Geostatistical modelling for predicting soil organic carbon change – a tutorial

Gerard Heuvelink Diana Collazos Cortes

2026-09-01

Funded by the European Union

MRV4SOC PROJECT

Table of contents

Preface

Citation Heuvelink, G. B. M., Collazos Cortes, D. F. (2026). "Geostatistical modelling for predicting soil organic carbon change – a tutorial". ISRIC - World Soil Information. https://doi.org/10.17027/kcwd-yq42

Acknowledgements This tutorial was developed within the **MRV4SOC** project, which received funding from the European Union's Horizon Europe research programme under grant agreement n° 101112754.

1 Introduction

This tutorial explains how geostatistics can be used to predict the total Soil Organic Carbon (SOC) stock for an area, or to predict the change in total SOC stock for an area between two points in time. Geostatistical interpolation benefits from spatial correlation of the variable of interest, but it can also include information provided through correlated explanatory variables. This is usually done by incorporating a trend defined through a statistical regression model, such as a multiple linear regression or random forest machine learning model. An important strength of geostatistical modelling and prediction is that the uncertainty of the predictions is also quantified, by means of a prediction error variance or a prediction interval.

Furthermore, geostatistics has the advantage that, unlike sampling theory and design-based statistical inference, it does not require that the SOC stock observations used for modelling and prediction are a probability sample from the area of interest. It also works in cases where a convenience or purposive sample has been collected. A disadvantage is that geostatistics must assume a statistical model, meaning that its results are only valid under the model assumptions made, while sampling theory and design-based inference are completely model-free. Readers interested in sampling theory for estimating SOC stock and SOC stock change are referred to Sampling theory for estimating soil organic carbon change - a tutorial, also available in the ISRIC Resource Library.

While this tutorial makes use of geostatistics it is not a geostatistics tutorial. For this we refer to Geostatistics for soil mapping. Likewise, readers specifically interested in machine learning for digital soil mapping are referred to Machine Learning for Digital Soil Mapping.

The tutorial explains the underlying theory but the emphasis is on practical application using a concrete example and the R language for statistical computing. The tutorial provides all R scripts and datasets for the example area.

We developed this tutorial such that it should not be difficult to adapt it to other case studies, thus supporting a wide range of Monitoring, Reporting and Verification (MRV) projects. For more information, we refer to the MRV4SOC Project (2026).

Appendix A explains all preparations that must be done prior to running the tutorial. This includes instructions for installing R and RStudio, R packages, and setting up the working directory. Note that we assume R version 4.4.3 or higher.

We assume that users of the tutorial have basic knowledge of statistics, geo-information science and basic experience with R.

2 Soil organic carbon stock of an example area

In this chapter we introduce the example area that is used throughout the tutorial. We present a map of the soil organic carbon (SOC) stock of the area and calculate summary statistics. Note that this is the same study area as previously used in Sampling theory for estimating soil organic carbon change - a tutorial.

All calculations are shown in code chunks, which should be run in the order as presented. We explain the most relevant aspects of the functions used, but we encourage the user to consult the R documentation and experiment with the script by making small modifications or extensions to support a better understanding.

The tutorial materials are available in the ISRIC WebDAV.

The set-up lines below are necessary for the code chunks in this chapter to run. Therefore, we define them at the beginning of the script.

Setup

In this chapter we only use functions from the `terra` package.

```
# load library
library(terra)
```

R automatically prints real numbers in scientific notation. We prefer, however, to have these numbers printed in standard notation. To avoid scientific notation, we adjust the `scipen` ("scientific penalty") option. By setting it to `999`, we essentially force standard notation.

```
# avoid scientific notation
options(scipen=999)
```

The tutorial makes use of a pseudo-random number generator. To encourage reproducible research we set the seed of this generator, so that anyone running the script gets the same results.

```
# set a seed (for reproducible research)
set.seed(12345)
```

```
# uncomment the two code lines below after replacing the file path
# with the location where you saved the tutorial files
# (i.e. that contains the 'soc_stock_kriging' folder)
#setwd("C:/tutorials/soc_stock_kriging")
#workingDir <- getwd()
```

```
# build input path
inputDir <- file.path(workingDir, "input")
```

Numeric results of calculations are often rounded, but since R does not print trailing zeros by default, we wrap the rounding in a number formatting function.

```
# custom rounding function
# x is the number, d the intended number of decimals
# p is the precision to round the number to the nearest factor
# for example, when x=87.8904, d=2 and p=10 returns "87.90"
# for example, when x=33745.5, d=0 and p=100 returns "33700"
round2 <- function(x, d, p = 1){
  p <- p*(10^-d)
  n <- round(x/p)*p
  format(n, nsmall = d)
  }
```

We selected a rectangular study area in the north-east of the Netherlands, shown in the figure below. The rectangle measures 4.8 km by 4.3 km. The main land use is agriculture (cropland and grassland) on clay soils.

> The bounding box was defined in *EPSG:28992 (Amersfoort / RD New)* to match the coordinate system of the input data. When reprojected to *WGS84 (EPSG:4326)* for visualization in Leaflet, the rectangle may appear rotated or skewed. This is a harmless visual effect caused by the transformation from a planar to a geographic coordinate system. All calculations in this tutorial are performed in *EPSG:28992*, so the distortion does not affect the analysis.

In order to sample SOC stock observations from the example area and use this sample to predict the 'true' SOC stock of the area using geostatistics, as well as to evaluate how close this prediction is to the 'true' SOC stock of the area, we need to know the SOC stock everywhere in the area. For the aims of this tutorial we therefore created a SOC stock synthetic reality raster. It is important to be aware that, in reality, we would not know the SOC stock at each and every location in a study area. While we need a map of the 'true' SOC stock for the purposes of this tutorial, in real-world situations we would only know the SOC stock at the sampling locations and derive our geostatistical predictions and prediction intervals from the sample.

Figure 2.1: Study area (red rectangle).

The primary data source for simulation of a synthetic SOC stock raster is Helfenstein et al. (2024a), which provides high-resolution soil property grid maps, including soil organic matter (SOM) and bulk density (BD) maps. We first combined the SOM and BD maps to derive SOC stock maps for the $0-5$ cm , $5-15$ cm and $15-30$ cm depth layers, summed these to obtain SOC stock maps for the $0-30$ cm topsoil, and finally added zero-mean spatially-correlated noise using spatial stochastic simulation. Noise was added to acknowledge that the $0-30$ cm SOC stock map derived from the maps provided in Helfenstein et al. (2024a) is a smoothed representation of the true SOC stock.

Load data

```
# load soc raster
soc_fn <- "soc_stock_2020.tif"
soc <- rast(file.path(inputDir, soc_fn))
```

Plot raster

```
# plot soc raster
plot(soc, main = "topsoil SOC stock",
     plg = list(title = "ton/ha"))
```

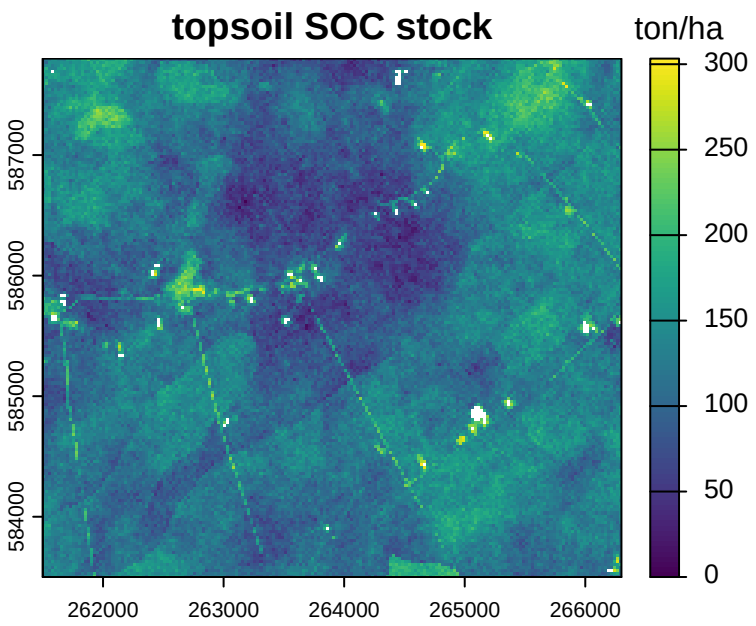


Figure 2.2: Topsoil SOC stock map of the example area, at 25m x 25m resolution. This map defines the 'reality' from which we will sample.

Number of NA pixels

```
# calculate the number of NA pixels
n_na <- global(is.na(soc), "sum", na.rm = TRUE)[[1]]

# check
n_na
```

```
[1] 92
```

To calculate the SOC stock for the example area we average all grid cell values and multiply by the size of the area. Note that there are 92 `NA` values in the rectangle shown in Figure 2.2 (i.e., houses and farm buildings). These cells are not part of the target area and must therefore be excluded from the calculations.

Total area

```
# terra::expanse() returns the area covered by all non-NA raster cells
area_ha <- expanse(soc, unit = "ha")[1, "area"]

# print
cat("Area:", round2(area_ha, 2), "ha")
```

```
Area: 2057.93 ha
```

'True' SOC stock

```
# calculate the mean soc stock across the non-NA pixels (ton/ha)
mean_stock <- global(soc, fun = "mean", na.rm=TRUE)[1, "mean"]

# print
cat("'True' mean SOC stock:", round2(mean_stock, 1), "ton/ha")
```

```
'True' mean SOC stock: 120.7 ton/ha
```

```
# multiply the mean SOC stock (ton/ha) by area (ha)
# to get the total SOC stock (ton)
tot_stock <- mean_stock * area_ha

# print
cat("'True' SOC stock:", round2(tot_stock, 0, 10), "ton")
```

```
'True' SOC stock: 248390 ton
```

This is 248.4 *Gg* (Gigagram).

Figure 2.2 also makes clear that there is considerable spatial variation in the SOC stock, with values ranging from almost 0 to more than 300~ton/ha. We can summarise spatial variation by calculating the standard deviation, while it is also instructive to plot a histogram:

Standard deviation and histogram

```
# calculate the standard deviation across the non-NA pixels (ton/ha)
sd_stock <- global(soc, fun = "sd", na.rm=TRUE)[1,"sd"]

# and from this the coefficient of variation (%)
# to compare spatial variation with the spatial mean
cv_stock <- sd_stock/mean_stock * 100

# print
cat("Standard deviation SOC stock:", round2(sd_stock, 1), "ton/ha")
```

```
Standard deviation SOC stock: 36.1 ton/ha
```

```
cat("Coefficient of variation SOC stock:", round2(cv_stock, 1), "%")
```

```
Coefficient of variation SOC stock: 29.9 %
```

```
# histogram
par(mar=c(4,4,0,0)) # c(bottom, left, top, right)
hist(soc,
     xlab = "SOC stock (ton/ha)", main = NULL,
     freq = TRUE,
     breaks = 24,
     cex.main = 0.8, cex.lab = 0.8, cex.axis = 0.8)
```

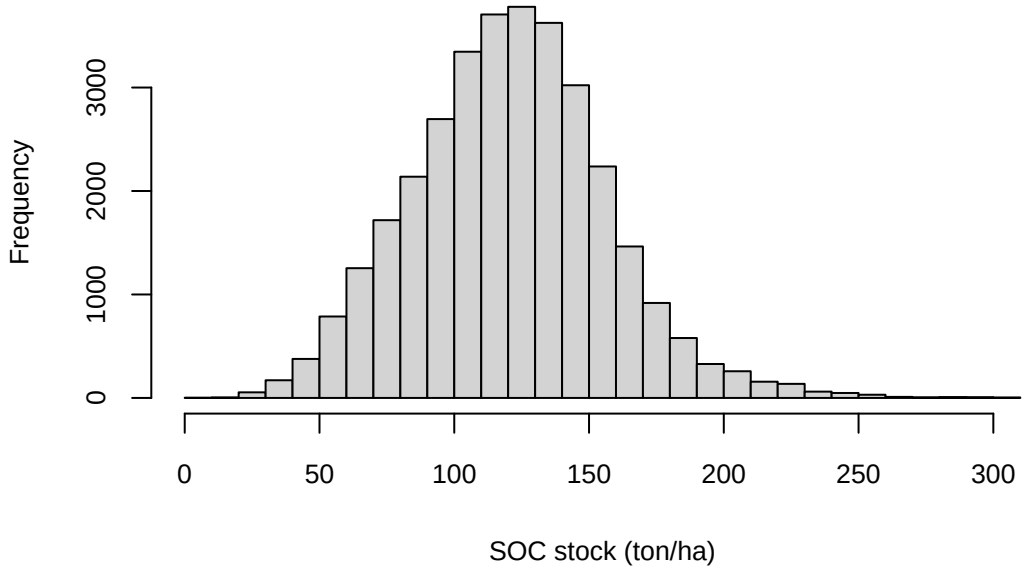


Figure 2.3: Frequency distribution of all SOC stock values in the example area.

3 Geostatistical theory and ordinary block kriging

In geostatistics, we assume that the variable of interest, denoted by $z = \{z(s)|s \in A\}$ is a realization of a statistical model, denoted by $Z = \{Z(s)|s \in A\}$. Here, s refers to a geographic location and A is the area of interest. Note the difference between the lower case $z(s)$, which refers to a deterministic value (i.e., a number), and the upper case $Z(s)$, which is a random variable. This important distinction can be illustrated with the example of rolling a die. If someone rolls a die and gets the outcome $x = 5$, but hides that outcome to you, then you would interpret x as a realization of a random variable X, where X has a discrete, uniform probability distribution, consisting of all six values from 1 to 6, and where each value has equal probability, namely 1/6. You do not know x and hence you work with X, which you do know, that is you know its probability distribution. Returning to our case, we interpret the SOC stock $z(s)$ at any unmeasured location s in the example area as a realization of a random variable $Z(s)$, and we continue by defining the probability distribution of that random variable for all locations $s \in A$. Note that the only reason that we adopt a stochastic model is because we are uncertain about the reality, not because reality itself is stochastic.

The first step in geostatistics is then to define the probability distribution of Z. This is not an easy task because Z is a collection of many random variables, one for each location s in A, and these random variables will likely be correlated. All this has to be incorporated in the geostatistical model. A common approach is to define the variable of interest as a sum of two components:

$$Z(s) = \mu(s) + \varepsilon(s), \quad \text{for all } s \in A$$

Here, μ is a deterministic **trend** and ε is a zero-mean, normally distributed stochastic **residual**, whose variance $var(\varepsilon(s))$ we will denote by $\sigma^2(s)$. In this tutorial we will mostly assume that μ is constant. Under this assumption, one could for example estimate it by the arithmetic mean of all observations. In fact, later on in this chapter we will see that in many geostatistical interpolation methods, such as in ordinary kriging, one actually does not need to know μ, and that it is enough to 'know' that it is a constant. It is only in Chapter 5 that we will relax the assumption of constant μ and allow it to be some function of explanatory variables.

Having assumed that the trend μ is constant we next turn to characterisation of the stochastic residual ε. This is a key step in any geostatistical analysis because it requires estimation of its

spatial correlation, through a function known as the **variogram**. This is explained in the next section.

3.1 Variogram estimation

Above we introduced the geostatistical model, where Z is the sum of a deterministic trend μ and a stochastic residual ε. We assumed that ε has zero mean and is normally distributed, which means that all that is required to fully characterise it statistically is specification of its variance $\sigma^2(s)$ for all $s \in A$ and the correlation $rho(s, s+h)$ between $\varepsilon(s)$ and $\varepsilon(s+h)$ for any pair of locations s and $s+h$ in A. Note that h refers to the distance between the two locations. Moreover, since μ is deterministic, $\rho(s, s+h)$ is also the correlation between $Z(s)$ and $Z(s+h)$. In geostatistics, this correlation is characterised by the variogram:

$$\gamma(h) = \frac{1}{2} E\left[(Z(s) - Z(s+h))^2\right]$$

Here, E refers to **expected value** (i.e., the mean of the random variable inside the square brackets). Note that in this equation we implicitly assumed that the variogram only depends on the distance h between the two locations, not on the locations s and $s+h$ themselves. This, together with assuming that μ is constant, is known as the **second-order stationarity** assumption. This assumption is often made because otherwise we would not be able to estimate γ from the available point observations. The second-order stationarity assumption also implies that $\sigma^2(s)$ is constant (i.e., it does not depend on s) and that $\rho(s, s+h)$ only depends on the distance h. The latter can easily be calculated from γ through the identity $\rho(h) = 1 - \gamma(h)/\gamma(\infty)$. Here, $\gamma(\infty)$ is the semivariance at distances that are so large that the variable is no longer spatially correlated. It is not difficult to show that $\gamma(\infty) = \sigma^2$.

Estimation of the variogram γ from point observations $z(s_i), i = 1 \ldots n$ is done by calculating half the squared difference between $z(s_i)$ and $z(s_j)$ for all observation pairs and grouping and averaging these for different distance bins or 'lags'. The result is a plot of the variogram value against distance, known as the **experimental variogram**, through which a mathematical curve is fitted, known as the **variogram model**. Typically, the variogram gradually increases with distance up to a distance known as the variogram **range**, after which it levels out and stays constant. The variogram range thus signifies the distance up to which there is spatial correlation. The constant, maximum variogram value is known as the **sill**. One also often finds that at short distances (i.e., locations not far apart), the variogram is larger than zero. This so-called **nugget** effect is caused by short-distance spatial variation and random measurement errors. Together, the nugget, sill and range constitute the three parameters of a variogram model, although some variogram models have more than three parameters.

The increase with distance of a variogram reflects the **first law of geography**, which states that "everything is related to everything else, but near things are more related than distant

things". In the next chapter we will see that this also holds for the SOC stock data in our example area, when we derive its variogram.

3.2 Ordinary kriging

Once the variogram is derived we can use it for spatial interpolation. In other words, we can use it to predict $Z(s_0)$ at any unobserved location $s_0 \in A$ from observations $z(s_i)$ at sampling locations $s_i, i = 1, ..., n$. In **ordinary kriging**, this is done by taking a weighted linear average of the observations:

$$\hat{z}(s_0) = \sum_{i=1}^{n} \lambda_i \cdot z(s_i)$$

Here, the λ_i are so-called **kriging weights**. They are computed from the variogram and the observation and prediction locations by solving a set of $n + 1$ linear equations, known as the kriging system. In ordinary kriging the weights λ_i are forced to sum to 1 (do you know why?). One usually finds that observations nearby the prediction point s_0 get a higher weight than observations further away from s_0, simply because observations nearby have a stronger spatial correlation and are therefore more informative than observations further away.

An attractive property of kriging is that it not only yields a prediction, but that it also quantifies the prediction error, through the so-called **kriging variance**. This is possible because a statistical model has been assumed. Thus, the kriging variance is defined in terms of Z rather than z. In case of ordinary kriging, it is given by:

$$var(Z(s_0) - \hat{Z}(s_0)) = \sum_{i=1}^{n} \{\lambda_i \cdot \gamma(s_i - s_0)\} + \phi$$

where ϕ is a Lagrange parameter, which is automatically computed when solving the kriging system. If we denote the square root of the ordinary kriging variance by $\sigma_K(s_0)$, also known as the **kriging standard deviation**, then we can derive the lower and upper limits of a 95% prediction interval for $z(s_0)$ as:

$$\text{lower limit} = \hat{z}(s_0) - 1.96 \cdot \sigma_K(s_0)$$
$$\text{upper limit} = \hat{z}(s_0) + 1.96 \cdot \sigma_K(s_0)$$

Note that here we used the normal distribution assumption. We can of course also compute the limits of broader or narrower prediction intervals. For example, by replacing 1.96 above by 1.64 we get the lower and upper limits of a 90% prediction interval.

3.3 Ordinary block kriging

The theory above explained how we can predict a variable of interest and quantify the associated prediction uncertainty at any location in the study area using ordinary kriging. This is called ordinary **point** kriging. But this tutorial is about predicting the spatial average (or total) of that variable in the area of interest A. In other words, our goal is not to predict $Z(s_0)$ but to predict:

$$\overline{Z} = \frac{1}{|A|} \int_{s \in A} Z(s)\, ds$$

where $|A|$ is the size of A. This prediction problem is solved by a technique called **block kriging**, where the term 'block' might suggest that A must be square or rectangular, but that in fact works for areas of any shape. In our case the 'block' is the entire example area.

As with any kriging method, ordinary block kriging yields a prediction as well as a prediction error standard deviation. The block kriging prediction is equal to the average of all point kriging predictions inside the block, meaning that it can easily be calculated from a map of the kriging predictions. However, the same does not hold for the block kriging standard deviation: this is not equal to the spatial average of the point kriging standard deviations. In fact, it will usually be much smaller than that, because positive and negative point prediction errors in the study area partially cancel out. To see why this is the case, consider the dice rolling example again: it is very difficult to predict the outcome of a roll with a single die (it could be any number between 1 and 6, high uncertainty), but it is much easier to predict the average of 1000 rolled dice (it will be pretty close to 3.5, low uncertainty). This is because high and low values partially cancel out.

Geostatistical text books explain how the ordinary block kriging variance is computed. The equation used is not very different from that for the ordinary point kriging variance presented above, but a key difference is that it subtracts the average semivariance within the block. This confirms that the block kriging variance is usually much smaller than the point kriging variance, especcially when the nugget is large.

While point kriging standard deviations can be large, especially when there is a weak spatial correlation and low sampling density, the block kriging standard deviation can be small, especially if we average over a large area. This is good news, because it means that we can obtain accurate predictions of the spatial average and total, even if the sampling density is not that high. In the next chapter we will apply ordinary block kriging to predict the SOC stock for the example area and quantify the prediction error by the block kriging standard deviation and a 95% prediction interval.

4 Application of ordinary block kriging to the example area

In Chapter 3 we presented the basics of geostatistical theory. In this chapter we will apply it to the example area. We will calculate the variogram of the SOC stock, create a SOC stock prediction map using ordinary kriging and quantify the associated prediction uncertainty with a kriging standard deviation map. We will also apply ordinary block kriging, thus predicting the total SOC stock in the area and quantifying the uncertainty of that prediction with a 95% prediction interval.

To prepare for the calculations in this chapter, we first run the following set-up lines.

Setup

```
# load libraries
library(terra)
library(sf)
library(gstat)
library(stars)
library(ggplot2)
library(patchwork)
```

```
# avoid scientific notation
options(scipen=999)
```

```
# set a seed
set.seed(987)
```

```
# uncomment the two code lines below after replacing the file path
# with the location where you saved the tutorial files
# (i.e. that contains the 'soc_stock_kriging' folder)
#setwd("C:/tutorials/soc_stock_kriging")
#workingDir <- getwd()
```

```
# build input path
inputDir <- file.path(workingDir, "input")
outputDir <- file.path(workingDir, "output")
```

```r
# custom rounding function
round2 <- function(x, d, p = 1, pad_number = TRUE){
  p <- p*(10^-d)
  n <- round(x/p)*p
  if (pad_number){
    format(n, nsmall = d)
  } else {n}
}
```

Load data

```r
# load soc raster
soc_fn <- "soc_stock_2020.tif"
soc <- rast(file.path(inputDir, soc_fn))
```

Prepare data

```r
# modify soc raster metadata to make it easier to call
soc_values <- "soc_ton_ha"
names(soc) <- soc_values
```

```r
# number of NA pixels
n_na <- global(is.na(soc), "sum", na.rm = TRUE)[[1]]

# terra::expanse() returns the area covered by all non-NA raster cells
area_ha <- expanse(soc, unit = "ha")[1, "area"]

# calculate the mean soc stock across the non-NA pixels (ton/ha)
mean_stock <- global(soc, fun = "mean", na.rm=TRUE)[1, "mean"]

# get the total SOC stock (ton)
total_stock <- mean_stock * area_ha

# print
cat("'True' SOC stock:", round2(total_stock, 0, 10), "ton")
```

```
'True' SOC stock: 248390 ton
```

Define sample size

```
sample_size <- 200
```

The code above loaded the map of the 'true' SOC stock in the example area and computed the total SOC stock from that. In reality, we do not know the SOC stock at all locations in the area, and therefore we cannot calculate the total SOC stock. Instead, we must estimate or predict it from a limited number of SOC stock observations. In this chapter we will assume that we measured the SOC stock at $n = 200$ sampling locations. We will use these 200 observations to predict the total SOC stock with ordinary block kriging.

The advantage of working with a 'synthetic reality' (i.e., the map of the 'true' SOC stock in the example area) is that we can check how close the ordinary block kriging prediction is to the 'true' total SOC stock, that we can check if the prediction interval that we will also compute includes the 'true' total SOC stock, and that we can apply different sampling schemes, because having the 'reality' at our disposal means that we can sample everywhere.

We continue by collecting the SOC stock values at the 200 sampling locations.

Randomly sample 200 locations from the example area

```
# sample non-NA locations as spatial points. In spatSample(),
# when as.points = TRUE, it converts the sampled cells into
# spatial points using xyFromCell(), which by default returns the
# cell centroid. When values = TRUE, raster cell values are returned
random_sample <- spatSample(
  soc,
  size = sample_size,
  method = "random",
  replace = FALSE,
  as.points = FALSE,
  na.rm = TRUE,
  values = TRUE,
  xy = TRUE # return cell centroid coordinates
)
```

Random sample jitter

```
# generate a random coordinates offset within ~1 meter
max_offset <- 1

# extract coordinates
coords <- random_sample[c("x", "y")]

# runif() generates random deviates from a uniform distribution
dx <- runif(nrow(coords), -max_offset, max_offset)
dy <- runif(nrow(coords), -max_offset, max_offset)

# add the offset to the coordinates
random_sample_offset <- cbind(
  random_sample["x"] + dx,
  random_sample["y"] + dy,
  random_sample[soc_values])

# from data frame to SpatVector
random_sample_sv <- vect(random_sample_offset,
                         geom = c("x", "y"),
                         crs = crs(soc),
                         keepgeom = TRUE)
```

Random sample checks

```
# check number of points
nrow(random_sample_sv)

# check that we don't have points falling on the same xy coordinates
sum(duplicated(random_sample_sv[[c("x", "y")]]))
```

Random sample plot

```
# check min and max values for plotting
ceiling(minmax(soc))

# since the ceiling of zero is zero, we can apply this rounding to both values
range_plot <- as.vector(ceiling(minmax(soc)))
```

```r
# terra::plot()
plot(soc, axes= FALSE, box=TRUE, cex.main = 1, range = range_plot,
     plg = list(title = "ton/ha", title.cex = 0.8, cex = 0.8))
plot(random_sample_sv, pch = 21, cex = 1,
     col = "black", bg = "orange", add = TRUE)
```

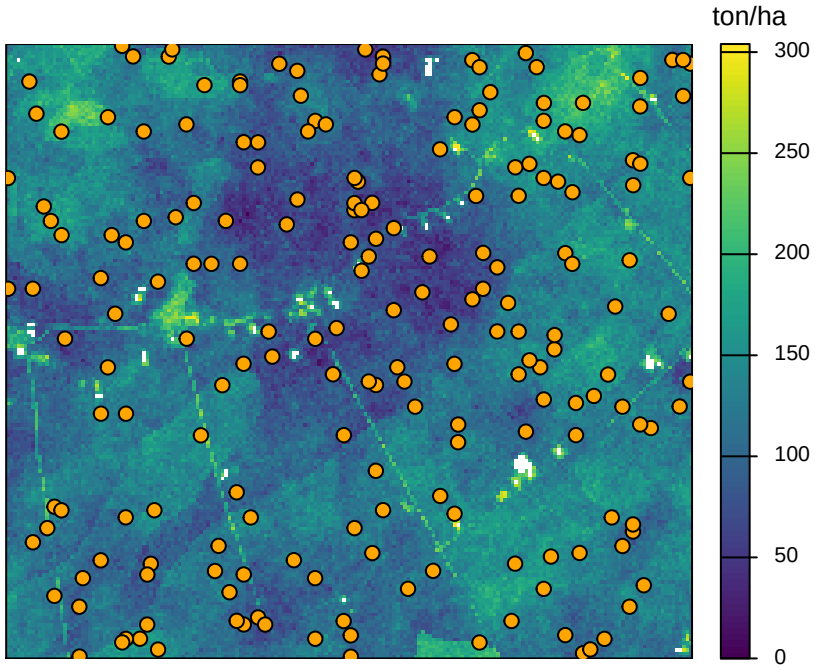


Figure 4.1: Random sample of 200 locations in the example area.

Notice that we selected the $n = 200$ locations randomly, but that we might also have used a different spatial sampling design, such as regular grid sampling or clustered sampling. Random sampling is not required in geostatistics and is likely suboptimal. We only did this because it meets the goals of this chapter and because it is easy to do in R (but not in the field!).

The script above also added a small jitter to the geographic coordinates of the observations. You might remember from the previous chapter that in ordinary block kriging we need to predict at all locations in the area of interest. In practice we discretise the area into grid cells, we predict at the centres of these grid cells, and use the average of these predictions to approximate the average of the predictions for the whole area. This introduces an approximation error, but that error will be small if the discretisation grid is sufficiently fine and if we ensure that none of the

grid centes coincides with a measurement location. Adding a small noise to the geographic coordinates of the measurement locations achieves the latter: it prevents that measurement locations are identical to grid cell centres.

4.1 Variogram estimation

To prepare for ordinary kriging we first need to fit a variogram from the 200 observations. This section shows how this is done. For a more detailed explanation, we refer to the Geostatistics for soil mapping tutorial, also stored in the ISRIC Resource Library.

It is always useful to start by calculating summary statistics of the data and plotting a histogram.

Summary statistics and histogram of the sample data

```
# summary statistics (ton/ha)
summary(random_sample_sv[soc_values])
```

```
   soc_ton_ha
 Min.   : 11.53
 1st Qu.: 89.02
 Median :116.28
 Mean   :117.15
 3rd Qu.:142.80
 Max.   :225.62
```

```
# plot margings
par(mar = c(4, 4, 1, 0))  # c(bottom, left, top, right)

# plot histogram
var_plot <- random_sample_sv[[soc_values]][[1]]
hist(var_plot, cex.main = 1, cex.axis = 0.9, cex.lab = 0.9, main = NULL,
     xlab = "SOC stock (ton/ha)", freq = TRUE, col = "lightblue")
rug(var_plot)
```

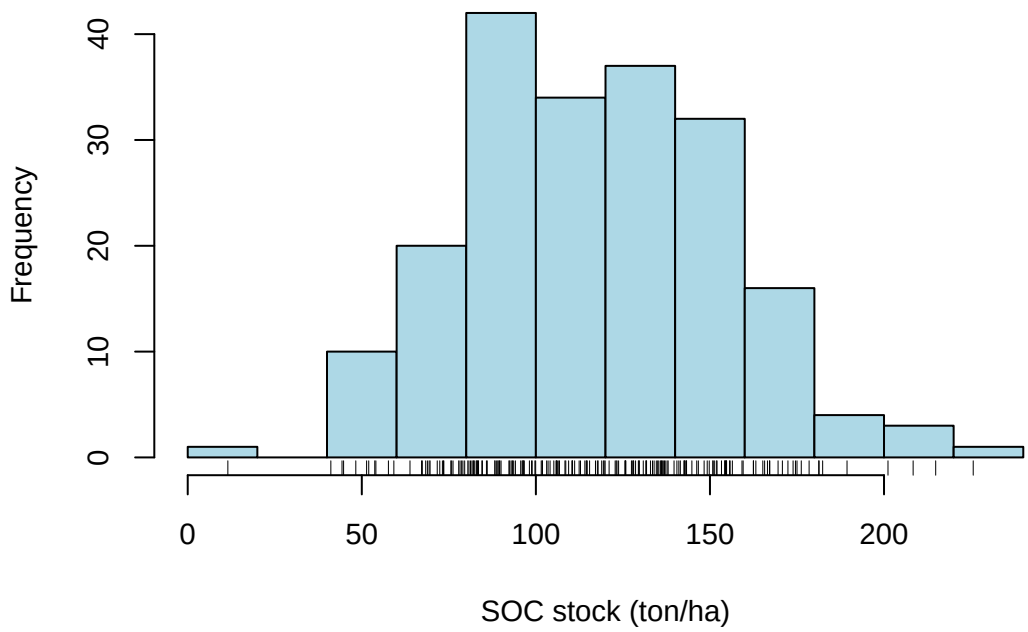


Figure 4.2: Histogram of a random sample of 200 SOC stock observations in the example area.

There are no suspicious values that require closer inspection. We therefore proceed with calculating the experimental variogram and fitting a variogram model. To calculate the variogram, we must first define a `gstat` object. We had sampled the 200 points with `terra::spatSample()`, which yields an object of class `SpatVector` (you may run `class(random_sample_sv)` to check). However, `gstat()` requires an `sf` object as input. Therefore, we first need to convert the `SpatVector` object to an `sf` object. Note that we could have done the spatial sampling directly with `sf` (we encourage the reader to give it a try), but in this tutorial we prefer to use the same sampling algorithm as in Sampling theory for estimating soil organic carbon change - a tutorial.

Create `gstat` object

```
# convert spatVector to sf object
random_sample_sf <- st_as_sf(random_sample_sv)
# define `gstat` object
# get() pulls the content of soc_values
g <- gstat(formula = get(soc_values) ~ 1, data = random_sample_sf)
```

The `formula` argument specifies the model that relates the dependent variable to explanatory variables. Those familiar with linear regression in R (i.e., function `lm`) will recognize the notation. The dependent variable is listed first, followed by a tilde symbol (`~`) and next all explanatory variables, with a `+` or `*` symbol in between (try `?formula` to learn more). In this case we are using ordinary kriging, which assumes a constant trend, as explained in Chapter 3. In other words, there are no explanatory variables. Therefore, the formula only includes an intercept (a constant, representing the mean μ defined in Chapter 3), indicated by the number 1, i.e. `formula = get(soc_values) ~ 1`. The data argument specifies the name of the dataset, which must be an `sf` object as explained above.

Next, we compute and plot the experimental variogram.

Experimental variogram

```
# experimental variogram
esvg <- variogram(g, boundaries = c(50,100,200,350,500,700,1000,1500,2000))
# plot is dispatched from gstat::plot.gstatVariogram
# because class(esvg)[1] == "gstatVariogram"
plot(esvg, xlim = c(0, 2100), ylim = c(0, 1650), plot.numbers = TRUE)
```

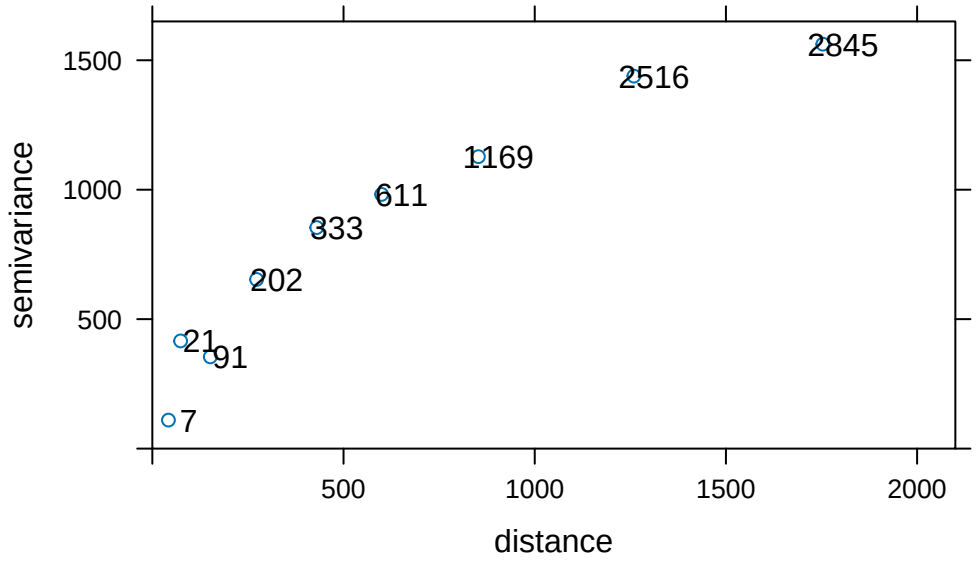


Figure 4.3: Plot of the experimental variogram.

The numbers next to the blue circles refer to the number of observation pairs used to calculate the experimental variogram value for each distance bin. The width of the bins is defined with the `boundaries` parameter. Note that we used smaller bin widths at shorter distances. This helps to reveal the short-distance spatial variation. The experimental variogram shows that SOC is spatially correlated, because the semivariance increases with distance.

Next we define a variogram model and fit it to the experimental variogram.

Fitting a variogram model

```
# define initial variogram model
svg_ini <- vgm(nugget = 200, psill = 1200, range = 600, model = "Exp")
# fit variogram model
svg <- fit.variogram(esvg, svg_ini, fit.method = 1)
# check
svg
```

```
  model     psill    range
1   Nug  182.3613   0.0000
2   Exp 1604.9949 872.1758
```

```
plot(esvg, svg, xlim = c(0, 2100), ylim = c(0, 1650), plot.numbers = TRUE)
```

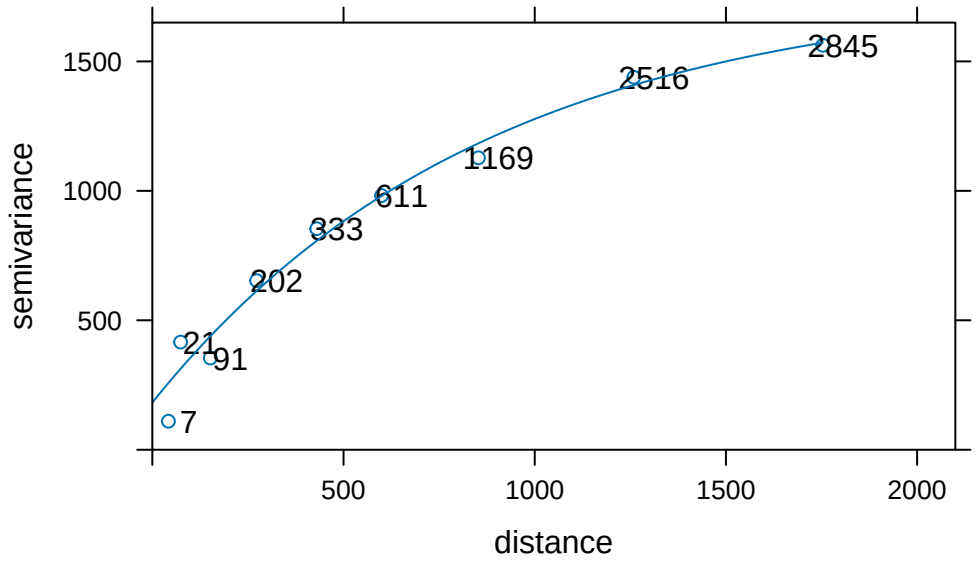


Figure 4.4: Experimental and fitted SOC stock variogram.

```r
# save nugget, sill and range
nugget <- svg[1,2]
sill <- svg[1,2] + svg[2,2]
range_parameter <- svg[2,3]

OK_params <- c(nugget, sill, range_parameter)
names(OK_params) <- c("nugget", "sill", "range")

print(OK_params)
```

```
   nugget      sill     range
 182.3613 1787.3562  872.1758
```

In Chapter 5 and Chapter 7 we will use this variogram again, therefore we print its parameters so that we can look them up later.

Note that variogram fitting starts by defining an initial variogram model. Here we used an exponential model that has three parameters: nugget, psill and range. We had introduced these variogram parameters in the previous chapter, but please note that psill is not equal to the sill but refers to the 'partial sill', that is the sill minus the nugget. Feel free to check the function documentation with `?gstat::vgm`. Initial values of the three parameters were chosen by eye. The call to `fit.variogram` starts an iterative, numerical optimization procedure to improve the parameter values. After convergence we obtained a decent fit to the experimental variogram. The semivariance increases up to a distance of about 1.7 km (recall that the example area measures 4.8 km by 4.3 km). The short-distance variability is quite small, because the nugget (i.e., the variogram value at distances close to zero, here it is 182.4) is small compared to the sill (here it is 1787.4). For more details about variogram fitting and interpretation of variogram parameters, we refer to the Geostatistics for soil mapping tutorial.

4.2 Ordinary kriging

In this section we create a map of the SOC stock with ordinary kriging, using the variogram model fitted in the previous section. We will also quantify the prediction uncertainty by computing the ordinary kriging standard deviation at each location in the study area.

Ordinary kriging is applied using the function `krige` of the `gstat` package. You can check with `?gstat::krige` which arguments are needed. One of these is `newdata`, in which we define the prediction locations (i.e. where we want to predict). For this we can use the `soc` raster we loaded at the beginning of this chapter as a mask. However, terra's `SpatRaster` class is not accepted by the `newdata` argument of `krige`. After consulting the documentation, we learned that one of the accepted formats is `stars`, and so we first convert to this format using the `st_as_stars()` function from the `stars` package.

Prepare mask

```
# the soc raster has many different values,
# but a mask only needs a constant value representing the area to predict
mask_terra <- soc*0 + 1
names(mask_terra) <- "mask"

# convert spatRaster to stars object
mask_stars <- st_as_stars(mask_terra)
```

We are now ready to run ordinary kriging. We store the output of `krige()` in the `ord_krig` object.

Ordinary kriging

```
# ordinary point kriging
ord_krig <- krige(formula = get(soc_values) ~ 1, locations = random_sample_sf,
                  newdata = mask_stars, model = svg)
```

The output object has two attributes, named `var1.pred` and `var1.var`, where `var1` refers to the variable of interest. We rename this to `soc` to make explicit which variable we are mapping.

```
# replace "var1" by "soc"
names(ord_krig) <- gsub("var1", "soc", names(ord_krig))
```

```
# inspect object
ord_krig
```

```
stars object with 2 dimensions and 2 attributes
attribute(s):
               Min.  1st Qu.  Median     Mean  3rd Qu.      Max. NAs
soc.pred   27.03372 100.3628 118.041 118.3242 136.9523  209.8643  92
soc.var   270.61217 489.3372 570.024 585.8398 661.6626 1302.9818  92
dimension(s):
  from  to offset delta              refsys x/y
x    1 192 261500    25 Amersfoort / RD New [x]
y    1 172 587800   -25 Amersfoort / RD New [y]
```

```
# output class is the same as that of the newdata argument
class(ord_krig)
```

```
[1] "stars"
```

The first element in `ord_krig`, `soc.pred`, is the map of ordinary kriging predictions. The second, `soc.var`, is the map of ordinary kriging variances.

At the beginning of this chapter we calculated the number of `NA` pixels in the SOC raster, which is 92. The same number is shown under the `NA` column in the `ord_krig` attributes. This means that, as expected, `krige` has not interpolated to the `NA` pixels in the mask. This is confirmed by the white cells in the ordinary kriging prediction map:

Kriging prediction map

```
# plot ordinary kriging predictions

# prepare colour breaks
n_colors <- 20 # the higher this number, the smoother the visualization
n_breaks = seq(range_plot[1], range_plot[2], length.out = n_colors + 1)

# stars::plot()
plot(ord_krig["soc.pred"], col = hcl.colors(n_colors, palette = "viridis"),
     main = "SOC stock predictions [ton/ha]", breaks = n_breaks, axes = FALSE)
```

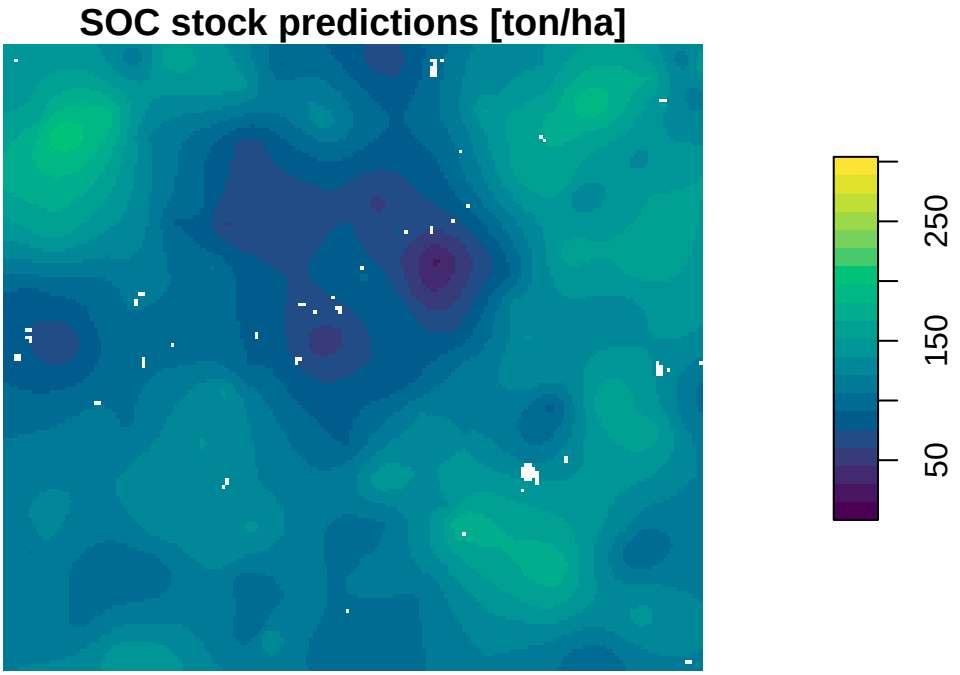


Figure 4.5: Ordinary kriging prediction of SOC stock in the example area.

We derive the kriging standard deviation from the kriging variance by taking the square root:

Kriging standard deviation map

```
# calculate the kriging standard deviation
ord_krig["soc.sd"] <- sqrt(ord_krig["soc.var"])
```

```
plot(ord_krig["soc.sd"], axes = FALSE,
     col = hcl.colors(9, palette = "YlOrBr"),
     main = "SOC stock standard deviation [ton/ha]", breaks = "equal",
     reset = FALSE)
plot(st_geometry(random_sample_sf),
     pch = 1, cex = 2, col = "black", add = TRUE)
```

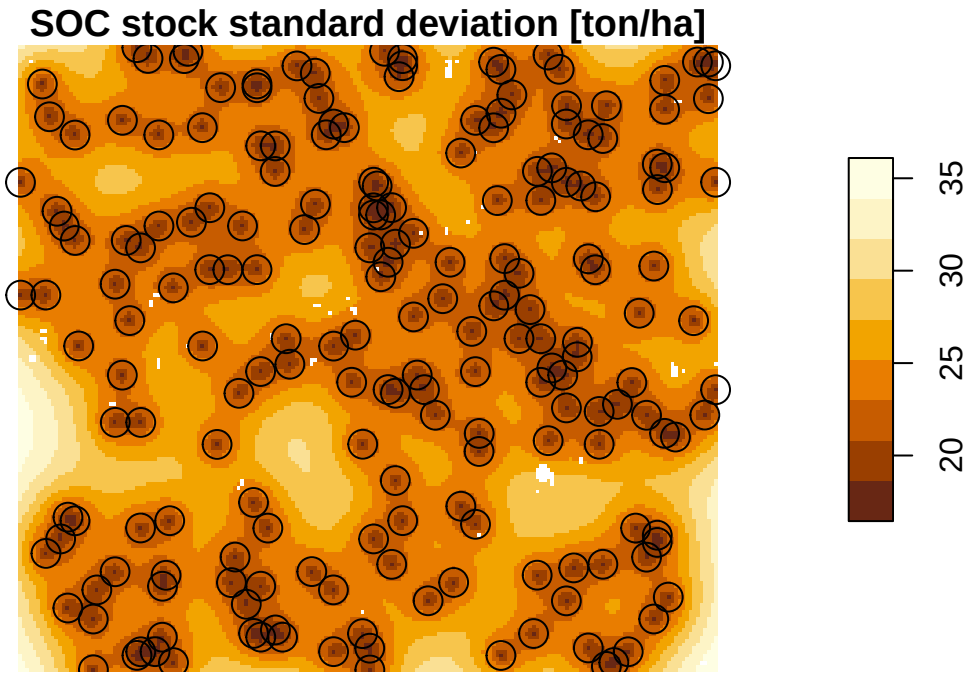


Figure 4.6: Ordinary kriging standard deviation of SOC stock in the example area.

The spatial pattern of the ordinary kriging standard deviation map is dominated by the spatial configuration of the sampling locations: it is small close to sampling locations and larger further away from them, particularly at the border of the example area. The spatial pattern of the ordinary kriging prediction map is very different: it is in essence a smoothed version of the `true` (but unknown) SOC stock map, shown in Figure 2.2.

Assuming that the kriging prediction error is normally distributed, we can derive the lower and upper bounds of a 95% prediction interval. Maps of these are plotted below, with the prediction map in the centre. Note that there were a few prediction locations where the lower bound was negative. This is a consequence of assuming a normal distribution. Since SOC stock can never be negative, we replace these negative values by zero.

Lower and upper bound of a 95% prediction interval

```
# 2 tails: 100% - 95% = 5%, 1 tail: 5% / 2 = 2.5%, therefore
# lower: 0.025
ord_krig["soc.025"] <- ord_krig["soc.pred"] + qnorm(0.025)*ord_krig["soc.sd"]
# assign zero to values lower than zero
ord_krig["soc.025"][ord_krig["soc.025"] < 0] <- 0
# upper: 1 - 0.025 = 0.975
ord_krig["soc.975"] <- ord_krig["soc.pred"] + qnorm(0.975)*ord_krig["soc.sd"]

# check
ord_krig[c("soc.025", "soc.pred", "soc.975")]
```

```
stars object with 2 dimensions and 3 attributes
attribute(s):
              Min.   1st Qu.    Median      Mean   3rd Qu.     Max. NAs
soc.025    0.00000  53.01459  69.17449  71.22816  90.52432 174.8324  92
soc.pred  27.03372 100.36278 118.04101 118.32422 136.95225 209.8643  92
soc.975   61.98736 146.14747 166.77551 165.42956 184.80673 247.7960  92
dimension(s):
  from  to offset delta                refsys x/y
x    1 192 261500    25 Amersfoort / RD New [x]
y    1 172 587800   -25 Amersfoort / RD New [y]
```

Prediction interval maps

```
# check min and max values for plotting
# we had already set min to a non-negative value
PI_min <- sapply(ord_krig["soc.025"], min, na.rm = TRUE)
PI_max <- sapply(ord_krig["soc.975"], max, na.rm = TRUE)

# since the ceiling of zero is zero, we can apply this rounding to both values
range_plot <- as.vector(ceiling(c(PI_min, PI_max)))
```

```r
# to plot side-by-side assign p1 <- ggplot() ... and then patchwork p1 + p2 + p3

# these arguments are used by the three plots
n_colors <- 10
n_breaks <- seq(range_plot[1], range_plot[2], length.out = n_colors + 1)
# round breaks to the nearest multiple of 5, but keep max to PI_max
n_breaks <- sapply(n_breaks, round2, d = 0, p = 5, pad_number = FALSE)
n_breaks[length(n_breaks)] <- ceiling(PI_max)

hcl_colors <- hcl.colors(n_colors, palette = "viridis")
scale_fill_args <- scale_fill_stepsn(colors = hcl_colors,
                                     limits = range_plot,
                                     breaks = n_breaks,
                                     name = "soc [ton/ha]",
                                     na.value = "transparent")

p1 <- ggplot() + geom_stars(data = ord_krig["soc.025"]) +
  scale_fill_args +
  coord_sf() + # to lock aspect ratio
  ggtitle("soc.025") + theme_void()

p2 <- ggplot() + geom_stars(data = ord_krig["soc.pred"]) +
  scale_fill_args +
  coord_sf() +
  ggtitle("soc.pred") + theme_void()

p3 <- ggplot() + geom_stars(data = ord_krig["soc.975"]) +
  scale_fill_args +
  coord_sf() +
  ggtitle("soc.975") + theme_void()

# since the three legends are the same, patchwork can "collect" them in one
p1 + p2 + p3 +
  plot_layout(guides = "collect") &
  theme(legend.position = "bottom",
        legend.key.width = unit(1.5, "cm"),
        legend.key.height = unit(0.3, "cm"))
```

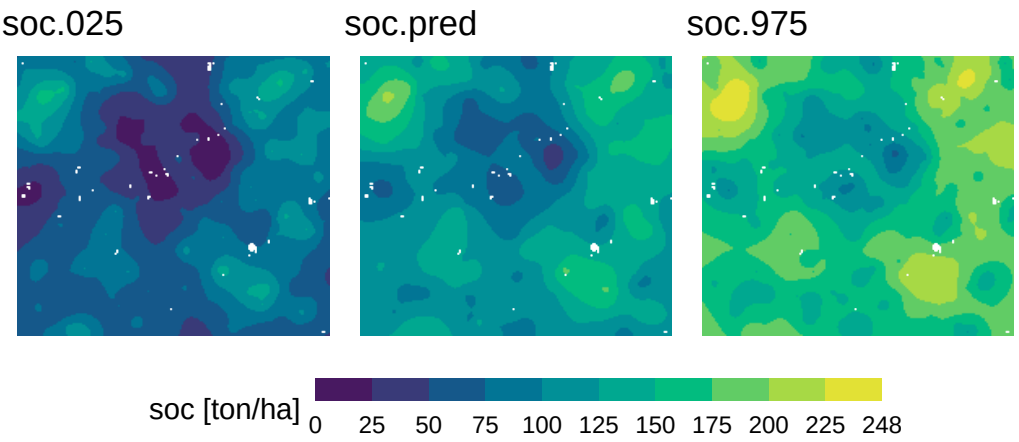


Figure 4.7: Maps of the lower (left) and upper (right) bounds of a 95% prediction interval of the SOC stock, together with the ordinary kriging prediction map (centre).

4.3 Ordinary block kriging

The previous section used ordinary kriging to predict the SOC stock at every location in the example area, with quantification of the prediction uncertainty. But the main interest of this tutorial is in predicting the total SOC stock in the area, and for this we use ordinary block kriging, with the `block` being the entire example area. We first derive the `block` from the mask map:

Create block

```
# convert mask to polygon
# mask_terra has a constant value of 1
# so all non-NA pixels are aggregated into a single polygon
block_sv <- as.polygons(mask_terra)
# spatVector to sf object
block_sf <- st_as_sf(block_sv)
```

The code chunk above defines the block by a polygon of the example area. But a polygon has an infinite number of locations, so that it would take forever to krige to all of them. We must therefore discretise the block to reduce the number of locations to a finite number. The `krige` function does this by defining a regular grid by default (see `?gstat::predict()`), but we can also specify customized prediction locations in the `block` argument of the `krige` function. In this way we have full control over the number of discretisation points and their locations. Following the logic of the default options, we opt for a regular grid composed of 30 rows and 30 columns.

Ordinary block kriging

```
# define a grid relative to (0,0); the centre of the block

# block has the size of the study area: 4800 x, 4300 y
# n_xy: number of grid points in x and y direction (columns, rows)
n_xy <- c(30, 30) # c(nx, ny)

# create a grid inside the block
grid <- st_make_grid(block_sf, n = n_xy, what = "centers")

# we need to make sure that none of these points fall in an NA pixel,
# so let's intersect the grid with the block
grid_sf <- st_as_sf(grid) # 30*30=900 points
grid_intersected <- st_filter(grid_sf, block_sf)
np_grid <- nrow(grid_intersected) # 897; 3 points in NA pixels and removed

# extract grid coordinates
grid_coords <- st_coordinates(grid_intersected)
# extract block centroid
block_centroid <- st_centroid(block_sf)

# the argument block must be relative to the centroid
# so the centroid represents (0,0)
# and the grid points are the distances between (0,0) and each point
# + or - depending on the direction
grid_relative <- sweep(grid_coords, 2, st_coordinates(block_centroid))
```

```
# ordinary block kriging
block_krig <- krige(formula = get(soc_values) ~ 1,
                    locations = random_sample_sf,
                    newdata = block_centroid,
                    model = svg,
                    block = grid_relative)
```

```r
# the object has two attributes, named `var1.pred` and `var1.var`
# we change the prefix `var` to `soc`
names(block_krig) <- gsub("var1", "soc", names(block_krig))

# inspect object
block_krig
```

```
Simple feature collection with 1 feature and 2 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 263899.2 ymin: 585649.7 xmax: 263899.2 ymax: 585649.7
Projected CRS: Amersfoort / RD New
  soc.pred  soc.var                   geometry
1 118.2949 3.215813 POINT (263899.2 585649.7)
```

```r
# output class is the same as that of the newdata argument
class(block_krig)
```

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

```r
# save values in new variables
block_pred <- block_krig$soc.pred
block_var <- block_krig$soc.var
```

```r
# plotting options
block_col <- "lightblue"
grid_col <- "red"
samples_col <- "black"
centroid_col <- "blue"
pch_grid <- 3
pch_samples <- 1
bbox <- st_bbox(block_sf)
x_expand <- (bbox["xmax"] - bbox["xmin"]) * 0.2  # expand by 20% to the right

# plot margins
par(mar = c(0, 0, 3, 0),  # c(bottom, left, top, right)
    mfrow = c(1, 1)) # reset plotting matrix

# sf::plot()
plot(st_geometry(block_sf), col = block_col,
     border = NA, main = "Block discretisation using a regular grid",
     xlim = c(bbox["xmin"], bbox["xmax"] + x_expand))
plot(st_geometry(grid_intersected), add = TRUE,
     col = grid_col, pch = pch_grid, cex = 0.5)
plot(st_geometry(random_sample_sf), add = TRUE,
     col = samples_col, pch = pch_samples, cex = 0.8)
plot(st_geometry(block_krig), add = TRUE,
     col = centroid_col, pch = pch_grid, cex = 0.5)

# add a manual legend
legend(x = bbox["xmax"],  # right edge of the map
       y = bbox["ymax"],  # top edge of the map
       legend = c(paste(
         "Pred:", round(block_pred, 1), "ton/ha"),
         "Regular grid",
         "Sampling locations",
         "Block centroid"),
       fill = c(block_col, NA, NA, NA),
       border = c(NA, NA, NA, NA),
       pch = c(NA, pch_grid, pch_samples, pch_grid),
       col = c(NA, grid_col, samples_col, centroid_col),
       cex = 0.7, bty = "n") # removes the box around the legend
```

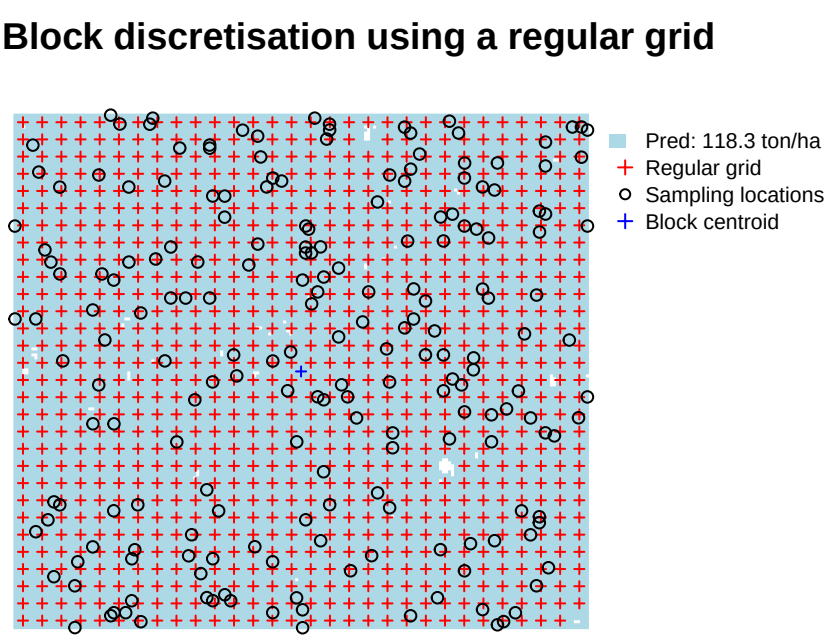


Figure 4.8: Ordinary block kriging prediction and discretisation grid.

We can compute the block kriging standard deviation from the block kriging variance in the usual way:

Block kriging standard deviation

```
# calculate the block kriging standard deviation
block_sd <- sqrt(block_var)

cat("With", np_grid, "grid points, soc.pred ~", round2(block_pred,2),
    "and soc.sd ~", round2(block_sd,2), "ton/ha")
```

```
With 897 grid points, soc.pred ~ 118.29 and soc.sd ~ 1.79 ton/ha
```

Feel free to calculate the block kriging prediction and standard deviation for different discretizations of the example area. For instance, you might try n_xy=50 or n_xy=80. The larger this number, the smaller the discretisation error, but of course it comes at the expense of longer computing times.

By multiplying with the area we obtain the prediction and prediction error standard deviation of the total SOC stock in the area. Assuming the normal distribution (which is now also supported by the Central Limit Theorem, see Sampling theory for estimating soil organic carbon change - a tutorial, we can also derive a 95% prediction interval.

Block kriging predicted total SOC stock

```
# calculate total SOC stock predicted with ordinary block kriging (ton)
block_krig_total_soc <- block_pred * area_ha

# print
cat("Predicted total SOC stock with ordinary block kriging:",
    round2(block_krig_total_soc, 0, 10), "ton")
```

```
Predicted total SOC stock with ordinary block kriging: 243440 ton
```

```
cat("Difference with the true SOC stock:",
    round2(total_stock - block_krig_total_soc, 0, 10), "ton")
```

```
Difference with the true SOC stock: 4950 ton
```

Prediction interval

```
# compute the lower and upper limits of the 95% prediction interval
# 2.5 % lower
lower_limit <- (block_pred + qnorm(0.025)*block_sd) * area_ha
# 100 % - 2.5 % = 97.5 % upper
upper_limit <- (block_pred + qnorm(0.975)*block_sd) * area_ha

# calculate prediction interval width
intv_width <- upper_limit - lower_limit

# print prediction interval
paste0("The 95% prediction interval is [", round2(lower_limit, 0, 10), ", ",
       round2(upper_limit, 0, 10), "] ton")
```

```
[1] "The 95% prediction interval is [236210, 250680] ton"
```

```
paste0("Prediction interval width: ", round2(intv_width, 0, 10), " ton")
```

```
[1] "Prediction interval width: 14470 ton"
```

The true total SOC stock is 248390 ton, and so we find that it is contained within the 95% prediction interval, given by [236210, 250680] ton. This is no surprise, since assuming that our geostatistical model is correct, there is a 95% probability that the true value is inside the prediction interval.

Finally, it is interesting to check whether the 95% prediction interval width obtained with ordinary block kriging is narrower, wider or similar to the width of the 95% confidence interval obtained with sampling theory, as derived in Section 4.3 of the Sampling theory for estimating soil organic carbon change - a tutorial, both with a sample size of 200. In that tutorial we obtained a confidence interval width of 20550 ton, which is 42.1% larger than the prediction interval width of 14470 that we obtained with ordinary block kriging. This shows that ordinary block kriging prediction from 200 observations is more accurate than using the mean of a random sample of $n = 200$. This also should not come as a surprise, because kriging benefits from spatial correlation, while sampling theory does not. However, we must also not forget that kriging makes assumptions (e.g., second-order stationarity, see Section 3.1), so that the kriging result is only valid if these assumptions are satisfied. Sampling theory has the important advantage that it makes no assumptions and is entirely model-free.

Before closing this chapter, we save the sample of 200 observations and its geographic locations in a GeoPackage that can be loaded into R with functions like `terra::vect()` or `sf::read_sf`. We do this because we will use that same sample of observations in the next chapters.

Save the sample in a GeoPackage (optional)

```
# this random sample already has an offset
st_write(
  random_sample_sf,
  file.path(outputDir, "random_sample.gpkg"),
  driver = "GPKG",
  delete_dsn = TRUE
)
```

We also save the variogram parameters we obtained for ordinary kriging.

Save OK variogram parameters

```
# save to re-use in Chapter 5 and Chapter 7
saveRDS(OK_params, file.path(outputDir, "OK_params.rds"))
```

5 Incorporating covariates in machine learning regression kriging

In this chapter, we will extend **ordinary block kriging** from the previous chapter to **regression block kriging**. The key difference is that unlike in ordinary kriging, the trend μ, as defined in Chapter 3, is no longer assumed to be constant but instead a function of explanatory variables, also known as **covariates**. These covariates must be known at all prediction locations, in other words we must have them in the form of maps.

We still work with the basic geostatistical model, which says:

$$Z(s) = \mu(s) + \varepsilon(s), \quad \text{for all } s \in A$$

In this chapter we define μ by a random forest (RF) machine learning algorithm, that is a statistical regression model that predicts the target variable from the covariates, but we could also have used a different algorithm. For instance, we could also have defined the trend with a multiple linear regression model, another machine learning regression model, or even a process-based mechanistic model.

After fitting the trend in Section 5.1, we will turn our attention to the residual ε, defined as the difference between the target variable and the trend. In this chapter, ε is the difference between the 'true' SOC stock and the SOC stock prediction from the RF model. Note that we only know the residual at the n observation locations, where we can subtract the RF prediction from the observed SOC stock. We compute a variogram from the n observed residuals, and use it to predict the residual at unobserved locations with ordinary kriging. This is done in Section 5.2. The procedure is the same as that of the previous chapter, the only difference being that we work with a RF residual instead of the SOC stock. You will remember from that chapter that one of the strengths of kriging is that it also quantifies the prediction uncertainty.

By summing the maps of the RF predictions and the kriged residuals, we obtain a regression kriging prediction map. The uncertainty of that map is characterized by the residual kriging standard deviation map. These steps are done in Section 5.3. We hypothesize that the prediction uncertainty of regression kriging is smaller than that of ordinary kriging, because regression kriging can benefit from covariates.

Finally, since our ultimate goal is to predict the total SOC stock in the example area, we aggregate the regression kriging predictions and derive a regression block kriging prediction of the total SOC stock in the example area, and we quantify the uncertainty of that prediction

with a regression block kriging prediction interval. This is done in the final section of this chapter, that is Section 5.4.

The above describes in a nutshell what we will do in this chapter. As you can see there are quite a few steps to be done. We begin with some preliminaries and by loading the data.

Setup

```
# load libraries
library(terra)
library(cvTools) # for creating data folds
library(ranger) # for fitting a random forest model
library(sf)
library(gstat)
library(stars)
```

```
# set a seed
set.seed(987)
```

```
# uncomment the two code lines below after replacing the file path
# with the location where you saved the tutorial files
# (i.e. that contains the 'soc_stock_kriging' folder)
#setwd("C:/tutorials/soc_stock_kriging")
#workingDir <- getwd()
```

```
# build input and output paths
inputDir <- file.path(workingDir, "input")
outputDir <- file.path(workingDir, "output")
```

```
# rounding function
round2 <- function(x, d, p = 1){
  p <- p*(10^-d)
  n <- round(x/p)*p
  format(n, nsmall = d)
  }
```

Load 'true' SOC stock map

```
# load soc raster
soc_fn <- "soc_stock_2020.tif"
soc <- rast(file.path(inputDir, soc_fn))
```

Prepare data

```
# modify soc raster metadata to make it easier to call
soc_values <- "soc_ton_ha"
names(soc) <- soc_values
```

```
# min and max values for plotting
# since the ceiling of zero is zero, we can apply this rounding to both values
range_plot <- as.vector(ceiling(minmax(soc)))
```

Calculate area

```
# compute the area (ha) (excluding NAs)
area_ha <- expanse(soc, unit = "ha")[1, "area"]

# calculate the mean and total SOC stock across the non-NA pixels (ton/ha)
mean_stock <- global(soc, fun = "mean", na.rm=TRUE)[1, "mean"]
total_stock <- mean_stock * area_ha
```

As explained in Section 4.2, to perform the operations in this chapter we require a mask that covers the example area. We create it from the `soc` raster by replacing all its non-NA values with 1.

Create mask

```
mask_terra <- soc*0 + 1
names(mask_terra) <- "mask"
```

SOC observations

To ease comparison of results, we use the same random sample as used in the previous chapter.

```
# load random sample
random_sample <- vect(file.path(outputDir, "random_sample.gpkg"))
sample_size <- dim(random_sample)[1]
```

Regression kriging also needs covariate maps. These were obtained from Helfenstein et al. (2024b). We selected a stack of 16 covariates from the full dataset (for details see Table 5.1), so that running the operations would not be that computationally expensive. We prepared these covariates in advance, ensuring that they have the same resolution, extent, and projection as the 'true' SOC stock raster and mask map. This is a condition that must be met to create a stacked or multi-band raster with `terra::rast()`.

Table 5.1: Summary of selected covariates in the raster stack.

name	values_type	description	postprocessing	zipped_folder
ahn_hillshade_25m	continuous	hillshade from AHN	-	relief
ahn3_25m	continuous	filled DTM	-	relief
grondwater_2018_25m	categorical	groundwater classes of NL	-	groundwater
nh3_2018_25m	continuous	ammonia (NH3) content	circular moving-window filter (radius 500)	management_other
ntot_2018_25m	continuous	total Nitrogen content	circular moving-window filter (radius 500)	management_other
paleogeografie1500nc_25m	categorical	paleogeographic maps of NL over 11000 years	-	paleogeography
precip_yearly_1981_2010_	continuous	average yearly precipitation 1981-2010	circular moving-window filter (radius 500)	climate
s2_NDVI_mean	continuous	NDVI monthly mosaic	s2 NDVI average 2016-2021	sentinel2_RGBNIR
s2_RGBNIR_1_mean	continuous	R monthly mosaic	Band 1 (R) average 2016-2021	sentinel2_RGBNIR
s2_RGBNIR_2_mean	continuous	G monthly mosaic	Band 2 (G) average 2016-2021	sentinel2_RGBNIR
s2_RGBNIR_3_mean	continuous	B monthly mosaic	Band 3 (B) average 2016-2021	sentinel2_RGBNIR
s2_RGBNIR_4_mean	continuous	NIR monthly mosaic	Band 4 (NIR) average 2016-2021	sentinel2_RGBNIR
temp_yearlymax_1981_2010_25m	continuous	average yearly maximum temperature 1981-2010	circular moving-window filter (radius 500)	climate
temp_yearlymean_1981_2010_25m	continuous	average yearly mean temperature 1981-2010	circular moving-window filter (radius 500)	climate
temp_yearlymin_1981_2010_25m	continuous	average yearly minimum temperature 1981-2010	circular moving-window filter (radius 500)	climate
water_wetness_probability_2015_25m	continuous	% wetness land surface (0% = dry, 100% = waterbodies)	-	landuse_COPERNICUS

Load covariates

```
# load covariates stack
covars <- rast(file.path(inputDir, "stack1.tif"))
```

We extend the covariate set with latitude and longitude, since this might improve the prediction accuracy of the RF model. Although not strictly needed, we normalize them before we add them to the covariate stack.

Add latitude and longitude to the covariate stack

```
# get x and y coordinate rasters
x_rast <- init(covars, "x")
y_rast <- init(covars, "y")

# normalize function
# set minimum to zero and maximum to 1
normalize_minmax <- function(r) {
  mm <- minmax(r)
  (r - mm["min", ]) / (mm["max", ] - mm["min", ])
}

# stack x and y rasters and apply normalization
x_y_covars <- normalize_minmax(c(x_rast, y_rast))

# plain "x" and "y" names can be confusing
# let's give them more meaningful names
names(x_y_covars) <-  c("x_coordinate", "y_coordinate")

# add to the covariate stack
covars <- c(covars, x_y_covars)

# we can remove x_y_covars from the environment since we
# no longer need them separately from the covars stack
rm(x_rast, y_rast, x_y_covars)
```

We mask the covariates to ensure that all have the same NA pixels as the 'true' SOC stock raster.

Mask covariates

```
# mask the covariates to omit the same NA cells as in the 'true' soc raster
covars <- mask(covars, mask_terra)
```

Explore covariates

Before continuing with training a RF model, let us first check the number of bands in the raster stack (i.e., the number of covariates) and compute summary statistics for each.

```
# number of bands
nlyr(covars) # 16 from Helfenstein (2024) + 2 xy spatial variation

# basic statistics
summary(covars)
```

Note that the number of NA cells is the same for all covariates and equal to 92. This agrees with the number of NA cells in the mask map, i.e. the white pixels in Figure 2.2. During modelling, if a pixel is NA in one or more of the covariates, no predictions will be generated for that pixel. Therefore, it is important to check that all pixels of the mask are covered by all covariates, otherwise we would be predicting the total SOC stock of a smaller area than the example area.
We must also make sure that all categorical covariates are treated as such during RF model training. From Table 5.1 we learn that 2 of the covariates are categorical.

```
# check if categorical rasters are factors
cat_covars <- c("grondwater_2018_25m",
                "paleogeografie1500nc_25m")
# check
is.factor(covars[[cat_covars]])
```

```
[1] FALSE FALSE
```

```
# declare as factors
covars[[cat_covars]] <- as.factor(covars[[cat_covars]])
is.factor(covars[[cat_covars]])
```

```
[1] TRUE TRUE
```

Below we plot all 18 covariates. Some show geomorphological features, other land use and vegetation, while climate covariates area also included.

```r
# select which bands to plot
plot_bands <- 1:nlyr(covars) #1:4 #c(3, 5, 7, 16)

# plotting options
n_cols <- 3
n_rows <- ceiling(length(plot_bands)/n_cols)

par(mfrow = c(n_rows, n_cols), # Create a n_row x n_col plotting matrix
    mar = c(0, 0, 0, 0), # margins c(bottom, left, top, right)
    oma = c(0, 0, 0, 0))

# terra::plot()
for (p in plot_bands){
  plot(covars[[p]], main = names(covars[[p]]), cex.main = 0.95,
       axes = FALSE, box = TRUE)
}
```

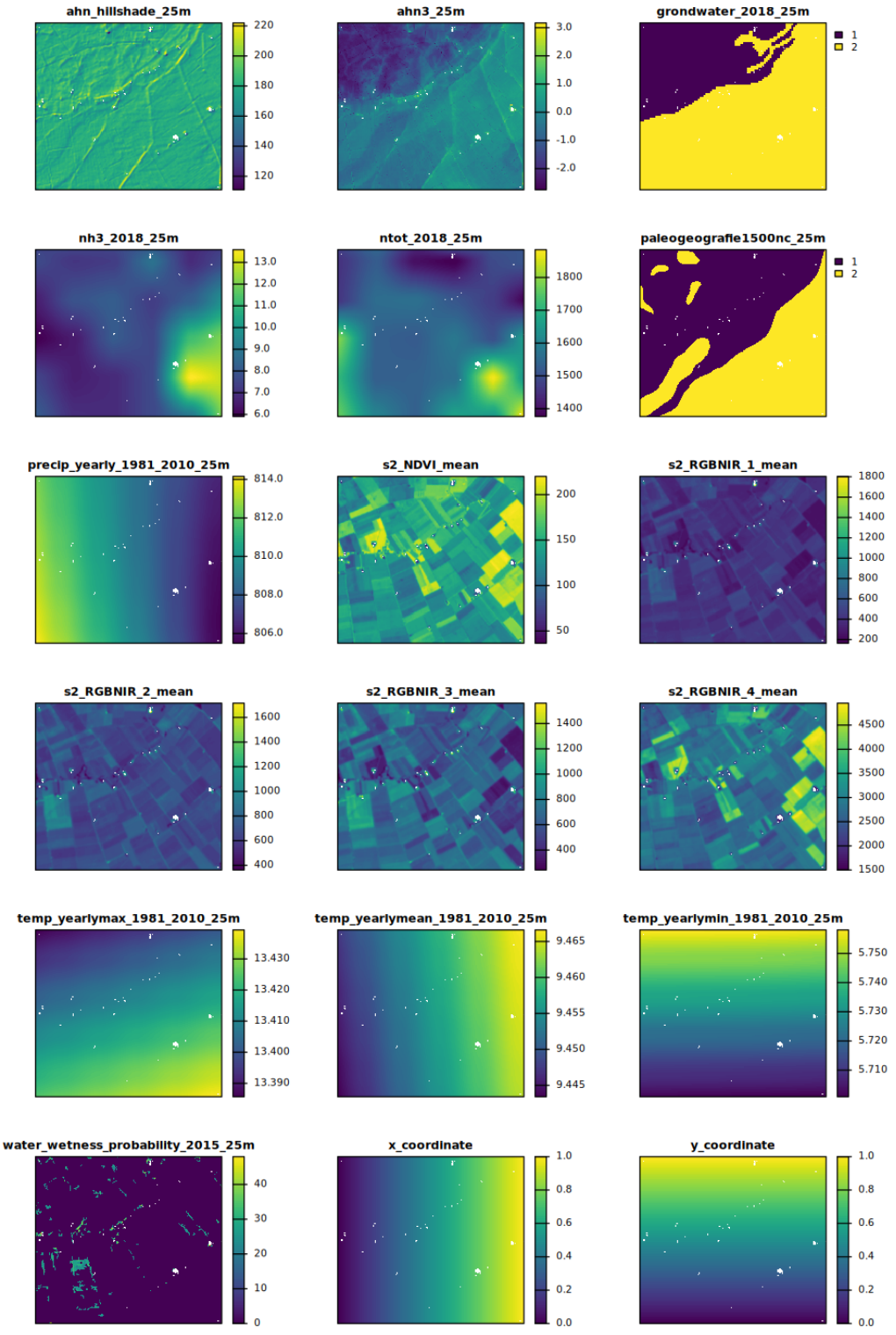


Figure 5.1: Covariates maps.

Save SOC stock change raster (optional)

```
# use writeRaster() to save a raster output
writeRaster(covars,
            filename = file.path(outputDir, "stack2.tif"),
            gdal=c("COMPRESS=DEFLATE"), # compression method
                    overwrite=TRUE)
```

5.1 Modelling the trend with Random Forest regression

To model the SOC stock trend using Random Forest, we follow the steps documented in two ISRIC tutorials:

- Data Preparation for Digital Soil Mapping
- Machine Learning for Digital Soil Mapping

Much of the preparation has already been done, but we still need to extract the covariate values at the observation locations to derive paired observations of the SOC stock and covariates. These paired observations are stored in a so-called **regression matrix**, which is used to train the RF model.

Generate regression matrix

```
# extract covariate values at sampling locations
rgmx <- as.data.frame(extract(covars, random_sample, bind = TRUE))

# check table contents
dim(rgmx)

# explore table by printing, for instance, row 1 to 4, from column 1 to 6
rgmx[1:4, 1:6]

# check completeness (the first covariate is stored in column 4)
summary(complete.cases(rgmx[ ,4:ncol(rgmx)]))
```

```
# check if the values extracted from categorical covariates
# inherited the factor datatype
if (all(cat_covars %in% names(rgmx))){
  sapply(rgmx[c(cat_covars)], is.factor)
}
```

```
grondwater_2018_25m paleogeografie1500nc_25m
               TRUE                       TRUE
```

We use the `ranger()` function from the `ranger` package (Wright & Ziegler, 2017) to fit the RF model.

The only two mandatory arguments are:

- `formula`: object of class `formula` that defines the model.
- `data`: regression matrix `data.frame` that contains the training data.

The formula has the form:

soil_variable ~ covariate_1 + covariate_2 + ... + covariate_m

where in our case $m = 18$.

Construct formula for `ranger()`

```
# extract column names from rgmx, where covariate values are stored
names_rgmx <- names(rgmx)
covars_in_rgmx <- names_rgmx[which(names_rgmx %in% names(covars))]

# construct covariate string
covar_string <- paste(covars_in_rgmx, collapse = " + ")

# this is the name inside the soc raster that we assigned in Prepare data
voi <- soc_values

# construct formula string
formula_string <- paste(voi, " ~ ", covar_string)
```

```r
# convert to formula object
mod_for <- as.formula(formula_string)

# show formula and object class
mod_for
```

```
soc_ton_ha ~ ahn_hillshade_25m + ahn3_25m + grondwater_2018_25m +
    nh3_2018_25m + ntot_2018_25m + paleogeografie1500nc_25m +
    precip_yearly_1981_2010_25m + s2_NDVI_mean + s2_RGBNIR_1_mean +
    s2_RGBNIR_2_mean + s2_RGBNIR_3_mean + s2_RGBNIR_4_mean +
    temp_yearlymax_1981_2010_25m + temp_yearlymean_1981_2010_25m +
    temp_yearlymin_1981_2010_25m + water_wetness_probability_2015_25m +
    x_coordinate + y_coordinate
```

```r
class(mod_for)
```

```
[1] "formula"
```

Fit Random Forest model using `ranger()`

```r
# fit Random Forest model
rf_model <- ranger(formula = mod_for, data = rgmx)
```

Now that we have a trained RF model we can predict the SOC stock for all grid cells in the example area. Note that the covariates must be provided as a data frame, so we convert them first.

Construct data for `predict()`

```r
# convert SpatRaster to data.frame
covars_df <- as.data.frame(covars, xy = TRUE, na.rm = TRUE)
```

Run Random Forest using `ranger`

```
# predict
prd <- predict(rf_model, data = covars_df, type = "response")
# inspect output
class(prd)
str(prd)
summary(prd$predictions)
```

Note that a call to `predict()` automatically dispatches to `predict.ranger()` because our model object is of class `ranger`. Therefore, to access the function documentation, you must type `?predict.ranger()` in the Console. The function has a large number of potential parameters, but only two are mandatory:

- `data`: covariate values at the prediction locations, i.e. the covariate stack converted to a `data.frame`.
- `type`: type of prediction required. The default is `response`, which returns predictions of the target variable (i.e., the SOC stock).

The prediction object is not yet a spatial object. To do this, we combine it with the coordinates of the covariate data frame.

Make predictions a spatial object

```
# compile predictions
p <- data.frame(x = covars_df$x, y = covars_df$y, pred = prd$predictions)

# make spatial
RF_pred <- rast(as.matrix(p), type = "xyz", crs = crs(covars))
names(RF_pred) <- paste0("RF_", soc_values)

# explore predicted SOC stock raster
RF_pred

# calculate the mean SOC stock across the non-NA pixels (ton/ha)
RF_mean_stock <- global(RF_pred, fun = "mean", na.rm=TRUE)[1, "mean"]

# number of NA pixels
n_na <- global(is.na(RF_pred), "sum", na.rm = TRUE)[[1]]
# check
n_na
```

Since we masked the covariates with a raster derived from the 'true' SOC stock, we obtained the same number of `NA`, that is 92, in the RF prediction map. Let us plot this map next to the 'true' SOC stock map. Please note that this is not yet the final regression kriging prediction map. It is 'only' the trend.

Predicted SOC stock map

```
# terra::plot()
cex_main <- 1
title_cex <- 1
cex_legend <- 1
plot_maps <- c(soc, RF_pred)
mains <- c("'True' SOC stock", "SOC stock predictions")

par(mfrow = c(1, 2), mar = c(0, 0, 0, 0))
for (i in 1:nlyr(plot_maps)){
  plot(plot_maps[[i]], main = mains[i],
     cex.main = cex_main, range = range_plot, axes = FALSE, box = TRUE,
     plg = list(title = "ton/ha", title.cex = title_cex, cex = cex_legend))
}
```

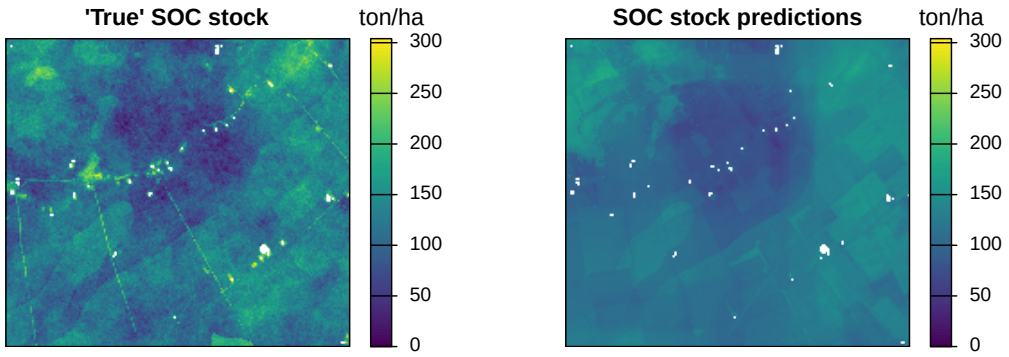


Figure 5.2: 'True' SOC stock and Random Forest SOC stock predictions [ton/ha].

We have now completed modelling the trend. In the next section we krige the residual, after which we sum the trend and the interpolated residual to get the regression kriging prediction map.

5.2 Modelling the residual variogram

Kriging the residual works in a similar way as kriging the SOC stock as done in Chapter 4. The only difference is that we need to work with the residuals, that is the difference between the SOC observations and RF predictions at all sampling locations. The residuals can easily be obtained by subtracting the RF predictions from the SOC stock observations, but there is one precaution that we must take. This is that we must avoid that we subtract a RF prediction from an observation if that same observation was also used to train the RF model from which the prediction was derived. If we would not take this precaution we would get residuals that are on average too close to zero, more so than residuals at locations where we have no observations. In other words, we must remove a SOC stock observation at a sampling location from the RF training set before we fit the RF model and make a prediction at that location. This will ensure that we get an "honest" residual at every sampling location — one that reflects how well the model would predict the SOC stock at a location if it had never "seen" the SOC stock at that location. This can be achieved by taking a **leave-one-out** model training approach. This works as follows.

We visit each of the n sampling locations one by one, each time removing the SOC stock observation at that location, and training the RF model on the remaining $n-1$ observations. Next we use the trained model to predict at that location, subtract the prediction from the put-aside SOC stock observation to obtain the residual at that location, and we move to the next location, where we repeat the procedure. Note that in the end we will have trained n RF models, and that this can be computationally demanding if n is large. In such case, one could opt for splitting the dataset in k approximately equally sized folds, with $k << n$, each time training the RF model on data from $k-1$ folds to predict the SOC stock at the locations of the put-aside fold. The code below takes the leave-one-out approach by setting the number of folds k equal to n.

Create folds

```
# number of folds for sub-setting the dataset
nrFolds <- sample_size
# cvFolds assigns each row to a fold
folds <- cvFolds(nrow(rgmx), K = nrFolds, type = "random")

# the row index is stored in "subsets" and the fold number in "which"
folds_df <- data.frame("which" = folds$which, "subsets" = folds$subsets)

# order by row index, select the folds column and
# add it to the regression matrix
rgmx["fold"] <- folds_df[order(folds_df$subsets), "which"]

# we can keep the index too
rgmx["id"] <- folds_df[order(folds_df$subsets), "subsets"]
```

Compute residuals

```
# create an empty column in the regression matrix to store the predictions
rgmx["pred"] <- NA

# fit the model for each hold-out fold
for (k in seq_len(nrFolds)) {

  # subset train and leave-out sets
  train <- subset(rgmx, fold != k)
  leave_out <- subset(rgmx, fold == k)

  # fit Random Forest model
  rf_model_k <- ranger(formula = mod_for, data = train)

  # predict the hold-out fold
  prd_k <- predict(rf_model_k, data = leave_out)$predictions

  # assign predictions back using id match
  rgmx$pred[match(leave_out$id, rgmx$id)] <- prd_k
}
```

```
# compute the residuals
rgmx["residuals"] <- rgmx[soc_values] - rgmx["pred"]

# explore residuals
summary(rgmx["residuals"])
```

```
   residuals
 Min.   :-69.77319
 1st Qu.:-16.37091
 Median : -0.53026
 Mean   : -0.02107
 3rd Qu.: 14.93039
 Max.   : 75.84057
```

Next we calculate the experimental variogram of the residuals and fit a variogram model, just as we did in the previous chapter.

Create `gstat` object

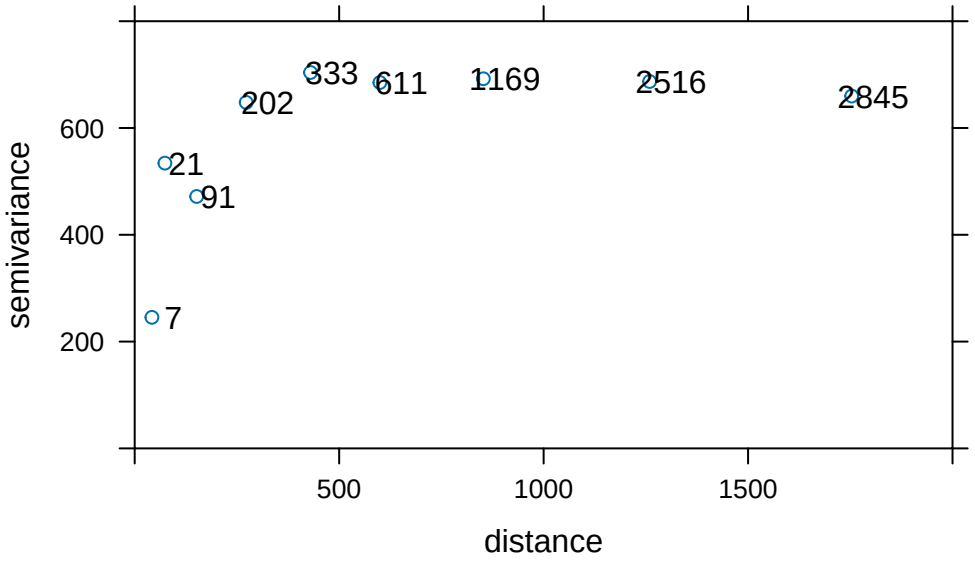
```
# convert dataframe to sf object
residuals_sf <- st_as_sf(rgmx[c("x", "y", "residuals")],
  coords = c("x", "y"), crs = crs(soc))

# define `gstat` object
g <- gstat(formula = residuals ~ 1, data = residuals_sf)
```

Experimental variogram

```
esvg <- variogram(g, boundaries = c(50,100,200,350,500,700,1000,1500,2000))
esvg
```

```
    np       dist    gamma dir.hor dir.ver   id
1    7   42.10002 245.4234       0       0 var1
2   21   73.86366 534.1813       0       0 var1
3   91  151.59092 471.7828       0       0 var1
4  202  272.61899 647.9413       0       0 var1
5  333  429.96384 703.9295       0       0 var1
6  611  599.58879 685.1730       0       0 var1
7 1169  852.70639 692.2246       0       0 var1
8 2516 1259.31037 687.1056       0       0 var1
9 2845 1753.43792 659.7840       0       0 var1
```

```
plot(esvg, xlim = c(0, 2000), ylim = c(0, 800), plot.nu = TRUE)
```

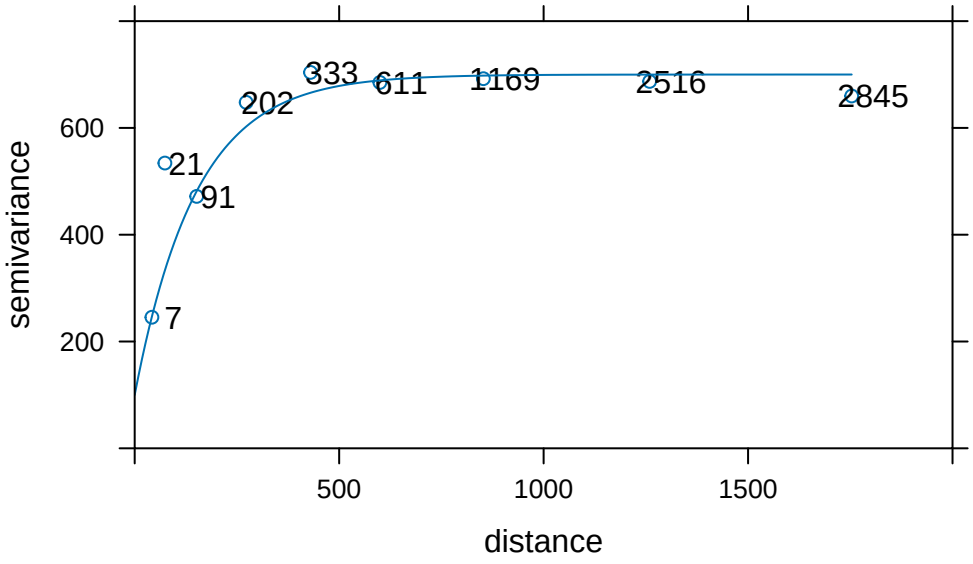


Initial variogram model

```
# define initial variogram model
svg_ini <- vgm(nugget = 100, psill = 600, range = 150, model = "Exp")
```

Fit variogram model

```
# fit variogram model
svg <- fit.variogram(esvg, svg_ini, fit.method = 1)
plot(esvg, svg_ini, xlim = c(0, 2000), ylim = c(0, 800), plot.nu = TRUE)
```




```
svg
```

```
  model    psill    range
1   Nug 106.1316   0.0000
2   Exp 572.3013 104.1252
```

Load Ordinary Kriging (OK) variogram parameters from Section 4.1

```
OK <- readRDS(file.path(outputDir, "OK_params.rds"))
```

Compare residual variogram with that of Section 4.1

```
# joint plot of original and residual variogram
svg_ok <- vgm(nugget = OK["nugget"], psill = OK["sill"], range = OK["range"],
              model = "Exp")
```

```
par(mfrow = c(1, 1), mar = c(4, 4, 1, 1))
plot(variogramLine(svg_ok, maxdist = 2000), type = "l", lwd = 2,
     col = "blue", ylim = c(0,2000))
lines(variogramLine(svg, maxdist=2000), lwd = 2, col = "red")
legend("topleft", legend = c("SOC stock (OK)", "Regression residual (RK)"),
       col = c("blue", "red"), lwd = 2, cex = 0.8)
```

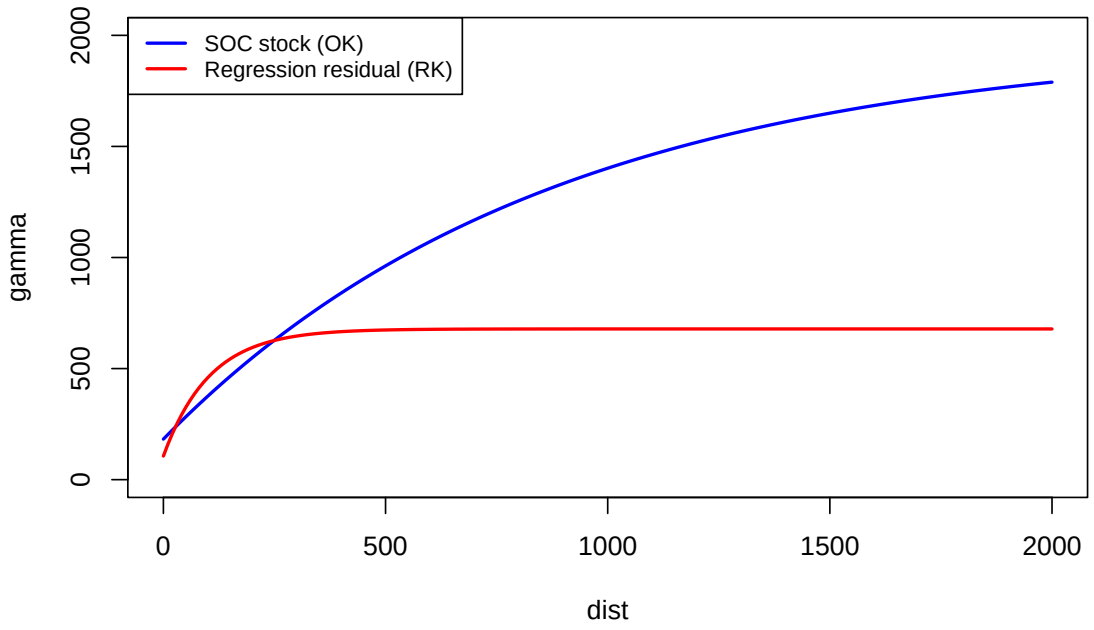


Figure 5.3: Variogram comparison. OK: Ordinary Kriging, RK: Regression Kriging.

The parameters in `svg_ok` come from Figure 4.4. Similarly, we will save the parameters of the fitted residual variogram to use in Chapter 7.

Save residual variogram parameters

```
# save nugget, sill and range
nugget <- svg[1,2]
sill <- svg[1,2] + svg[2,2]
range_parameter <- svg[2,3]

RK_params <- c(nugget, sill, range_parameter)
names(RK_params) <- c("nugget", "sill", "range")

print(RK_params)
```

```
  nugget     sill    range
106.1316 678.4329 104.1252
```

```
# save to re-use in Chapter 7
saveRDS(RK_params, file.path(outputDir, "RK_params.rds"))
```

Figure 5.3 above shows both the SOC stock variogram (blue line, from Figure 4.4) and the residual variogram (red line, this chapter) in one plot. This tells us a lot. To begin with, the sill has much decreased, which indicates that the RF model was quite able to explain much of the spatial variation of the SOC stock. But we also see that the covariates could not explain the spatial variation at short distances, say up to about 300~m, because the blue and red lines are quite close at distances smaller than 300~m. In fact, the red line is mostly above the blue line at short distances, which would suggest that at these separation distances, residual spatial variation is larger than SOC stock spatial variation. This is remarkable, and has implications for kriging, as we shall see later in this chapter.

5.3 Regression kriging

In this section we first interpolate the residual using ordinary kriging, and next add the interpolated residual to the RF prediction map. We quantify prediction uncertainty with the residual kriging standard deviation map. The kriging steps below are very similar to what we did in Section 4.2.

Prepare mask

```
# convert spatRaster mask to stars
mask_stars <- st_as_stars(mask_terra)
```

Run ordinary kriging

```
# ordinary point kriging
ord_krig <- krige(formula = residuals ~ 1, locations = residuals_sf,
                  newdata = mask_stars, model = svg)
```

The output object has two attributes, named `var1.pred` and `var1.var`. We change the prefix `var1` to `resid` to make clear which variable we kriged.

```
names(ord_krig) <- gsub("var1", "resid", names(ord_krig))
```

We derive the kriging standard deviation map by taking the square root of the variance.

```
ord_krig["resid.sd"] <- sqrt(ord_krig["resid.var"])
```

We calculate the regression kriging prediction map by summing the maps of RF predictions and kriged residuals.

Sum RF prediction and kriged residual

```
# convert stars object to terra spatRaster
resid_pred <- rast(ord_krig["resid.pred"])
resid_sd <- rast(ord_krig["resid.sd"])

# sum the Random Forest prediction and kriged residuals
RK_pred <- RF_pred + resid_pred
names(RK_pred) <- "RFplusResid"

# compare
summary(c(soc, RK_pred, RF_pred, resid_pred))
```

Plot the regression kriging prediction and standard deviation maps.

```
# terra::plot()
cex_main <- 1
title_cex <- 1
cex_legend <- 1
plg_opts <- list(title = "ton/ha", title.cex = title_cex, cex = cex_legend)

par(mfrow = c(1, 2))
plot(RK_pred, main = "RK prediction",
     cex.main = cex_main, range = range_plot,
     axes = FALSE, box = TRUE, plg = plg_opts)
plot(resid_sd,  main = "RK standard deviation",
     cex.main = cex_main, col = hcl.colors(9, palette = "YlOrBr"),
     axes = FALSE, box = TRUE, plg = plg_opts)
```

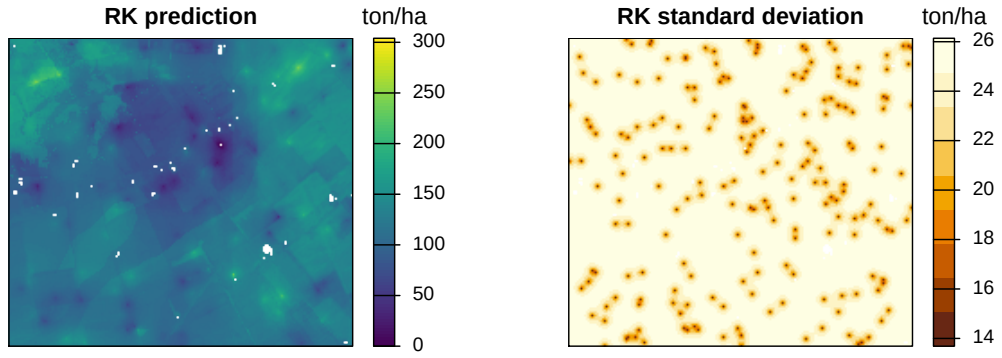


Figure 5.4: Random Forest + Residuals SOC stock predictions [ton/ha] (left) and kriged residuals standard deviation [ton/ha] (right)

The regression kriging prediction map is not very different from the RF prediction map shown in Figure 5.2. This is because the interpolated residual is often close to zero, partly because the residuals are often close to zero, and partly because it has little spatial correlation. One might therefore ask, why go through all the trouble of modelling the residual spatial variation if it hardly influences the prediction map? Well, the answer is that modelling the residual spatial autocorrelation is crucially important to be able to quantify the uncertainty of the total SOC stock in the example area, as we will see in the next and final section of this chapter.

5.4 Regression block kriging

In this section we derive the regression block kriging prediction of the total SOC stock and quantify the associated uncertainty with a 95% prediction interval. We use a similar approach as in the previous chapter, but with ordinary block kriging replaced by regression block kriging. We first define the 'block', that is the area over which we aggregate.

Create block

```
# convert mask to polygon
# mask_terra has a constant value of 1
# so all non-NA pixels are aggregated into a single polygon
block_sv <- as.polygons(mask_terra)

# spatVector to sf object
block_sf <- st_as_sf(block_sv)
```

Next we apply ordinary block kriging to the RF residuals. This is where we need the variogram of the RF residuals.

Ordinary block kriging of RF residuals

```
# define a grid relative to (0,0); the centre of the block

# block has the size of the study area: 4800 x, 4300 y
# n: number of grid points in x and y direction (columns, rows)
n_xy <- c(30, 30) # c(nx, ny)

# create a grid inside the block
grid <- st_make_grid(block_sf, n = n_xy, what = "centers")
```

```
# intersect the grid with the block and remove points in NA pixels
grid_sf <- st_as_sf(grid) # 30*30=900 points
grid_intersected <- st_filter(grid_sf, block_sf)
np_grid <- nrow(grid_intersected) # 897; 3 points dropped

# extract grid coordinates
grid_coords <- st_coordinates(grid_intersected)

# extract block centroid
block_centroid <- st_centroid(block_sf)
```

```
# the argument block must be relative to the centroid
# so the centroid represents (0,0)
# and the grid points are the distances between (0,0) and each point
# + or - depending on the direction
grid_relative <- sweep(grid_coords, 2, st_coordinates(block_centroid))
```

```
# ordinary block kriging
block_krig <- krige(formula = residuals ~ 1,
                    locations = residuals_sf,
                    newdata = block_centroid,
                    model = svg,
                    block = grid_relative)
```

```
# change variable names
names(block_krig) <- gsub("var1", "residuals", names(block_krig))

# inspect object
block_krig
```

```
Simple feature collection with 1 feature and 2 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 263899.2 ymin: 585649.7 xmax: 263899.2 ymax: 585649.7
Projected CRS: Amersfoort / RD New
  residuals.pred residuals.var                   geometry
1       0.224168       3.35276 POINT (263899.2 585649.7)
```

```
# output class is same as newdata argument
class(block_krig)
```

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

```
# save values in new variables
block_pred <- block_krig$residuals.pred
block_var <- block_krig$residuals.var
block_sd <- sqrt(block_var)
```

Finally, we derive the regression block kriging prediction and associated prediction interval of the total SOC stock in the example area.

Regression block kriging prediction and prediction interval

```
# regression block kriging prediction of total SOC stock
RK_prediction <- (RF_mean_stock + block_pred) * area_ha

paste0("Difference with the true SOC stock: ",
       round2(total_stock - RK_prediction, 0, 10), " ton")
```

```
[1] "Difference with the true SOC stock: 5180 ton"
```

```
# regression block kriging standard deviation of total SOC stock
RK_sd <- block_sd * area_ha

# print prediction and standard deviation
cat("The regression block kriging prediction of the total SOC stock is",
    "\n", round2(RK_prediction, 0, 10),"ton.")
```

```
The regression block kriging prediction of the total SOC stock is
 243210 ton.
```

```
cat("The regression block kriging standard deviation of the total SOC stock is",
    "\n", round2(RK_sd, 0, 10),"ton.")
```

```
The regression block kriging standard deviation of the total SOC stock is
 3770 ton.
```

```
# lower and upper bounds of the 95% prediction interval
lower_bound <- (RF_mean_stock + block_pred + qnorm(0.025)*block_sd) * area_ha
upper_bound <- (RF_mean_stock + block_pred + qnorm(0.975)*block_sd) * area_ha

# calculate prediction interval width
intv_width <- upper_bound - lower_bound

# print prediction interval
paste0("The 95% RF + residuals prediction interval is [",
       round2(lower_bound, 0, 10), ", ",
       round2(upper_bound, 0, 10), "] ton")
```

```
[1] "The 95% RF + residuals prediction interval is [235820, 250590] ton"
```

```
paste0("Regression kriging 95% prediction interval width: ",
       round2(intv_width, 0, 10), " ton")
```

```
[1] "Regression kriging 95% prediction interval width: 14770 ton"
```

Comparing these results with those of ordinary block kriging, we find that there is no improvement. The difference with the 'true' SOC stock of the area is a little bit bigger than before, and the prediction interval is slightly wider. Apparently, we could not benefit from covariate information to improve prediction. Why is that? The reason is that covariates were only able to explain the large-scale spatial SOC stock variation, and not the spatial variation at shorter distances. We had already concluded this from Figure 5.3. Given the fairly high sampling density (see Figure 4.1), with observations within 300~m distance from every grid cell in the area, we benefit little from the covariates.

You might be disappointed that regression kriging did not pay off, in spite of the substantial effort we put into it, but in fact this result is not uncommon. We often find that with large sample sizes ordinary kriging performs equally well as regression kriging, unless covariates explain a substantial part of the short-distance spatial variation of the target variable.

The above also tells us that we might expect regression block kriging to perform better than ordinary block kriging in a case where we have a lower sampling density, that is a smaller dataset. We will find out if this is indeed the case in Chapter 7.

6 Predicting SOC stock change

In previous chapters we only considered SOC stock spatial variation. But soil organic carbon is not static and also varies in time. This implies that the total SOC stock of the example area also varies over time. The purpose of this chapter is to use kriging to predict the total SOC stock change of the example area between two points in time, and quantify the associated prediction uncertainty. As mentioned in the firsts chapter of this tutorial, being able to do this is highly relevant to carbon farming and MRV projects.

In this chapter we will assume a constant trend and use ordinary kriging, but note that we could also have used regression kriging for predicting SOC stock change. We refer to Szatmári, Pásztor, & Heuvelink (2021) for a real-world example of regression block kriging for prediction of the SOC stock change in Hungary.

We had not mentioned it before, but the SOC stock raster that we used in previous chapters referred to the SOC stock raster for the year 2020. Similarly, we also have a SOC stock raster (i.e., a synthetic map of the 'true' SOC stock) for the year 1990. In this chapter we will sample SOC stock data from both years and use these data to predict the total SOC stock change during the 30-year period from 1990 to 2020. We will consider two cases: (**1**) the measurement locations in 1990 and 2020 are the same (i.e., measure-remeasure, Section 6.1); (**2**) the sampling locations in 2020 are not the same as those in 1990 (Section 6.2).

As usual we start with running some set-up lines.

Setup

```
# load library
library(terra)
library(sf)
library(gstat)
library(stars)
```

```
# avoid scientific notation
options(scipen=999)
```

```
# set a seed (for reproducible research)
set.seed(987)
```

```
# uncomment the two code lines below after replacing the file path
# with the location where you saved the tutorial files
# (i.e. that contains the 'soc_stock_kriging' folder)
#setwd("C:/tutorials/soc_stock_kriging")
#workingDir <- getwd()
```

```
# build input and output paths
inputDir <- file.path(workingDir, "input")
outputDir <- file.path(workingDir, "output")
```

```
# rounding function
round2 <- function(x, d, p = 1){
  p <- p*(10^-d)
  n <- round(x/p)*p
  format(n, nsmall = d)
  }
```

To ensure colour scale comparability when plotting multiple rasters, the `range` argument can be set in the `terra::plot()` function. The function below saves us a few lines of code when we need to find the minimum and maximum values for one or more rasters.

```
# function to compute raster min max values
get_minmax <- function(c_rasters){
  mimax_rasters <- global(c_rasters, fun = "range", na.rm = TRUE)
  min_range <- floor(min(mimax_rasters["min"]))
  max_range <- ceiling(max(mimax_rasters["max"]))
  rasters_range <- c("min" = min_range,
                     "max" = max_range)
  return(rasters_range)
}
```

For Section 6.2, we will need random samples in both years. We took the code from Chapter 4 and turned it into a function.

```r
# function to generate random sample of size n with random offset
# n = sample size
# r = raster (area, pixel centroids) on which the sample will be generated
random_sample_fn <- function(n, r){

  # generate random sample
  random_sample <- spatSample(
    r,
    size = n,
    method = "random",
    replace = FALSE,
    as.points = FALSE,
    na.rm = TRUE,
    values = TRUE,
    xy = TRUE # return cell centroid coordinates
  )

  # random sample jitter
  # generate a random coordinates offset within ~1 meter
  max_offset <- 1

  # extract coordinates
  coords <- random_sample[c("x", "y")]

  # runif() generates random deviates from a uniform distribution
  dx <- runif(nrow(coords), -max_offset, max_offset)
  dy <- runif(nrow(coords), -max_offset, max_offset)

  # add the offset to the coordinates
  random_sample_offset <- cbind(
    random_sample["x"] + dx,
    random_sample["y"] + dy,
    random_sample[names(r)])

  # from data frame to sf object
  random_sample_sf <- st_as_sf(random_sample_offset,
                        coords = c("x", "y"),
                        crs = "EPSG:28992")

  return(random_sample_sf)
}
```

We load and plot the SOC stock rasters from 1990 and 2020, and derive from that the 'true' SOC stock change map:

Load data

```
# load soc rasters
soc_1990 <- rast(file.path(inputDir, "soc_stock_1990.tif"))
soc_2020 <- rast(file.path(inputDir, "soc_stock_2020.tif"))
```

Prepare data

```
# rename to make it easier to call
names(soc_1990) <- "soc_1990"
names(soc_2020) <- "soc_2020"
```

Derive the 'true' SOC stock change map, i.e. delta SOC

```
# calculate the soc stock change
soc_change <- soc_2020 - soc_1990

# fix metadata
names(soc_change) <- "soc_stock_change_ton_ha_1990.2020"
varnames(soc_change) <- names(soc_change)
```

Prepare elements for ordinary block kriging

```
# prepare mask
mask_terra <- soc_change*0 + 1
names(mask_terra) <- "mask"

# convert spatRaster to stars object
mask_stars <- st_as_stars(mask_terra)
```

```
# create block
# convert mask to polygon
# mask_terra has a constant value of 1
# so all non-NA pixels are aggregated into a single polygon
block_sv <- as.polygons(mask_terra)
# spatVector to sf object
block_sf <- st_as_sf(block_sv)
```

```r
# create grid, code copied from Chapter 4
n_xy <- c(30, 30)
grid <- st_make_grid(block_sf, n = n_xy, what = "centers")
grid_sf <- st_as_sf(grid)
grid_intersected <- st_filter(grid_sf, block_sf)
np_grid <- nrow(grid_intersected)
grid_coords <- st_coordinates(grid_intersected)
block_centroid <- st_centroid(block_sf)
grid_relative <- sweep(grid_coords, 2, st_coordinates(block_centroid))
```

Prepare elements for plotting

```r
# find the min-max across the 1990 and 2020 rasters
soc_range <- get_minmax(c(soc_1990, soc_2020))
```

```r
# min-max values of the soc stock change raster
diff_range <- get_minmax(soc_change)

# get the absolute maximum to mirror the colour palette around it
max_abs <- max(abs(diff_range))

# define the palette colours
colour_palette <- colorRampPalette(c("red", "white", "blue"))

# define number of colour codes to generate from the colour palette
n_colours <- 100

# get the colour codes
colours <- colour_palette(n_colours)
```

Plot SOC stock in 1990, 2020, and delta SOC

```
# terra::plot()
cex_main <- 1
title_cex <- 1
cex_legend <- 1
plg_opts <- list(title = "ton/ha",
                 title.cex = title_cex, cex = cex_legend)
plot_maps <- c(soc_1990, soc_2020)
mains <- c("SOC stock 1990", "SOC stock 2020")

par(mfrow = c(1, 2)) # Create a n_row x n_col plotting matrix

# plot soc stock 1990
for (i in 1:nlyr(plot_maps)){
  plot(plot_maps[[i]], main = mains[i],
     range = soc_range, cex.main = cex_main, plg = plg_opts,
     axes = FALSE, box = TRUE)
}
```

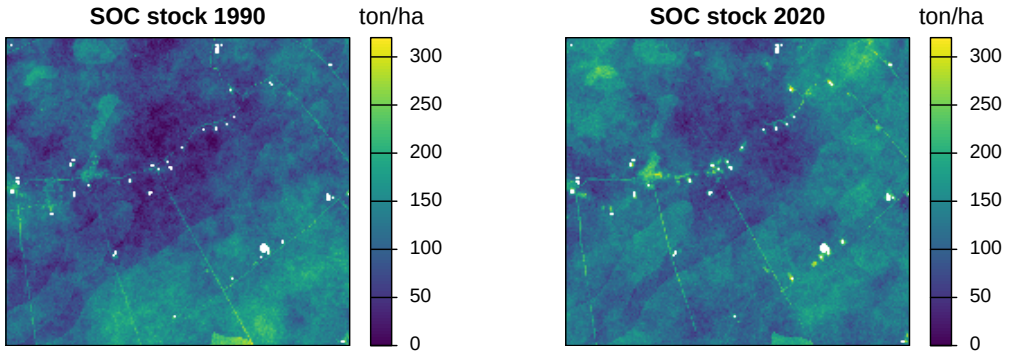


Figure 6.1: Topsoil SOC stock in the years 1990 and 2020.

```
par(mfrow = c(1, 1)) # reset plotting matrix
# plot delta soc
plot(soc_change,
     main = paste0("SOC stock change"), col = colours,
     plg = list(title = "ton/ha"), range = c(-max_abs, max_abs),
     axes = FALSE, box = TRUE)
```

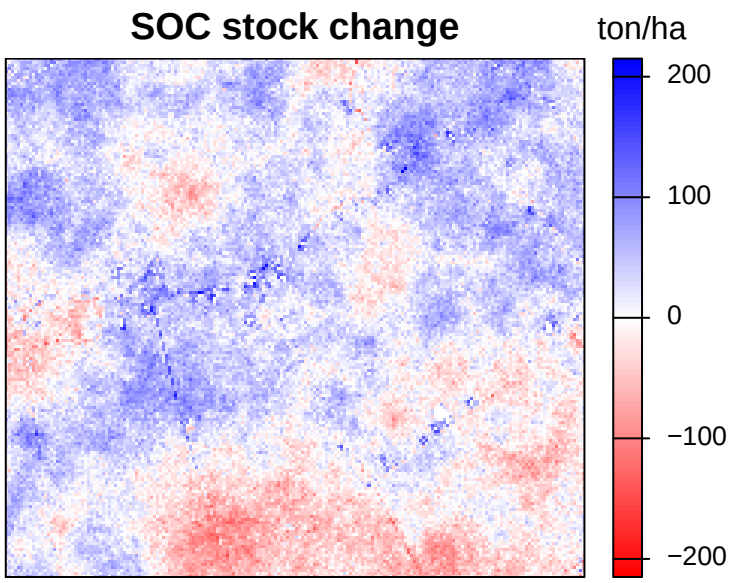


Figure 6.2: SOC stock change between 1990 and 2020.

While the 1990 and 2020 maps have a similar spatial pattern, there are also meaningful differences. It appears that overall the SOC stock has increased during the 30-year period, because there is more blue than red in the change map. This is confirmed by calculating the mean and total SOC stock change:

Mean and total SOC stock change

```
# terra::expanse() returns the area covered by all non-NA raster cells
area_ha <- expanse(soc_change, unit = "ha")[1, "area"]

# calculate the mean SOC stock changes across the non-NA pixels (ton/ha)
mean_stock_change <- global(soc_change, fun = "mean", na.rm=TRUE)[1, "mean"]
cat("'True' mean SOC stock change:", round2(mean_stock_change, 1), "ton/ha")
```

```
'True' mean SOC stock change: 16.4 ton/ha
```

```r
# total SOC stock change
tot_stock_change <- mean_stock_change * area_ha
cat("'True' SOC stock change:", round2(tot_stock_change, 0, 10), "ton")
```

```
'True' SOC stock change: 33750 ton
```

In reality, we do not know the SOC stocks in 1990 and 2020 and the SOC stock change everywhere in the example area. We can only measure the SOC stock at a finite number of sampling locations in 1990 and 2020. In the next section we explain how kriging can be used to predict the total SOC stock change from samples of SOC stock observations in 1990 and 2020, in a case where we measure the SOC stock at the same locations in both years.

6.1 Block kriging SOC stock change

In this section we assume that we measure the SOC stock at exactly the same locations in 1990 and 2020. Note that in practice this might not be that easy, since there might be positional errors and because soil sampling is destructive (but note that composite soil sampling would address both problems). Ignoring these problems, we randomly select $n = 200$ locations and extract the SOC stock at these locations for both years.

SOC observations

For consistency, we use the same random sample locations that we used in previous chapters.

```r
# load random sample
random_sample <- vect(file.path(outputDir, "random_sample.gpkg"))
sample_size <- dim(random_sample)[1]
```

We created this random sample in Chapter 4 using the 'true' 2020 SOC stock raster. The attributes table contains the random sample coordinates, and the `soc_ton_ha` column stores the extracted 2020 SOC stock values. We now also extract the 1990 SOC stock values, and calculate their difference. Since the SOC stock change is all we need in this section, we replace all contents of `random_sample` with the subtract the extracted values of `soc_1990` from those of `soc_2020`.

```
# extract soc values at sample locations
extr_1990 <- terra::extract(soc_1990, random_sample, ID = FALSE)
extr_2020 <- terra::extract(soc_2020, random_sample, ID = FALSE)

# calculate the difference
extr_diff <- extr_2020 - extr_1990

# extr_diff is a data frame, but we need a spatial object
# assign back to the random sample SpatVector
values(random_sample) <- extr_diff

# rename extracted values column
names(random_sample) <- "delta_soc"

# check again
head(random_sample)
```

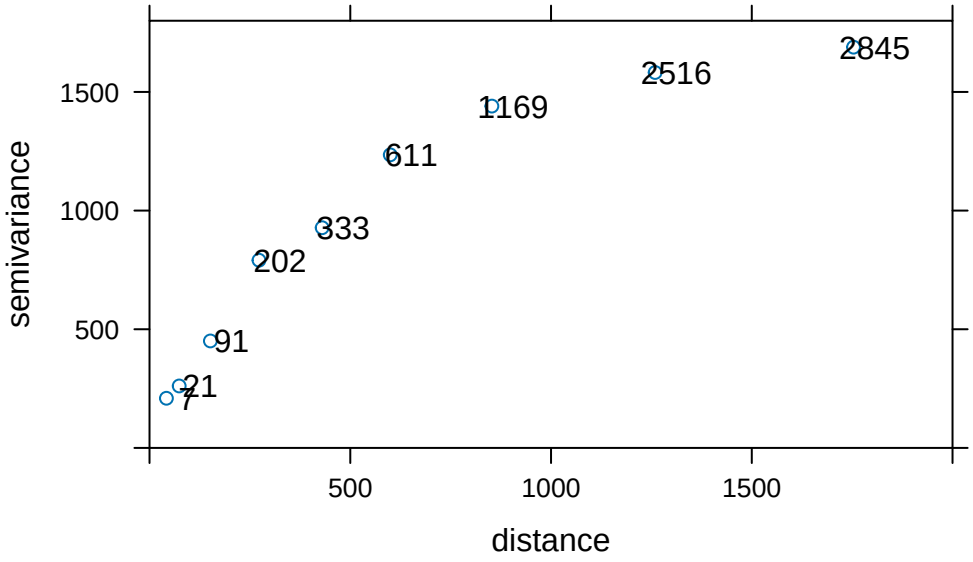
The way forward is obvious: we simply apply the ordinary block kriging that we did in Chapter 4 to the 200 ΔSOC stock change observations. We first create a `gstat` object, next calculate the experimental variogram and fit a variogram model, and use this to predict the spatial average SOC stock change with ordinary block kriging. This is done in the code chunks below.

Create `gstat` object

```
# convert spatVector to sf object
random_sample_sf <- st_as_sf(random_sample)

# define `gstat` object
# slightly different from previous chapters, we do not need get() because
# delta_soc is not a variable; it is the column name where the data
# are stored inside random_sample_sf
g <- gstat(formula = delta_soc ~ 1, data = random_sample_sf)
```

Experimental variogram

```
esvg <- variogram(g, boundaries = c(50,100,200,350,500,700,1000,1500,2000))
plot(esvg, xlim = c(0, 2000), ylim = c(0, 1800), plot.nu = TRUE)
```



Fitting a variogram model

```
# define initial variogram model
svg_ini <- vgm(nugget = 200, psill = 1400, range = 500, model = "Exp")

# fit variogram model
svg <- fit.variogram(esvg, svg_ini, fit.method = 7)
svg
```

```
  model      psill    range
1   Nug   56.93752   0.0000
2   Exp 1699.87108 532.5124
```

```
# save nugget, sill and range
nugget <- svg[1,2]
sill <- svg[1,2] + svg[2,2]
range_parameter <- svg[2,3]
```

```r
plot(esvg, svg, xlim = c(0, 2000), ylim = c(0, 1800), plot.nu = TRUE)
```

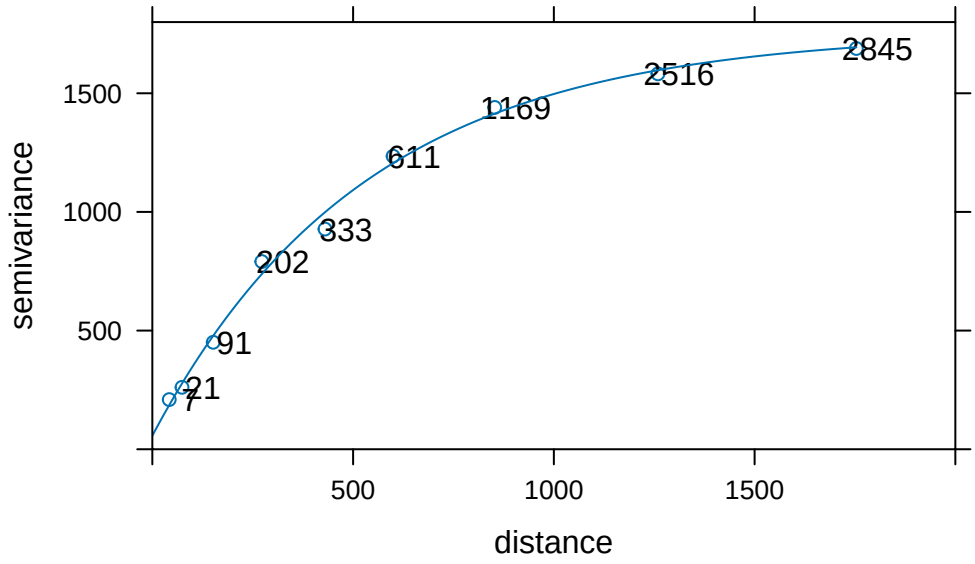


Figure 6.3: Fitted variogram of SOC stock change.

Note that the SOC stock change variogram is different from the SOC stock variogram shown in Figure 4.4. For example, we find that the SOC stock change variogram has a smaller nugget. In fact, it is surprising that the sill of the SOC stock change variogram is similar to that of the SOC stock variogram. This tells us that the SOC stock change over 30 years is quite large for many locations in the example area, and that it has similar magnitude as the spatial variability of the SOC stock. You can verify this yourself by comparing the variance of the SOC stock observations in 2020 with the variance of the SOC stock change observations.

Note also that in the script above we used `fit.method=7` in the `fit.variogram` function, whereas we used `fit.method=1` in Chapter 4 and Chapter 5. We did this because this resulted in visually more attractive variogram models, but we invite users to check this out themselves by changing the parameter value and consult the help function of `fit.variogram` for explanation of the different parameter settings.

Now that we have a variogram we can apply ordinary block kriging and predict the total SOC stock change in the example area, and quantify the associated uncertainty with a 95% prediction interval. The R script is very similar to that used in Chapter 4.

Ordinary block kriging

```
# ordinary block kriging
block_krig <- krige(formula = delta_soc ~ 1,
                    locations = random_sample_sf,
                    newdata = block_centroid,
                    model = svg,
                    block = grid_relative)
```

```
names(block_krig) <- gsub("var1", "soc_change", names(block_krig))
block_pred <- block_krig$soc_change.pred
block_var <- block_krig$soc_change.var
```

Block kriging standard deviation

```
# calculate the block kriging standard deviation
block_sd <- sqrt(block_var)

cat("When discretizing the area using", np_grid, "grid points,", "\n",
    "soc_change.pred ~", round2(block_pred,2),
    "and soc_change.sd ~", round2(block_sd,2), "ton/ha")
```

```
When discretizing the area using 897 grid points,
 soc_change.pred ~ 14.41 and soc_change.sd ~ 1.94 ton/ha
```

Block kriging predicted total SOC stock change between 1990 and 2020

```
# calculate total SOC stock change predicted with block kriging (ton)
block_krig_total_soc_change <- block_pred * area_ha
cat("Predicted total SOC stock change with ordinary block kriging:",
    round2(block_krig_total_soc_change, 0, 10), "ton")
```

```
Predicted total SOC stock change with ordinary block kriging: 29660 ton
```

Prediction interval

```
lower_limit <- (block_pred + qnorm(0.025)*block_sd) * area_ha
upper_limit <- (block_pred + qnorm(0.975)*block_sd) * area_ha
intv_width <- upper_limit - lower_limit
cat(paste0("The 95% prediction interval is [", round2(lower_limit, 0, 10),
           ", ", round2(upper_limit, 0, 10), "] ton"), "\n")
```

```
The 95% prediction interval is [21830, 37480] ton
```

```
cat(paste0("Prediction interval width: ", round2(intv_width, 0, 10), " ton"))
```

```
Prediction interval width: 15660 ton
```

The predicted SOC stock change is 29660 ton, meaning that we predict that in 30 years the SOC stock in the example area has increased by 13.8%. Moreover, we find that this predicted change is statistically significant, because both the lower and upper limit of the prediction interval are greater than zero. Note also that the 'true' SOC stock change, that we had calculated at the start of this chapter and that was equal to 33750 ton, is inside the prediction interval.

With $n = 200$ paired observations we are almost certain that the SOC stock has increased from 1990 to 2020. You can check this yourself by calculating the limits of a 99.9% prediction interval, and verifying that the lower and upper limits of the associated prediction interval are both greater than zero.

It is also interesting to compare the results above with those obtained in Section 6.1 of the Sampling theory for estimating soil organic carbon change - a tutorial. In that tutorial, the SOC stock change was estimated as 34970 ton, which is similar to our predicted SOC stock change. But more interestingly, we find that the confidence interval width obtained in that tutorial, which was 17080 ton, is larger than the prediction interval width obtained in this section, which is 15660 ton. This is remarkable, because in this chapter we used $n = 200$ SOC stock change observations, while in Section 6.1 of the sampling tutorial, we used $n = 400$ observations. In other words, replacing a sampling theory estimation method by a geostatistical prediction method achieved a higher accuracy, even though it used only half the sample size.

6.2 Block cokriging SOC stock change

In Section 6.1 we assumed that we measured the SOC stock at the same locations in 1990 and 2020. This simplified the analysis quite a lot because we could work on the SOC stock change directly, by subtracting the SOC stock measured in 1990 from that measured in 2020 and applying block kriging to the difference. In most cases this will be the preferred approach.

Not only because it is relatively easy, but also because using paired observations filters out the static spatial variability, and can thus achieve higher prediction accuracy than methods that do not remove static spatial variability.

But the downside is that we must measure at the same locations in each time period, while in practice this may not always be feasible. For instance, due to budget cuts or budget increases we might have a different sample size in 1990 than in 2020, or we might find that some land owners do not give permission to take soil samples in both periods. One other reason to avoid measure-remeasure is that such practice is sensible to fraud: land owners who benefit financially from SOC stock increase might be tempted to pay more effort to increase the SOC stock at locations that were sampled in 1990 than elsewhere.

In this section we therefore address the case that the sampling locations and/or sample sizes at the two points in time are not identical. This does not mean that there can be no overlap, for example we could have a case where we had n=200 measurements in the first time period, revisit these same locations in 2020, but have extra budget so that we can add another 300 observations in 2020.

The method presented in this section is therefore more flexible, but this comes at the price of increased complexity and perhaps also reduced prediction accuracy. We will know at the end of this section.

We begin this section with explaining the theory of cokriging, because that is the method that we need in this case.

In Chapter 3 we had one variable of interest (i.e., the SOC stock in 2020), for which we defined a geostatistical model $Z = \{Z(s)|s \in A\}$. But now we have two variables of interest, that is SOC stock 1990 and SOC stock 2020, which we denote by Z_1 and Z_2, respectively.

We assume that both variables satisfy the conventional geostatistical model, given by:

$$Z_i(s) = \mu_i(s) + \varepsilon_i(s), \quad \text{for all } s \in A \text{ and } i = 1, 2$$

We will further assume second-order stationarity, that is that the μ_i are constant and that the direct variograms of Z_1 and Z_2 only depend on the distance h between locations:

$$\gamma_i(h) = \frac{1}{2} E\left[(Z_i(s) - Z_i(s+h))^2\right], \quad i = 1, 2$$

This is all familiar from Chapter 3. In cokriging we also need a **cross-variogram**, to model the correlation between Z_1 and Z_2:

$$\gamma_{12}(h) = \frac{1}{2} E\left[(Z_1(s) - Z_1(s+h)) \cdot (Z_2(s) - Z_2(s+h))\right]$$

Note that here we assumed that the cross-variogram also only depends on the distance between locations. Note also that a cross-variogram need not be positive, because it multiplies instead

of taking a square. However, in our case we expect it will be positive for all h, because if the SOC stock in 1990 is higher at location $s+h$ than at location s, then it is likely that the SOC stock in 2020 will also be higher at location $s+h$ than at location s.

The advantage of cokriging over kriging is that it uses observations of all variables for prediction of a variable at unmeasured locations. In our case we have only two variables, so that we get:

$$\hat{z}_1(s_0) = \sum_{i=1}^{n} \lambda_i \cdot z_1(s_i) + \sum_{i=n+1}^{n+m} \nu_i \cdot z_2(s_i)$$

Here, we assumed that variable z_1 was measured at locations s_1 to s_n and variable z_2 at locations s_{n+1} to s_{n+m}. Note that this still allows locations to partly overlap, we might for example have that $s_i = s_{n+j}$ for some $i = 1, \ldots, n$ and $j = 1, \ldots, m$.

Similar to Chapter 3, the cokriging weights λ_i and ν_i are calculated by solving a kriging system, but in cokriging this system is larger because it consists of $n+m+2$ equations that are derived from the two direct variograms and the cross-variogram.

Cokriging uses more information than kriging and can potentially do better, but in practice the accuracy gain is often limited. This is because to ensure that predictions are unbiased we need to impose that the λ_i sum to 1 and that the ν_i sum to 0. If our only goal were to predict the SOC stock change, we might as well apply block kriging twice, one to the 1990 case and once to the 20202 case, and subtract the predictions. But we also want to quantify the uncertainty of that prediction, and for this we need cokriging (unless we have paired observations, see the previous section). The advantage of cokriging is that it not only quantifies the prediction error variances of Z_1 and Z_2, but also the covariance of the prediction errors, that is it calculates $cov(Z_1(s_0) - \hat{Z}_1(s_0), Z_2(s_0) - \hat{Z}_2(s_0))$ for all prediction locations $s_0 \in A$. We will need this covariance when we calculate the uncertainty of the SOC stock change at the end of this section.

There is only one more issue that we need to address before we can apply cokriging to our example area. That is that the cross-variogram defined above can only be estimated if we have measurements of both Z_1 and Z_2 at sampling locations, as is obvious from its definition above. But we do not have that, because this section is about a case where the sampling locations in 1990 and 2020 are not the same. The solution is to replace the cross-variogram with the so-called **pseudo cross-variogram**, defined as:

$$\tilde{\gamma}_{12}(h) = \frac{1}{2} E\left[(Z_1(s) - Z_2(s+h))^2\right]$$

It can be shown that under stationarity and symmetry assumptions the pseudo cross-variogram and cross-variogram differ only by a constant. In the application below we will choose this constant such that the cross-variogram gets a nugget equal to the smallest of the nuggets of the two direct variograms.

Let us now apply cokriging to the example area. We first sample the SOC stock at 200 locations, both for 1990 and 2020. Note that these are not the same locations.

Generate random samples for 1990 and 2020

```
n_samples <- 200
random_sample_1990 <- random_sample_fn(n = n_samples, r = soc_1990)
random_sample_2020 <- random_sample_fn(n = n_samples, r = soc_2020)

summary(random_sample_1990[["soc_1990"]])
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.318  75.143  97.147 104.098 134.873 239.536
```

```
summary(random_sample_2020[["soc_2020"]])
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  37.61   96.65  119.87  119.26  142.55  214.17
```

Plot random samples for 1990 and 2020

```
# plot margins
par(mar = c(0, 0, 4, 0), # c(bottom, left, top, right)
    cex.main = 1)

bbox <- st_bbox(mask_stars)
# factor to expand extent by 10% to the right
x_expand <- (bbox["xmax"] - bbox["xmin"]) * 0.2

col_mask <- "#b4dab4"
col_1990 <- "indianred1"
col_2020 <- "deepskyblue"
pch_1990 <- 21
pch_2020 <- 21
```

```r
plot(mask_stars, col = col_mask, axes = FALSE, key.pos = NULL,
     breaks = "equal", xlim = c(bbox["xmin"], bbox["xmax"] + x_expand),
     reset = FALSE, main = "Study area with sample points")

plot(st_geometry(random_sample_1990), add = TRUE,
     pch = pch_1990, bg = col_1990, col = "black", cex = 0.8)

plot(st_geometry(random_sample_2020), add = TRUE,
     pch = pch_2020, bg = col_2020, col = "black", cex = 0.8)

legend("topright",
       legend = c("Study area", "1990 locations", "2020 locations"),
       fill   = c(col_mask, NA, NA),
       border = c(NA, NA, NA),
       pch    = c(NA, pch_1990, pch_2020),
       pt.bg  = c(NA, col_1990, col_2020),
       col    = c(NA, "black", "black"),
       cex = 0.8,
       bty = "n")
```

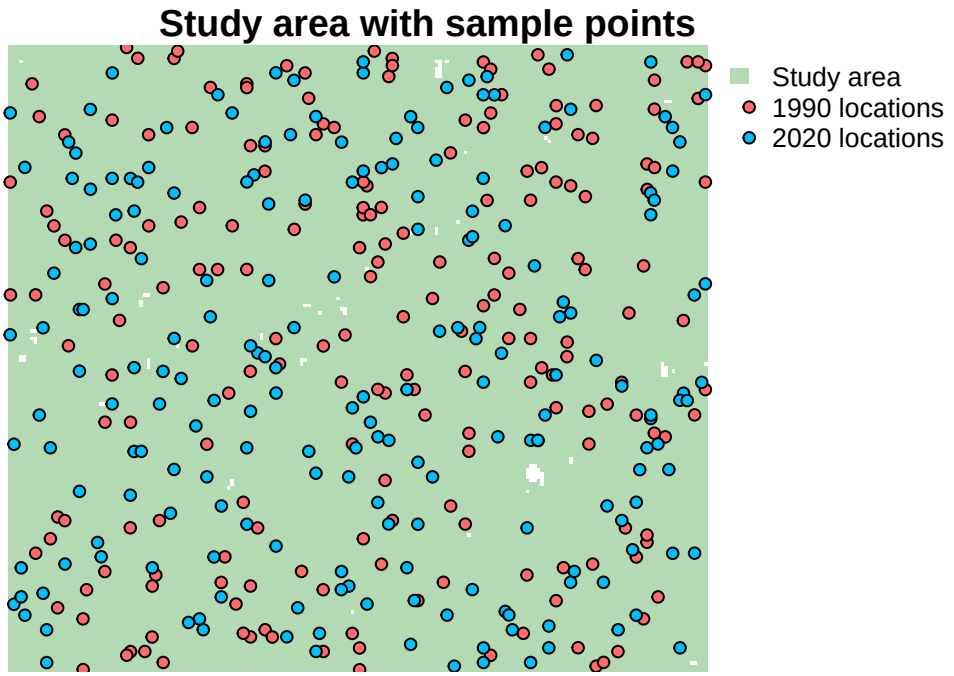


Figure 6.4: Spatial distribution of 200 randomly sampled locations in 1990 and 2020.

Next we calculate the experimental variograms and fit variogram models. Note that we will get two direct variograms and one pseudo cross-variogram.

Cokriging - experimental variograms

```
# create multivariable gstat object
g <- gstat(id = "soc_1990", formula = soc_1990 ~ 1,
           data = random_sample_1990)
g <- gstat(g, id = "soc_2020", formula = soc_2020 ~ 1,
           data = random_sample_2020)

# plot experimental variograms
bounds = c(100,200,350,500,700,1000,1500,2000)
vgm <- variogram(g, boundaries = bounds)
plot(vgm)
```

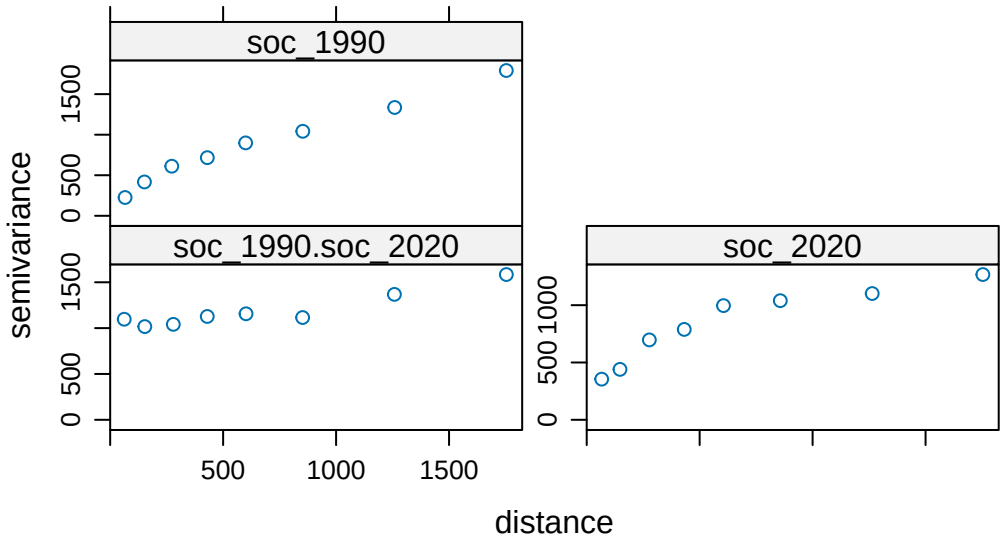


Figure 6.5: Cokriging - experimental variograms.

Cokriging - initial variogram models

```
# define initial variogram models (eye fitting)
common_range <- 800
svg_ini_1990 <- vgm(nugget=200, psill=1800, range=common_range, model="Exp")
svg_ini_2020 <- vgm(nugget=200, psill=1100, range=common_range, model="Exp")
svg_ini_1990_2020 <- vgm(nugget=1000, psill=500, range=common_range, model="Exp")

g <- gstat(g, id = "soc_1990", model = svg_ini_1990)
g <- gstat(g, id = "soc_2020", model = svg_ini_2020)
g <- gstat(g, id=c("soc_1990", "soc_2020"), model = svg_ini_1990_2020)

plot(vgm, model=g$model)
```

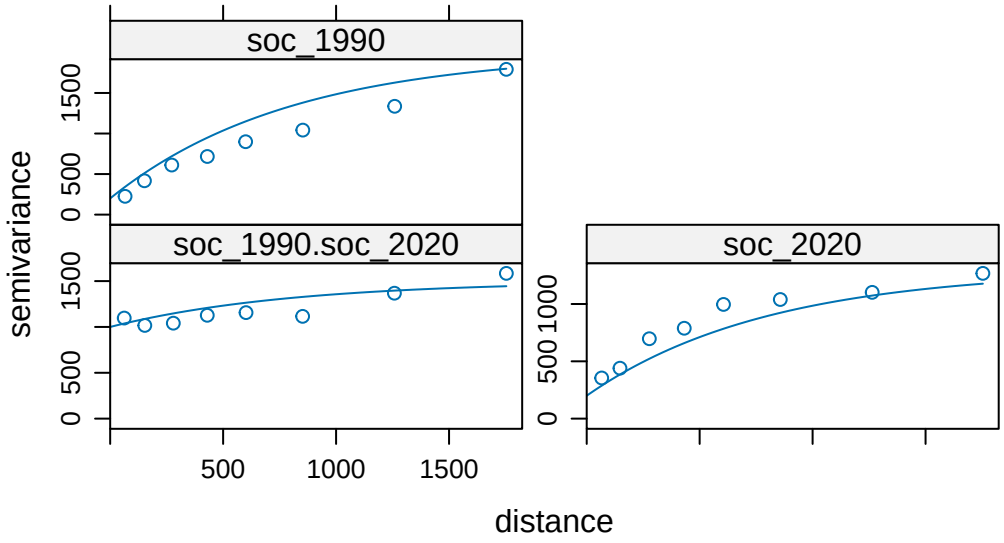


Figure 6.6: Cokriging - initial variogram models (eye fitting).

Cokriging - fit variogram

```
# fit a multivariable sample variogram using gstat::fit.lmc()
g_fit <- fit.lmc(vgm, g, fit.lmc=FALSE, fit.ranges = FALSE)
plot(vgm, model = g_fit$model)
```

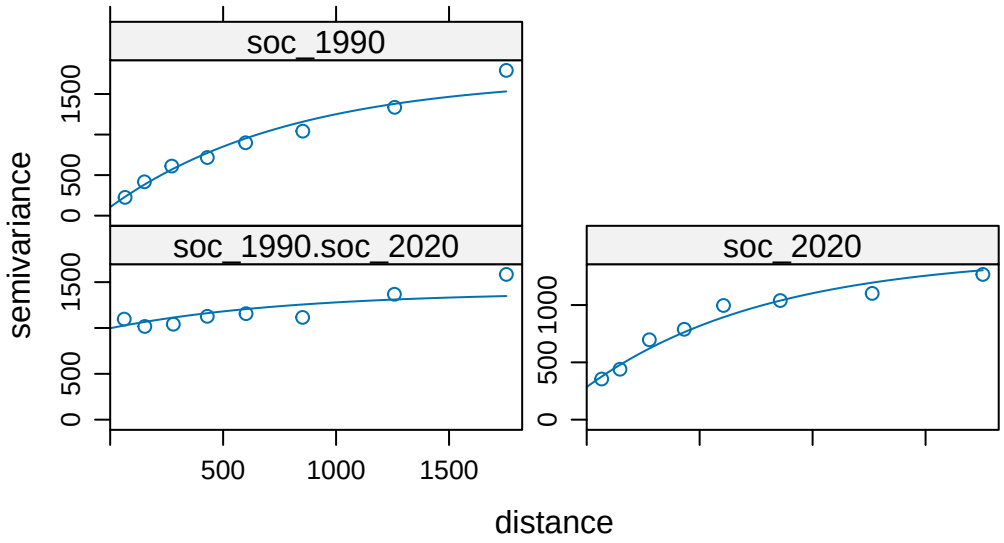


Figure 6.7: Cokriging - fit multivariable variogram - Pseudo cross-variogram.

The fitted models are satisfactory, but we must not forget that the variogram shown in the bottom-left corner of Figure 6.7 is a pseudo cross-variogram, while in cokriging we need a cross-variogram. As explained earlier, the two differ by a constant, and we choose this constant such that the cross-variogram gets a nugget equal to the smallest nugget of the two direct variograms:

Cokriging - cross-variogram

```
# change nugget of the cross-variogram
g_fit $model$soc_1990.soc_2020[1,2] <- min(g_fit$model$soc_1990[1,2],
                                           g_fit$model$soc_2020[1,2])
plot(vgm, model = g_fit$model)
```

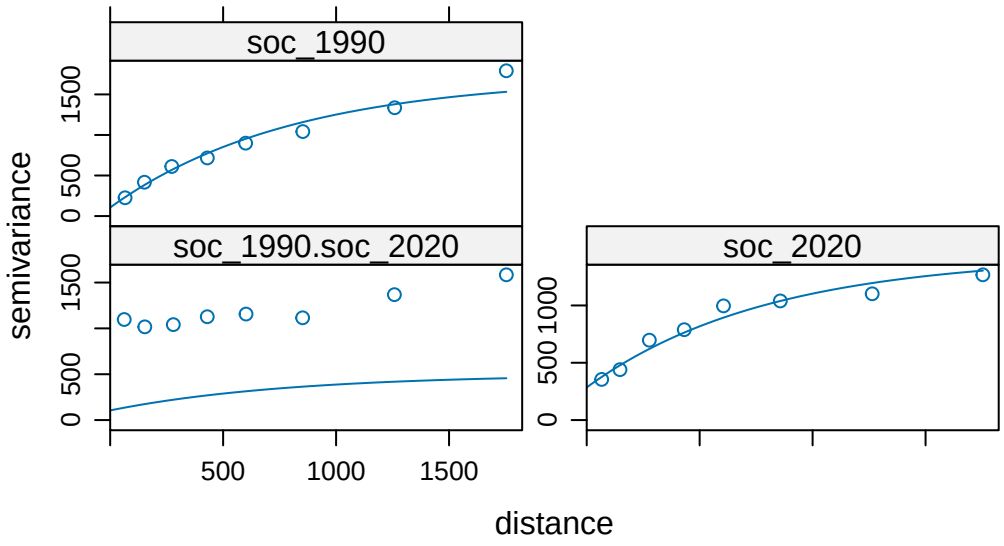


Figure 6.8: Cokriging - Cross-variogram.

With this model we can now apply ordinary block cokriging. This yields predictions of the mean SOC stock in the example area for 1990 and 2020, from which we calculate the predicted SOC stock change. It also yields prediction error variances for 1990 and 2020, as well as the covariance of the prediction errors for both years. From this we compute the prediction error variance of the SOC stock change, using the identity:

$$var(X - Y) = var(X) + var(Y) - 2 \cdot cov(X, Y)$$

where in our case X and Y are the prediction errors of the total SOC stock in 2020 and 1990, respectively.

Block cokriging predicted total SOC stock change between 1990 and 2020

```
# block cokriging
co.krig <- predict(object=g_fit, newdata=block_centroid, block=grid_relative)
```

```
Linear Model of Coregionalization found. Good.
[using ordinary cokriging]
```

```
# calculate total SOC stock change predicted with block cokriging (ton)
co.krig.change <- co.krig[["soc_2020.pred"]] - co.krig[["soc_1990.pred"]]
co.krig.total.change <- co.krig.change * area_ha
# print
cat("Predicted total SOC stock change with block cokriging:",
    round2(co.krig.total.change, 0, 10), "ton")
```

```
Predicted total SOC stock change with block cokriging: 30170 ton
```

Cokriging prediction interval

```
# calculate the prediction error variance of the SOC stock change
co.krig.var <- co.krig[["soc_1990.var"]] +
  co.krig[["soc_2020.var"]] - 2*co.krig[["cov.soc_1990.soc_2020"]]

# compute the lower and upper limit of the 95% prediction interval
co.krig.sd <- sqrt(co.krig.var)
lower_limit <- (co.krig.change + qnorm(0.025)*co.krig.sd) * area_ha
upper_limit <- (co.krig.change + qnorm(0.975)*co.krig.sd) * area_ha

# calculate prediction interval width
intv_width <- upper_limit - lower_limit

# print prediction interval
cat(paste0("The 95% prediction interval is [", round2(lower_limit, 0, 10),
           ", ", round2(upper_limit, 0, 10), "] ton"), "\n")
```

```
The 95% prediction interval is [20120, 40220] ton
```

```
cat(paste0("Prediction interval width: ", round2(intv_width, 0, 10), " ton"))
```

```
Prediction interval width: 20110 ton
```

Comparison with the results of Section 6.1 show that the predicted SOC stock change is not very different and now a bit closer to the 'true' SOC stock change (but with a different seed this might have been the other way round), but more importantly that the prediction interval width has increased from 15660 to 20110 ton. We had already anticipated that because unlike in Section 6.1, we did not have paired observations and so could not eliminate any static spatial variation.

7 Effect of sample size on prediction accuracy

In previous chapters we analysed how block kriging can be used to predict the total SOC stock and its change between two points in time for the example area. We always quantified the accuracy of a prediction with a prediction interval. In all cases we used a sample size of $n = 200$. In this last content chapter we will analyse how the prediction interval width depends on sample size. We expect that the larger the sample size, the smaller the prediction interval width, but we do not know the exact form of that relationship.

We will analyse the sample size effect on the prediction interval width for the case where we predict the total SOC stock of the example area for the year 2020, so that we can use the same approach and scripts as used in Chapter 4 and Chapter 5. Please note that a similar analysis could be done for prediction of the total SOC stock change, using the scripts from Chapter 6.

At the end of this chapter we will plot the prediction interval width against sample size for ordinary block kriging and regression block kriging. In addition, we will also plot in the same figure the confidence interval width against sample size as obtained in Sampling theory for estimating soil organic carbon change - a tutorial. We are interested to learn which method has the smallest width!

In this chapter, some code chunks might take a while to run. We have therefore made the intermediate results available in the `outputDir` folder — if a chunk is taking too long, you can load these instead.

As usual we start with running some set-up lines.

Setup

```
# load libraries
library(terra)
library(sf)
library(gstat)
library(stars)
library(ggplot2)
library(cvTools) # for creating data folds
library(ranger) # for fitting a random forest model
```

```
# avoid scientific notation
options(scipen=999)
```

```
# uncomment the two code lines below after replacing the file path
# with the location where you saved the tutorial files
# (i.e. that contains the 'soc_stock_kriging' folder)
#setwd("C:/tutorials/soc_stock_kriging")
#workingDir <- getwd()
```

```
# build input and output paths
inputDir <- file.path(workingDir, "input")
outputDir <- file.path(workingDir, "output")
```

```
# rounding function
round2 <- function(x, d, p = 1){
  p <- p*(10^-d)
  n <- round(x/p)*p
  format(n, nsmall = d)
  }
```

Since in this chapter we repeatedly need to sample from the example area, it is useful to write an R function that does this. We took the code from Chapter 4. Note that this includes adding a little noise to the geographic coordinates to avoid that observation locations are identical to grid cell centres.

```r
# function to generate random sample of size n with random offset
# n = sample size
# r = raster (area, pixel centroids) on which the sample will be generated
random_sample_fn <- function(n, r){

  # generate random sample
  random_sample <- spatSample(
    r,
    size = n,
    method = "random",
    replace = FALSE,
    as.points = FALSE,
    na.rm = TRUE,
    values = TRUE,
    xy = TRUE # return cell centroid coordinates
  )

  # random sample jitter
  # generate a random coordinates offset within ~1 meter
  max_offset <- 1

  # extract coordinates
  coords <- random_sample[c("x", "y")]

  # runif() generates random deviates from a uniform distribution
  dx <- runif(nrow(coords), -max_offset, max_offset)
  dy <- runif(nrow(coords), -max_offset, max_offset)

  # add the offset to the coordinates
  random_sample_offset <- cbind(
    random_sample["x"] + dx,
    random_sample["y"] + dy,
    random_sample[names(r)])

  # from data frame to sf object
  random_sample_sf <- st_as_sf(random_sample_offset,
                        coords = c("x", "y"),
                        crs = "EPSG:28992")

  return(random_sample_sf)
}
```

We also define two functions used in random forest regression kriging.

```r
# function to build formula for ranger()
# names_rgmx: column names in the regression matrix
# covars_names: terra::names(covars)

build_formula_fn <- function(names_rgmx, covars_names){
  # extract column names from rgmx where covariates values are stored
  covars_in_rgmx <- names_rgmx[which(names_rgmx %in% covars_names)]

  # construct covariate string
  covar_string <- paste(covars_in_rgmx, collapse = " + ")

  # soc_values stores the name that was assigned to the soc raster
  # in the Prepare data chunk
  voi <- soc_values

  # construct formula string
  formula_string <- paste(voi, " ~ ", covar_string)

  # convert to formula object
  mod_for <- as.formula(formula_string)

  # return formula
  return(mod_for)
}
```

```r
# nrFolds: number of folds to be created
# nrow_rgmx: regression matrix number of rows

create_folds_fn <- function(nrFolds, nrow_rgmx){

  # cvFolds assigns each row to a fold
  folds <- cvFolds(nrow_rgmx, K = nrFolds, type = "random")

  # the row index is stored in "subsets" and the fold number in "which"
  folds_df <- data.frame("which" = folds$which, "subsets" = folds$subsets)

  return(folds_df)
}
```

The code below loads the SOC stock 2020 raster and computes the 'true' mean and total SOC stock of the example area, in the same way as done before in Chapter 4.

Load data

```
# load soc raster
soc_fn <- "soc_stock_2020.tif"
soc <- rast(file.path(inputDir, soc_fn))
```

Prepare data

```
# modify soc raster metadata to make it easier to call
soc_values <- "soc_ton_ha"
names(soc) <- soc_values
```

```
area_ha <- expanse(soc, unit = "ha")[1, "area"]
mean_stock <- global(soc, fun = "mean", na.rm=TRUE)[1, "mean"]
total_stock <- mean_stock * area_ha
```

To prepare for block kriging we also need to define the prediction grid. We repeat the steps from Chapter 4.

Prepare elements for block kriging as explained in Section 4.3

```
# create prediction grid, code copied from Chapter 4
mask_terra <- soc*0 + 1
names(mask_terra) <- "mask"
block_sv <- as.polygons(mask_terra)
block_sf <- st_as_sf(block_sv)
n_xy <- c(30, 30)
grid <- st_make_grid(block_sf, n = n_xy, what = "centers")
grid_sf <- st_as_sf(grid)
grid_intersected <- st_filter(grid_sf, block_sf)
grid_coords <- st_coordinates(grid_intersected)
block_centroid <- st_centroid(block_sf)
grid_relative <- sweep(grid_coords, 2, st_coordinates(block_centroid))
```

7.1 Effect of sample size with ordinary block kriging

We use 8 different sample sizes, starting with $n = 25$ and doubling this until the largest sample size of $n = 1600$.

Generate n random samples

```
# define sample size
sample_size <- c(25, 50, 100, 200, 400, 800, 1600)
```

```
# create a list to store the random sample for each sample size
random_sample_list <- vector(mode = "list", length = length(sample_size))
names(random_sample_list) <- paste0("n", sample_size)

for (i in seq_along(sample_size)){
  set.seed(987 + 1)
  # generate a random sample of size n with random offset
  # and save it in a list
  random_sample_list[[i]] <- random_sample_fn(n = sample_size[i], r = soc)
}
```

```
# optional: load random_sample_list
#random_sample_list <- readRDS(file.path(outputDir, "random_sample_list.rds"))
```

To avoid confounding the results with differences in fitted variograms, we use the SOC stock variogram from Section 4.1 for all sample sizes.

We apply ordinary block kriging for each sample size and store the SOC stock prediction, the 0.025 and 0.975 quantiles, as well as the 95% prediction interval width.

Varying sample size in block kriging

```
# load ordinary kriging parameters from Chapter 4
OK <- readRDS(file.path(outputDir, "OK_params.rds"))
```

```
# use variogram from Chapter 4: Ordinary Kriging (OK)
svg_ok <- vgm(nugget = OK["nugget"], psill = OK["sill"],
              range = OK["range"], model = "Exp")
```

```
# warning! It might take a while to run
# below there is a code chunk to load the df_ok from the outputDir
```

```r
# create a data frame to store values per iteration
df_ok <- data.frame(matrix(nrow=0, ncol=4))
colnames(df_ok) <- c("OK_pred_mean", "lower_limit", "upper_limit", "PIW")

for (i in seq_along(random_sample_list)){

  # get random sample of size n
  random_sample_sf <- random_sample_list[[i]]

  # ordinary block kriging
  block_krig <- krige(formula = get(soc_values) ~ 1,
                      locations = random_sample_sf,
                      newdata = block_centroid,
                      model = svg_ok,
                      block = grid_relative)

  # extract prediction and standard deviation
  block_pred <- block_krig$var1.pred
  block_sd <- sqrt(block_krig$var1.var)

  # compute prediction mean in the area
  OK_pred_mean <- block_pred * area_ha

  # compute the lower and upper limit of the 95% prediction interval
  lower_limit <- (block_pred + qnorm(0.025)*block_sd) * area_ha
  upper_limit <- (block_pred + qnorm(0.975)*block_sd) * area_ha

  # calculate prediction interval width
  PIW <- upper_limit - lower_limit

  # save mean, lower and upper limits and PIW per iteration
  df_ok[i,] <- c(OK_pred_mean, lower_limit, upper_limit, PIW)
}

# add sample size to the data frame
df_ok <- cbind(sample_size, df_ok)
```

```r
# optional: load df_ok
#df_ok <- readRDS(file.path(outputDir, "df_ok.rds"))
```

We plot the predicted total SOC stock and the lower and upper limits of the 95% prediction interval against sample size, and in a separate figure the prediction interval width against sample size.

Sample size vs total SOC stock (OK)

```
# n vs SOC (predicted mean with 95% PI, and true value)
y_range <- range(df_ok$lower_limit, df_ok$upper_limit, total_stock)

ggplot(df_ok, aes(x = sample_size)) +
  geom_errorbar(aes(ymin = lower_limit, ymax = upper_limit,
                    color = "OK prediction (95% PI)"),
                width = 20) +
  geom_point(aes(y = OK_pred_mean, color = "OK prediction (95% PI)"),
             size = 2) +
  geom_hline(aes(yintercept = total_stock, color = "True SOC stock"),
             linetype = "dashed") +
  scale_color_manual(name = NULL,
                     values = c("OK prediction (95% PI)" = "steelblue",
                                "True SOC stock" = "black")) +
  scale_x_continuous(breaks = sample_size) +
  scale_y_continuous(breaks = pretty(y_range, n = 8)) +
  labs(x = "Sample size n", y = "Total SOC stock [ton]") +
  theme_bw() +
  theme(legend.position = c(0.74, 0.89),
        legend.background = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1))
```

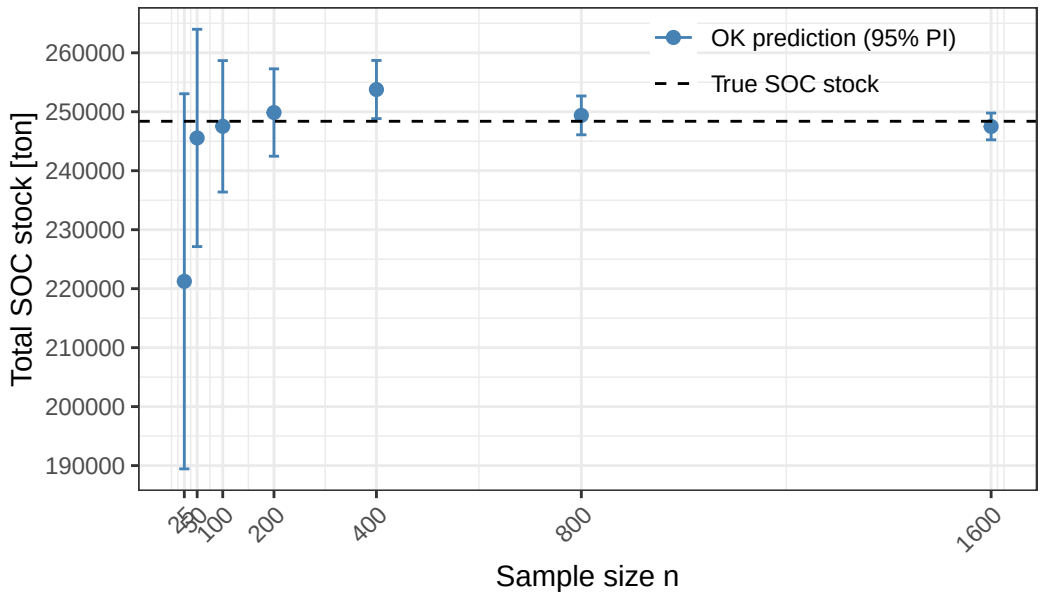


Figure 7.1: Total SOC stock prediction and 95% prediction interval against sample size, ordinary block kriging case.

Sample size vs PIW (OK)

```
# n vs PIW
par(mar = c(4, 5, 2, 2)) # margins c(bottom, left, top, right)
plot(sample_size, df_ok[["PIW"]], type = "b", pch = 16,
     col = "steelblue", cex = 1, main = NULL,
     xlab = "Sample size n",
     ylab = "Prediction Interval Width (PIW)",
     xlim = c(0, max(sample_size)),
     ylim = c(0, max(df_ok[["PIW"]])),
     xaxt = "n", yaxt = "n")  # suppress default axes
# add actual values of n and PIW to the axis
axis(1, at = c(0, sample_size), labels = c(0, sample_size), cex.axis = 0.7)
axis(2, at = round(c(0, df_ok[["PIW"]])),
     labels = round(c(0, df_ok[["PIW"]])), las = 1, cex.axis = 0.7)
```

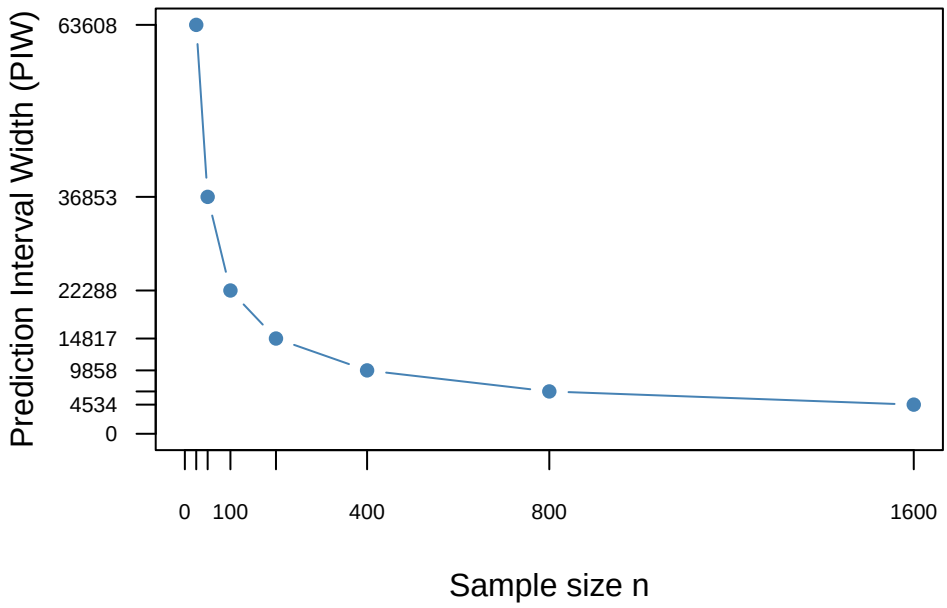


Figure 7.2: Prediction interval width against sample size, ordinary block kriging case.

Figure 7.1 shows, as expected, that predictions tend to be closer to the true SOC stock as the sample size increases. It is also interesting to see that in all cases the true SOC stock is inside the 95% prediction interval. Figure 7.2 shows that the PIW decreases with increased sample size, but also that the accuracy gain becomes smaller with larger sample sizes. This

is definitely true in an absolute sense, but also relatively. Initially, doubling from $n = 25$ to $n = 50$ leads to a prediction interval that is 42.1% narrower, while doubling the sample size from $n = 800$ to $n = 1600$ leads to a reduction of 31.2%.

7.2 Effect of sample size with regression block kriging

In case of regression block kriging, we need to fit RF models to represent the trend and apply residual block kriging. Below we copy R scripts from Chapter 5. Note that we use the same set of covariates as in Chapter 5 and the same residual variogram. The residuals themselves are not the same between different sample sizes, because each time the RF model is different and because each time we have different sample sizes and sampling locations.

We begin with copying the residual variogram parameters from Chapter 5 and loading the covariate maps.

Load Regression Kriging (RK) variogram parameters from Section 5.2

```
# load regression kriging variogram parameters from Chapter 5
RK <- readRDS(file.path(outputDir, "RK_params.rds"))
```

Load covariates

```
# load covariates prepared in Chapter 5
covars <- rast(file.path(outputDir, "stack2.tif"))
```

```
# sanity check
# declare categorical rasters are factors
cat_covars <- c("grondwater_2018_25m",
                "paleogeografie1500nc_25m")
# declare as factors
covars[[cat_covars]] <- as.factor(covars[[cat_covars]])
```

```
# extract covariate names
covars_names <- names(covars)
```

```
# construct data for `predict()`
# convert SpatRaster to data.frame
covars_df <- as.data.frame(covars, xy = TRUE, na.rm = TRUE)
```

We next fit the RF model for all sample sizes and store the RF prediction grids.

Global mean Random Forest

```
# warning! It might take a while to run
# below there is a code chunk to load the df_rf from the outputDir
```

```
# create data frame to save RF prediction per sample size
df_rf <- data.frame(sample_size)
df_rf["RF_pred"] <- NA

for (i in seq_along(random_sample_list)){
  # extract random sample from the list
  random_sample <- random_sample_list[[i]]
  # print message to inform about progress
  print(paste0("Running RF for sample size: ", nrow(random_sample)))

  # extract covariate values at sampling locations
  rgmx <- as.data.frame(extract(covars, random_sample, bind = TRUE))
  # build formula for ranger
  mod_for <- build_formula_fn(colnames(rgmx), covars_names)
  # fit Random Forest model
  rf_model <- ranger(formula = mod_for, data = rgmx,
                     seed = 987 + i, # vary per iteration, but deterministic
                     num.threads = 1) # optional
  # run Random Forest using ranger
  prd <- predict(rf_model, data = covars_df, type = "response")

  # compile predictions
  p <- data.frame(x = covars_df$x, y = covars_df$y, pred = prd$predictions)
  # make spatial
  RF_pred <- rast(as.matrix(p), type = "xyz", crs = crs(covars))
  # calculate the mean soc stock across the non-NA pixels (ton/ha)
  RF_mean_stock <- global(RF_pred, fun = "mean", na.rm=TRUE)[1, "mean"]
  # save RF prediction per iteration [ton/ha]
  df_rf[i, "RF_pred"] <- RF_mean_stock * area_ha
}
```

```
# optional: load df_rf
#df_rf <- readRDS(file.path(outputDir, "df_rf.rds"))
```

We extract residuals by subtracting the RF predictions from the observations and apply residual block kriging using the residual variogram from Section 5.2. We compute and store the regression kriging prediction, the associated lower and upper prediction interval limits, and the PIW per sample size.

Note that in the script below we use 10-fold cross-validation instead of leave-one-out cross-validation, as was done in Chapter 5. We did this to speed up calculations since leave-one-out cross-validation would take too long for the larger sample sizes.

Residuals from varying sample size in Random Forest

```
# warning! It might take a while to run
# below there is a code chunk to load the residuals_list from the outputDir

# initialize list to save the residuals per iteration
residuals_list <- vector(mode = "list", length = length(sample_size))
names(residuals_list) <- paste0("n", sample_size)
```

```
for (i in seq_along(random_sample_list)){
  # extract random sample from the list
  random_sample <- random_sample_list[[i]]
  # sample size per iteration
  sample_size_i <- nrow(random_sample)
  # print message to inform about progress
  print(paste0("Calculating residuals for sample size: ", sample_size_i))

  # generate a regression matrix
  rgmx <- as.data.frame(extract(covars, random_sample, xy = TRUE, bind = TRUE))

  # build formula. Notice that, since we are using the same covariates,
  # the formula will be the same for every iteration
  mod_for <- build_formula_fn(colnames(rgmx), covars_names)

  # number of folds for sub-setting the dataset
  nrFolds <- 10 #sample_size_i # leave-one-out would take too long
  nrow_rgmx <- nrow(rgmx)
  # create folds
  folds_df <- create_folds_fn(nrFolds, nrow_rgmx)
  # order by row index, select the folds column and
  # add it to the regression matrix
  rgmx["fold"] <- folds_df[order(folds_df$subsets), "which"]
  # we can keep the index too
  rgmx["id"] <- folds_df[order(folds_df$subsets), "subsets"]
  # create an empty column in the regression matrix to store the predictions
  rgmx["pred"] <- NA

  # fit the model for each hold-out fold
  for (k in seq_len(nrFolds)) {
    # subset train and leave-out sets
    train <- subset(rgmx, fold != k)
    leave_out <- subset(rgmx, fold == k)
    # fit Random Forest model
    rf_model_k <- ranger(formula = mod_for, data = train)
    # predict the hold-out fold
    prd_k <- predict(rf_model_k, data = leave_out)$predictions
    # assign predictions back using id match
    rgmx$pred[match(leave_out$id, rgmx$id)] <- prd_k
  }

  # compute and save the residuals
  rgmx["residuals"] <- rgmx[soc_values] - rgmx["pred"]
  residuals_list[[i]] <- rgmx[c("x", "y", "residuals")]
}
```

```
# optional: load residuals_list
#residuals_list <- readRDS(file.path(outputDir, "residuals_list.rds"))
```

Varying sample size in regression kriging

```
# use variogram from Chapter 5: Regression Kriging (RK)
svg_rk <- vgm(nugget = RK["nugget"], psill = RK["sill"],
              range = RK["range"], model = "Exp")
```

```
# warning! It might take a while to run
# below there is a code chunk to load the df_rk from the outputDir
```

```
# create a data frame to store values per iteration
df_rk <- data.frame(matrix(nrow = 0, ncol = 4))
colnames(df_rk) <- c("RK_pred","lower_limit", "upper_limit", "PIW")

# apply ordinary block kriging to the residuals
for (i in seq_along(residuals_list)){

  # convert dataframe to sf object
  residuals_sf <- st_as_sf(residuals_list[[i]],
                           coords = c("x", "y"), crs = crs(soc))

  # print message to inform about progress
  print(paste0("Running RK for sample size: ", nrow(residuals_sf)))

  # ordinary block kriging
  block_krig <- krige(formula = residuals ~ 1,
                      locations = residuals_sf,
                      newdata = block_centroid,
                      model = svg_rk,
                      block = grid_relative)

  # extract residual prediction and standard deviation
  block_pred <- block_krig$var1.pred
  block_sd <- sqrt(block_krig$var1.var)

  # get the RF predicted mean per sample size
  RF_pred_mean <- df_rf[i, "RF_pred"]

  # compute the RK predicted mean per sample size
  RK_pred_mean <- RF_pred_mean + (block_pred*area_ha)

  # compute the lower and upper limit of the 95% prediction interval
  lower_limit <- RK_pred_mean + qnorm(0.025)*block_sd*area_ha
  upper_limit <- RK_pred_mean +  qnorm(0.975)*block_sd*area_ha

  # calculate prediction interval width
  PIW <- upper_limit - lower_limit

  # save mean, lower and upper limits and PIW per iteration
  df_rk[i,] <- c(RK_pred_mean, lower_limit, upper_limit, PIW)
}

# add RF prediction
df_rk <- cbind(df_rf, df_rk)
```

```
# optional: load df_rk
#df_rk <- readRDS(file.path(outputDir, "df_rk.rds"))
```

Sample size vs total SOC stock (RK)

```
# n vs SOC (predicted mean with 95% PI, and true value)
y_range <- range(df_rk$lower_limit, df_rk$upper_limit, total_stock)

ggplot(df_rk, aes(x = sample_size)) +
  geom_errorbar(aes(ymin = lower_limit, ymax = upper_limit,
                    color = "RK predicted mean (95% PI)"),
                width = 20) +
  geom_point(aes(y = RK_pred, color = "RK predicted mean (95% PI)"),
             size = 2) +
  geom_hline(aes(yintercept = total_stock, color = "True SOC stock"),
             linetype = "dashed") +
  scale_color_manual(name = NULL,
                     values = c("RK predicted mean (95% PI)" = "steelblue",
                                "True SOC stock" = "black")) +
  scale_x_continuous(breaks = sample_size) +
  scale_y_continuous(breaks = pretty(y_range, n = 8)) +
  labs(x = "Sample size n", y = "Total SOC stock [ton]") +
  theme_bw() +
  theme(legend.position = c(0.74, 0.86),
        legend.background = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1))
```

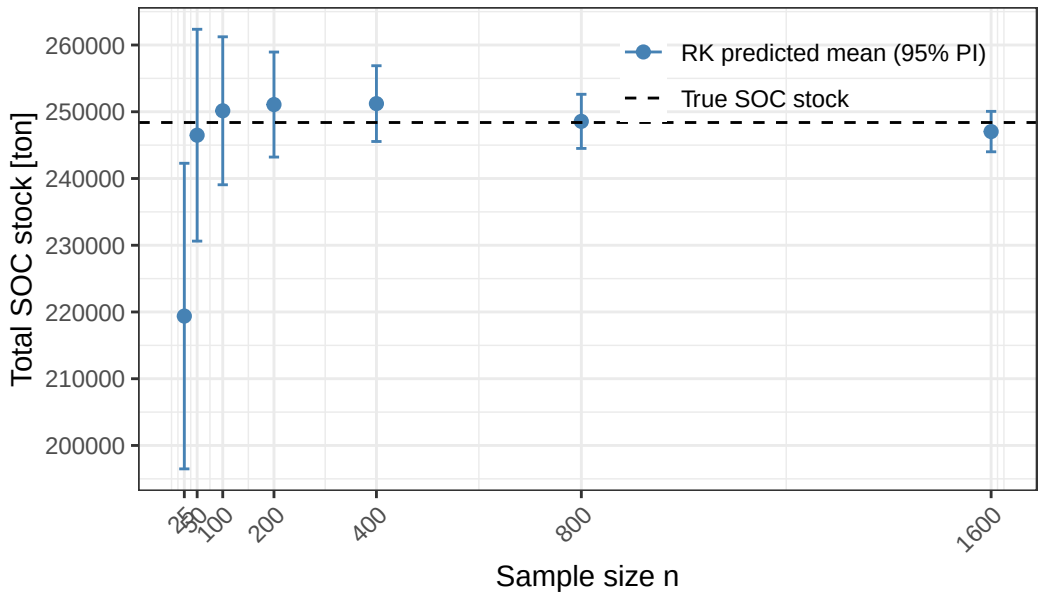


Figure 7.3: Total SOC stock prediction and 95% prediction interval against sample size, regression block kriging case.

We plot the regression block kriging PIW together with the ordinary block kriging PIW in one figure.

Comparison OK and RK: sample size vs PIW

```
# common y-range across both series
y_range <- range(0, df_ok[["PIW"]], df_rk[["PIW"]])

par(cex.lab = 0.8, mar = c(4, 5, 1, 2)) # margins c(bottom, left, top, right)
plot(sample_size, df_ok[["PIW"]], type = "b", pch = 16,
     col = "blue", cex = 1, main = NULL,
     xlab = "Sample size n",
     ylab = "Prediction Interval Width (PIW)",
     xaxt = "n", yaxt = "n",
     xlim = c(0, max(sample_size)),
     ylim = y_range)
lines(sample_size, df_rk[["PIW"]], type = "b", pch = 16,
      col = "red", cex = 1) # add RK series
# add actual values of n to the x-axis
axis(1, at = c(0, sample_size), labels = c(0, sample_size), cex.axis = 0.7)
# add a merged, non-overlapping set of tick labels for both PIW series
y_ticks <- c(0, sort(unique(round(c(df_ok[["PIW"]], df_rk[["PIW"]])))))
axis(2, at = y_ticks, labels = y_ticks, las = 1, cex.axis = 0.7)
legend("topright", legend = c("OK", "RK"),
       col = c("blue", "red"), lwd = 2, pch = 16, cex = 0.8, bty = "n")
```

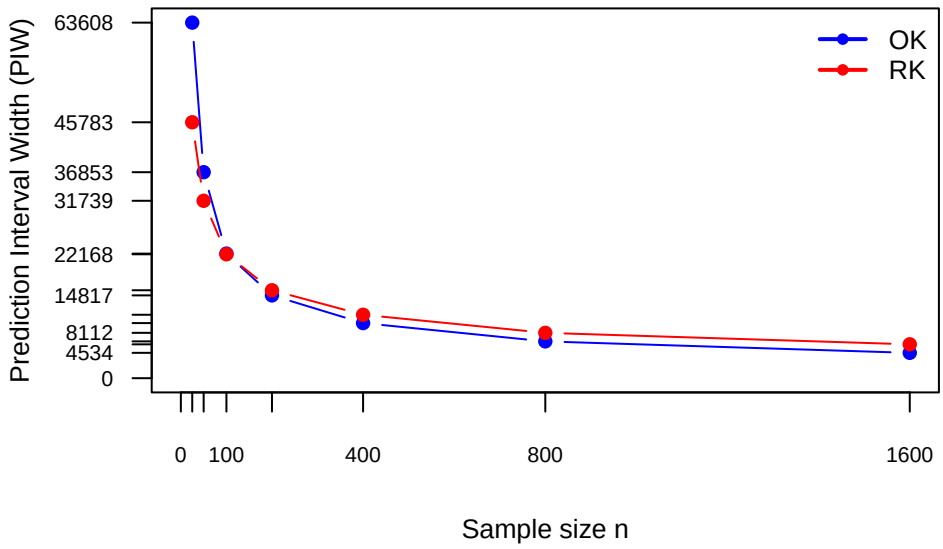


Figure 7.4: Ordinary block kriging (OK) and regression block kriging (RK) PIW against sample size.

Figure 7.4 shows that RK outperforms OK for smaller sample sizes, while OK has higher accuracy for larger sample sizes. We had noted already in Chapter 5 that for $n = 200$ RK was slightly worse than OK, which was explained by higher semivariance values at short distances for the residual variogram (see Figure 5.3). This effect remains if we further increase the sample size, because with an increased sample size we have even shorter distances between observation and prediction locations in the example area.

The narrower PIW for RK than for OK with small sample sizes is in some sense a relief: at least for these sample sizes we find that prediction of the total SOC stock in the example area benefits from covariates. Again, this result can be explained from Figure 5.3, because at larger distances the RK residual variogram is considerably smaller than the OK variogram. It also makes sense intuitively: with smaller sample sizes the SOC stock observations provide less information, so that the covariates become a more important source of information. The accuracy gain is largest for the smallest sample size, when $n = 25$, where the PIW decreased with 31.8 %.

In this chapter we only looked at the effect of sample size on the OK and RK prediction accuracy, but it is important to note that the spatial configuration of the sampling locations will also have an effect. We used random sampling, which is known to be sub-optimal for kriging. Kriging tends to be more accurate if the observations are more uniformly spread out over the area of interest, and slightly pushed to the study area boundaries, so that there is less spatial extrapolation. In case of RK it is also important to pay attention to the distribution of the observations in the `feature space`. And to make things even more complicated, often we also need the observations to estimate the variogram, which requires sufficient short-distance comparisons. Spatial sampling design optimisation is beyond the scope of this tutorial, instead we refer to Groenigen, Siderius, & Stein (1999), Webster & Oliver (2007), Brus & Heuvelink (2007) and Alexandre M. J-C. Wadoux, Brus, & Heuvelink (2019) for more information.

7.3 Comparison with design-based SOC stock estimation

Before we close this chapter it is also interesting to compare the prediction accuracies of kriging with those derived using statistical sampling theory, in the companion tutorial Sampling theory for estimating soil organic carbon change - a tutorial. Section 4 of that tutorial showed in its Figure 4.3 how the width of a 95% confidence interval depends on sample size, using simple random sampling and the same sample sizes as used in this tutorial. We copy the means of the confidence interval widths for each sample size from that tutorial in the figure below.

Load data from Figure 4.3 from the Sampling tutorial

```
# sampling tutorial results
df_st <- readRDS(file.path(inputDir, "df_st.rds"))
```

Comparison OK, RK and St: sample size vs PIW

```
# common y-range across all three series
y_range <- range(0, df_ok[["PIW"]], df_rk[["PIW"]], df_st[["PIW_median"]])

par(cex.lab = 0.8, mar = c(4, 5, 2, 2))
plot(sample_size, df_ok[["PIW"]], type = "b", pch = 16,
     col = "blue", cex = 1, main = NULL,
     xlab = "Sample size n",
     ylab = "Prediction Interval Width (PIW)",
     xaxt = "n", yaxt = "n",
     xlim = c(0, max(sample_size)),
     ylim = y_range)
lines(sample_size, df_rk[["PIW"]], type = "b", pch = 16,
      col = "red", cex = 1)
lines(df_st[["sample_size"]], df_st[["PIW_median"]], type = "b", pch = 16,
      col = "darkgreen", cex = 1)  # add Sampling tutorial series

axis(1, at = c(0, sample_size), labels = c(0, sample_size), cex.axis = 0.7)

y_ticks <- c(0, sort(unique(round(c(df_ok[["PIW"]],
                                    df_rk[["PIW"]],
                                    df_st[["PIW_median"]]))))))
axis(2, at = y_ticks, labels = y_ticks, las = 1, cex.axis = 0.7)

legend("topright", legend = c("OK", "RK", "Sampling tutorial"),
       col = c("blue", "red", "darkgreen"),
       lwd = 2, pch = 16, cex = 0.8, bty = "n")
```

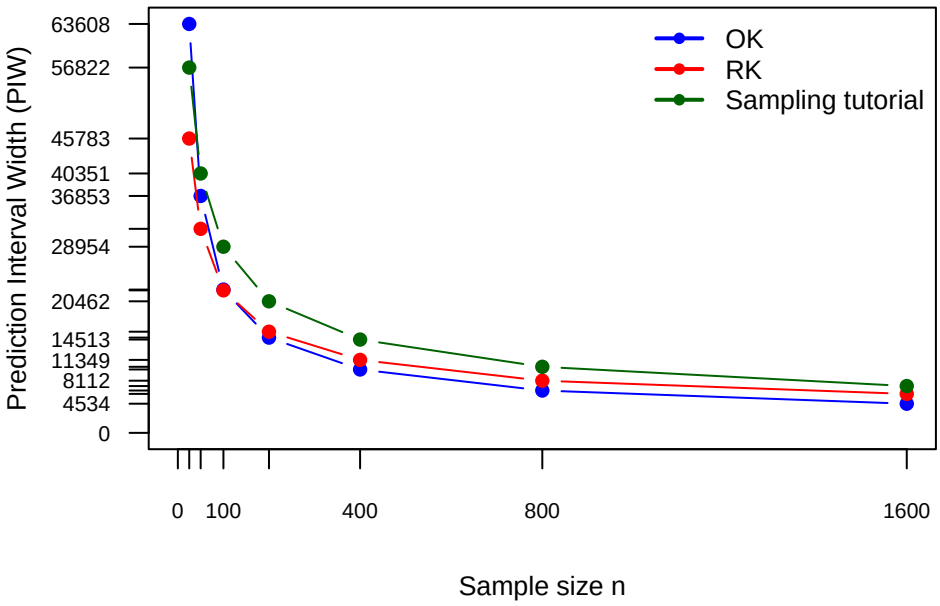


Figure 7.5: Prediction/confidence interval width against sample size for Ordinary Kriging (OK), Regression Kriging (RK) and the sampling theory tutorial.

We find that in almost all cases the `model-based` geostatistical approach has higher accuracy than the `design-based` sampling theory approach. For instance, the OK prediction interval for $n = 200$ is narrower than the sampling theory confidence interval for $n = 400$. However, the difference is smaller for larger sample sizes.

Results might turn out differently in other case studies. Also, we should not forget that the model-based prediction intervals are only valid under the assumptions made during geostatistical modelling, while sampling theory is completely 'model-free'. We should therefore not conclude from Figure 7.5 that geostatistics should always be preferred over sampling theory.

8 Conclusion

This tutorial explained the use of geostatistics for prediction of the SOC stock and SOC stock change in an area. We covered ordinary kriging, regression kriging and cokriging, and we extended all these kriging variants from point prediction to block prediction, thus accounting for a spatial **change of support**.

A key strength of geostatistics is that it also quantifies the prediction error, by means of a kriging standard deviation or a prediction interval, and that it can do so both at the point and block support.

We found that regression kriging did not outperform ordinary kriging, and that for SOC stock change prediction it is advantageous to revisit the same measurement locations (i.e., measure-remeasure), not only because it is more accurate but also because the geostatistical methodology is more straightforward.

As expected, prediction uncertainty decreased with increased sample sizes. Being able to quantify this is very useful because it allows soil carbon projects to calculate how large a sample of soil observations is needed to reach a desired prediction accuracy.

We also found that for the example area, for a given sample size, model-based geostatistical prediction reached higher accuracy levels than design-based sampling theory. This indicates that considerable savings in field and laboratory costs can be achieved, but please note that results might be different in other areas. The comparison might also turn out differently when using more advanced sampling methods and associated statistical inference. In fact, Chapter 7 in Sampling theory for estimating soil organic carbon change - a tutorial, also available in the ISRIC Resource Library, explained how considerable savings can be achieved by using composite soil sampling and merging soil samples from far away locations in design-based approaches. The conclusion is that model-based and design-based approaches are both valuable and that both should be carefully considered when choosing a method for a specific project.

This tutorial was not meant to be a geostatistics tutorial (for that we refer to Geostatistics for soil mapping), but looking back we did cover quite a bit of geostatistics in previous chapters, as well as some machine learning for digital soil mapping (with many more details provided in Machine Learning for Digital Soil Mapping).

Nonetheless, there is much more to geostatistics than what was covered in this tutorial. Readers who wish to dig deeper into the topic are advised to consult geostatistics text books, such as Webster & Oliver (2007), Wackernagel (2003) or Wikle, Zammit-Mangion, & Cressie (2019). If you are specifically interested in the application of geostatistics to SOC stock prediction, some

relevant publications are Nussbaum, Papritz, Baltensweiler, & Walthert (2014), Kempen et al. (2019), Szatmári et al. (2021) and A. M. J.-C. Wadoux et al. (2026).

Readers who would like to learn more about the measurement, modelling and prediction of SOC stock changes in the context of Monitoring, Reporting and Verification (MRV) are invited to read Batjes et al. (2024), Smith et al. (2020), Ihasusta et al. (2026) and visit the MRV4SOC Project (2026).

References

Batjes, N. H., Ceschia, E., Heuvelink, G. B. M., Demenois, J., Maire, G. le, Cardinael, R., … Egmond, F. van. (2024). Towards a modular, multi-ecosystem monitoring, reporting and verification (MRV) framework for soil organic carbon stock change assessment. *Carbon Management*, *15*(1), 2410812. https://doi.org/10.1080/17583004.2024.2410812

Brus, D. J., & Heuvelink, G. B. M. (2007). Optimization of sample patterns for universal kriging of environmental variables. *Geoderma*, *138*(1), 86–95. https://doi.org/https://doi.org/10.1016/j.geoderma.2006.10.016

Groenigen, J. W. van, Siderius, W., & Stein, A. (1999). Constrained optimisation of soil sampling for minimisation of the kriging variance. *Geoderma*, *87*(3), 239–259. https://doi.org/https://doi.org/10.1016/S0016-7061(98)00056-1

Helfenstein, A., Mulder, V. L., Hack-ten Broeke, M. J. D., Doorn, M. van, Teuling, K., Walvoort, D. J. J., & Heuvelink, G. B. M. (2024a). BIS-4D: Mapping soil properties and their uncertainties at 25 m resolution in the netherlands. *Earth System Science Data*, *16*(6), 2941–2970. https://doi.org/10.5194/essd-16-2941-2024

Helfenstein, A., Mulder, V. L., Hack-ten Broeke, M. J. D., Doorn, M. van, Teuling, K., Walvoort, D. J. J., & Heuvelink, G. B. M. (2024b). *Spatially explicit environmental variables at 25m resolution for spatial modelling in the Netherlands*. 4TU.ResearchData. https://doi.org/10.4121/6af610ed-9006-4ac5-b399-4795c2ac01ec.v3

Ihasusta, A., Bitar, A. A., Batjes, N. H., Egmond, F. van, Cardinael, R., Karunaratne, S., … Ceschia, E. (2026). Choosing the appropriate methodology to monitor soil organic carbon (SOC) in croplands: Aligning methods with evolving monitoring reporting verification (MRV) frameworks. *Carbon Management*, *17*(1), 2638317. https://doi.org/10.1080/17583004.2026.2638317

Kempen, B., Dalsgaard, S., Kaaya, A. K., Chamuya, N., Ruipérez-González, M., Pekkarinen, A., & Walsh, M. G. (2019). Mapping topsoil organic carbon concentrations and stocks for Tanzania. *Geoderma*, *337*, 164–180. https://doi.org/https://doi.org/10.1016/j.geoderma.2018.09.011

MRV4SOC Project. (2026). *MRV4SOC: Monitoring, reporting and verification of soil organic carbon and greenhouse gas balance*. The project official website. Retrieved from https://mrv4soc.eu/

Nussbaum, M., Papritz, A., Baltensweiler, A., & Walthert, L. (2014). Estimating soil organic carbon stocks of swiss forest soils by robust external-drift kriging. *Geoscientific Model Development*, *7*(3), 1197–1210. https://doi.org/10.5194/gmd-7-1197-2014

Smith, P., Soussana, J.-F., Angers, D., Schipper, L., Chenu, C., Rasse, D. P., … Klumpp, K. (2020). How to measure, report and verify soil carbon change to realize the potential of

soil carbon sequestration for atmospheric greenhouse gas removal. *Global Change Biology*, *26*(1), 219–241. https://doi.org/10.1111/gcb.14815

Szatmári, G., Pásztor, L., & Heuvelink, G. B. M. (2021). Estimating soil organic carbon stock change at multiple scales using machine learning and multivariate geostatistics. *Geoderma*, *403*, 115356. https://doi.org/https://doi.org/10.1016/j.geoderma.2021.115356

Wackernagel, H. (2003). *Multivariate Geostatistics: An Introduction with Applications* (3rd ed.). Berlin, Heidelberg: Springer-Verlag. https://doi.org/10.1007/978-3-662-05294-5

Wadoux, Alexandre M. J-C., Brus, D. J., & Heuvelink, G. B. M. (2019). Sampling design optimization for soil mapping with random forest. *Geoderma*, *355*, 113913. https://doi.org/https://doi.org/10.1016/j.geoderma.2019.113913

Wadoux, A. M. J.-C., Donovan, M., Fu, P., Leach, N. R., Maddalena, J., Rustowicz, R., … Kellner, J. R. (2026). A digital soil mapping approach to soil carbon monitoring, reporting and verification (MRV). *npj Sustainable Agriculture*, *4*(1), 58. https://doi.org/10.1038/s44264-026-00125-0

Webster, R., & Oliver, M. A. (2007). *Geostatistics for Environmental Scientists* (2nd ed.). Chichester, UK: John Wiley & Sons. https://doi.org/10.1002/9780470517277

Wikle, C. K., Zammit-Mangion, A., & Cressie, N. (2019). *Spatio-Temporal Statistics with R*. Boca Raton, FL: Chapman & Hall/CRC. Retrieved from https://spacetimewithr.org/Spatio-Temporal/%20Statistics/%20with/%20R.pdf

Wright, M. N., & Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, *77*(1), 1–17. https://doi.org/10.18637/jss.v077.i01

A Running the tutorial in R

A.1 Materials

The chapters of this tutorial can be accessed via: (1) the links in the left-side menu in the HTML version, or (2) the links in the table of contents in the PDF version. Each chapter that includes code chunks has its own `R` script file.

The scripts are made available with this tutorial in a dedicated folder on ISRIC's file service and can be accessed here. The folder contains the zip file `soc_stock_kriging.zip` that includes all materials. Download this folder, save it in a dedicated folder on your computer and extract the zip file from there. This will create a folder named `soc_stock_kriging` that includes a set of sub-folders and various files.

IMPORTANT:

1. Please carefully read the `README` file that explains the folder structure and what can be found within these folders.
2. Do **not** change any of the folder or file names, including the name of the 'parent folder' `soc_stock_kriging`. Changing folder and file names might cause the code stop running and produce errors.
3. Do **not** change any code. The entire code should execute without errors. Changing the code might cause the code to stop running and produce errors. If you wish to experiment with the code (which we encourage!), please do so in a copy of the code file. The only code line you need to modify is the file path in the `setwd()` function (further explanation below).

A.2 RStudio and required R packages

You can open the scripts in a dedicated software that provides a more user-friendly Graphical User Interface (GUI), such as RStudio, and run the code from there. Hence, it is not necessary to copy the code from this tutorial to `R`.

R can be downloaded here www.r-project.org.

RStudio can be downloaded here download/rstudio-desktop.

The required R packages are:

- **terra**: spatial data analysis
- **sf**: vector (point, line and polygon) data handling
- **gstat**: spatial and spatio-temporal geostatistical modelling, prediction and simulation
- **stars**: spatiotemporal arrays, raster and vector data cubes
- **ggplot2**: create elegant data visualisations using the grammar of graphics
- **patchwork**: composer of plots (only used for plotting side-by-side Figure 4.7)
- **cvTools**: cross-validation tools for regression models
- **ranger**: fast implementation of the random forest machine learning algorithm

In case a package is not installed, the `install.packages()` function can be used to download and install it.

```
# To check if a package is installed
system.file(package = 'terra')

# it prints the directory where the package is installed
# if the package is not installed, it prints an empty string ''

# To install a package
install.packages('terra')
```

Alternatively, packages can be installed using specific options in Rstudio, in case it is used to run the R code. Instructions can be found here.

A.3 Setting the working directory

To be able to execute the code you have to tell R where it can find the materials on your computer. In other words, you will have to set the **working directory**.

This can be done with the `setwd()` function. Within the `()` you will define the file path to the soc_stock_kriging folder, unzipped from the `soc_stock_kriging.zip` that you have downloaded.

As an example, suppose you created a folder called `tutorials` on the C-drive of a Windows computer and saved and unzipped the downloaded `soc_stock_kriging` files here.

Then the working directory can be set as follows:

```
# modify this file path with the location where you saved the tutorial files
# (i.e. that contains the 'soc_stock_kriging' folder
setwd("C:/tutorials/soc_stock_kriging")
workingDir <- getwd()
```

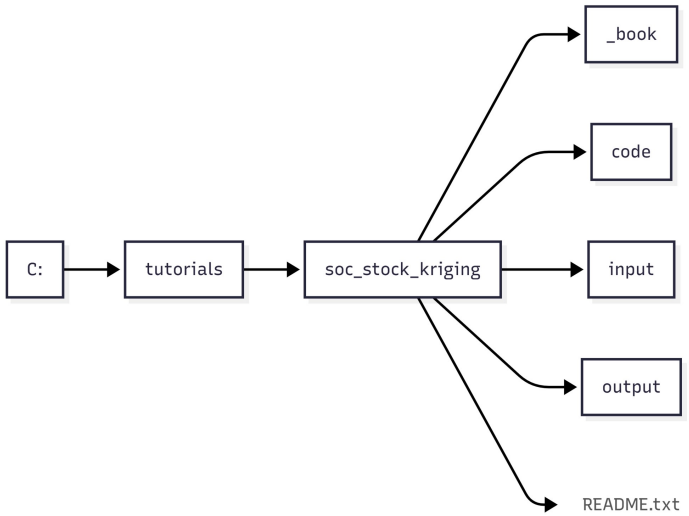


Figure A.1: Folder structure.

Note that quotes must be used to define the file path.

IMPORTANT: the code line with the `setwd()` function is the *only* code line you need to modify in each of the scripts.

Alternatively, for RStudio users, the working directory can be set using specific options in the GUI. Instructions can be found here.

The working directory is set at the beginning of each chapter of this tutorial. The tutorial assumes that this is `C:\tutorials`, but obviously this can be changed to any preferred folder.

In case R returns an error after running the `setwd()` function, you probably have an error in the file path definition: check and correct.