Practical 1: Disease mapping study of lung cancer incidence in Greater London, 2001-2005

18 February, 2019

In this practical you will be using R as well as OpenBUGS to carry out a disease mapping study as discussed in Lecture 1.

The R packages used in this practical are: rgdal, maptools, sp, spdep, SpatialEpi, R2OpenBUGS, mcmcplots, coda and RColorBrewer.

Before starting the practical

Organise your work:

- Create a separate subdirectory in your home directory to save your files created during this practical (e.g. "C:/SpatialAnalysis2019/Practicals/Practical1").
- Copy all the files from the blackboard to your subdirectory (created just above). The data set includes the number of cases of lung cancer and population counts for the 628 wards of Greater London, stratified by sex and age groups.
- To check whether a package is installed you can use the query:

```r
is.element("rgdal", installed.packages())
```

- If FALSE is returned, then install the package function `install.packages()`, e.g.

```r
install.packages("rgdal", dep=TRUE)
```

- Then, load needed libraries:

```r
library(rgdal)        # Geospatial Data Abstraction Library,
```

```
## Loading required package: sp

## rgdal: version: 1.3-6, (SVN revision 773)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 2.2.3, released 2017/11/20
##  Path to GDAL shared files: C:/MyR_packages/R/rgdal/gdal
##  GDAL binary built with GEOS: TRUE
##  Loaded PROJ.4 runtime: Rel. 4.9.3, 15 August 2016, [PJ_VERSION: 493]
##  Path to PROJ.4 shared files: C:/MyR_packages/R/rgdal/proj
##  Linking to sp version: 1.3-1
```

```
                        # projection/transformation operations
library(maptools)       # Functions for manipulating and reading geographical data
```

```
## Checking rgeos availability: TRUE
```

```
library(sp)             # Classes and methods for spatail analysis
library(spdep)          # Functions and tests for evaluating spatial patterns
```

```
## Loading required package: Matrix

## Loading required package: spData

## To access larger datasets in this package, install the spDataLarge
## package with: `install.packages('spDataLarge',
## repos='https://nowosad.github.io/drat/', type='source')`
```

```
                        # and autocorrelation
library(SpatialEpi)     # Methods and Data for Spatial Epidemiology
library(R2OpenBUGS)     # A Package for Running OpenBUGS from R
library(mcmcplots)      # A Package for plotting and viewing of MCMC output
```

```
## Loading required package: coda
```

```
library(coda)           # A Package for summarizing and plotting of MCMC output
                        # and diagnostic tests
library(RColorBrewer)   # A pakage providing colour palettes for shading maps
                        # and other plots
```

Data

- Import the health data

```
lung <- read.csv("Lungcancer_strata_GL.csv")
```

- Print only the first rows of the data

```
head(lung)
```

```
##   STwardcode POLY_ID STWardName SEX AGE_GROUP POPULATION CASES
## 1     00AAFA       1 Aldersgate   M         0     18.608     0
## 2     00AAFA       1 Aldersgate   M         1     18.608     0
## 3     00AAFA       1 Aldersgate   M         2     18.608     0
## 4     00AAFA       1 Aldersgate   M         3     18.608     0
## 5     00AAFA       1 Aldersgate   M         4     18.608     0
## 6     00AAFA       1 Aldersgate   M       5_9     71.000     0
```

- What are the names of the variables (columns)?

```
names(lung)
```

```
## [1] "STwardcode" "POLY_ID"    "STWardName" "SEX"        "AGE_GROUP"
## [6] "POPULATION" "CASES"
```

- How many levels for the variables SEX and AGE_GROUPS? What are the levels?

```
levels(lung$SEX)
```

```
## [1] "F" "M"
```

```
nlevels(lung$SEX)
```

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

```
levels(lung$AGE_GROUP)
```

```
##  [1] "0"      "1"      "10_14"  "15_19"  "2"      "20_24"  "25_29"
##  [8] "3"      "30_34"  "35_39"  "4"      "40_44"  "45_49"  "5_9"
## [15] "50_54"  "55_59"  "60_64"  "65_69"  "70_74"  "75_79"  "80_84"
## [22] "85PLUS"
```

```
nlevels(lung$AGE_GROUP)
```

```
## [1] 22
```

- The levels of the age variable (AGE_GROUPS) are not in the right order. Re-order the data from `0` to `85PLUS`

```
lung$AGE_GROUP <- ordered(lung$AGE_GROUP, levels = c("0","1","2","3","4","5_9",
               "10_14","15_19","20_24","25_29","30_34",
               "35_39","40_44","45_49","50_54","55_59","60_64",
               "65_69","70_74","75_79", "80_84","85PLUS"))
levels(lung$AGE_GROUP)
```

```
##  [1] "0"      "1"      "2"      "3"      "4"      "5_9"    "10_14"
##  [8] "15_19"  "20_24"  "25_29"  "30_34"  "35_39"  "40_44"  "45_49"
## [15] "50_54"  "55_59"  "60_64"  "65_69"  "70_74"  "75_79"  "80_84"
## [22] "85PLUS"
```

- Compute the total number of cases of lung cancer in Greater London

```
sum(lung$CASES)
```

```
## [1] 18587
```

- Compute the total number of cases of lung cancer by SEX

```
aggregate(lung$CASES, by=list(lung$SEX), FUN=sum)
```

```
##   Group.1     x
## 1       F  7579
## 2       M 11008
```

- Compute the number of cases of lung cancer and population counts by AGE_GROUP

```
aggregate(lung[,c("CASES", "POPULATION")],
          by=list(lung$AGE_GROUP), FUN=sum)
```

```
##    Group.1 CASES POPULATION
## 1        0    15   481072.8
## 2        1    19   481072.8
## 3        2    21   481072.8
## 4        3    22   481072.8
## 5        4    27   481072.8
## 6      5_9    14  2222602.0
## 7    10_14    13  2158418.5
## 8    15_19    20  2177301.2
## 9    20_24    26  2712779.7
## 10   25_29    35  3669932.1
## 11   30_34    53  3674631.1
## 12   35_39   101  3358548.4
## 13   40_44   194  2764339.3
## 14   45_49   441  2222323.0
## 15   50_54   828  1967550.2
## 16   55_59  1369  1780678.5
## 17   60_64  1856  1397130.2
## 18   65_69  2583  1245716.9
## 19   70_74  3209  1087526.4
## 20   75_79  3322   900613.2
## 21   80_84  2641   670622.7
## 22  85PLUS  1778   552952.8
```

- Create a new data frame, called newLung, with the number of cases per ward and the two variables STwardcode and POLY ID

```
newLung <- aggregate(lung$CASES,
                    by=list(STwardcode=lung$STwardcode,
                    POLY_ID=lung$POLY_ID), FUN=sum)
```

- Change the name of the column corresponding to the number of observed cases to O

```
names(newLung)
```

```
## [1] "STwardcode" "POLY_ID"    "x"
```

```
names(newLung)[3]<-"O"
```

Estimating and mapping the SMRs

To calculate the expected number of cases in each ward, follow the steps:

1. Order the data set lung, so that the strata (i.e. SEX and AGE_GROUP) are the same in each ward

```
lung <- lung[order(lung$POLY_ID, lung$SEX, lung$AGE_GROUP),]
```

2. Calculate the expected numbers of cases using the function `expected` of the package `SpatialEpi`. Think carefully about the number of strata

```
newLung$E <- expected(population=lung$POPULATION,
                     cases=lung$CASES, n.strata=2*22)
# nb of strata = 44 (2 strata levels for sex * 22 strata levels for age groups)
```

3. Check if everything is OK:

- Sum of the *expected* numbers of cases = sum of the *observed* numbers of cases

```
sum(newLung$E)
```

```
## [1] 18587
```

```
sum(newLung$O)
```

```
## [1] 18587
```

- number of rows = number of wards (remember that the number of wards is 628)

```
dim(newLung)
```

```
## [1] 628   4
```

4. Calculate the SMRs and add them to the data frame newLung

```
newLung$SMR <- newLung$O/newLung$E
```

- Summarise the dataset newLung

```
summary(newLung)
```

```
##    STwardcode     POLY_ID             O               E
##  00AAFA :  1   Min.   :  1.0   Min.   : 1.0   Min.   : 2.616
##  00AAFQ :  1   1st Qu.:157.8   1st Qu.:22.0   1st Qu.:24.100
##  00AAFT :  1   Median :314.5   Median :29.0   Median :27.508
##  00AAFX :  1   Mean   :314.5   Mean   :29.6   Mean   :29.597
##  00ABFX :  1   3rd Qu.:471.2   3rd Qu.:36.0   3rd Qu.:34.090
##  00ABFY :  1   Max.   :628.0   Max.   :71.0   Max.   :61.368
##  (Other):622
##       SMR
##  Min.   :0.3286
##  1st Qu.:0.7790
##  Median :0.9795
##  Mean   :1.0274
##  3rd Qu.:1.2590
##  Max.   :2.2038
##
```

5. In order to map the SMRs, follow the steps:

- Read the Shapefiles of Greater London with the Thames river using the function `readOGR` of the package `rgdal`. Call this object as `shpriver`, and plot it.

```
shpriver <- readOGR(dsn = ".", layer = "GreaterLondon_ward_river")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "C:\SpatialAnalysis2019\Practicals\Practical1", layer: "GreaterLondon_ward_river"
## with 628 features
## It has 6 fields
## Integer64 fields read as strings:  POLY_ID
```

```r
plot(shpriver, border = "blue")
title(main = "Map of Greater London")
```

Map of Greater London

In the code above the `dsn` argument specifies the data source name. If the shapefile is in your current working directory, the `dsn` refers to that directory and all you need is symply a `"."`. Note that the Shapefile could be also imported using the function `readShapeSpatial` of the package `maptools`, such as `shpriver <- readShapeSpatial("GreaterLondon_ward_river.shp", IDvar="STwardcode")`, however this function is deprecated.

- Examine the object `shpriver`: the class and the names of the columns of the data frame (to see what it contains). Then produce result summaries.

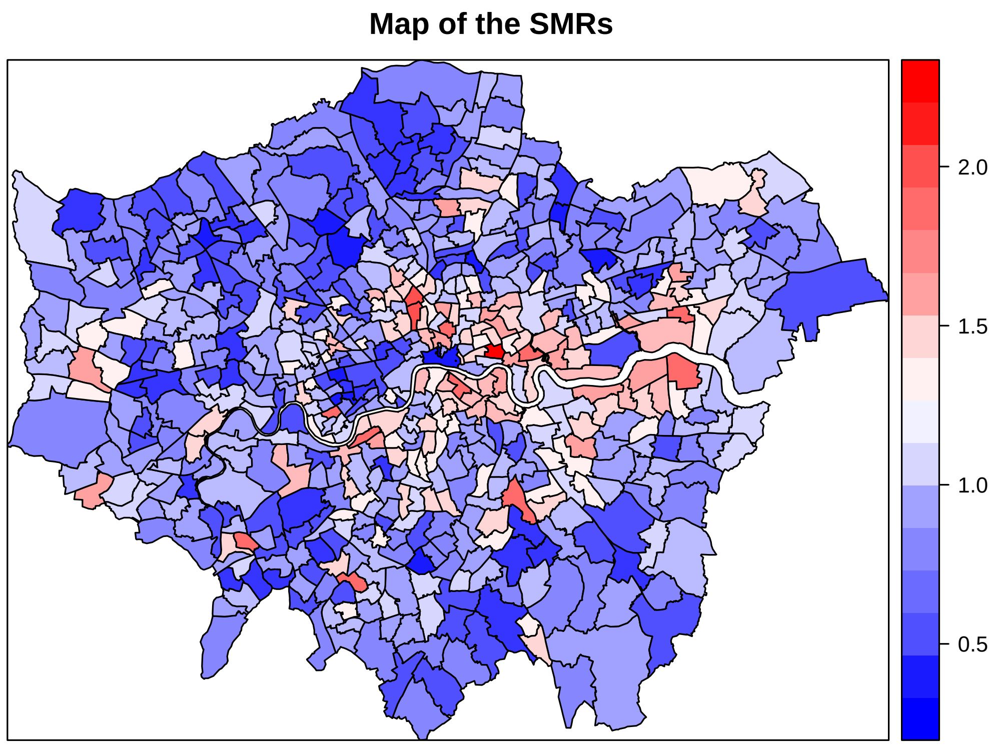
```r
class(shpriver) # SpatialPolygonsDataFrame
names(shpriver)
summary(shpriver)
```

- Merge the health data with the shapefile by using the common variable STwardcode (i.e. the code of the wards)

```r
data.London <- attr(shpriver, "data")
attr(shpriver, "data") <- merge(data.London, newLung,
                               by="STwardcode")
```

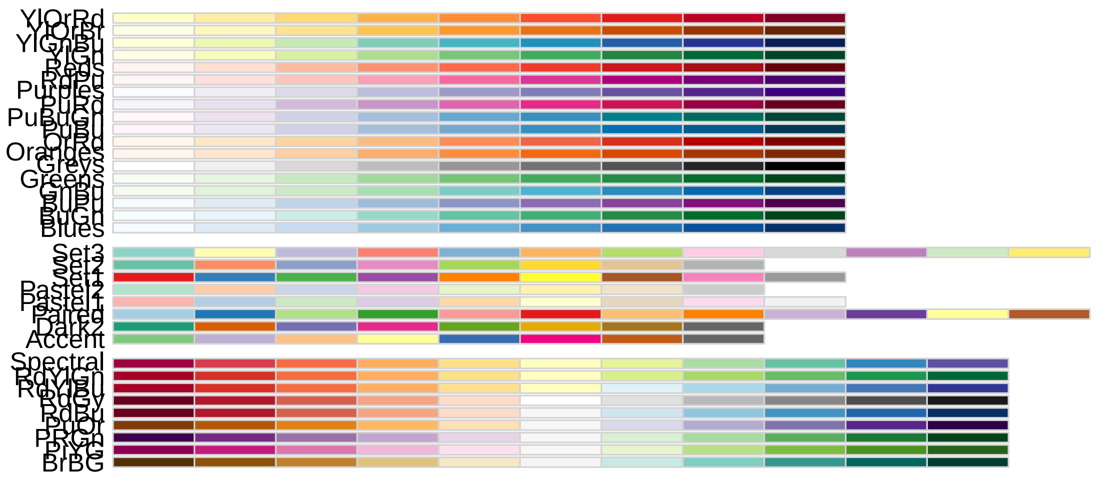
- Plot the SMRs using the function `spplot` of the package sp, that is an extension of plot specifically for making maps of spatial objects.

```r
color.palette <- colorRampPalette(c("blue", "white", "red")) #
spplot(obj=shpriver, zcol="SMR", col.regions=color.palette(20), asp=1,
       main = "Map of the SMRs")
```

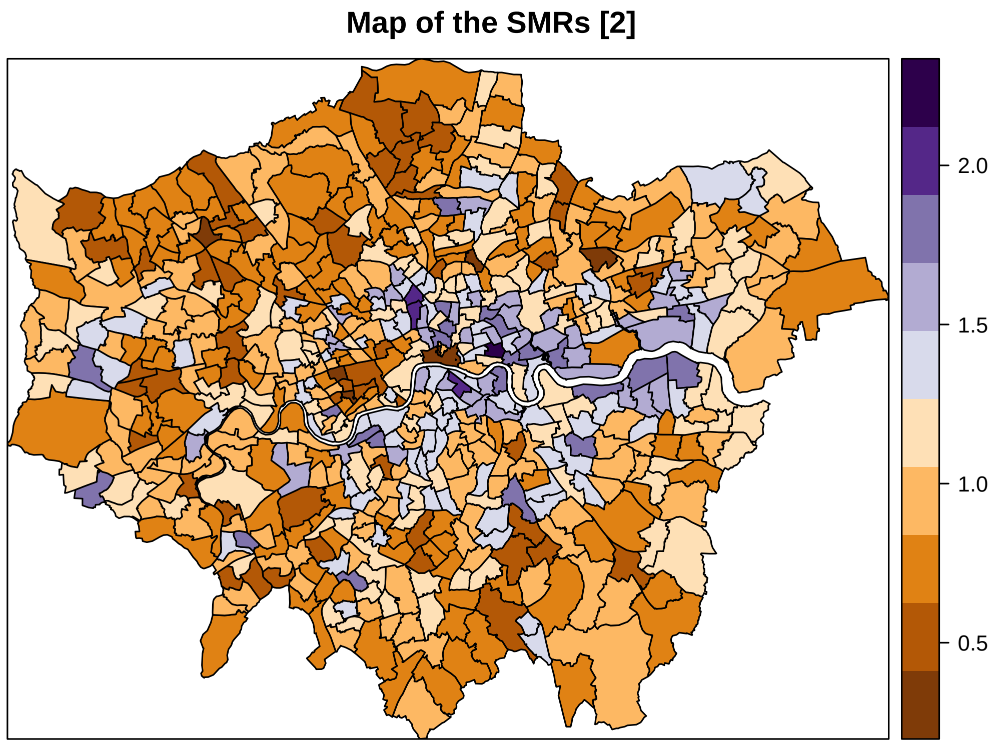


Here we use the function `colorRampPalette` to create custom palettes for `spplot`. A useful package that can be alternatively used is `RColorBrewer`. Once loaded, you can see a list of all the color pallets typing `display.brewer.all()`, then pick a palette you like a plot the SMRs

```r
display.brewer.all()
```



```
my.palette <- brewer.pal(n = 10, name = "PuOr")
# then, pass this pallet to spplot,
# making sure to set the number of cuts to 1 minus the number of colors.
spplot(obj=shpriver, zcol="SMR", col.regions=my.palette, cut=9,
       main = "Map of the SMRs [2]")
```



Note that for writing a map out to PDF, PNG, JPG etc. files, you can use the following code:

```r
jpeg(file="MapSMR.jpg") # to save the map as a JPEG file
spplot(obj=shpriver, zcol= "SMR", col.regions=color.palette(20), asp=1,
       main = "Map of the SMRs")
dev.off()
# dev.off() deletes your current plot in the RStudio Plots Pane.
# This tells R that you are finished plotting; otherwise your map will not show up.
# If you have multiple graphics devices open, repeat this command until the output
# displays null device.
```

Obtaining the posterior RR

The RRs will be smoothed using the Poisson-logNormal model. The inference is done with OpenBUGS called through R. The model is provided in the *model_HET.txt*.

- DO NOT CLOSE R and open the model file *model_HET.txt*. Have a look to the code and try to understand it (suggestion: discuss the code with your colleagues).
- Once the file *model_HET.txt* is clearly understood close it.
- Back to R, format the health data in a list. Do not forget to order the data so that POLY_ID is from 1 to 628 (same order as the shapefile)

```r
# Order the data according to POLY_ID
newLung <- newLung[order(newLung$POLY_ID),]

# Obtain the number of wards
n.wards <- dim(shpriver)[1]

# Format the data for OpenBUGS
data <- list(N=n.wards, # nb of wards
           O=newLung$O, # observed nb of cases
           E=newLung$E) # expected nb of cases
```

- What are the parameters to be initialised? With R, create a list with two elements (each a list) with different initial values for the parameters

```r
# Initialise the unknown parameters, 2 chains
inits <- list(
  list(alpha=0.01, prec.v=10, V=rep(0.01,times=628)),  # chain 1
  list(alpha=0.5, prec.v=1, V=rep(-0.01,times=628)))   # chain 2
```

- Set the parameters that will be monitored by OpenBUGS, e.g.

```r
# Monitored parameters
parameters <- c("sigma2.v", "overallRR", "resRR", "RR", "e")
```

Note that the parameters that are not set, will NOT be monitored!

- Specify the MCMC setting

```r
# MCMC setting
ni <- 3000   # nb iterations
nt <- 1      # thinning interval
nb <- 1000   # nb iterations as burn-in
nc <- 2      # nb chains
```

The burn-in should be long enough to discard the initial part of the Markov chains that have not yet converged to the stationary distribution.

- Run the MCMC simulations calling OpenBUGS from R using the function `bugs()`

```r
modelHET.sim <- bugs(data = data, parameters = parameters,
                     inits = inits, model.file = "model_HET.txt",
                     n.chains = nc, n.iter = ni, n.burnin = nb,
                     n.thin = nt, debug = FALSE,
                     working.directory = getwd(),
                     codaPkg = FALSE, summary.only = FALSE,
                     bugs.seed = 9)
```

Note that specifying `codaPkg = FALSE`, the function `bugs()` returns a bugs object (if `codaPkg = TRUE`, file names of OpenBUGS output are returned for easy access by the coda package). In this case, to produce a coda object from the `modelHET.sim` output, we can use the function `read.bugs`, e.g. `model.coda <- read.bugs(modelHET.sim)`. Here, we proceed setting the output from OpenBUGS as a bugs object.

- Inspect object `modelHET.sim`

```
attributes(modelHET.sim)
```

```
## $names
##  [1] "n.chains"        "n.iter"          "n.burnin"
##  [4] "n.thin"          "n.keep"          "n.sims"
##  [7] "sims.array"      "sims.list"       "sims.matrix"
## [10] "summary"         "mean"            "sd"
## [13] "median"          "root.short"      "long.short"
## [16] "dimension.short" "indexes.short"   "last.values"
## [19] "isDIC"           "DICbyR"          "pD"
## [22] "DIC"             "model.file"
##
## $class
## [1] "bugs"
```

```
names(modelHET.sim$sims.list)
```

```
## [1] "sigma2.v"  "overallRR" "resRR"     "RR"        "e"         "deviance"
```

```
# bugs() automatically saves the "deviance" as a measure of
# goodness of fit of the model
```

- Summarize posteriors from `modelHET.sim`

```
print(modelHET.sim, digits = 3)
```

- You can produce the summary statistics of all the monitored parameters typing

```
modelHET.sim$summary
# or
modelHET.sim$summary[, c(1, 2, 3, 7)]
```

- You can also check the 95% and 90% CI of the posterior distribution of the parameters such as

```r
quantile(modelHET.sim$sims.list$overallRR,
         probs = c(0.025, 0.975))
```

```
##   2.5%  97.5%
## 0.9591 1.0110
```

```r
quantile(modelHET.sim$sims.list$overallRR,
         probs = c(0.05, 0.95))
```

```
##     5%    95%
## 0.9633 1.0070
```

- Check the convergence. We expect the chains to eventually converge to the stationary distribution. However, there is no guarantee that the chains have converged after a number of draws. There is a combination of several standard ways to check the convergence. Here some useful checks. The package `mcmcplot` allows us to visualise the main diagnostic plots (trace, density, autocorrelation) from MCMC simulations. It creates an HTML file, opening the default browser straight away.

```r
mcmcplot(modelHET.sim, random=3) # diagnostic plots
```

Here, specifying the `random` option, only a random subset of parameters in the model is plotted. NOTE that to use `mcmcplot()`, you need to specify `codaPkg = FALSE` within the function `bugs()`.

Moreover, the `gelman.diag` statement gives us the median *potential scale reduction factor (psrf)* and its 97.5% quantile (the psrf is estimated with uncertainty because the chain lengths are finite). The scale reduction factor measures basically whether there is a difference between the variance within the chains and the variance between the chains. When this is high (perhaps greater than 1.1), then we should run our chains out longer to improve convergence to the stationary distribution.

```r
gelman.diag(modelHET.sim) # Gelman Rubin diagnostic
```

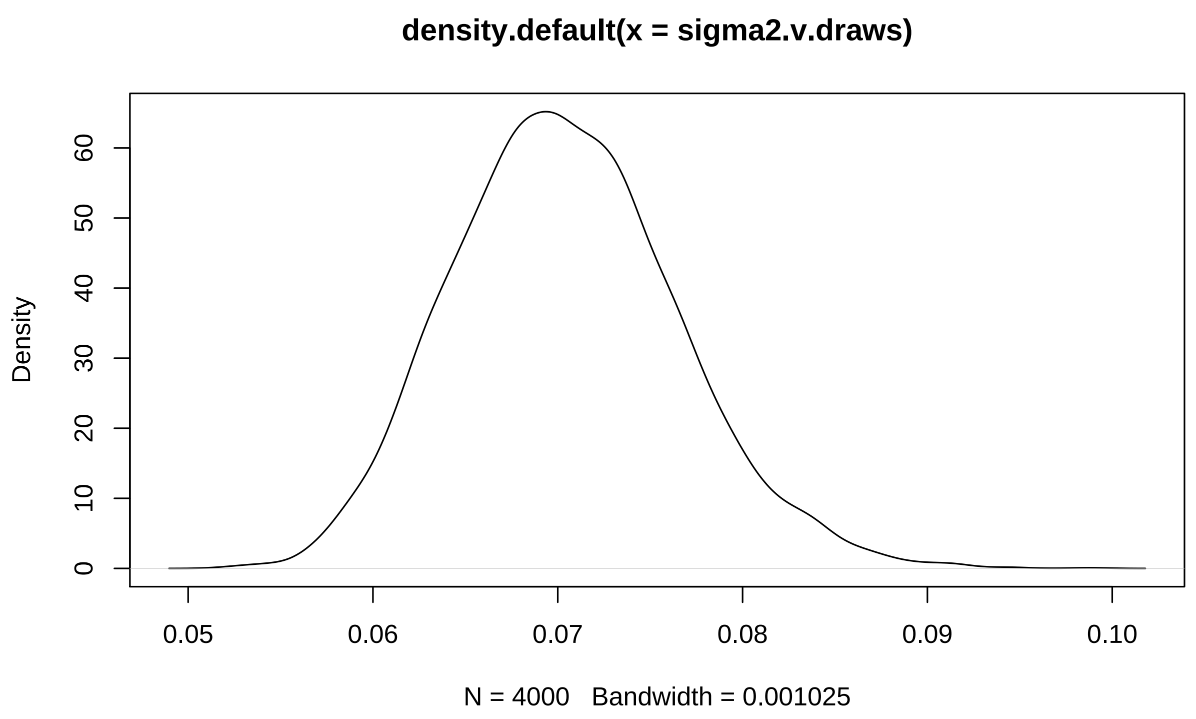
For models with many parameters it is not practical to check the convergence for every parameter, so a choice should be made of the relevant parameters to monitor. In this way the convergence of the OpenBUGS output can be assessed with a reasonable degree of confidence.

To use the package `coda` we should turn our chains into `mcmc` objects. For example, looking at the parameter `sigma2.v`:

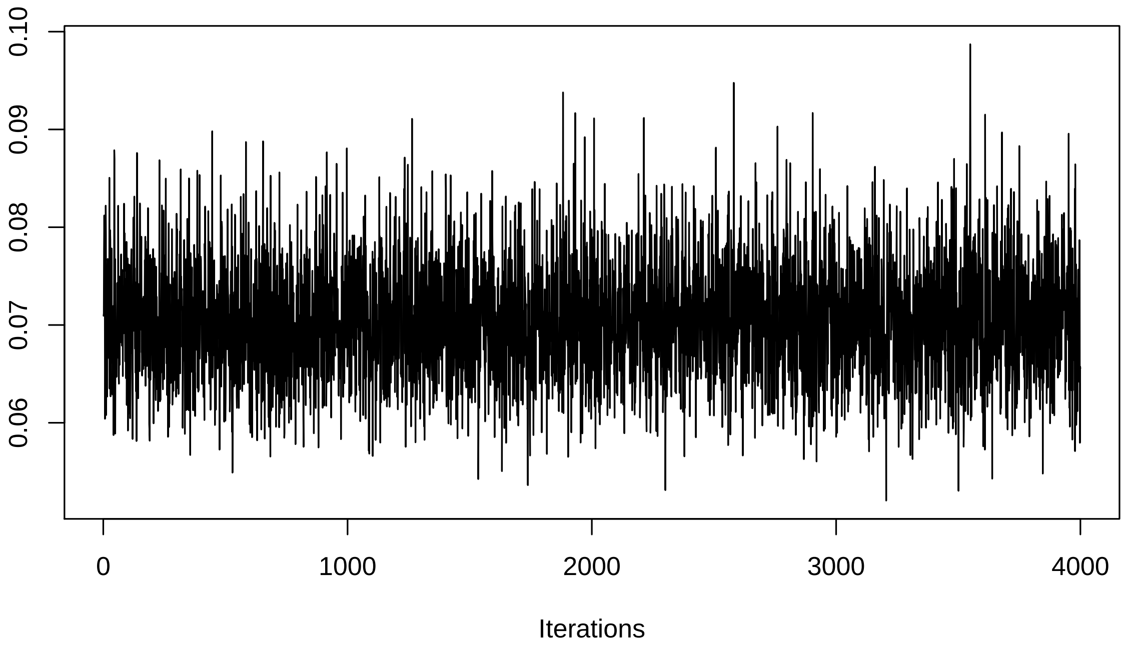
```r
attach.bugs(modelHET.sim) # attach bugs object
sigma2.v.draws <- mcmc(sigma2.v) # turn the chains into mcmc object
```

```r
summary(sigma2.v.draws) # posterior mean,
# posterior standard deviation and posterior quantiles
```

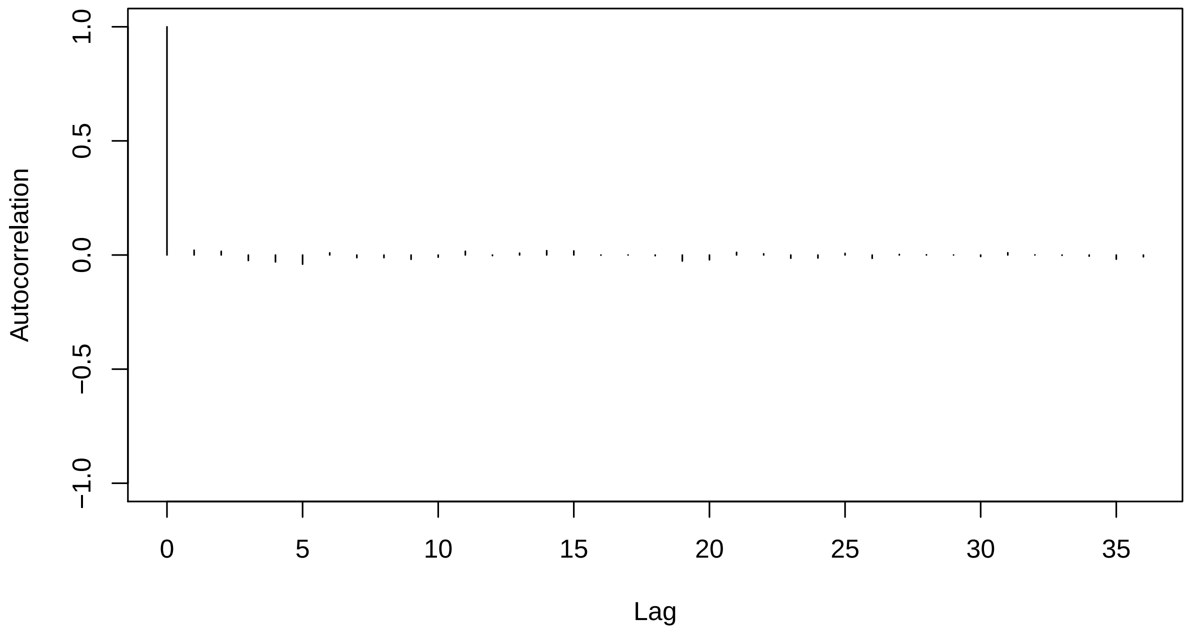
```
plot(density(sigma2.v.draws)) # density plot
```



```
traceplot(sigma2.v.draws) # traceplot
```



```
autocorr.plot(sigma2.v.draws) # autocorrelation plot
```



- Take note of DIC (we will use it in another practical)

```
modelHET.sim$DIC
```

```
## [1] 4287
```

- Now we want to map the smoothed RRs in R using spplot. To do so, extract the relative risk RR

```
RR_HET <- as.data.frame(apply(RR,2,mean)) # posterior mean
```

- Create an ID for each ward, from 1 to 628

```
RR_HET$POLY_ID <- 1:628
```

- Change the name of the column corresponding to the posterior mean to HET

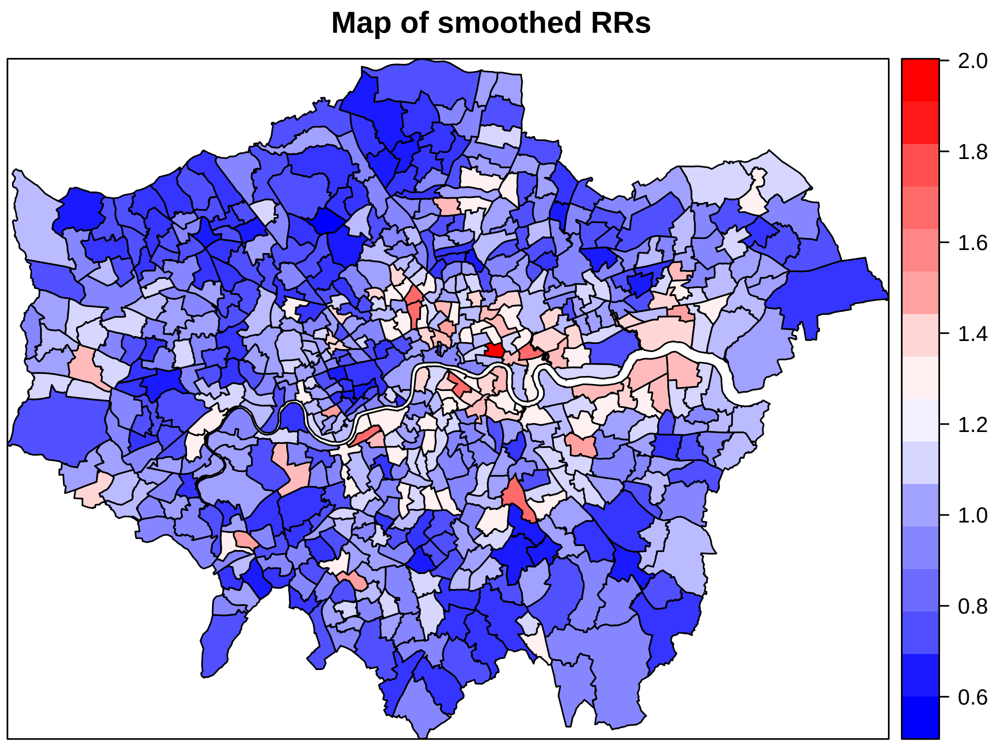
```
colnames(RR_HET) <- c("HET","POLY_ID")
```

- Merge RR_HET with newLung using POLY_ID as column for merging

```
newLung <- merge(newLung, RR_HET, by="POLY_ID")
```

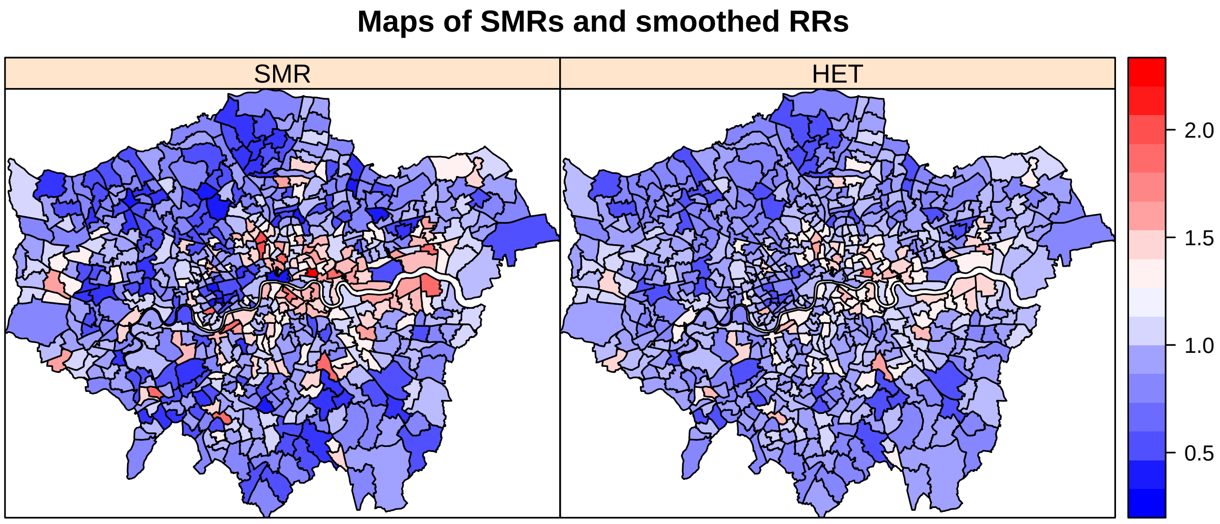
- Using `spplot` function, produce a map of the smoothed RR

```
attr(shpriver, "data") <- merge(data.London, newLung, by="STwardcode")
spplot(obj=shpriver, zcol= "HET", col.regions=color.palette(20),
       asp=1,  main="Map of smoothed RRs")
```



- Include the maps in the same plot. Comment the maps

```
spplot(obj=shpriver, zcol= c("SMR","HET"), col.regions=color.palette(20),
       asp=1, main="Maps of SMRs and smoothed RRs")
```



- Now, obtain the posterior mean of the residuals

```
resid.HET <- as.data.frame(apply(e,2,mean)) # posterior mean of the residuals
names(resid.HET)[1] <- paste("resid")
```

- Evaluate the *Moran's I statistic* for spatial autocorrelation on the residuals (permutation test using the function `moran.mc` from the package `spdep`). To do so, first define the neighbours of each area. As we saw in Lecture 1, neighbours can be defined in several ways, but a common definition is that a pair of areas which share some part of their boundaries are neighbours. The function `poly2nb` in the package `spdep` allows the extraction of the neighbours list specifying the definition of polygon adjacency (rook or queen criterion). The list of neighbours is stored in an `nb` object. Then, from the `nb` object we can obtain a `listw` object that stores the list of neighbours, together with their weights using the `nb2listw` function. Finally we compute Moran's I using a simulation-based approach using the function `moran.mc`

```
# Neighbours (queen is the default, alternatively rook)
W.nb <- poly2nb(shpriver, queen=FALSE)

# Weights matrix, W=row standardized, B=binary, C=globally standardized
W.list <- nb2listw(W.nb, style = "B", zero.policy=TRUE)

# Moran's I permutation test
moran.mc(x = resid.HET$resid, listw = W.list, nsim = 10000, zero.policy=TRUE)
```

```
##
##  Monte-Carlo simulation of Moran I
##
## data:  resid.HET$resid
## weights: W.list
## number of simulations + 1: 10001
##
## statistic = 0.0047621, observed rank = 9326, p-value = 0.06749
## alternative hypothesis: greater
```

```
# Note: zero.policy=TRUE permits the weights list to be formed
# with zero-length weights vectors
```

The Moran's I statistic does not seem to suggest substantive positive spatial autocorrelation.