

# Geostatistical COVID-19 infection risk maps with R

*Manuel Ribeiro, CERENA-IST/UL*

*August 28, 2020*

## 1 Introduction

This tutorial shows how to do block sequential simulation to model the spatial distribution of a disease, as in Azevedo et al. 2020<sup>1</sup>, using R. For that purpose we will run an example using COVID-19 data.

The R code in this tutorial calculates parameters and generate files to be read by an .exe program (dss.c.64.exe) performing block sequential simulation (using direct sequential simulation algorithm, Soares 2000<sup>2</sup>).

## 2 Data and R code

A set of functions written in R will generate the required files and call dss.c.64.exe for block sequential simulation.

To run the code, you will need :

- a COVID-19 datafile and,
- a grid with id region values at all simulation locations.

## 3 Basic instructions

Make sure to put all data and R functions in default (working) directory. Inside that directory create a folder called “input” and put the dss.c.64.exe inside.

## 4 COVID-19 example

For our example we will use a COVID-19 dataset for Portugal mainland with all cases notified to health authorities on 01/06/2020. The example is a step-by-step guide to obtain geostatistical maps of COVID-19. All tools needed to run this task (R functions, datasets and .exe) are available on github to reproduce example.

### 4.1 Source functions

Start by running the code via source. I created separate files for each function:

```
source("f1_irates.R", echo = T)
source("f2_blockfile.R", echo = T)
source("f3_maskfile.R", echo = T)
source("f4_varexp.R", echo = T)
source("f5_varmodel.R", echo = T)
source("f6_ssdpars.R", echo = T)
source("f7_outraster.R", echo = T)
```

### 4.2 Import COVID-19 data

As input you should provide a data frame with id of region, x, y and z cartesian coordinates at region mass center, number of COVID-19 cases by region and population at risk by region.

Therefore we start by reading an ascii file with the COVID-19 data (available on github) and create a data frame from it.

---

<sup>1</sup><https://doi.org/10.1186/s12942-020-00221-5>

<sup>2</sup><https://doi.org/10.1023/A:1012246006212>

```
covid = read.table("covid19_data.txt", header = TRUE, sep = "\t", dec = ".")
```

```
head(covid)
```

```
##   id_region      name_region      xcoord      ycoord      zcoord      poprisk      ncases
## 1         1         Abrantes -5332.936 -18739.44         0      35377         17
## 2         2         Agueda -24009.776  100641.94         0      45992         65
## 3         3   Aguiar da Beira  50540.025  122094.85         0       4740         NA
## 4         4     Alandroal  60012.722 -116382.18         0       5064         NA
## 5         5 Albergaria-a-Velha -29294.848  113819.38         0      24128         91
## 6         6     Albufeira -8581.783 -285532.71         0      41123         76
```

```
str(covid)
```

```
## 'data.frame':   278 obs. of  7 variables:
## $ id_region : int  1 2 3 4 5 6 7 8 9 10 ...
## $ name_region: Factor w/ 278 levels "Abrantes","Agueda",...: 1 2 3 4 5 6 7 8 9 10 ...
## $ xcoord : num -5333 -24010 50540 60013 -29295 ...
## $ ycoord : num -18739 100642 122095 -116382 113819 ...
## $ zcoord : int  0 0 0 0 0 0 0 0 0 0 ...
## $ poprisk : int  35377 45992 4740 5064 24128 41123 11712 12860 53641 19505 ...
## $ ncases : int  17 65 NA NA 91 76 7 9 40 23 ...
```

You may see the dataset contains regions where number of cases is missing (id regions 3 and 4). These NA must be replaced by an integer, for kriging to run. Function `irates()` (next section) includes an argument to overcome this limitation.

### 4.3 Compute rates ( $/10^4$ ) and variance-error terms with `irates()`

Use `irates()` to compute rates ( $/10^4$ ), variance-error terms by region. The arguments of the function are:

- `dfobj`, string, dataframe name with COVID-19 data
- `id`, character, field name for region id
- `x`, character, field name for x-coordinates
- `y`, character, field name for y-coordinates
- `z`, character, field name for z-coordinates
- `cases`, character, field name for number of cases
- `pop`, character, field name for population size
- `casesNA`, numeric, an integer used to replace rows with `cases = NA`,
- `day`, character, string indicating date (format “yyyymmdd”) of COVID-19 data

```
rates = irates(df = covid, oid = "id_region", x = "xcoord", y = "ycoord", z = "zcoord",
               cases = "ncases", pop = "poprisk", casesNA = 2, day = "20200601")
```

The function writes a text file (.out) with rates and store it in input folder. `irates()` also returns the following list of objects:

```
str(rates)
```

```
## List of 4
## $ rates : 'data.frame':   278 obs. of  7 variables:
## ..$ id : int [1:278] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ x : num [1:278] -5333 -24010 50540 60013 -29295 ...
## ..$ y : num [1:278] -18739 100642 122095 -116382 113819 ...
## ..$ z : int [1:278] 0 0 0 0 0 0 0 0 0 0 ...
## ..$ rate: num [1:278] 4.81 14.13 4.22 3.95 37.72 ...
## ..$ err : num [1:278] 0.000885 0.000681 0.006607 0.006184 0.001298 ...
## ..$ pop : int [1:278] 35377 45992 4740 5064 24128 41123 11712 12860 53641 19505 ...
```

```
## $ mrisk      : num 31.3
## $ file       :List of 3
## ..$ day      : chr "20200601"
## ..$ name     : chr "20200601notified.out"
## ..$ folder   : chr "Z:/pos_doc/covid/bkrig/tutorial_example/input"
## $ ssdirpars :List of 7
## ..$ nvars    : int 4
## ..$ xcolumn  : int 1
## ..$ ycolumn  : int 2
## ..$ zcolumn  : int 3
## ..$ varcol   : int 4
## ..$ minval   : num 1.31
## ..$ maxval   : num 122
```

#### 4.4 Create block data with blockfile()

Use `blockfile()` to transform grid file in block format. The function requires some libraries to be loaded. If not installed they will be first installed.

You should provide a georeferenced grid file with id region values at simulation locations. The arguments of the function are:

- `rateobj`, character, name of list, output of function `irates()`.
- `gridimage`, character, name of georeferenced grid file (e.g. tif)
- `na.value`, numeric, integer with grid value for “No data”

```
block = blockfile(rates, "grid2k.tif")
```

The grid file values should refer to the region id's at simulation locations (nodes). All regions in covid data should be represented by 1 or more node.

The function writes a text file (.out) with blockdata and store it in input folder. `blockfile()` also returns the following list of objects:

```
str(block, max.level = 2)
```

```
## List of 4
## $ gridpars:List of 4
## ..$ nodes      : int [1:2] 141 288
## ..$ resolution : num [1:2] 2000 2000
## ..$ origin     : num [1:2] -119191 -300405
## ..$ NAs        : num -999
## $ outgrid :List of 3
## ..$ values : num [1:40608] -999 -999 -999 -999 -999 -999 -999 -999 -999 -999 ...
## ..$ idblock: int [1:278] 1 2 3 4 5 6 7 8 9 10 ...
## ..$ nblock  : int 278
## $ file      :List of 3
## ..$ day     : chr "20200601"
## ..$ name    : chr "20200601blockdata.out"
## ..$ folder  : chr "Z:/pos_doc/covid/bkrig/tutorial_example/input"
## $ ingrid   :Formal class 'RasterLayer' [package "raster"] with 12 slots
```

#### 4.5 Create mask file with maskfile()

The function `maskfile()` creates a mask for the block file. The only argument of the function is the name of list, output of function `blockfile()`.

```
mask = maskfile(block)
```

Generates a file with values  $\{-1,0\}$  where -1 are assigned to nodata locations and 0 are assigned to nodes with values (id region). A text file (.out) with mask data is created and stored in input folder.

```
str(mask)
```

```
## List of 2
## $ file :List of 3
## ..$ day : chr "20200601"
## ..$ name : chr "20200601mask.out"
## ..$ folder: chr "Z:/pos_doc/covid/bkrig/tutorial_example/input"
## $ zones:List of 2
## ..$ nzones : int 2
## ..$ zoneval: num [1:2] -1 0
```

#### 4.6 Calculate experimental variogram with varexp()

Use `varexp()` to calculate experimental variogram from COVID-19 rates. Only implemented in omnidirectional case. The arguments are:

- `dfobj`, character, name of object, output of function `irates()`
- `lag`, numeric, the lag distance used for variogram estimates
- `nlags`, numeric, the number of lags to calculate variogram.

```
vexp = varexp(rates, lag = 7000, nlags = 15)
```

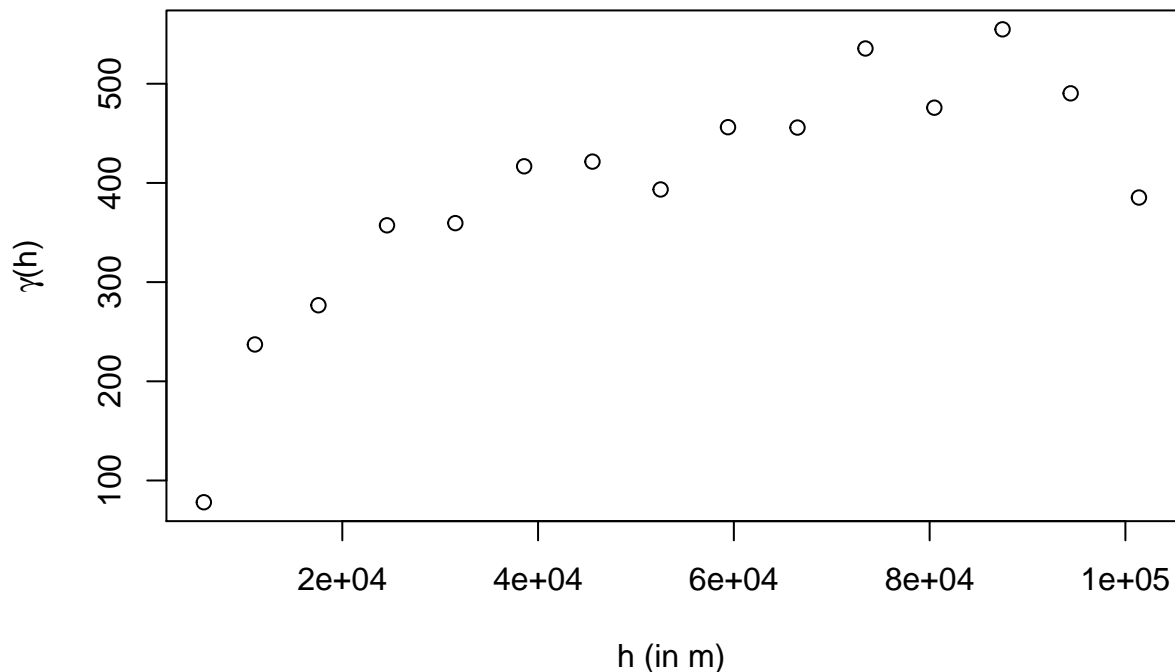
The function returns a list with the weighted variance (by population size) and variogram estimates at nlags.

```
str(vexp)
```

```
## List of 2
## $ weightsvar: num 501
## $ semivar : 'data.frame': 15 obs. of 2 variables:
## ..$ dist : num [1:15] 5855 11072 17561 24565 31547 ...
## ..$ semivariance: num [1:15] 78.1 237.1 276.7 357.3 359.5 ...
```

You may plot results to evaluate main structural patterns in the data:

```
plot(vexp[["semivar"]], ylab = expression(paste(gamma, "(h)")), xlab = "h (in m)")
```



#### 4.7 Fit variogram model with `varmodel()`

Function `varmodel()` fits (manually) a theoretical variogram. You should provide the experimental variogram data to evaluate fit by visual inspection, the variogram model type and the variogram parameters. The arguments of `varmodel()` are:

- `varexp`, character, name of object, output of function `varexp()`
- `mod`, character, the variogram model type (available are: “Sph” or “Exp”)
- `nug`, numeric, nugget-effect value of the variogram
- `ran`, numeric, range value of the variogram
- `sill`, numeric, sill (or partial sill) value of the variogram

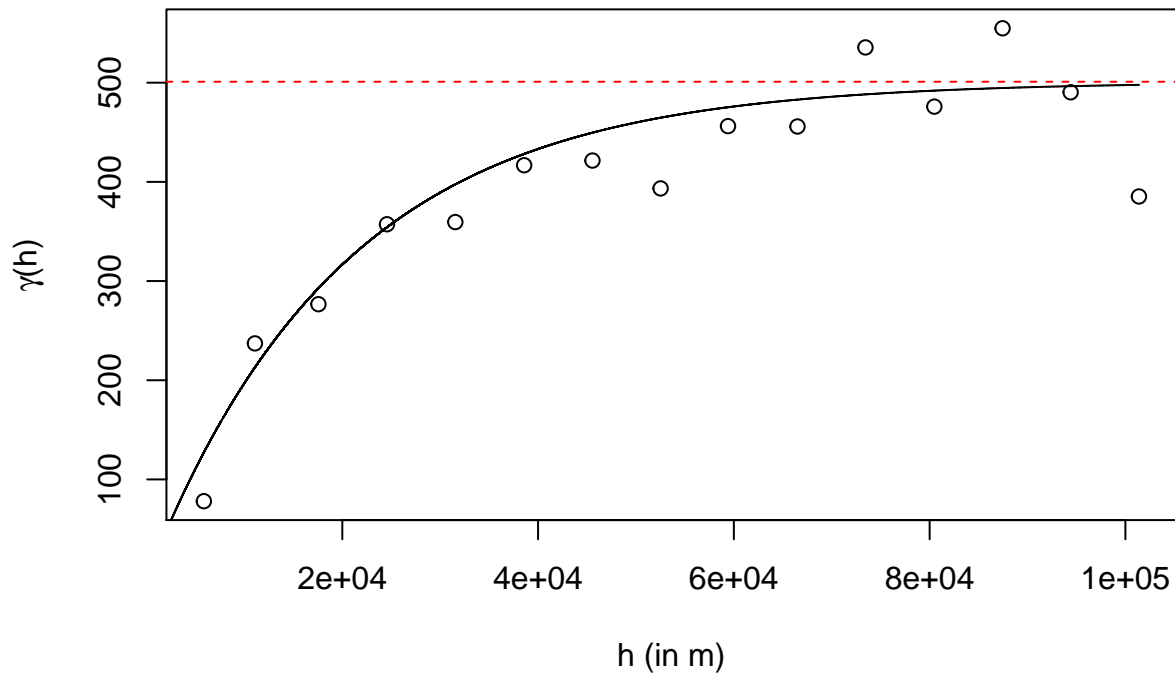
```
vmod = varmodel(vexp, mod = "Exp", nug = 0, ran = 60000, sill = vexp[["weightsvar"]])
```

```
str(vmod)
```

```
## List of 3
## $ structures: num 1
## $ parameters:'data.frame': 1 obs. of 5 variables:
## ..$ model : Factor w/ 1 level "Exp": 1
## ..$ modeltype: num 2
## ..$ nugget : num 0
## ..$ range : num 60000
## ..$ psill : num 501
## $ fittedval : num [1:101385] 0 0.025 0.0501 0.0751 0.1002 ...
```

```
# plot experimental variogram
plot(vexp[["semivar"]], ylab = expression(paste(gamma, "(h)")), xlab = "h (in m)")
```

```
# add sill
abline(h = vexp[["weightsvar"]], col = "red", lty = 2)
# add theoretical model
lines(vmod[["fittedval"]])
```



#### 4.8 Create parameters file with `ssdparams()` and calls `dss.c.64.exe`

Function `ssdparams()` writes the parameters file (.par) for .exe and runs block sequential simulations. Function arguments are the names of objects returned above and parameter values for block-kriging and simulation processes:

- `blockobj`, character, name of list, output of function `blockfile()`
- `maskobj`, character, name of list, output of function `maskfile()`
- `dfobj`, character, name of list, output of function `irates()`
- `varmobj`, character, name of list, output of function `varmodel()`
- `simulations`, numeric, number of simulations
- `nrbias`, numeric, nr simulations for bias correction
- `biasecor`, num vector, flag for (mean, variance) correction (yes = 1, no = 0)
- `ndMin`, numeric, min number of neighbour observations used in kriging
- `ndMax`, numeric, max number of neighbour observations used in kriging
- `nodMax`, numeric, max number of previously simulated nodes used in kriging
- `radius1`, numeric, search radii in the major horizontal axis
- `radius2`, numeric, search radii in the axis orthogonal (horizontal) to radius1
- `radius3`, numeric, search radii in the vertical axis
- `ktype`, numeric, the kriging type to be used (available are: 0 = simple, 1 = ordinary)

```
ssdpars(blockobj = block, maskobj = mask, dfobj = rates, varmobj = vmod,
        simulations = 3, radius1 = 60000, radius2 = 60000)
```

The function generates a text file (.par), calls dss.c.64.exe, run block simulations and returns the simulated maps (.out). This process may take a while depending mostly on the number of simulation nodes and number of simulations.

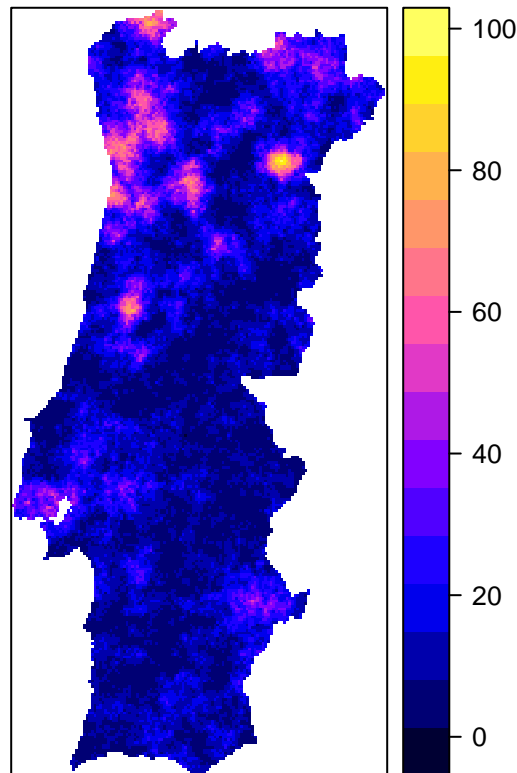
Both text file (.par) and simulations files (.out) are stored in input folder.

## 4.9 Export simulations from .out format to raster format

Function `outraster()` transforms simulations .out returned by `ssdpars()` into grid (raster) files. Argument `blockobj` = is the name (string) of the object returned by `blockfile()`. If argument `emaps` = T, the function also computes e-type and uncertainty maps and plots e-type map.

```
outraster(block, emaps = T)
```

```
## [1] "20200601sim_1.out"
## [1] "20200601sim_2.out"
## [1] "20200601sim_3.out"
## [1] "20200601sim_4.out"
## [1] "20200601sim_5.out"
```



## Acknowledgements

Manuel Ribeiro acknowledges the financial support of the CERENA (project FCT-UIDB/04028/2020) and Fundação para a Ciência e Tecnologia (research contract IF2018-CP1384). Manuel Ribeiro gratefully

acknowledge CERENA-IST/UL researchers Leonardo Azevedo, Maria João Pereira and Amilcar Soares for the code in Matlab and Fortran.