|  |
| --- |
| R-Markdown  SW Tool for Radiological Mapping |

NSIL, IAEA

Version 2.2

Updated: 4.10.2024

Last verified software versions:

*R-4.4.1-win.exe*

*RStudio-2024.09.0-375.exe*

*rtools43-5550-5548.exe*

*jre-8u421-windows-x64.exe*

# Installation

# Installation instructions on Windows

## Install R in your computer

<https://cran.r-project.org/>

Select base and Download R-XXX for Windows

## Install R-studio (Last version)

https://www.posit.co

Choose the latest version to install from the list in the page.

## **Install R-tools**

<https://cran.r-project.org/bin/windows/Rtools/>

## Install JRE (Java Runtime Environment)

required by the "OpenStreetMap" package

<https://www.java.com/en/download/>

## Create an R folder

* Create a subfolder 'data' in your R folder and store in it your Excel data files
* Copy the following packages into you 'R' folder in 'documents'

*packages-installation.Rmd*

*surveys.Rmd*

## [Optional] Activate the "renv" package manager in RStudio

The "renv" option allows to install R packages in isolated environments for each individual projects. It helps ensuring the reproducibility of the environment and in particular possible competing package versions between projects.

In main menu:

a. "Tools > Project Options", Tab "Environments"

b. Check the "Use renv with this project"

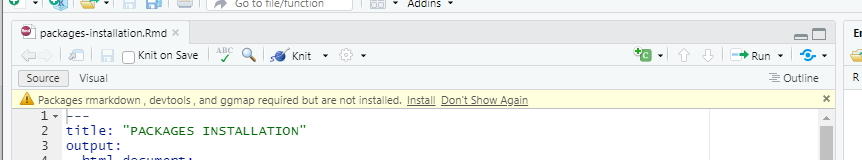
c. And click "OK"

## Start R Studio

NOT NEEDED IN THE NEW VERSION:

Open the Rmd file *packages-installation.Rmd*

Run it to install all the required packages. Install also markdown, devtools and ggmap, if needed:



it will take a few minutes, be patient.

You can make your first run using the provided surveys.Rmd

# Introduction

## Interpretation of GIS-referenced in-situ radiation measurements

The interpretation of GIS-referenced in-situ radiation measurements is carried out through five "typical" steps:

1. data loading;

2. data cleaning and pre-processing (including Geo-processing);

3. Exploratory Data Analysis (EDA - visualize, summary statistics, getting acquainted with the dataset involved);

4. spatial estimation (exceedance probability, ...);

5. results dissemination.

Each of these steps will be detailed and exemplified in the following chapters.

## Structure of the Rmd file

This Rmd has been developed for IAEA to support the production of maps depicting surface distribution of radioactive contamination in different scenarios.

The file consists of three sections:

1. Definition of parameters
2. Read file
3. Get and sort data from more sources (detectors)
4. Process sorted data
5. Show and draw data
6. Print maps of the walks
7. Interpolation
8. Show maps with the interpolated values
9. Print final maps

It is recommended to keep a copy of this file, and to create new copies for each case of study, in which you can remove the explanations and the sections of code (chunks) that are not needed for your particular case.

# `R` free software environment

`R` free software environment for statistical computing and graphics has been used by a large community of Statisticians and data analysts for decades. `R`'s killer feature is its extremely rich ecosystem of library/packages allowing to perform (among others): Statistical analysis, modelling, data cleaning, data transformation & visualization, machine learning, deep learning, GIS, Geospatial Analysis, ...

Having to learn and use `R` programming language may look daunting for people used to "point & click" user interfaces but we do believe that the learning experience might be much pleasant and approachable than you first imagined and that it will pay off in the long term.

Here below some motivations:

\* using `R` to script (a sequence of data analysis steps - workflow) your analysis pipeline is definitely much easier than developing a full software or complex library;

\* `R` ecosystem includes packages like the `tydiverse` suite and in particular `dplyr`, `sf`, ... allowing to implement your "ideal" data analysis pipeline in an extremely intuitive and seamless manner;

\* `RStudio` \*\*IDE\*\* (\*\*I\*\*ntegrated \*\*D\*\*evelopment \*\*E\*\*nvironment) offers an ergonomic interface where you can seamlessly code, experiment, debug your data analysis, access packages documentation, experiment different scenarios, visualize your analysis outputs, disseminate your results, ...

\* there is a huge amount of extremely valuable and free learning resources online;

\* you will be able to implement reproducible research http://reproducibleresearch.net/ workflows: access, pre-process, analyse and visualize your data, document and publish your outcomes in a reproducible way.

In our particular use case, we will tap into `R` packages particularly relevant to the tasks at hand. It includes:

\* `dplyr` https://dplyr.tidyverse.org/ for data loading, cleaning, selection, ...

\* `ggplot2` https://ggplot2.tidyverse.org/ for data visualization

\* `mapview`https://r-spatial.github.io/mapview/ for interactive data geo-visualization

\* `sf` https://r-spatial.github.io/sf/ for GIS-like features (reprojecting, clipping, and many more)

\* `gstat` https://cran.r-project.org/web/packages/gstat/vignettes/gstat.pdf for Spatial Interpolation (Geostatistics, ...)

\* ...

## How to best use this "template" data analysis workflow

This tutorial targets two distinct types of audience with different objectives:

- \*\*researchers, engineers\*\*, ... willing to \*\*develop thorough data analysis skills\*\* and use them on a regular basis: execute, modify code snippets provided and explore/study further resources and supplemental material pointed out;

- \*\*decision makers\*\* willing to \*\*understand typical data analysis workflows\*\* and to \*\*assess toolbox's suitability and potential\*\* without delving into the programming details: understand the rationale and sequence of a typical workflow; [optionally] tweack code snippets, model parameters, visualizations, ...

## RStudio and R markdown crash course

\*\*RStudio\*\* is a free and open-source Integrated Development Environment (IDE) for R. In a nutshell, it is a single entry point to a wide range of features and functionalities allowing to perform data loading, transformation, analysis, modelling and visualization. Additionally, used in combination with RMarkdown package, it allows to document and publish your data analysis pipeline/workflow efficiently, thus ensuring reproducibility and fostering collaboration.

RStudio IDE, RMarkdown and R packages in general are extremely well documented. In particular, it is highly recommended to get access to the series of \*\*RStudio Cheat Sheets\*\* available here: https://www.rstudio.com/resources/cheatsheets. For now the two most important one are: \*\*RStudio IDE Cheat Sheet\*\*, \*\*R Markdown Cheat Sheet\*\*.

For further reference, take a look as well at: http://r4ds.had.co.nz/r-markdown.html#r-markdown-basics

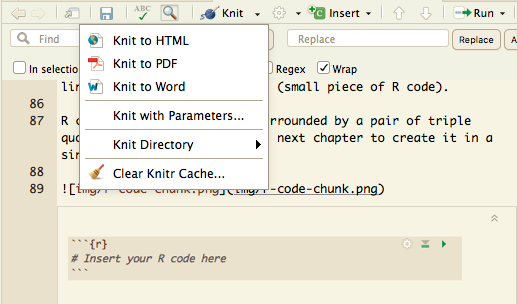
### Inserting and executing code "chunks" in RMarkdown

A RMarkdown document is a mix of formatted text, images, links, graphs and code chunks (small piece of R code).

R code chunks are inserted surrounded by a pair of triple quote as follows (look at the next chapter to create it in a single keyboard stroke):



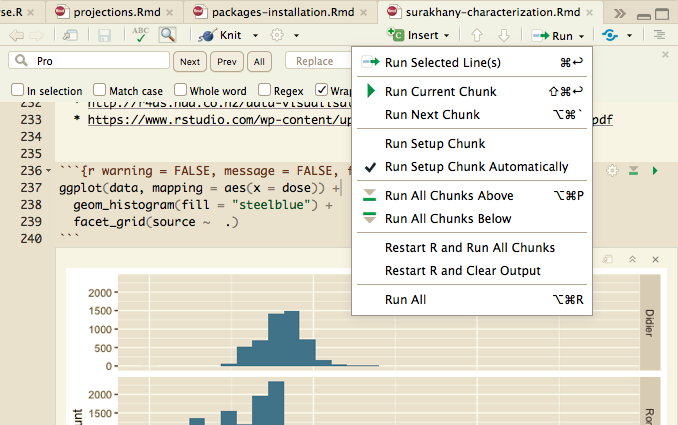
Then the whole RMarkdown document can be "Knitted" to html or whatever format you find relevant through RStudio User Interface (or associated keyboard shortcut - see next chapter).



It is important to note that each chunk can be executed individually and sequentially by clicking the green left arrow at the top right of a code chunk. This way, before generating/knitting the whole RMarkdown document, you can experiment/debug it.

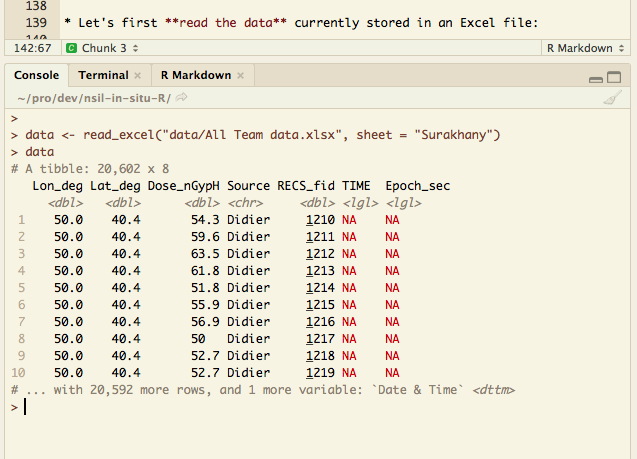
It is \*\*important to remember\*\* than code chunks must be executed from top to bottom sequentially as later chunks will depend on calculations of previous ones.

Rstudio "Run" menu accessible at the top of a RMarkdown document is as well extremely convenient as it allows to execute all chunks above, below the current selected one



Pro tip: You can as well select any line in a chunk and execute it by pressing "Shift + Enter" keyboard keys.

Pro tip: Last, you can experiment any piece of code in the R console itself. This is an ideal way to interactively analyse, visualize your data, debug your code, ...



### Some useful keyboard shortcuts

There are dozens of keyboard shortcuts that would improve your user experience and productivity. Many of them are listed in RStudio "Help" menu. Below a curated list of extremely useful ones:

\*Note: "Cmd/Ctrl" means you should use "Cmd" key for a Mac keyboard and"Ctrl" for others\*

\* \*\*Cmd/Ctrl-Alt-I\*\*: Create code chunk

\* \*\*Cmd/Ctrl-Shift-K\*\*: "Knit" all document

\* \*\*Cmd/Ctrl-Shift-Enter\*\*: Executing single code chunk (you can use as well the green top-left arrow)

\* \*\*alt + -\*\*: <- (R variable assignment symbol)

\* \*\*Cmd/Ctrl-Shift-M\*\*: %>% (Piping commands - see below)

\* \*\*Cmd + alt + R\*\*: Run all

\* \*\*Cmd/Ctrl-Shift-C\*\*: Comment block selected/highlighted

# Exploratory Data Analysis

As nicely summarized by H. Wickham and G. Grolemund in their outstanding book "R for Data Science" (see reference below): Exploratory Data Analysis is an iterative cycle where you:

1. Generate questions about your data;

2. Search for answers by visualizing, transforming and modelling your data;

3. Use what you learn to refine your questions and/or generate new questions.

For further reference: http://r4ds.had.co.nz/exploratory-data-analysis.html (Accessible online)

In our specific case, we will load the dataset collected, clean it, look at summary statistics, transform it, look at values distribution, spot potential outliers, spatialize it, ... Our ultimate goal being to gain enough knowledge to inform and prepare the spatial estimation of doses in our area of interest.

## Definition of the parameters

In the first part the parameters are defined:

**OPEN\_WINDOWS = FALSE/TRUE**

define OPEN\_WINDOWS as TRUE if you want to have the results opened in separate windows while running the code. It is just opening the printed results in the files using the function viewer(…).

**crs <- 4326**

the Coordinate Reference System as described in 5.2.

Next, the source files with the code need to be listed. These files are then read in the part 2. Here, in the main file, we only make a list (array) of the files. Note, there need to be comma (,) between the items.

**source\_name <- c(**

**"inputs/file1.R"**

**,**

**"inputs/file2.R"**

**)**

This is the part that needs to be edited by the user to point to real files linked to data. Typically, there is only one file. Data from several sources from one location can be combined, as shown in the example above.

The input files include the name of the data file and parameters that are needed to be set up for a particular analysis.

A typical input file looks like:

# \\\\\\\\\\\\\\\\\\\\\ INPUT PARAMETERS \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

input <- data.frame(

**filename <- "inputfile.csv",** *# the name of the inputfile in .xlsx or .csv format*

*the files is supposed to be in the data directory*

*# columns to be read:*

**longitude <- "lon",** *# the columns “lon” in the csv/xlsx file will be used as longitude*

**latitude <- "lat",** *# the columns “lat” in the csv/xlsx file will be used as latitude*

**column <- "gamma\_cps",** *in the csv/xlsx file # the columns with the values to be used*

*Sometimes this value needs to be multiplied by some constant, for example if the value in pSv/s is multiplied by 3.6, the final value is in nSv/h.*

**constant <- 1,** *# the constant the value needs to be multiplied by*

*The following parameters are used in the* *titles or names of the output file*

**site <- "Seibersdorf Arktis UAV"**, *#this appears on the figures and printout file names*

**detector <- "Arktis",** *# for the name of the printout files*

**quantity <- "cps",** *#this appears on the figures*

**unitName <- "cps",** *# for the name of the printout files*

**utm = 32633,**  *# Universal Transverse Mercator system (32633 Vienna)*

**resolution <- 1,** *# grid resolution for the interpolation in [m]*

**subsetting <- 3** # the subsetting is used to produce a subset of data to make the visualization of the points easier if they are too many and overlap.

)

# /////////////////////////////////////////////////////////////////////

Parameters for the titles or names of the output file (Except for the detector that actually needs to be different for each source) , utm, resolution, and subsetting are expected to be identical in all input files; if not, only the last one is used.

## Data Loading & Transformation (ReaD File)

The first .R source is read first

**source(source\_name[1])**

with the data file listed as *input$filename*.

Data can come in a variety of formats and structures, our code recognizes the `.csv` and `.xlsx` formats. For the read\_excel function the `readxl` package is used, the read.csv is a base function.

Let's first read the data currently stored in an Excel file into the *data* array:

**if(endsWith(input$filename, ".xlsx")) {data <- read\_excel(paste0("data/",input$filename))}**

**if(endsWith(input$filename, ".csv")) {data <- read.csv(paste0("data/",input$filename))}**

Notice that some special characters, such as "-" cannot be used, in some functions... use "\_" instead in the Excel worksheet or Variable names.

Pro tip: From RStudio, the documentation of any function (usage, function parameters, returned value, ...) can be accessed by writing in RStudio console `?funcion\_name`, in our case: `?read\_excel`. For further reference: http://r4ds.had.co.nz/data-import.html

In the next step only the longitude, latitude and the column with the value of interest are selected and inserted in a ‘new’ data array:

**data <- data %>% rename(lon = input$longitude, lat = input$latitude, value = input$column) %>% select("lon","lat","value")**

Selecting columns, filtering rows, renaming column/field names, ... can be done very easily and intuitively using the `dplyr` package. A first "naive" approach could be to perform:

\* first the selection then storing the result;

\* then to perform the rows selection or renaming;

\* then storing the result;

\* ... till the end of your transformation/pipeline.

\*\*Important remark\*\*: The following line `data\_selected <- dplyr::select(data, 1:4)` could be simplified as: `data\_selected <- select(data, 1:4)`. Here we prefix `select` by `dplyr::` to explicitly specify that the `select` function we want is the one provided by the `dplyr` package. It helps avoiding name conflicts with other packages that might provide functions with the same name (in our case the `raster` package used later on).

You can perform as many operations as you want in one go using the `%>%` (pronounce "pipe") operator as follows (see curated RStudio shortcuts in the introduction):

For further reference: http://r4ds.had.co.nz/transform.html and <http://r4ds.had.co.nz/pipes.html>

Sometimes, the value is not in the unit that you want and you can multiply it by a constant defined in the input file. This can be useful if data from two detectors with different unit (e.g. nSv/h and mSv/h) are being combined.

**data$value <- data$value \* input$constant**

For the comparisons of the detectors, the name of the detector is also added in the array as *dataset*.

**data$dataset <- input$detector**

and the name of the detector is also used for the titles:

**detectorname <- input$detector**

## More Sources (Detectors)

The part 3 is activated only if there are more (than 1) source – or lines in the source\_name. The same procedure for reading the file is followed, the new data are then attached to the old data.

**data <- rbind(data,data\_temp)** # repeated with each dataset

and the new detector name is attached as well

**detectorname <- paste0(detectorname,"+",input$detector)** # for the detector name in the printouts

Now we will have a first look at the data

### Getting summary statistics

It's might be a good idea to take a quick look at the first and last rows of a just loaded dataset.

**head(data)**

**tail(data)**

\*Note: By default the first and last 5 rows are displayed but it can be further specified, for instance: `head(data, 10)`.\*

Use the command for the summary

**summary(data, digits=6)**

# notice that the length (digits) can be specified... we have selected 6 to have accurate estimates for median values of longitud and latitude!

### Getting counts per `dataset`

Look at the number of the counts, in our case, the source field can contain the name of the data collector, instrument, ...

data %>% count(dataset)

### Histogram of `value` per `source`

R has several systems for making graphs, but `ggplot2` is one of the most elegant and most versatile. `ggplot2` implements the \*\*grammar of graphics\*\*, a coherent system for describing and building graphs. With `ggplot2`, you can do more and faster by learning one system and applying it in many places.

# 1. first ggplot's layer: ggplot(data, mapping = aes(x = value)) specifying data

# and aesthetics mapping (here the dose variable to x positioning)

# 2. second layer: geom\_histogram(fill = "steelblue") specifying we want an

# histogram

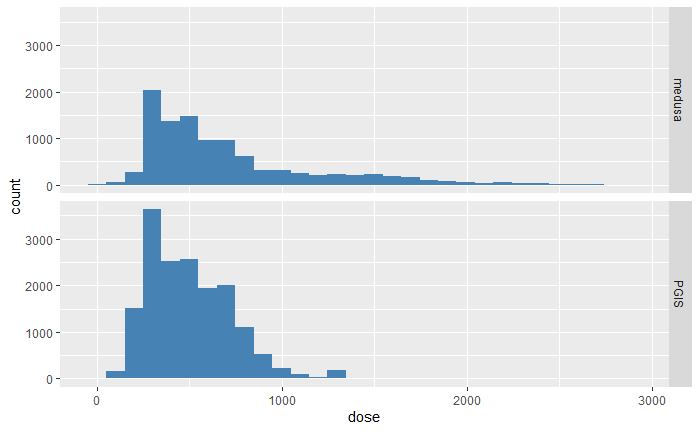
# 3. last layer: facet\_grid(source ~ .) specifying we want to breakdown the

# histogram by source (here name of data collector)

**ggplot(data, mapping = aes(x = value)) +**

**geom\_histogram(fill = "steelblue") +**

**facet\_grid(dataset ~ .)**



Again, data visualization is a wide topic and ggplot offers a lot of flexibility and expressive power. Hopefully, there are great resource materials available.

For further reference:

\* http://r4ds.had.co.nz/data-visualisation.html

\* https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

\* https://ggplot2-book.org (the Third version of this book is now available online)

### Quick value map

To quickly \*\*"spatialize"\*\* all data:

1. Base layer: pass data and map "lon" and "lat" variables respectively to x and y aesthetics

2. add a "point" layer and map "dose" variable to the color aesthetic with

alpha (opacity) to 0.8 (0 is transparent and 1 is opaque) and points size

3. specify a color palette. "direction = -1" reverses the color order

4. specity a map projection. Look at ?coord\_map in RStudio console for further details. Map projections will be covered more carefully later on.

# Same as previous but now "faceted"

**ggplot(data, mapping = aes(x = lon, y = lat)) +**

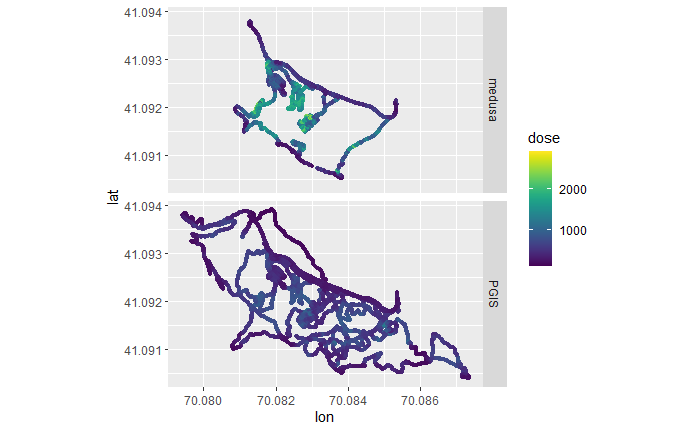
**geom\_point(aes(color = dose), alpha = .8, size = 1) +**

**scale\_color\_viridis(direction = 1) +**

**coord\_quickmap() +**

**facet\_grid(detector ~ .)**

and we get walk overviews:



## Process sorted data

Here we prepare our data in various formats for later use.

### Bin Spacing and Subsetting

Some parameters are needed later at some maps.

Space between white isolines is defined from the difference between the maximum and minimum value:

**signif(((data %>% pull(value) %>% max()) - (data %>% pull(value) %>% min()))/2,1) -> ISObinwidth**

The data in some graphs can overlap, so we reduce them. We do it by subsetting a dataframe by index of rows to be selected by factor that is defined by subsetting (default 5 defined in the beginning).

**data\_sub <- data[sample(1:nrow(data), (nrow(data)/subsetting)),]**

### Coordinate Reference System

To be able to use the a "sf" package, we need to convert our subset to a "sf" package "compatible" object with associated crs (Coordinate Reference System) and specifying "lon" and "lat" columns in original dataset.

WGS84 (EPSG: 4326) is commonly used by organizations that provide GIS data for the entire globe or many countries. CRS is used by Google Earth and also in our project. The value of crs is defined as 4326 in the beginning.

For further reference on Coordinated Reference Systems and Map projections, you can refer to the document titled "6-understanding-coordinate-reference-system.pdf" under the `/manuals` folder. This document is part of a wider course on GIS developed for the Incident & Emergency Center of the IAEA.

For further reference on `sf` package: https://r-spatial.github.io/sf/articles/sf1.html

**data\_sf\_all <- st\_as\_sf(data, coords = c("lon", "lat"), crs = crs)**

**data\_sf\_sub <- st\_as\_sf(data\_sub, coords = c("lon", "lat"), crs = crs)**

### Duplicated locations

Duplicated location can be reduced by

**d = st\_is\_within\_distance(data\_sf\_all, dist=0.01)**

**dupl = unlist(mapply(function(x,y) x[x < y], d, seq\_along(d)))**

**data\_sf <- data\_sf\_all[-dupl, ]**

Based on <https://github.com/r-spatial/sf/issues/669>

### Universal Transverse Mercator coordinate system

Some of the map routines use the UTM.

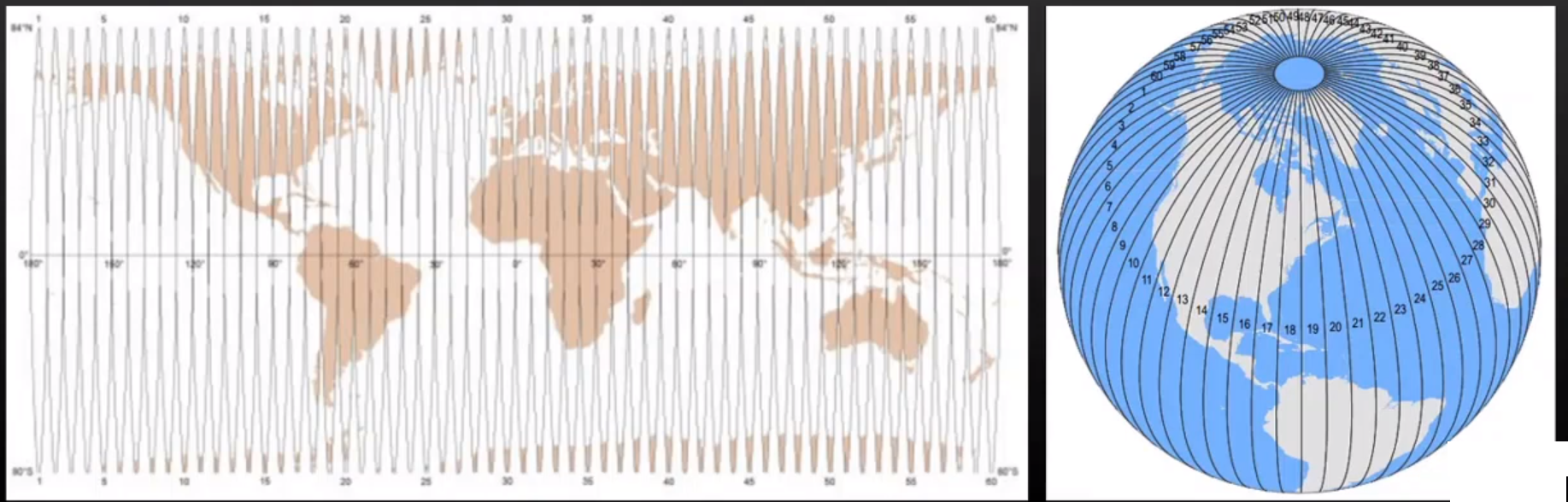
https://www.wikiwand.com/en/Universal\_Transverse\_Mercator\_coordinate\_system

Worldwide UTM zones (WGS84 based) are numbered 326 + zone number for the northern hemisphere and 327 + zone number for us in the south.

There are webpages to find your utm <https://www.geoplaner.com/>

Vienna UTM is 32633. The utm value needs to be defined in the input file. If more input files are defined, the last utm is used.

**data\_utm <- st\_transform(data\_sf, crs = input$utm)**



### Perimeters, boxes, grids

# Produce perimeters and bboxes

**site\_perimeter\_sf <- st\_convex\_hull(st\_union(data\_sf))**

**bbox\_sf <- st\_bbox(data\_sf)**

# Generate grids covering the bounding box and cropped by the site's perimeter.

**bbox\_sf %>%**

**st\_as\_stars(dx = resolution, dy = resolution) %>%**

**st\_crop(site\_perimeter\_sf) -> grd\_sf**

## Show and draw data and print the maps

Data:

* data - raw data
* data\_sf - data in the format of the sf library
* data\_utm - data in the utm format of the sf library

Plotting tools:

* plot - data format: data, data\_sf, data\_utm
* ggplot - data format: data, data\_sf, data\_utm
* mapview - data format: data\_sf, data\_utm
* OpenStreetMap - data format: data

OpenStreetMap:

By default the Coordinate Reference System (CRS) of web maps is "Google Mercator" (CRS: 3785). We have re-projected it to WGS84 above to make it compatible with our dataset\*

The base map `type` available is listed above:

"osm", "osm-bw",

"maptoolkit-topo", "waze", "bing", "stamen-toner", "stamen-terrain",

"stamen-watercolor", "osm-german", "osm-wanderreitkarte", "mapbox", "esri",

"esri-topo", "nps", "apple-iphoto", "skobbler", "hillshade", "opencyclemap",

"osm-transport", "osm-public-transport", "osm-bbike", "osm-bbike-german"

The list above comes from the documentation of the `openmap` function accessed through the following command to be typed in the console: `?openmap`. Feel free to choose the base layer of your choicer. However, note that at this geographical scale, such layer is not as informative as the Google Satellite imagery.

The various ways how to plot the data, including the walks and posted over maps, are shown in the Chapter 4. The Chapter 5 shows how to print the map in a file:

**ggsave("printouts/dose-rate-above-threshold.jpg", dpi=600)**

## Printing

We found the *OpenStreetMap* package to be ideal for printing:

**my\_osmmap <- openmap(c(bbox\_sf$ymax+0.2\*(bbox\_sf$ymax-bbox\_sf$ymin),**

**bbox\_sf$xmin-0.2\*(bbox\_sf$xmax-bbox\_sf$xmin)),**

**c(bbox\_sf$ymin-0.2\*(bbox\_sf$ymax-bbox\_sf$ymin),**

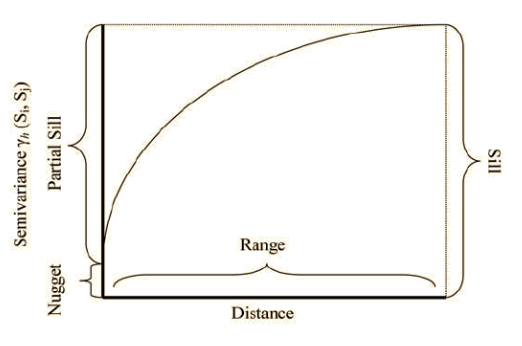
**bbox\_sf$xmax+0.2\*(bbox\_sf$xmax-bbox\_sf$xmin)),**

**zoom=18, type="bing", mergeTiles = TRUE)**

Various background maps can be used: "osm","bing","osm-german","esri","esri-topo","esri-physical","esri-shaded","esri-imagery","esri-terrain","esri-natgeo","nps","apple-iphoto". The disadvantage of this function is that sometimes it cannot find a map with a corresponding zoom. The zoom needs to be increased or decreased by 1 to find a working one. The *ggsave* then prints a *pdf* or *jpg* file with the resolution defined in the *dpi* parameter. If *OPEN\_WINDOWS* is defined as *true*, the file is open in a new window.

## Interpolations

### Kriging



Define the model:

**value\_vgm <- variogram(value ~ 1, data\_utm, cloud = FALSE)**

or if you want to predefine the binning and range:

**value\_vgm <- variogram(value ~ 1, data\_utm, cloud = FALSE,**

**boundaries = c(seq(2,10,1), seq(11, 40, 3), seq(41, 200, 5)))**

We can estimate the starting parameters:

**nugget <- value\_vgm$gamma[1]** # first variogram value

**psill <- max(value\_vgm$gamma) - value\_vgm$gamma[1]** # partial sill

**range <- value\_vgm$dist[which.max(value\_vgm$gamma)] #**

and fit the model with a function of the choice – typically Sph:

**v\_model <- fit.variogram(dose\_vgm, model = vgm(**

**model = "Sph",**

**nugget = nugget,**

**psill = psill,**

**range = range,**

**fit.kappa = TRUE**

**) )**

Model functions include e.g. "Exp", "Sph", "Gau", "Mat", "Ste", “Lin”.

Check the fit:

**print(v\_model)**

The fitted Kriging model is then used to produce the interpolated values

**value\_krig <- krige(value ~ 1, data\_utm[, "value"], newdata = grd\_utm, model = v\_model, nmax = 100)**

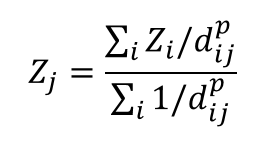
Note that the gridding has the resolution defined in the input file.

The uncertainty of kriging in % is calculated from the variance values.

**value\_krig$cov <- 100 \* sqrt( value\_krig$var1.var) / value\_krig$var1.pred**

### Inverse distance weighting

The value at an unsampled location 𝑗 is computed as



where 𝑑\_𝑖𝑗 is the distance between locations 𝑖 and 𝑗, 𝑝 is the weight parameter. Only points having the mutual distance lower than a predefined value are considered.

**dose\_idw <- idw(dose ~1, data\_utm, grd\_utm, idp = 10)**

**dose\_idw**

## Production of the maps with interpolated values

Several packages are used to print the interpolated data over maps, see part 7.

The Krigging method provides predictions, but also varians (estimation precision).

Always check the estimated presition.

The IDW provides predictions.

## Printing

The Openstreemap is again used for printouts:

**ggsave("printouts/dose-rate-krig.jpg", dpi=600)**

**ggsave("printouts/dose-rate-krig.pdf", dpi=600)**