Assignment Block A

Bianca Neubert, Franziska Wehrmann

Theory

Spatial Processes - Moving Average

Define a process over the set of n locations $s_1, ..., s_n \in S$. For $\epsilon(s_1), ..., \epsilon(s_n) \sim (0, \sigma^2)$, iid, we define

$$Y(s_i) = \epsilon(s_i) + \beta \cdot \sum_{j \in N(s_i)} \omega_{ij} \epsilon(s_j) \quad , i = 1...n$$

where $N(s_i) := \{j : \ \|s_i - s_j\| < \delta\}$, ω_{ij} are MA weights and β controls for the strength of spatial interrelation.

1. Determine $\mathbb{E}[Y(s_i)]$ for $i \in \{1, ..., n\}$:

$$\begin{aligned}
\mathbb{E}[Y(s_i)] &= \mathbb{E}\left[\epsilon(s_i) + \beta \cdot \sum_{j \in N(s_i)} \omega_{ij} \epsilon(s_j)\right] \\
&= \mathbb{E}[\epsilon(s_i)] + \beta \cdot \sum_{j \in N(s_i)} \omega_{ij} \mathbb{E}[\epsilon(s_j)] \\
&= 0 + \beta \cdot \sum_{j \in N(s_i)} \omega_{ij} \cdot 0 \\
&= 0
\end{aligned}$$

Here, we use that $\mathbb{E}[\epsilon(s_i)0] = 0$ for all $i \in \{1, ..., n\}$ and the linearity of the expectation.

2. Determine $\mathbb{V}[Y(s_i)]$ for $i \in \{1, ..., n\}$:

$$\begin{aligned}
\mathbb{V}[Y(s_i)] &= \mathbb{V}\left[\epsilon(s_i) + \beta \cdot \sum_{j \in N(s_i)} \omega_{ij} \epsilon(s_j)\right] \\
&= \mathbb{V}[\epsilon(s_i)] + \mathbb{V}\left[\beta \cdot \sum_{j \in N(s_i)} \omega_{ij} \epsilon(s_j)\right] + 2\mathbb{C}\text{ov}\left[\epsilon(s_i), \beta \cdot \sum_{j \in N(s_i)} \omega_{ij} \epsilon(s_j)\right] \\
&= \sigma^2 + \beta^2 \mathbb{V}\left[\sum_{j \in N(s_i)} \omega_{ij} \epsilon(s_j)\right] + 2\beta \sum_{j \in N(s_i)} \omega_{ij} \mathbb{C}\text{ov}[\epsilon(s_i), \epsilon(s_j)] \\
&= \sigma^2 + \beta^2 \sum_{j \in N(s_i)} \omega_{ij}^2 \mathbb{V}\left[\epsilon(s_j)\right] + 2\beta \omega_{ii} \sigma^2 \\
&= \sigma^2 \left(1 + \beta^2 \sum_{j \in N(s_i)} \omega_{ij}^2 + 2\beta \omega_{ii}\right)
\end{aligned}$$

Here, we use that $s_i \in N(s_i)$, $\mathbb{C}\text{ov}[\epsilon(s_i), \epsilon(s_j)] = 0$ for $i \neq j$ and else σ^2.

3. Determine $\mathbb{C}\mathrm{ov}[Y(s_i), Y(s_j)]$ for $i, j \in \{1, ..., n\}$ and $i \neq j$:

$$\begin{aligned}
\mathbb{C}\mathrm{ov}[Y(s_i), Y(s_j)] &= \mathbb{C}\mathrm{ov}\left[\epsilon(s_i) + \beta \cdot \sum_{l \in N(s_i)} \omega_{il}\epsilon(s_l), \epsilon(s_j) + \beta \cdot \sum_{k \in N(s_j)} \omega_{jk}\epsilon(s_k)\right] \\
&= \mathbb{C}\mathrm{ov}[\epsilon(s_i), \epsilon(s_j)] + \mathbb{C}\mathrm{ov}\left[\epsilon(s_i), \beta \cdot \sum_{k \in N(s_j)} \omega_{jk}\epsilon(s_k)\right] \\
&\quad + \mathbb{C}\mathrm{ov}\left[\beta \cdot \sum_{l \in N(s_i)} \omega_{il}\epsilon(s_l), \epsilon(s_j)\right] + \mathbb{C}\mathrm{ov}\left[\beta \cdot \sum_{l \in N(s_i)} \omega_{il}\epsilon(s_l), \beta \cdot \sum_{k \in N(s_j)} \omega_{jk}\epsilon(s_k)\right] \\
&= 0 + \beta \cdot \sum_{k \in N(s_j)} \omega_{jk}\mathbb{C}\mathrm{ov}[\epsilon(s_i), \epsilon(s_k)] + \beta \cdot \sum_{l \in N(s_i)} \omega_{il}\mathbb{C}\mathrm{ov}\left[\epsilon(s_l), \epsilon(s_j)\right] \\
&\quad + \beta^2 \sum_{l \in N(s_i)} \sum_{k \in N(s_j)} \omega_{il}\omega_{jk}\mathbb{C}\mathrm{ov}[\epsilon(s_l), \epsilon(s_k)] \\
&= \beta\sigma^2\omega_{ji}\mathbb{1}_{\{i \in N(s_j)\}} + \beta\sigma^2\omega_{ij}\mathbb{1}_{\{j \in N(s_i)\}} + \beta^2\sigma^2 \sum_{l=1}^{n} \mathbb{1}_{\{j \in N(s_i) \cap N(s_j)\}}\omega_{il}\omega_{jl} \\
&= \mathbb{1}_{\{i \in N(s_j)\}}\beta\sigma^2(\omega_{ji} + \omega_{ij} + \beta \sum_{l=1}^{n} \mathbb{1}_{\{j \in N(s_i) \cap N(s_j)\}}\omega_{ij}\omega_{jl})
\end{aligned}$$

where $\mathbb{1}$ indicates the indicator function and we have that $i \in N(s_j)$ is equivalent to $j \in N(s_i)$.

4. Is this a second-order stationary process? The random field $\{Y(s)\}$ is second-order stationary if

i) $\mathbb{E}[Y(s)] = m$ for all $s \in S$. This is fulfilled with $m = 0$, see 3.

ii) $\mathbb{E}[Y(s)Y(s+h)] = C(h)$ for any $s, s+h \in S$ and C only depends on h. This is not the case without further constraints on the weights.

Hence $\{Y(s)\}$ is not a second-order stationary process.

(Semi-)Variogram and Correlation Function

Let $Y = \{Y(s), s \in S\}$ denote a second-order IRF where $S \subset \mathbb{R}^2$ with $m = 0$ and covariogram $C(h)$, $C(0) = \sigma^2 = 1$.

To show: For the correlogram $\rho(h)$ of Y we have that

$$\gamma(h) = 1 - \rho(h)$$

Proof: First we have per definition and with $C(0) = 1$ that $\rho(h) = \frac{C(h)}{C(0)} = C(h)$. For the variogram γ we get with its defintion and the defintion of C for $s, s+h \in S$

$$\begin{aligned}
2\gamma(h) &= \mathbb{V}[Y(s+h) - Y(s)] \\
&= \mathbb{V}[Y(s+h)] + \mathbb{V}[Y(s)] - 2\mathbb{C}\mathrm{ov}[Y(s+h), Y(s)] \\
&= \mathbb{C}\mathrm{ov}[Y(s+h), Y(s+h+0)] + \mathbb{C}\mathrm{ov}[Y(s), Y(s+0))] - 2\mathbb{C}\mathrm{ov}[Y(s+h), Y(s)] \\
&= C(0) + C(0) - 2C(h) \\
&= 2 - 2C(h).
\end{aligned}$$

From this we get for the correlogram that

$$\gamma(h) = 1 - C(h) = 1 - \rho(h)$$

which concludes the proof.

Computation

The Jura Data

For this task, we look at the Jura data, originally collected by the Swiss federal Institute of Technology at Lausanne which contains information on the concentrations of seven heavy metals (cadium, cobalt, chromium, copper, nickel, lead and zinc) in the top soil at each location. For this, we use the packages **gstat** and **geoR**. First, let us have a look at the data.

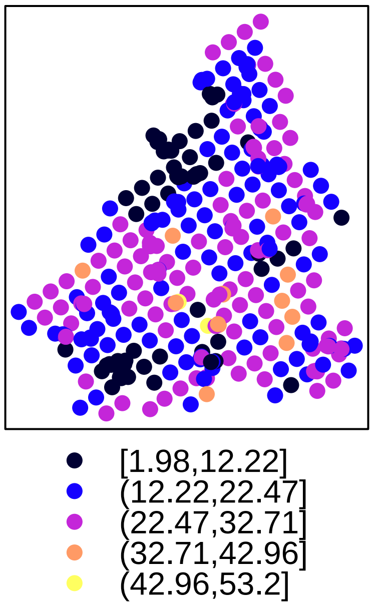
```
jura <- read.table("jura.txt", header = TRUE)
head(jura)
```

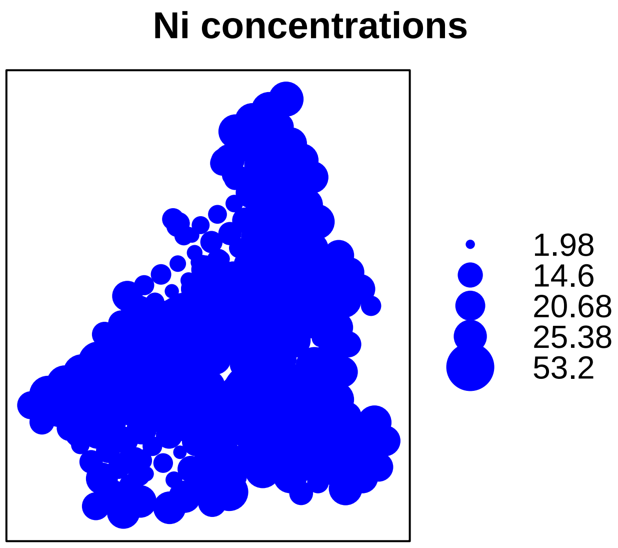
```
##       x     y Landuse Rock    Cd    Co    Cr    Cu    Ni    Pb    Zn
## 1 2.672 3.558       3    5 1.570  8.28 37.12 18.60 18.60 38.20  65.2
## 2 3.589 4.443       3    1 2.045 10.80 40.80 11.48 21.52 33.36 112.8
## 3 4.010 4.713       2    1 1.203 12.00 53.20 13.04 23.92 26.56  91.6
## 4 2.942 3.137       2    5 0.490 10.92 23.40  5.64 14.60 25.88  41.2
## 5 1.409 2.748       3    3 0.692  8.12 27.16 10.32 14.64 31.16  50.4
## 6 3.978 2.910       1    2 1.750  9.12 35.48  8.36 26.40 37.72  63.2
```

Now, we want our data to be saved as a SpatialPointsDataFrame, so we add coordinates (We keep `jura` just for convenience). Also, we generate a convex grid using the function `build.convex.grid` as a surface for predictions later and ensure that it is of class SpatialPoints:

```
cj <- read.table("jura.txt", header = TRUE)
coordinates(cj) = ~x+y
jg <- data.frame(build.convex.grid(jura[,1], jura[,2],10000))
names(jg) <- c("x","y")
jg <- SpatialPoints(jg)
```

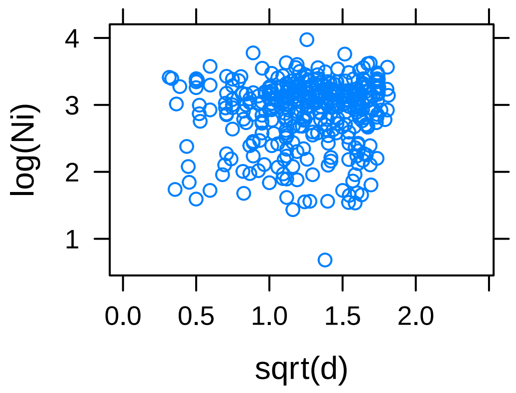
For this task, we only consider the concentration of nickel (`Ni`).To get a first impression, we map the measured concentration of nickel with `spplot` and `bubble` (the bubble plot is not very helpful in this display, but we did not want to include very big plots).





We could also compute distances (e.g. Euclidean distance) and look if there is a relation between it and the concentration of nickel:

```
d <- dist(jura[1:2])
xyplot(log(Ni) ~ sqrt(d), as.data.frame(jura))
```



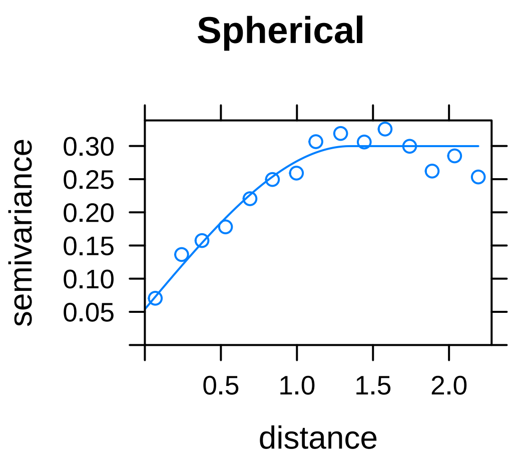
Why does it not seem to be as if there is a relation? And what exactly was plotted here?

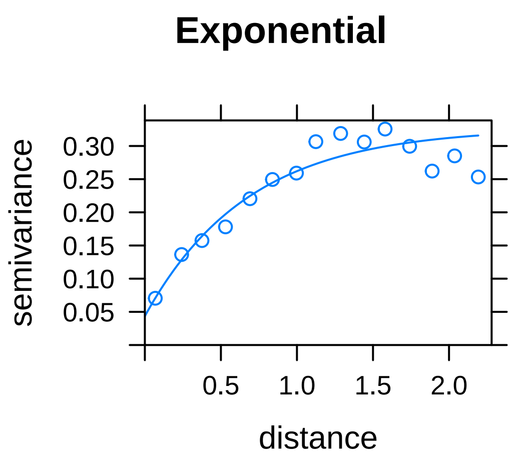
(Semi-)Variograms

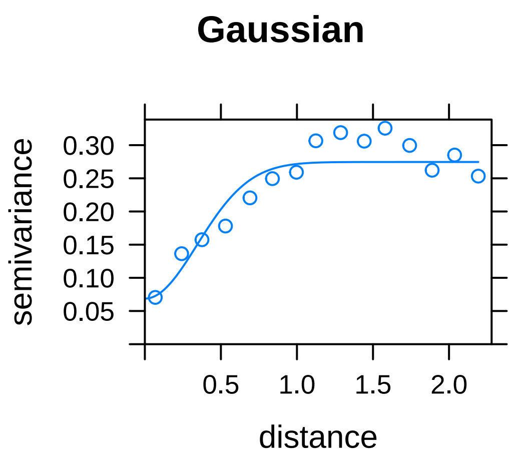
Let us compute (semi-)variograms from the data using the `variogram` function, where we assume no trend for variable `log(Ni)` and fit different variogram models (Spherical, Exponential, Gaussian and Matern). First, we use the default classical method of moments variogram estimate for `variogram` and produce plots.

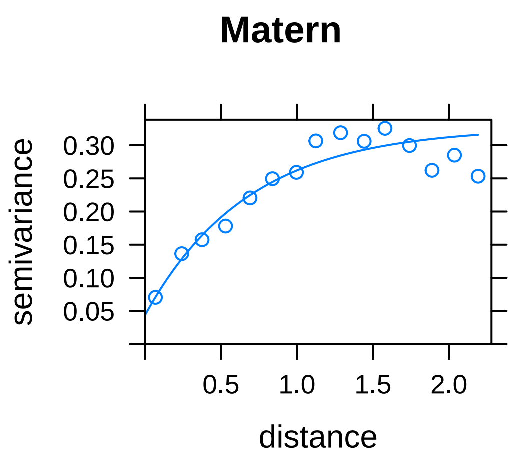
```
v1  = variogram(log(Ni)~1, cj)
v11.fit <- fit.variogram(v1, model = vgm(1, "Sph", 10, 1))
v12.fit <- fit.variogram(v1, model = vgm(1, "Exp", 10, 1))
v13.fit <- fit.variogram(v1, model = vgm(1, "Gau", 10, 1))
```

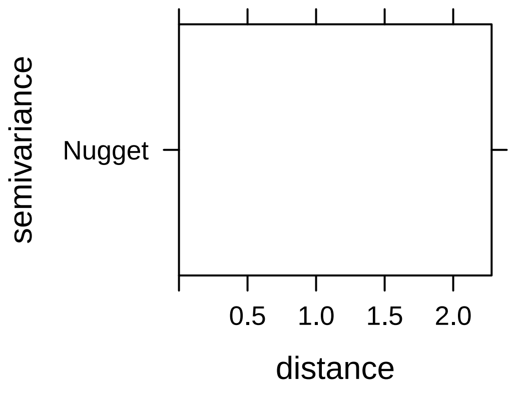
```r
v14.fit <- fit.variogram(v1, model = vgm(1, "Mat", 10, 1))
v15.fit <- fit.variogram(v1, model = vgm("Nug"))
v16.fit <- fit.variogram(v1, model = vgm("Lin"))
```

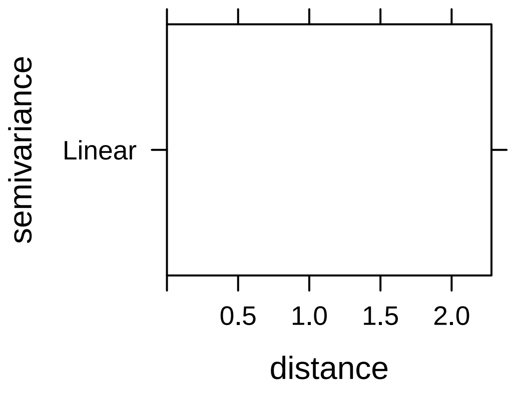






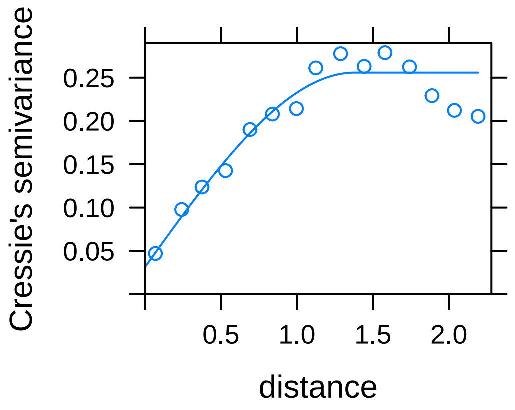






We can also use Cressie's version for `variogram` and, hence, compute a robustified version, here we only fit the spherical model.

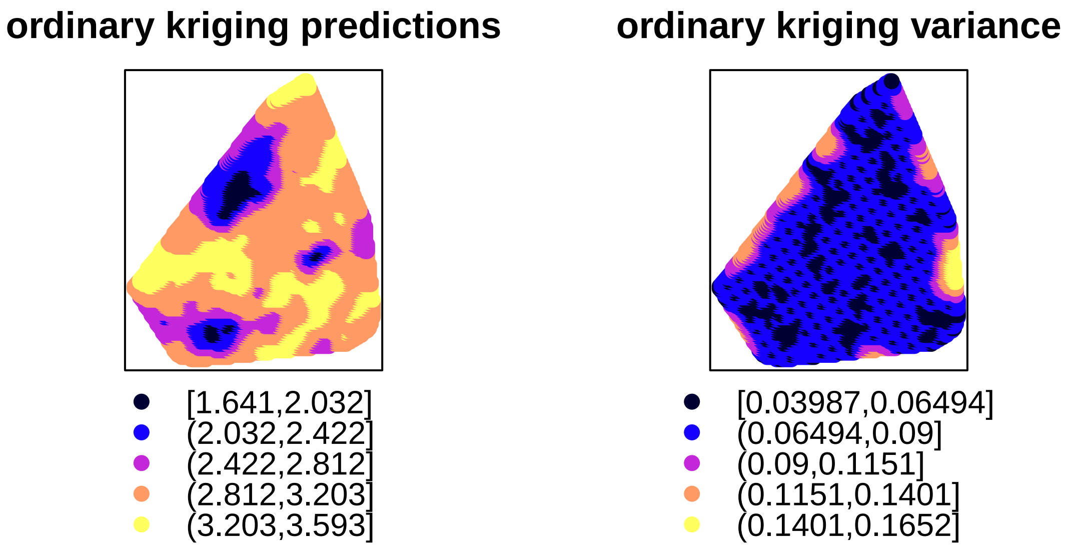
```r
v2 = variogram(log(Ni)~1, cressie  = TRUE, cj)
v2.fit <- fit.variogram(v2, model = vgm(1, "Sph", 10, 1 ))
```



Finally, we can perform kriging with the `krige` function where we use the spherical fitted model from Cressie's version. With `ssplot` we can plot the predictions and its variances.

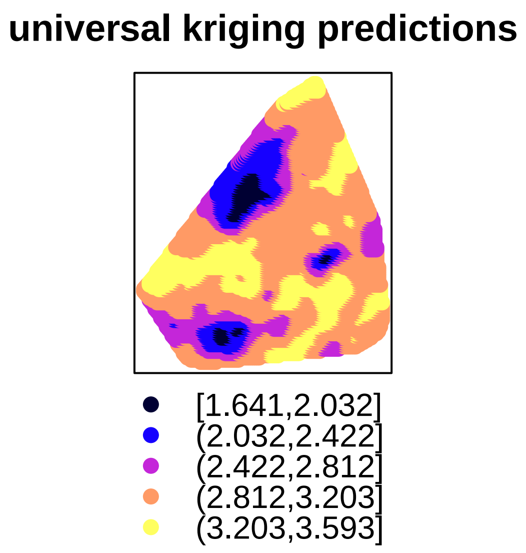
1. Ordinary Kriging (OK)

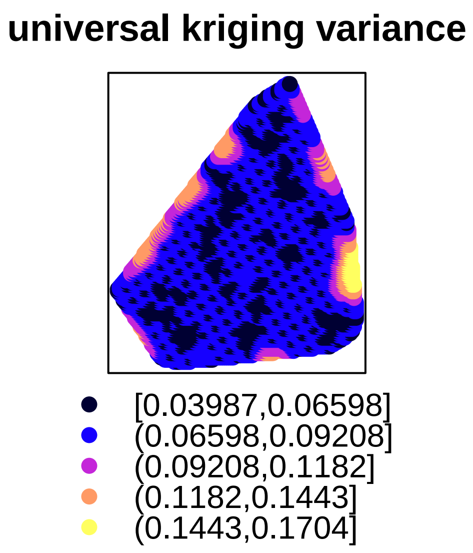
```r
x <- krige(log(Ni)~1, cj, jg, model = v2.fit)
```



2. Universal Kriging (UK)

```r
y <- krige(log(Ni)~x+y, cj, jg, model = v2.fit)
```





Why is there not much difference?

In addition, we can perform variography and kriging with the R package **geoR**. For this we need to transform the data into a geodata object. We again just look at the nickel concentration which is in column 9 of the `jura` dataframe. Inspect the data and the geodata object.

```
ju.geo <- as.geodata(jura, data.col = 9)
attributes(ju.geo)
```

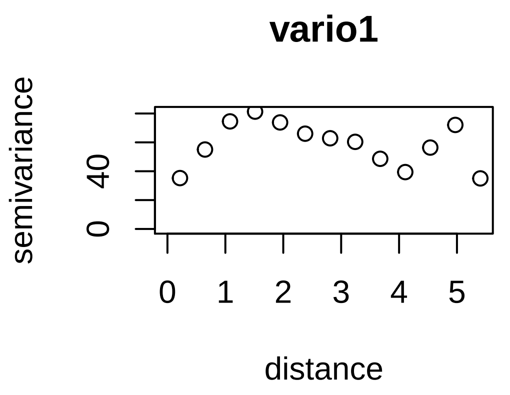
```
## $names
## [1] "coords" "data"
##
## $class
## [1] "geodata"
```

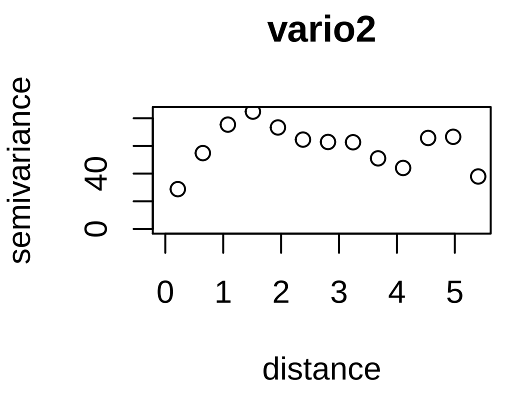
```
summary(ju.geo)
```

```
## Number of data points: 359
##
## Coordinates summary
##         x     y
## min 0.491 0.524
## max 4.920 5.690
##
## Distance summary
##      min      max
## 0.005000 5.619847
##
## Data summary
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##  1.98000 14.60000 20.68000 20.01822 25.38000 53.20000
```

Here, there are multiple possible functions to compute variograms. First, we use `variog` to compute an empirical variogram again with the classical method of moments (‘`vario1`) and with the estimator suggested by Cressie (`vario2`) to get an object of class `variogram` which is needed for the `variofit` function.

```
vario1 <- variog(ju.geo)
vario2 <- variog(ju.geo, estimator.type = "modulus")
```





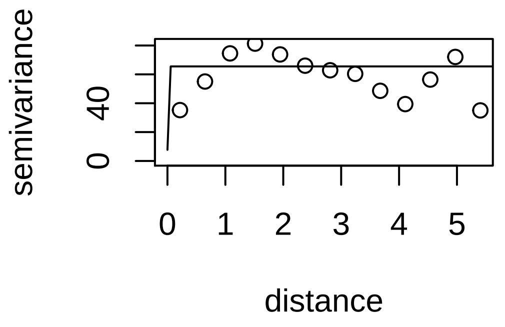
Now, we can apply `variofit` with both models. For the first, we fit the default linear model and for the second we fit a matern variogram with cressie weights in the loss function.

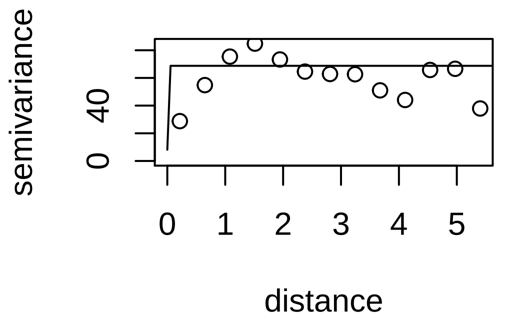
```
f1 <- variofit(vario1)
```

```
## variofit: covariance model used is matern
## variofit: weights used: npairs
## variofit: minimisation function used: optim

## Warning in variofit(vario1): initial values not provided - running the default
## search

## variofit: searching for best initial value ... selected values:
##               sigmasq phi   tausq  kappa
## initial.value "60.96" "0"   "8.13" "0.5"
## status        "est"   "est" "est"  "fix"
## loss value: 10360839.7920689
```

```
f2 <- variofit(vario2, cov.model = "matern", weights = "cressie")
```

```
## variofit: covariance model used is matern
## variofit: weights used: cressie
## variofit: minimisation function used: optim

## Warning in variofit(vario2, cov.model = "matern", weights = "cressie"): initial
## values not provided - running the default search

## variofit: searching for best initial value ... selected values:
##               sigmasq phi   tausq  kappa
## initial.value "63.59" "0"   "8.48" "0.5"
## status        "est"   "est" "est"  "fix"
## loss value: 2565.88358755168
```

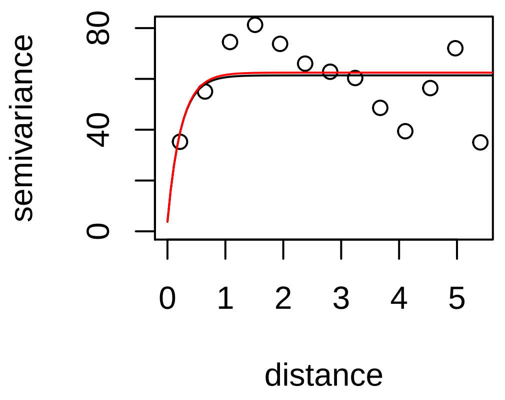




Why do we have so steep curves? Is there an error?

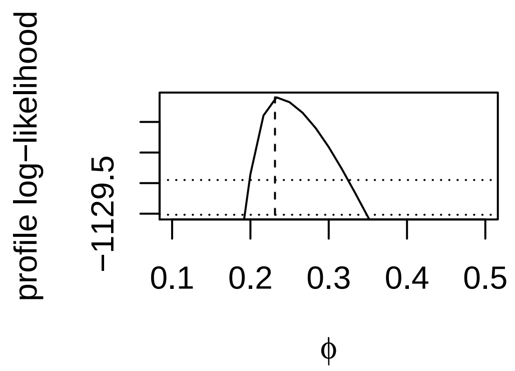
Next, we use `likfit` to estimate a variogram with maximum likelihood (ML) estimation - the default - and resticted maximum likelihood (REML) estimation. In the plot one can compare the empirical variogram `vario1` with the result from `ml`(black line) and `reml`(red line).

```
ml <- likfit(ju.geo, ini.cov.pars = c(0.5, 0.5))
reml <- likfit(ju.geo, ini.cov.pars = c(0.5,0.5), lik.method = "REML")
```



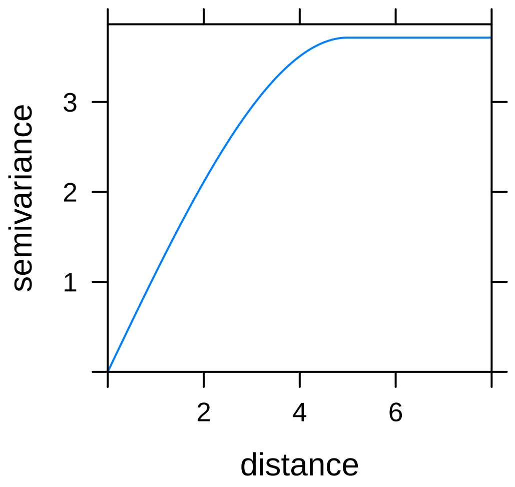
Then, compute profile likelihoods for model parameters with `proflik`.

```
pl  <- proflik(ml, ju.geo,ill.values=seq(0.5, 1.5, l=4),
               range.val=seq(0.1, .5, l=4))
plot(pl, nlevels = 16)
```

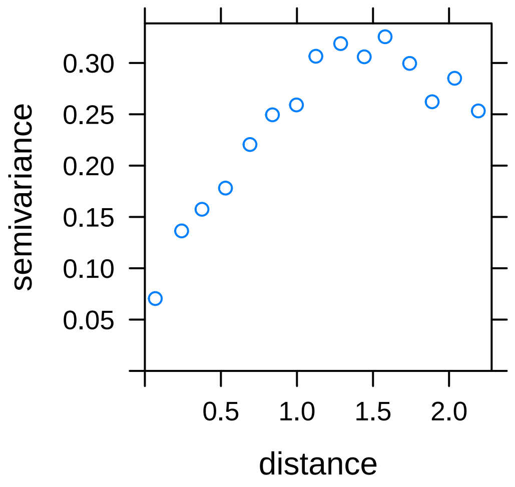


Use `gstat::fit.variogram.reml`:

```r
reml.fit <- fit.variogram.reml(log(Ni)~1, cj, jg, model = vgm(1, "Sph", range=5))
plot(reml.fit, cutoff=8)
```



```r
va3 <- variogram(log(Ni)~1, cj)
plot(va3)
```



Why do they have totally different domain and values? Semivariance with REML are probably scaled by 10, but the domain is still much larger. With a cutoff of 2, we would not see the the variogram reaching the sill yet.

Finally, on could perform a visual modelling using the `eyefit` funciton via

```
eyefit(vario1)
```

How should one compare the results? And how can we interpret the results from the profile likelihoods? Why is there not much difference between ML and REML?

The Gambia Malaria Data

For this task, we use data on malaria prevalence in children obtained at 65 villages in The Gambia. Inspect the data. We see that this data does meet the assumptions of geostatistical data since there are multiple observations for some locations (the same coordinates, that is the same `x` and the same `y` value).

```
data(gambia)
head(gambia)
```

```
##             x       y pos  age netuse treated green phc
## 1850 349631.3 1458055   1 1783      0       0 40.85   1
## 1851 349631.3 1458055   0  404      1       0 40.85   1
## 1852 349631.3 1458055   0  452      1       0 40.85   1
## 1853 349631.3 1458055   1  566      1       0 40.85   1
## 1854 349631.3 1458055   0  598      1       0 40.85   1
## 1855 349631.3 1458055   1  590      1       0 40.85   1
```

```
dim(unique(gambia[,c("x","y")]))
```

```
## [1] 65  2
```

We see that there are 65 different locations. For our anlaysis we need to transform the data which we do using the package **dplyr**. Since we are interested in the prevalence of malaria per location (number of positively tested children divided by the total number of tested children) we add this value to our data. The transformed data is saved in a dataframe `d`.

```
gambia$count <- rep(1, 2035)
d <- gambia %>%
      group_by(x,y) %>%
      summarize(positive = sum(pos), total = sum(count)) %>%
      mutate(prev = positive / total) %>% ungroup
```

The data is in UTM format (Easting/Northing). We change the projection to CRS("+proj=longlat +datum=WGS84"). Finally, we add the longitude and latitude variables to `d`. Inspect the transformed object.

```
spd <- SpatialPoints(d[, c("x", "y")],
                     proj4string = CRS("+proj=utm +zone=28 +datum=WGS84")
)
spdt <- spTransform(spd, CRS("+proj=longlat +datum=WGS84"))

d[,c("long", "lat")] <- coordinates(spdt)
head(d)
```

```
## # A tibble: 6 x 7
##         x       y positive total  prev  long   lat
##     <dbl>   <dbl>    <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 349631. 1458055       17    33 0.515 -16.4  13.2
## 2 358543. 1460112       19    63 0.302 -16.3  13.2
## 3 360308. 1460026        7    17 0.412 -16.3  13.2
## 4 363796. 1496919        8    24 0.333 -16.3  13.5
## 5 366400. 1460248       10    26 0.385 -16.2  13.2
## 6 366688. 1463002        7    18 0.389 -16.2  13.2
```

Now, we construct a map with the locations of the villages and the malaria prevalence using the **leaflet** package. However, this produces html-plots which we don't know how to display here. This is the code we used (which was already provided in the assignment):

```
pal <- colorBin("viridis", bins = c(0, 0.25, 0.5, 0.75, 1))
leaflet(d) %>%
  addProviderTiles(providers$CartoDB.Positron) %>%
  addCircles(lng = ~long, lat = ~lat, color = ~pal(prev)) %>%
  addLegend("bottomright", pal = pal, values = ~prev, title = "Prevalence") %>%
  addScaleBar(position = c("bottomleft"))
```

To join covariate information on the evelation in The Gambia to our model, we use the `getData` function from the **raster** library.

```
r <- getData(name = "alt", country = "GMB", mask = T)
```

We can again compute a map of the relevant raster using the capacities of the **leaflet** via

```
pal <- colorNumeric("viridis", values(r), na.color = "transparent")
leaflet(d) %>%
  addProviderTiles(providers$CartoDB.Positron) %>%
  addRasterImage(r, color = pal) %>%
  addLegend("bottomright",
            pal = pal, values = values(r),
            title = "Altitude") %>%
  addScaleBar(position = c("bottomleft"))
```

We now extract the altitude values at the villages locations using the `extract` function of the **raster** package

```
d$alt <- raster::extract(r, d[, c("long", "lat")])
head(d)
```

```
## # A tibble: 6 x 8
##         x       y positive total  prev  long   lat   alt
##     <dbl>   <dbl>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 349631. 1458055       17    33 0.515 -16.4  13.2    14
## 2 358543. 1460112       19    63 0.302 -16.3  13.2    30
## 3 360308. 1460026        7    17 0.412 -16.3  13.2    32
## 4 363796. 1496919        8    24 0.333 -16.3  13.5    20
## 5 366400. 1460248       10    26 0.385 -16.2  13.2    28
## 6 366688. 1463002        7    18 0.389 -16.2  13.2    17
```

Now, we turn to fitting the prevalence model using the INLA and SPDE approach where we consider the following specifications of the prevalences

$$Y_i|P(s_i) \sim \text{Bin}(N_i, P(s_i))$$

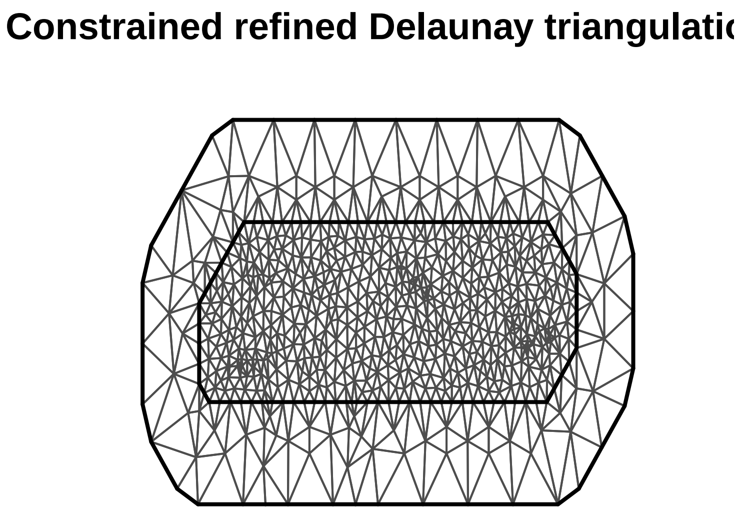
where $P(s_i)$ is the true prevalence at location $s_i,\ i = 1, ..., n$, and Y_i is the number of positive results out ofN_i people sampled at s_i such that

$$logit(P(s_i)) = beta0 + beta1 * altitude + f(s_i)$$

where $f(s_i)$ is a spatial random effect following a zero-mean Gaussian process with Matern covariance function.

To begin with, define a mesh using the **INLA** library and plot the mesh:

```
coo <- cbind(d$long, d$lat)
mesh <- inla.mesh.2d(
  loc = coo, max.edge = c(0.1, 5),
  cutoff = 0.01
)
```



Here, max.edge determines the largest allowed triangle edge length for the outer edges and the inner edges. Changing the value for cutoff changes the minimum allowed distance between points and replaces points closer than this value with one single edge. So if the value is very large there are only very few points considered and, thus, not very many triangles and if the value is very small there are all points considered and the mesh is very finely structured not combing any similar points.

For the prediction with inla, first, generate an index set and projection matrix and set the stochastic partial differential equation.

```
spde <- inla.spde2.matern(mesh = mesh, alpha = 2, constr = TRUE)
A <- inla.spde.make.A(mesh = mesh, loc = coo)
indexs <- inla.spde.make.index("s", spde$n.spde)
```

Compute the predicting location through rasterToPoints and inspect the new object.

```
s0 <- rasterToPoints(r)
dim(s0)
```

```
## [1] 12964     3
```

Compute a lower number of predicting locations by aggregation from r specifying fact = 5 in the aggregate function and extract the coordinates from this reduced set of prediction points and compute a prediction matrix Ap.

```
ag <- aggregate(r, fact = 5)
c <- rasterToPoints(ag)
coop <- c[, c("x", "y")]

Ap <- inla.spde.make.A(mesh, loc = coop)
```

We need to set up the stack data for the estimation and prediction:

```
stk.e <- inla.stack(tag = "est",
                    data = list(y = d$positive, numtrials = d$total),
                    A = list(1, A),
                    effects = list(data.frame(b0 = 1, cov = d$alt),  s = indexs)
)

stk.p <- inla.stack(tag = "pred",
                    data = list(y = NA, numtrials = NA),
                    A = list(1, Ap),
                    effects = list(data.frame(b0 = 1, cov = c[, 3]), s = indexs)
)

stk.full <- inla.stack(stk.e, stk.p)
```

For the model, we specify the following formula.

```
formula <- y ~ 0 + b0 + cov + f(s, model = spde)
```

Here, 0 removes the intercept and we add a covariate term b0 so that all coovariate terms can be captured in the projection matrix. Finally, we can call inla:

```
res <- inla(formula, family = "binomial", Ntrials = numtrials,
            control.family = list(link = "logit"),
            data = inla.stack.data(stk.full),
            control.predictor = list(compute = TRUE, link = 1, A = inla.stack.A(stk.full)))
```

To inspect the results, we again create a map of the posteriori means from the results using a predefined index. Since the predicted values correspond to points, we use th predicted values correspond to a set of points (which could be seen in the first plot below). We can create a raster with `rasterize`. This is depicted in the second map (which again can not be seen in this pdf).

```
index <- inla.stack.index(stack = stk.full, tag = "pred")$data
prev_mean <- res$summary.fitted.values[index, "mean"]
pal <- colorNumeric("viridis", c(0, 1), na.color = "transparent")

leaflet() %>%
  addProviderTiles(providers$CartoDB.Positron) %>%
  addCircles(
    lng = coop[, 1], lat = coop[, 2],
    color = pal(prev_mean)
  ) %>%
  addLegend("bottomright",
            pal = pal, values = prev_mean,
            title = "Prev."
  ) %>%
  addScaleBar(position = c("bottomleft"))

r_prev_mean <- rasterize(
  x = coop, y = ag, field = prev_mean,
  fun = mean
)

pal <- colorNumeric("viridis", c(0, 1), na.color = "transparent")

leaflet() %>%
  addProviderTiles(providers$CartoDB.Positron) %>%
  addRasterImage(r_prev_mean, colors = pal, opacity = 0.5) %>%
  addLegend("bottomright",
            pal = pal,
            values = values(r_prev_mean), title = "Prev."
  ) %>%
  addScaleBar(position = c("bottomleft"))
```