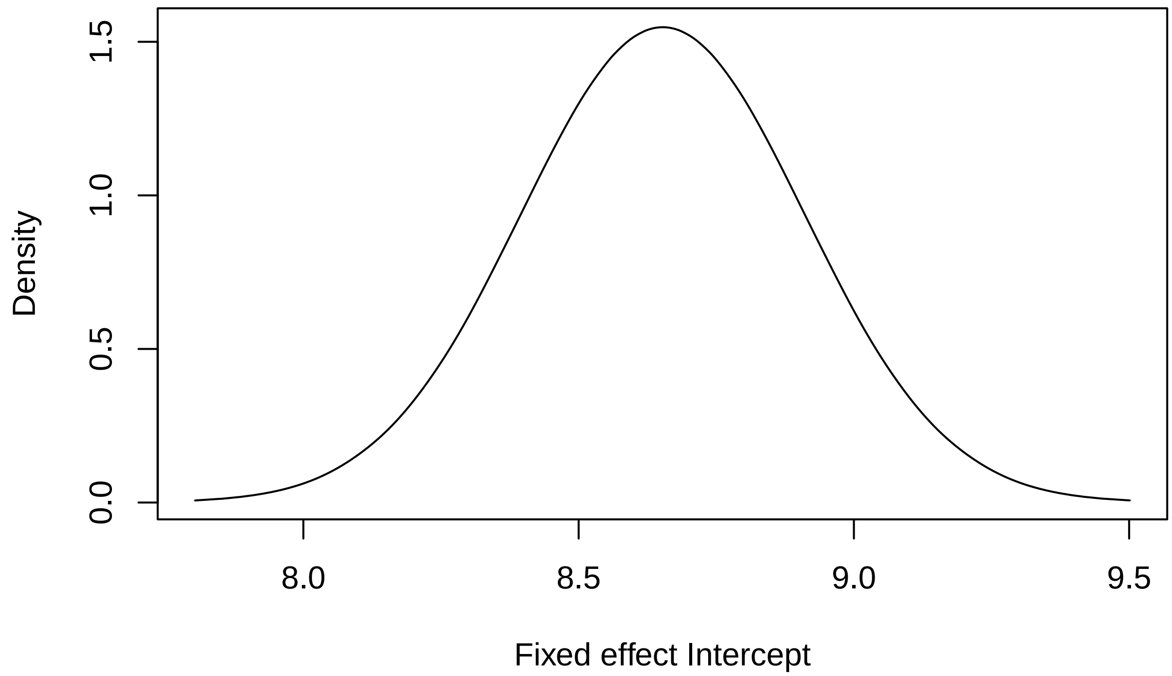
See `?inla.spde2.result` for extracting results.

```r
summary(res)
##
## Call:
##    c("inla.core(formula = formula, family = family, contrasts = contrasts,
##    ", " data = data, quantiles = quantiles, E = E, offset = offset, ", "
##    scale = scale, weights = weights, Ntrials = Ntrials, strata = strata,
##    ", " lp.scale = lp.scale, link.covariates = link.covariates, verbose =
##    verbose, ", " lincomb = lincomb, selection = selection, control.compute
##    = control.compute, ", " control.predictor = control.predictor,
##    control.family = control.family, ", " control.inla = control.inla,
##    control.fixed = control.fixed, ", " control.mode = control.mode,
##    control.expert = control.expert, ", " control.hazard = control.hazard,
##    control.lincomb = control.lincomb, ", " control.update =
##    control.update, control.lp.scale = control.lp.scale, ", "
##    control.pardiso = control.pardiso, only.hyperparam = only.hyperparam,
##    ", " inla.call = inla.call, inla.arg = inla.arg, num.threads =
##    num.threads, ", " blas.num.threads = blas.num.threads, keep = keep,
##    working.directory = working.directory, ", " silent = silent, inla.mode
```

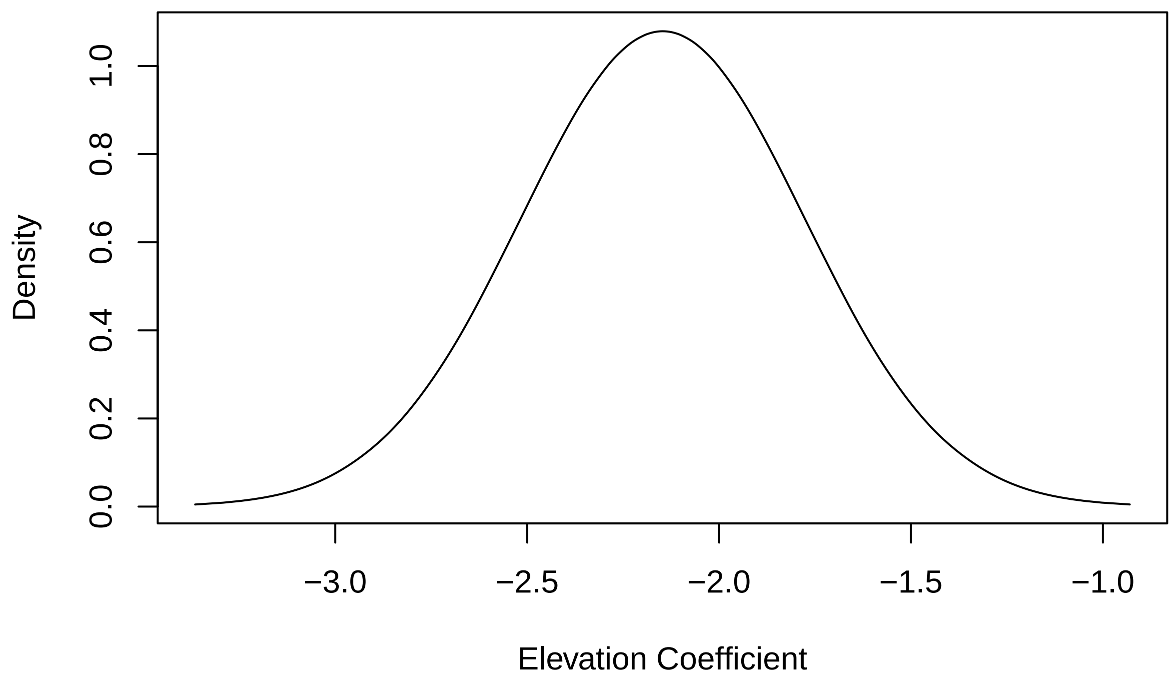
```
##    = inla.mode, safe = FALSE, debug = debug, ", " .parent.frame =
##    .parent.frame)")
## Time used:
##     Pre = 1.52, Running = 0.392, Post = 0.0191, Total = 1.93
## Fixed effects:
##         mean    sd 0.025quant 0.5quant 0.975quant   mode kld
## Xcov1  8.653 0.258      8.148    8.653      9.158  8.653   0
## Xcov2 -2.147 0.370     -2.872   -2.147     -1.422 -2.147   0
## Xcov3 -0.270 0.031     -0.331   -0.270     -0.209 -0.270   0
##
## Random effects:
##   Name     Model
##     s SPDE2 model
##
## Model hyperparameters:
##                                            mean      sd 0.025quant 0.5quant
## Precision for the Gaussian observations  29.620  11.223     14.586   27.335
## Range for s                             673.029 172.448    387.199  655.548
## Stdev for s                               0.423   0.062      0.316    0.418
##                                         0.975quant    mode
## Precision for the Gaussian observations     57.904  23.296
## Range for s                               1062.102 623.696
## Stdev for s                                  0.559   0.408
##
## Marginal log-Likelihood:  -81.11
##  is computed
## Posterior summaries for the linear predictor and the fitted values are computed
## (Posterior marginals needs also 'control.compute=list(return.marginals.predictor=TRUE)')
print("MLE SE BAYES SD")
## [1] "MLE SE BAYES SD"
for (i in 1:3) {
    cat(cbind(mlfit$beta[i], sqrt(mlfit$beta.var[i, i])), res$summary.fixed[i, 1],
        res$summary.fixed[i, 2], "\n")
}
## 8.616183 0.2557962 8.65279 0.2577952
## -2.107206 0.3414008 -2.146881 0.3697624
## -0.2689692 0.02989409 -0.2701307 0.03097456
```

Visual summarization

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

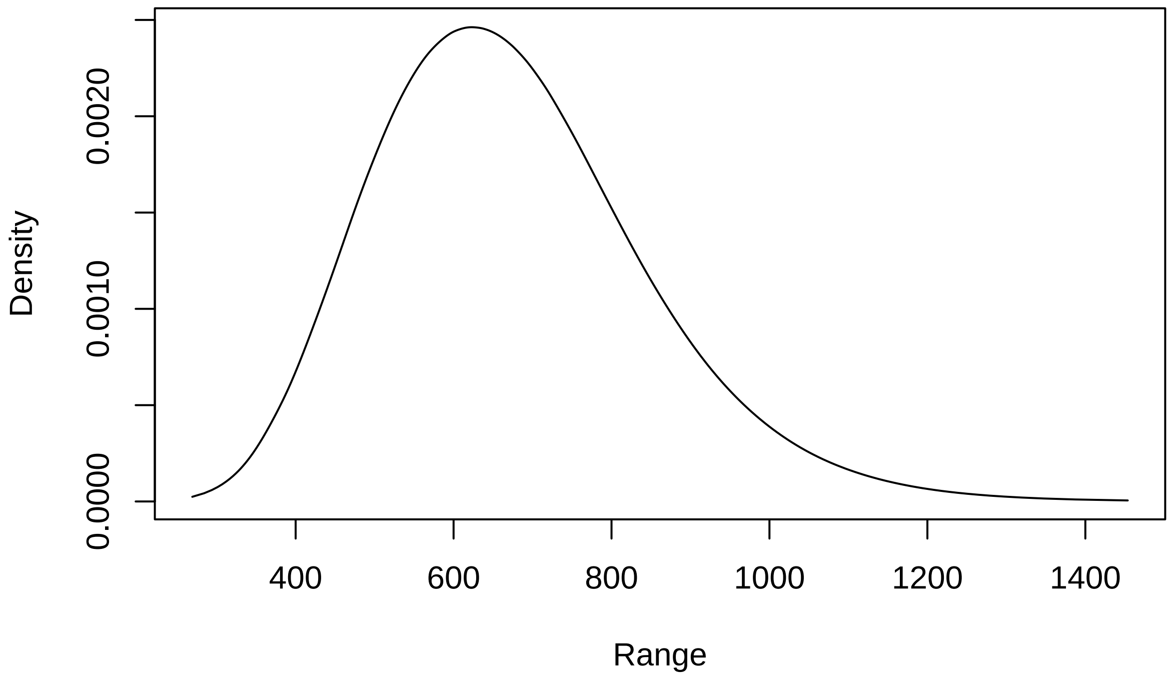



```
tmp = inla.tmarginal(function(x) x, res$marginals.fixed[[2]])
plot(tmp, type = "l", xlab = "Elevation Coefficient", ylab = "Density")
```



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")
```

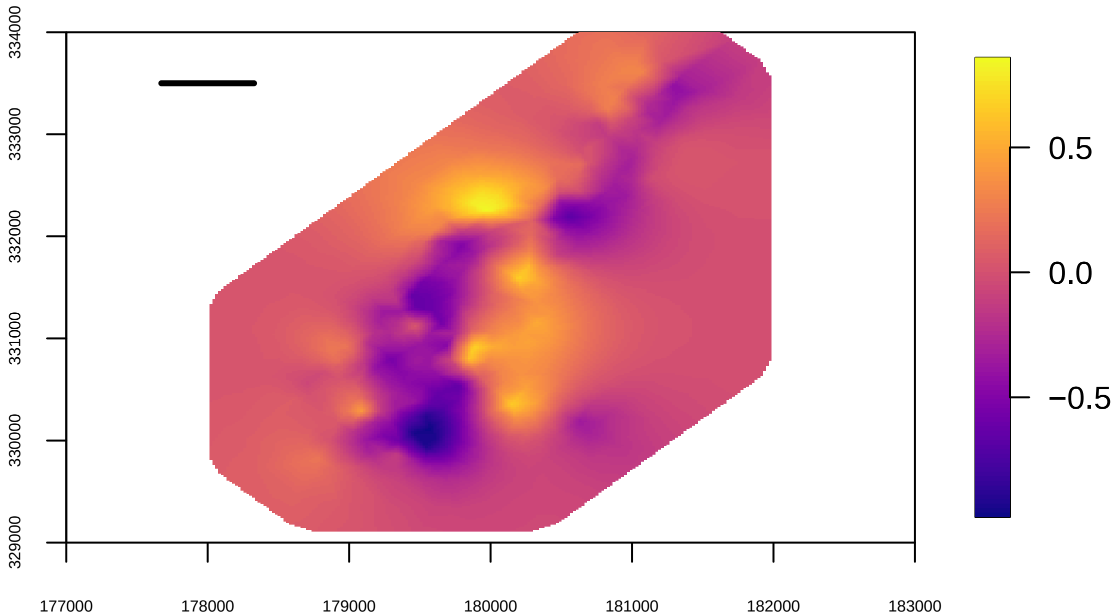


We define a function for plotting spatial fields for this application.

```
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), ...)
}
```

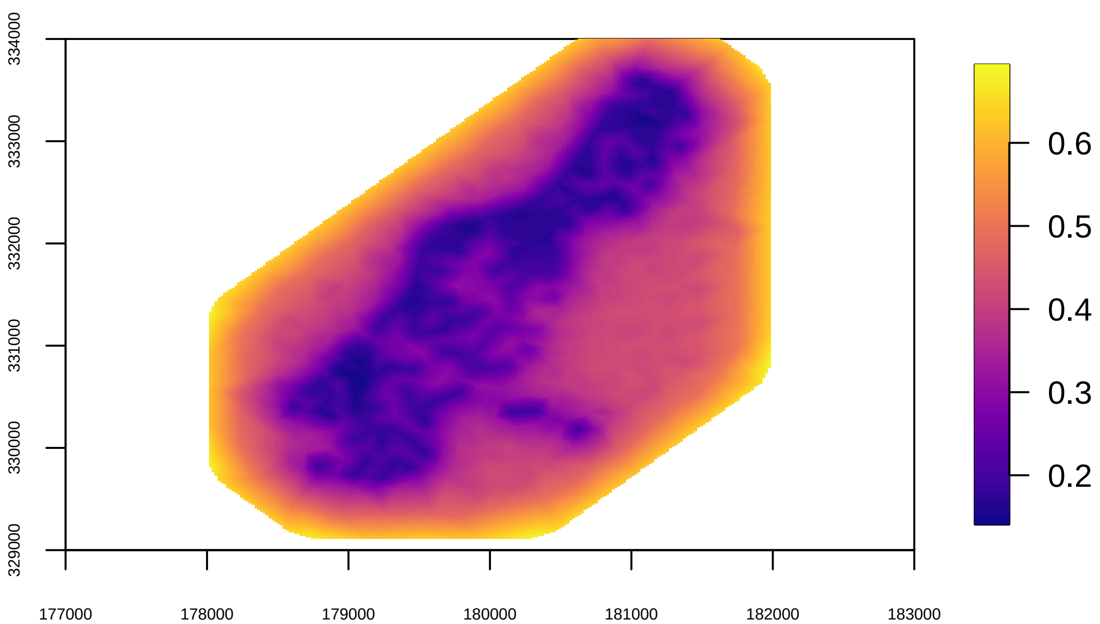
We now plot the predictive mean of the spatial field.

```
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
```



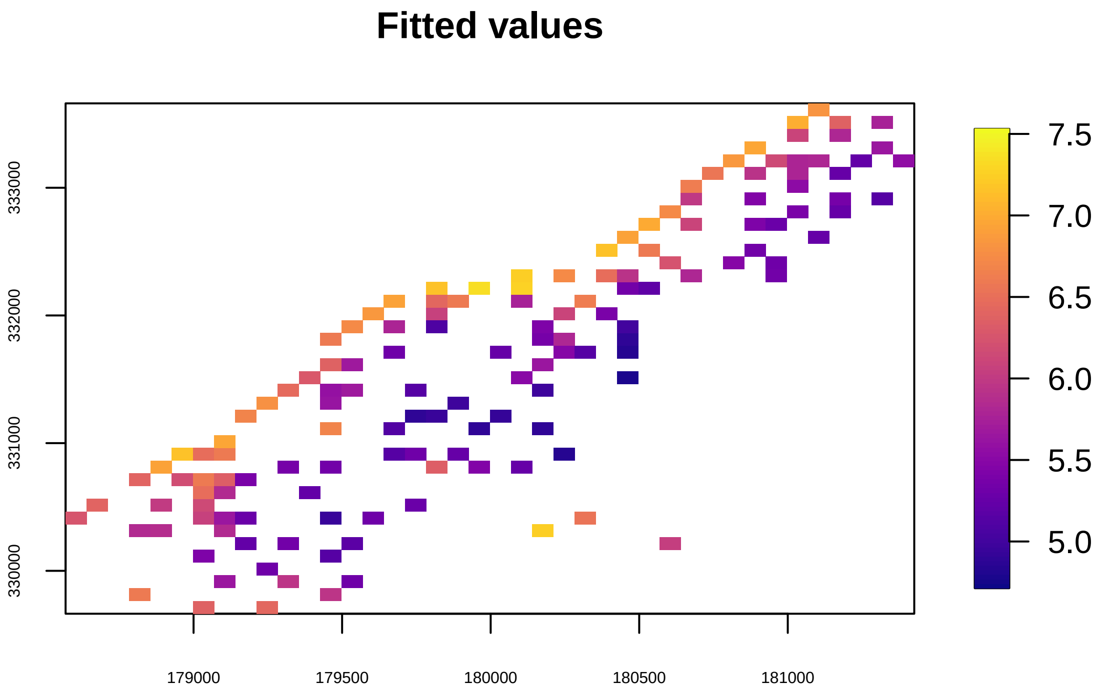
We now plot the predictive standard deviation of the spatial field.

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



And finally, we plot the fitted values.

```
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