R Scripts related to Analyzing spatio-temporal data with R: Everything you always wanted to know – but were afraid to ask

Ghislain Geniaux and Davide Martinetti and Edith Gabriel and Eric Parent and Nicolas Desassis and Denis Allard and Thomas Opitz and Liliane Bel and Thomas Romary

16 septembre 2016

Contents

1) Installing R packages

```
install.packages('raster')
install.packages('shiny')
install.packages('spacetime')
install.packages('leaflet')
install.packages('foreign')
install.packages('maptools')
install.packages('gstat')
install.packages('plm')
install.packages('sp')
install.packages('xts')
install.packages('rgeos')
install.packages('diveMove')
install.packages('maps')
install.packages('googleVis')
install.packages('rgdal')
install.packages('ggmap')
install.packages('data.table')
install.packages('mapdata')
install.packages('cshapes')
install.packages('rworldmap')
install.packages('dplyr')
install.packages('RColorBrewer')
install.packages('ncdf4')
install.packages('rworldmap')
install.packages('plotKML')
install.packages('tseries')
install.packages('ggfortify')
install.packages('ggplot2')
install.packages('CompRandFld')
install.packages('fields')
install.packages('spam')
install.packages('scatterplot3d')
install.packages('RandomFields')
```

2) Importing data and converting to STSDF and SDIDF spacetime formats.

2.a) Importing stations

```
### AirBase stations database (cleaned and purged) and their plot
stations <- read.csv("data/AirBase_v8_stations.csv",header=TRUE,sep="\t")
names(stations)

coord <- cbind(stations$station_longitude_deg,stations$station_latitude_deg)
stations <-stations[which(coord[,2]>20),]
coord <- cbind(stations$station_longitude_deg,stations$station_latitude_deg)
```

```r
not_duplicated <- which(!duplicated(coord))
stations <- stations[not_duplicated,]
coord <- cbind(stations$station_longitude_deg,stations$station_latitude_deg)

plot(coord)
save(stations,file='data/stations.Rdata')
```

2.b) Importing hourly data

```r
library(data.table)

file=list.files('data/OBS')
length(file)
head(file)

### Select only file containing hours data (file names with more than 15-17 characters)
filePM10 <- file[grep('PM10',file)]
filePM10 <- filePM10[nchar(filePM10)>17]
filePM25 <- file[grep('PM25',file)]
filePM25 <- filePM25[nchar(filePM25)>17]
fileO3   <- file[grep('O3',file)]
fileO3   <- fileO3[nchar(fileO3)>15]
fileNO2  <- file[grep('NO2',file)]
fileNO2  <- fileNO2[nchar(fileNO2)>16]
filePM10 <- paste('data/OBS/',filePM10,sep="")
filePM25 <- paste('data/OBS/',filePM25,sep="")
fileO3   <- paste('data/OBS/',fileO3,sep="")
fileNO2  <- paste('data/OBS/',fileNO2,sep="")

date_station <- gsub(".csv","",filePM10)
date_station <- gsub("PM10_","",date_station)

### Checking if all files have the same lenght and order
{
  length(filePM10)
  length(filePM25)
  length(fileO3)
  length(fileNO2)
  filePM10h <- gsub(".csv","",filePM10)
  filePM10h <- gsub("PM10_","",filePM10h)
  filePM25h <- gsub(".csv","",filePM25)
  filePM25h <- gsub("PM25_","",filePM25h)
  fileO3h   <- gsub(".csv","",fileO3)
  fileO3h   <- gsub("O3_","",fileO3h)
  fileNO2h  <- gsub(".csv","",fileNO2)
  fileNO2h  <- gsub("NO2_","",fileNO2h)
  all(filePM10h==filePM25h)
  all(filePM10h==fileO3h)
  all(filePM10h==fileNO2h)
}
```

```
### Preparing list for rbindList
PM10l <- vector(mode = "list", length = length(filePM10))
PM25l <- vector(mode = "list", length = length(filePM25))
O3l   <- vector(mode = "list", length = length(fileO3))
NO2l  <- vector(mode = "list", length = length(fileNO2))

for(i in 1:length(PM10l)){
  date  <- strptime(substr(date_station[i],10,19),format= "%Y%m%d%H")
  PM10t <- try(read.table(filePM10[i],header = FALSE,sep=" ",na.strings="-999",
                          col.names = c("ID","long","lat","PM10"),
                          colClasses = c("factor","numeric","numeric","numeric")))
  PM10t <- cbind(PM10t,rep(date,nrow(PM10t)))
  PM10l[[i]] <- PM10t
  PM25t <- try(read.table(filePM25[i],header = FALSE,sep=" ",na.strings="-999",
                          col.names = c("ID","long","lat","PM25"),
                          colClasses=c("factor","numeric","numeric","numeric")))
  PM25t <- cbind(PM25t,rep(date,nrow(PM25t)))
  PM25l[[i]] <- PM25t
  NO2t  <- try(read.table(fileNO2[i],header = FALSE,sep=" ",na.strings="-999",
                          col.names = c("ID","long","lat","NO2"),
                          colClasses=c("factor","numeric","numeric","numeric")))
  NO2t <- cbind(NO2t,rep(date,nrow(NO2t)))
  NO2l[[i]] <- NO2t
  O3t <- try(read.table(fileO3[i],header = FALSE,sep=" ",na.strings="-999",
                        col.names = c("ID","long","lat","O3"),
                        colClasses=c("factor","numeric","numeric","numeric")))
  O3t <- cbind(O3t,rep(date,nrow(O3t)))
  O3l[[i]] <- O3t
  cat(i,' ')
}

### Using rbindlist for faster row binding of all dataframes
PM10 <- rbindlist(PM10l)
names(PM10)[5] <- 'date'
PM10$ID2 <- paste(PM10$ID,PM10$date,sep='_')
PM25 <- rbindlist(PM25l)
names(PM25)[2:3] <- c("long1","lat1")
names(PM25)[5]   <- 'date1'
PM25$ID2 <- paste(PM25$ID,PM25$date1,sep='_')
NO2 <- rbindlist(NO2l)
names(NO2)[2:3] <- c("long2","lat2")
names(NO2)[5]   <- 'date2'
NO2$ID2 <- paste(NO2$ID,NO2$date2,sep='_')
O3 <- rbindlist(O3l)
names(O3)[2:3] <- c("long3","lat3")
names(O3)[5]   <- 'date3'
O3$ID2 <- paste(O3$ID,O3$date3,sep='_')


### Using data.table for faster merge
PM10 <- data.table(PM10)
PM25 <- data.table(PM25)
NO2  <- data.table(NO2)
```

```
O3   <- data.table(O3)
res  <- merge(PM10[,.(PM10,ID2,date,long,lat)],PM25[,.(PM25,ID2,date1,long1,lat1)],
              by="ID2",all=TRUE)
res  <- merge(res,NO2[,.(NO2,ID2,date2,long2,lat2)] ,by="ID2",all=TRUE)
res  <- merge(res, O3[,.(O3,ID2,date3,long3,lat3)] ,by="ID2",all=TRUE)
res$ID <- sapply(res$ID2, function(x) strsplit(x,"_")[[1]][1])
res$ID <- as.factor(res$ID)
res    <- data.frame(res)

### Finding locations and dates avoiding NAs
nodate <- is.na(res$date)
res$date[nodate] <- res$date1[nodate]
nodate <- is.na(res$date)
res$date[nodate] <- res$date2[nodate]
nodate <- is.na(res$date)
res$date[nodate] <- res$date3[nodate]

nolong <- is.na(res$long)
res$long[nolong] <- res$long1[nolong]
nolong <- is.na(res$long)
res$long[nolong] <- res$long2[nolong]
nolong <- is.na(res$long)
res$long[nolong] <- res$long3[nolong]

nolat <- is.na(res$lat)
res$lat[nolat] <- res$lat1[nolat]
nolat <- is.na(res$lat)
res$lat[nolat] <- res$lat2[nolat]
nolat <- is.na(res$lat)
res$lat[nolat] <- res$lat3[nolat]

### Building the final dataframe
OBS_horaire <- res[!is.na(res$date),c('ID','long','lat','date','PM10','PM25','NO2','O3')]
summary(OBS_horaire)

save(OBS_horaire,file='data/OBS_horaire.Rdata')
```

2.c) Importing daily data

```
file <- list.files('data/OBS')
length(file)
head(file)

filePM10 <- file[grep('PM10',file)]
filePM10 <- filePM10[nchar(filePM10)<=17]
filePM25 <- file[grep('PM25',file)]
filePM25 <- filePM25[nchar(filePM25)<=17]
fileO3   <- file[grep('O3',file)]
fileO3   <- fileO3[nchar(fileO3)<=15]
fileNO2  <- file[grep('NO2',file)]
fileNO2  <- fileNO2[nchar(fileNO2)<=16]
```

```
date_station <- gsub(".csv","",filePM10)
date_station <- gsub("PM10_","",date_station)

### Checking if all files have the same length and order
{
  length(filePM10)
  length(filePM25)
  length(fileO3)
  length(fileNO2)
  filePM10h <- gsub(".csv","",filePM10)
  filePM10h <- gsub("PM10_","",filePM10h)
  filePM25h <- gsub(".csv","",filePM25)
  filePM25h <- gsub("PM25_","",filePM25h)
  fileO3h   <- gsub(".csv","",fileO3)
  fileO3h   <- gsub("O3_","",fileO3h)
  fileNO2h  <- gsub(".csv","",fileNO2)
  fileNO2h  <- gsub("NO2_","",fileNO2h)
  all(filePM10h==filePM25h)
  all(filePM10h==fileO3h)
  all(filePM10h==fileNO2h)
}

PM10l <- vector(mode = "list", length = length(filePM10))
PM25l <- vector(mode = "list", length = length(filePM25))
O3l   <- vector(mode = "list", length = length(fileO3))
NO2l  <- vector(mode = "list", length = length(fileNO2))

filePM10 <- paste('data/OBS/',filePM10,sep="")
filePM25 <- paste('data/OBS/',filePM25,sep="")
fileO3   <- paste('data/OBS/',fileO3,sep="")
fileNO2  <- paste('data/OBS/',fileNO2,sep="")

for(i in 1:length(PM10l)){
      date  <- strptime(date_station[i],format= "%Y%m%d")
      PM10t <- try(read.table(filePM10[i],header = FALSE,sep=" ",na.strings="-999",
      col.names = c("ID","long","lat","PM10"),colClasses = c("factor","numeric","numeric","numeric")))
      PM10t <- cbind(PM10t,rep(date,nrow(PM10t)))
      PM10l[[i]] <- PM10t
      PM25t <- try(read.table(filePM25[i],header = FALSE,sep=" ",na.strings="-999",
      col.names = c("ID","long","lat","PM25"),colClasses = c("factor","numeric","numeric","numeric")))
      PM25t <- cbind(PM25t,rep(date,nrow(PM25t)))
      PM25l[[i]] <- PM25t
      NO2t <- try(read.table(fileNO2[i],header = FALSE,sep=" ",na.strings="-999",
      col.names = c("ID","long","lat","NO2"),colClasses = c("factor","numeric","numeric","numeric")))
      NO2t <- cbind(NO2t,rep(date,nrow(NO2t)))
      NO2l[[i]] <- NO2t
      O3t <- try(read.table(fileO3[i],header = FALSE,sep=" ",na.strings="-999",
      col.names = c("ID","long","lat","O3"),colClasses = c("factor","numeric","numeric","numeric")))
      O3t <- cbind(O3t,rep(date,nrow(O3t)))
      O3l[[i]] <- O3t
      cat(i,'')
}
```

```
### Using rbindlist for faster row binding of all loaded dataframes
PM10 <- rbindlist(PM10l)
names(PM10)[5]='date'
PM10$ID2 <- paste(PM10$ID,PM10$date,sep='_')
PM25 <- rbindlist(PM25l)
names(PM25)[2:3] <- c("long1","lat1")
names(PM25)[5]   <- 'date1'
PM25$ID2 <- paste(PM25$ID,PM25$date1,sep='_')
NO2 <- rbindlist(NO2l)
names(NO2)[2:3] <- c("long2","lat2")
names(NO2)[5]   <- 'date2'
NO2$ID2 <- paste(NO2$ID,NO2$date2,sep='_')
O3 <- rbindlist(O3l)
names(O3)[2:3] <- c("long3","lat3")
names(O3)[5]   <- 'date3'
O3$ID2 <- paste(O3$ID,O3$date3,sep='_')

### Using data.table for faster merge
PM10 <- data.table(PM10)
PM25 <- data.table(PM25)
NO2  <- data.table(NO2)
O3   <- data.table(O3)
res <- merge(PM10[,.(PM10,ID2,date,long,lat)],PM25[,.(PM25,ID2,date1,long1,lat1)],
           by="ID2",all=TRUE)
res <- merge(res,NO2[,.(NO2,ID2,date2,long2,lat2)] ,by="ID2",all=TRUE)
res <- merge(res, O3[,.(O3,ID2,date3,long3,lat3)] ,by="ID2",all=TRUE)

res$ID<-sapply(res$ID2, function(x) strsplit(x,'_')[[1]][1])
res$ID<-as.factor(res$ID)

res=data.frame(res)

### Selecting the filled dates and locations
nodate <- is.na(res$date)
res$date[nodate] <- res$date1[nodate]
nodate <- is.na(res$date)
res$date[nodate] <- res$date2[nodate]
nodate=is.na(res$date)
res$date[nodate] <- res$date3[nodate]

nolong <- is.na(res$long)
res$long[nolong] <- res$long1[nolong]
nolong <- is.na(res$long)
res$long[nolong] <- res$long2[nolong]
nolong <- is.na(res$long)
res$long[nolong] <- res$long3[nolong]

nolat <- is.na(res$lat)
res$lat[nolat] <- res$lat1[nolat]
nolat <- is.na(res$lat)
res$lat[nolat] <- res$lat2[nolat]
nolat <- is.na(res$lat)
res$lat[nolat] <- res$lat3[nolat]
```

```
OBS_jour <- res[!is.na(res$date),c('ID','long','lat','date','PM10','PM25','NO2','O3')]
summary(OBS_jour)
save(OBS_jour,file='data/OBS_jour.Rdata')
```

2.d) Importing CHIMERE grid data

```
library(ncdf4)

file <- list.files('data/CHM')
length(file)
head(file)

file <- file[grep('chim_',file,fixed=TRUE)]
file <- file[grep('.nc',file)]
file <- paste('data/CHM/',file,sep='')

a <- vector(mode = "list", length = length(file))

for(i in 1:length(file)){
  ncfile <- nc_open(file[i])
  print(file[i])
  lat  <- as.vector(ncvar_get(ncfile,"lat"))
  lon  <- as.vector(ncvar_get(ncfile,"lon"))
  PM10 <- as.vector(ncvar_get(ncfile,"PM10"))
  PM25 <- as.vector(ncvar_get(ncfile,"PM25"))
  NO2  <- as.vector(ncvar_get(ncfile,"NO2"))
  O3   <- as.vector(ncvar_get(ncfile,"O3"))
  time <- strptime(strsplit(file[i],'.',fixed = TRUE)[[1]][2],format = "%Y%m%d")
  dfr  <- data.frame(lon=lon,lat=lat,PM10=PM10,PM25=PM25,NO2=NO2,O3=O3,
                     time=rep(time,each=length(lon)))
  a[[i]]<-dfr
}
CHM <- rbindlist(a)
CHM <- data.frame(CHM)

save(CHM,file='data/CHM.Rdata')
```

2.e) Converting dataframes to spacetime objects

```
rm(list=ls())
library(spacetime)
load('data/OBS_horaire.Rdata')
load('data/OBS_jour.Rdata')
load('data/CHM.Rdata')
load('data/stations.Rdata')

### WARNING: daily data contain 2 stations with only 214 recorded days.
### This can cause problems for the construction of a ST object
```

```
STFDF_jour <- stConstruct(OBS_jour,space=c('long','lat'),time='date',
                          SpatialObj=SpatialPoints(OBS_jour[,c('long','lat')]))
STFDF_jour    <- as(STFDF_jour, "STFDF")
STFDF_30jour  <- STFDF_jour[,"2014-01-01::2014-01-31"]


### The entire dataset is too large and difficult to handle
### STIDF_obs=stConstruct(OBS_horaire,space=c('long','lat'),time='date')
### with long format and STIDF dimension s and t are wrong.
### For monitoring station data, spacetime::stplot is particularly useful if
### it knows that these are repeated measurements for fixed stations.
### Provided with an STIDF, stplot assumes they are not.
### Pebesma https://stat.ethz.ch/pipermail/r-sig-geo/2015-July/023101.html)
#STIDF_obs =as(STIDF_obs, "STFDF") ### too long

### Considering two subsamples of hourly data of 30 or 7 days

STIDF_Horaire_30j <- stConstruct(OBS_horaire[OBS_horaire$date>= "2013-12-31" &
                     OBS_horaire$date <= "2014-02-01",],space=c('long','lat'),time='date')
STIDF_Horaire_30j <- as(STIDF_Horaire_30j, "STSDF")

STIDF_Horaire_7j <- stConstruct(OBS_horaire[OBS_horaire$date>= "2013-12-31" &
                     OBS_horaire$date <= "2014-01-08",],space=c('long','lat'),time='date')
STIDF_Horaire_7j <- as(STIDF_Horaire_7j, "STSDF")

CHM_jour <- stConstruct(CHM[CHM$time>= "2013-12-31" & CHM$time <= "2014-02-01",],
                      space=c('lon','lat'),time='time')
CHM_jour <- as(CHM_jour, "STSDF")


add_ID<-function(STFDF){
  ID  <- paste('ID_',1:(dim(STFDF)[1]),sep='')
  IDs <- rep(ID,dim(STFDF)[2])
  attr(STFDF,'data') <- cbind(IDs,attr(STFDF,'data'))
  return(STFDF)
}
CHM_jour=add_ID(CHM_jour)

### Adding projections
library(sp)
proj4string(CHM_jour)   <- "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"
proj4string(STFDF_jour) <- "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"
proj4string(STIDF_Horaire_30j) <- "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"
proj4string(STIDF_Horaire_7j)  <- "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"

save(CHM_jour,file='data/CHM_jour.Rdata')
save(STFDF_jour,file='data/STFDF_jour.Rdata')
save(STIDF_Horaire_30j,file='data/STIDF_Horaire_30j.Rdata')
save(STIDF_Horaire_7j,file='data/STIDF_Horaire_7j.Rdata')
```

```
rm(list=ls())
load('data/STFDF_jour.Rdata')
```

```
### Understanding STFDF object
dim(STFDF_jour)
```

```
##     space      time variables
##       507       365         5
```

```
str(STFDF_jour)
```

```
## Formal class 'STFDF' [package "spacetime"] with 4 slots
##   ..@ data   :'data.frame':  185055 obs. of  5 variables:
##   .. ..$ ID  : Factor w/ 507 levels "FR01001","FR01005",..: 1 2 3 4 5 6 7 8 9 10 ...
##   .. ..$ PM10: num [1:185055] 6.8 9.3 8.1 9.3 12 6.9 8.3 7.7 6.2 NA ...
##   .. ..$ PM25: num [1:185055] NA NA NA NA 8.5 NA NA NA NA NA ...
##   .. ..$ NO2 : num [1:185055] 5.6 6.4 8.6 NA 10 8.3 7.8 NA 9.1 10 ...
##   .. ..$ O3  : num [1:185055] 52 NA NA NA 45 NA 55 NA NA 56 ...
##   ..@ sp     :Formal class 'SpatialPoints' [package "sp"] with 3 slots
##   .. .. ..@ coords     : num [1:507, 1:2] 5.8 6.08 6.14 6.13 6.18 ...
##   .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. .. ..$ : chr [1:507] "1" "366" "731" "1096" ...
##   .. .. .. .. ..$ : chr [1:2] "long" "lat"
##   .. .. ..@ bbox       : num [1:2, 1:2] -4.49 41.92 9.48 51.05
##   .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. .. ..$ : chr [1:2] "long" "lat"
##   .. .. .. .. ..$ : chr [1:2] "min" "max"
##   .. .. ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
##   .. .. .. .. ..@ projargs: chr "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs +towgs84=0,0,0"
##   ..@ time   :An 'xts' object on 2014-01-01/2014-12-31 containing:
##   Data: int [1:365, 1] 1 2 3 4 5 6 7 8 9 10 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr "timeIndex"
##   Indexed by objects of class: [POSIXct,POSIXt] TZ:
##   xts Attributes:
##  NULL
##   ..@ endTime: POSIXct[1:365], format: "2014-01-02" ...
```

```
names(STFDF_jour@data)
```

```
## [1] "ID"   "PM10" "PM25" "NO2"  "O3"
```

```
dim(STFDF_jour@data)
```

```
## [1] 185055      5
```

```
as.numeric(dim(STFDF_jour)[1]*dim(STFDF_jour)[2])
```

```
## [1] 185055
```

```
### Select and subset of STFDF object: STFDF[location_subset,time_subset,var_selection]
### Some examples:
#### 1) Select first location, first day, second variable
STFDF_jour[1,1,2]
```

```
## [1] 6.8
```

```
#### object of class numeric
class(STFDF_jour[1,1,2])
```

```
## [1] "numeric"
```

```
#### 2) Select first two locations, first day, second variable
STFDF_jour[1:2,1,2]
```

```
##            coordinates PM10
## 1   (5.804448, 49.535)  6.8
## 2 (6.076108, 49.32472)  9.3
```

```
####  object of class SpatialPointDataFrame
class(STFDF_jour[1:2,1,2])
```

```
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

```
#### 3) Select first location, first two days, second variable
STFDF_jour[1,1:2,2]
```

```
##            PM10 timeIndex
## 2014-01-01  6.8         1
## 2014-01-02  4.0         2
```

```
####  object of class xts (time serie)
class(STFDF_jour[1,1:2,2])
```

```
## [1] "xts" "zoo"
```

```
#### 4) Select first ten locations, first two days and ID + PM10 variables
STFDF_jour[1:10,1:2,1:2]
```

```
## An object of class "STFDF"
## Slot "data":
##         ID PM10
## 1  FR01001  6.8
## 2  FR01005  9.3
## 3  FR01006  8.1
## 4  FR01009  9.3
```

```
## 5  FR01011 12.0
## 6  FR01012  6.9
## 7  FR01014  8.3
## 8  FR01015  7.7
## 9  FR01016  6.2
## 10 FR01018   NA
## 11 FR01001  4.0
## 12 FR01005  6.5
## 13 FR01006  8.6
## 14 FR01009  6.2
## 15 FR01011 11.0
## 16 FR01012 13.0
## 17 FR01014  9.6
## 18 FR01015  7.4
## 19 FR01016  4.8
## 20 FR01018   NA
##
## Slot "sp":
## SpatialPoints:
##          long      lat
## 1    5.804448 49.53500
## 366  6.076108 49.32472
## 731  6.140000 49.32083
## 1096 6.132500 49.27139
## 1461 6.180833 49.11941
## 1826 6.223336 49.11028
## 2191 6.058336 48.91500
## 2556 6.051392 48.88473
## 2921 6.093611 48.88636
## 3286 6.122772 49.10750
## Coordinate Reference System (CRS) arguments: +proj=longlat
## +ellps=WGS84 +datum=WGS84 +no_defs +towgs84=0,0,0
##
## Slot "time":
##            timeIndex
## 2014-01-01         1
## 2014-01-02         2
##
## Slot "endTime":
## [1] "2014-01-02 CET" "2014-01-03 CET"
```

```
####  object of class STFDF
class(STFDF_jour[1:10,1:2,1:2])
```

```
## [1] "STFDF"
## attr(,"package")
## [1] "spacetime"
```

```
#### 5) select 10 days time interval (using date format) and two variables (PM10 and 03)
STFDF_jour[1,"2014-01-01::2014-01-10",c('PM10','03')]
```

```
##            PM10 03 timeIndex
## 2014-01-01  6.8 52         1
```

```
## 2014-01-02  4.0 55         2
## 2014-01-03 10.0 55         3
## 2014-01-04 10.0 44         4
## 2014-01-05  7.9 57         5
## 2014-01-06  5.8 48         6
## 2014-01-07  8.7 50         7
## 2014-01-08  8.8 33         8
## 2014-01-09  6.1 49         9
## 2014-01-10  9.2 37        10
```

```
#### 6) select one station by ID (not as natural as expected), 10 days by date and 2 variables
IDs=levels(STFDF_jour$ID)
STFDF_jour[which(IDs %in% c('FR01011')),"2014-01-01::2014-01-10",c('PM10','O3')]
```

```
##            PM10 O3 timeIndex
## 2014-01-01   12 45         1
## 2014-01-02   11 43         2
## 2014-01-03   16 40         3
## 2014-01-04   16 27         4
## 2014-01-05   11 51         5
## 2014-01-06   13 33         6
## 2014-01-07   14 25         7
## 2014-01-08   16 20         8
## 2014-01-09   14 45         9
## 2014-01-10   22 23        10
```

```
#### 7) select 2 stations by ID (not as natural as expected), 10 days by date and 2 variables
temp<-STFDF_jour[which(IDs %in% c('FR01011','FR03063')),"2014-01-01::2014-01-10",c('PM10','O3')]

### Get dataframe from spacetime STFDF/STIDF
mydata<-temp@data
summary(mydata)
```

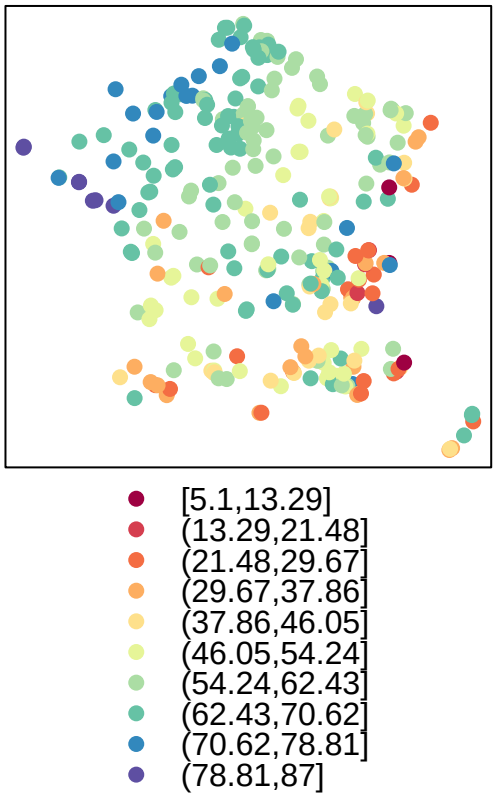
```
##       PM10             O3
##  Min.   :11.00   Min.   :16.00
##  1st Qu.:12.25   1st Qu.:27.75
##  Median :14.00   Median :39.00
##  Mean   :14.50   Mean   :37.40
##  3rd Qu.:16.00   3rd Qu.:45.00
##  Max.   :22.00   Max.   :69.00
##  NA's   :10
```

3) Plots of STFDF/STIDF

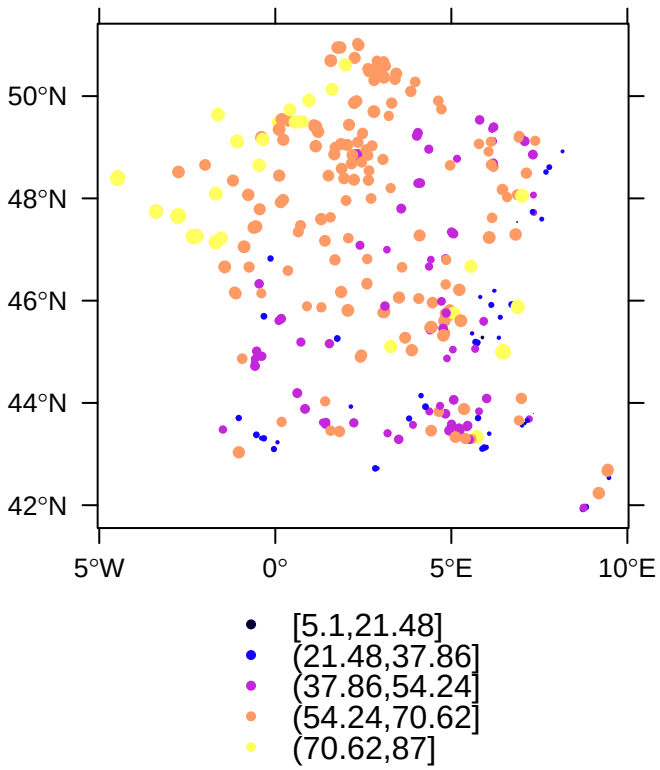
3.a) Static plots

```
#####################################
rm(list=ls())
load('data/STFDF_jour.Rdata')
library(RColorBrewer)
```

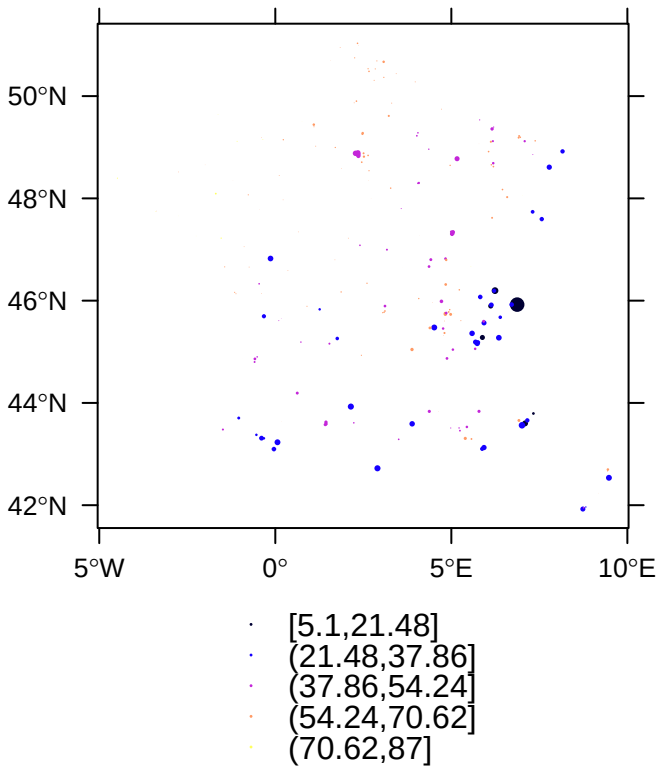
```
library(sp)

###  Some examples of spplot
#### 1) Plot all locations, 1 day (by date), 1 variable (O3)
spplot(STFDF_jour[,"2014-01-01",'O3'],col.regions=brewer.pal(10,"Spectral"),cuts=10)
```




```
#### 2) The same, but point size is proportional to O3 value for that station
cx03 = attr(STFDF_jour[,"2014-01-01",'O3'],'data')/max(attr(STFDF_jour[,"2014-01-01",'O3'],
            'data'),na.rm=T) * 5
spplot(STFDF_jour[,"2014-01-01",'O3'],cex= 0.2*as.numeric(cx03$O3),scales=list(draw=TRUE))
```




```
#### 3) Plot all locations, 1 day (by date), 2 variables (O3 and NO2)
#### point size is proportional to NO2 value, color to O3 value
cxNO2 = attr(STFDF_jour[,"2014-01-01",'NO2'],'data')/max(attr(STFDF_jour[,"2014-01-01",'NO2']
            ,'data'),na.rm=T) * 5
spplot(STFDF_jour[,"2014-01-01",'O3'],cex= 0.2*as.numeric(cxNO2$NO2),scales=list(draw=TRUE))
```



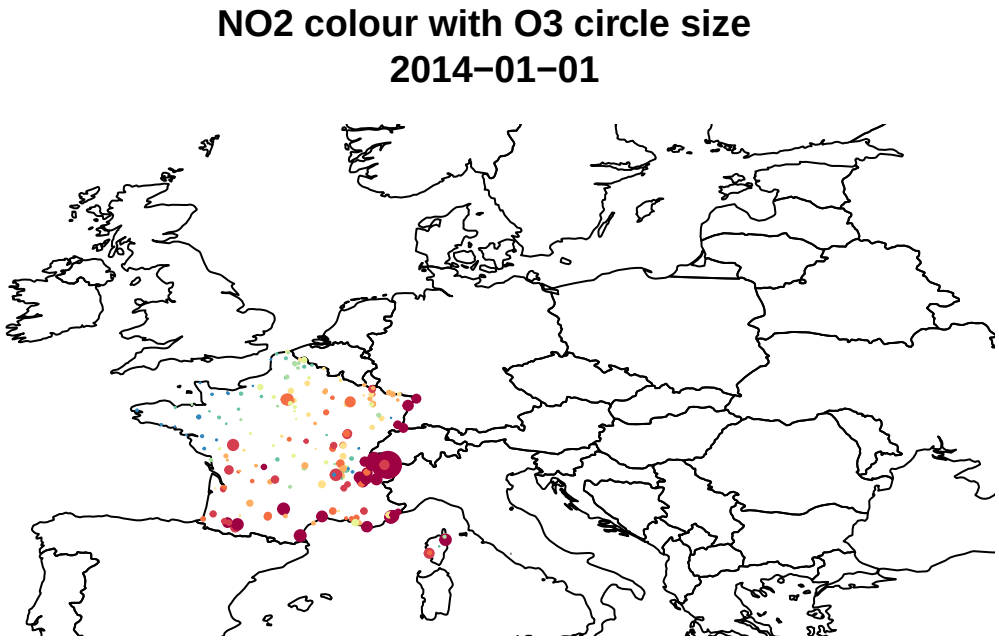

```
### Download and add map of countries
par(mfrow=c(1,1))
library(rworldmap)
library(maptools)
library(ggmap)
world <- getMap(resolution="low")
proj4string(world)
```

```
## [1] "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"
```

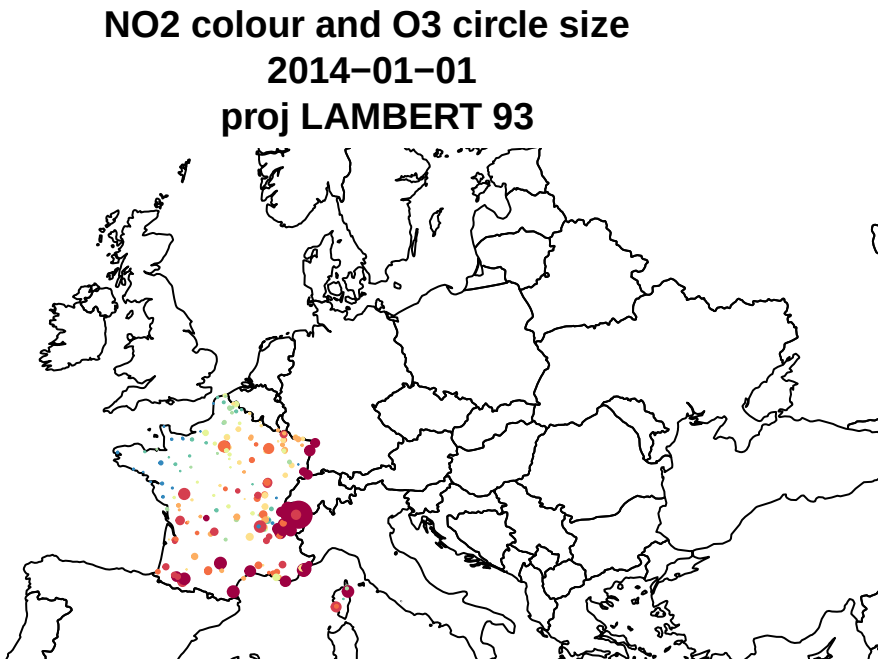
```
europe.limits <- geocode(c("Spain","Greece","Ireland","Norway","Ukraine"))
```

```
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Spain&sensor=false

## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Greece&sensor=false

## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Ireland&sensor=false

## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Norway&sensor=false

## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Ukraine&sensor=false
```

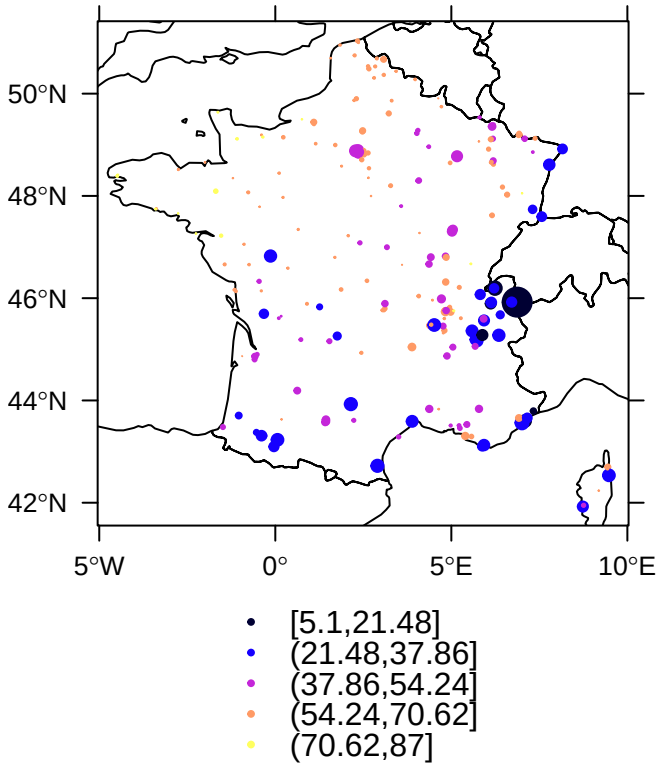
```
plot(world,xlim=range(europe.limits$lon),ylim=range(europe.limits$lat),asp=1,
     main='NO2 colour with O3 circle size \n 2014-01-01')
colours <- brewer.pal(10,"Spectral")
O3      <- attr(STFDF_jour[,"2014-01-01",'O3'],'data')
plot(STFDF_jour[,"2014-01-01",'O3'],col=colours[findInterval(O3$O3,
     quantile(O3$O3,seq(0,1,length.out =10),na.rm=T), all.inside=TRUE)],
     cex= 0.4*as.numeric(cxNO2$NO2),scales=list(draw=TRUE),add=TRUE,pch=19)
```



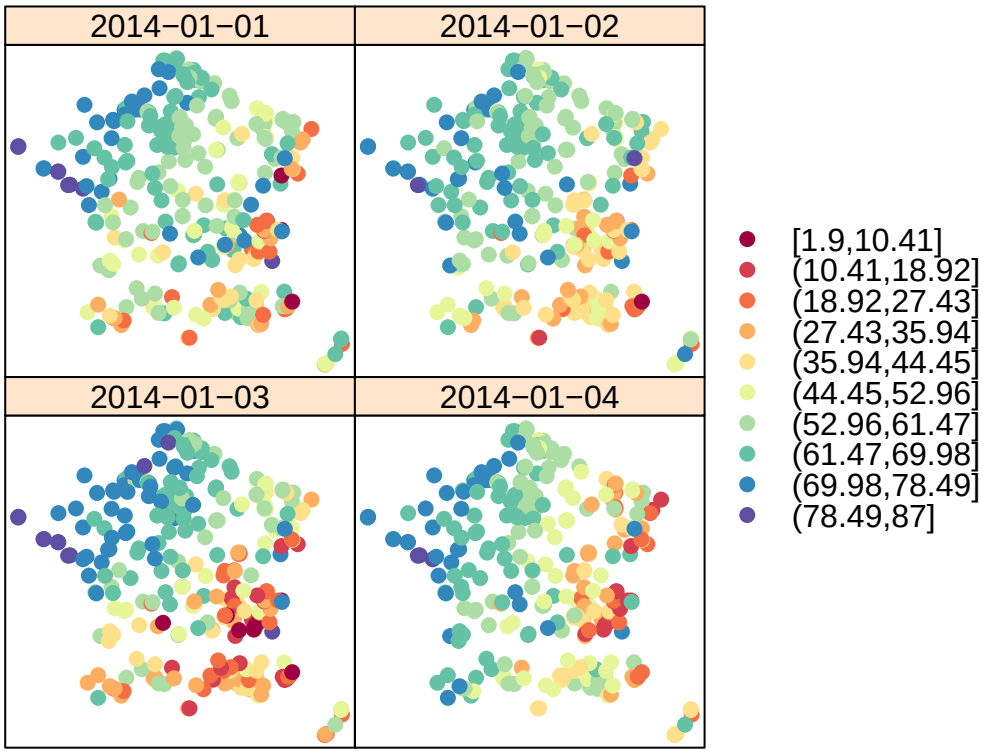

```
### Same as before but using Lambert 93 (L93) projection
splimit <- SpatialPoints(europe.limits)
proj4string(splimit) <- proj4string(world)
splimit_l93 <- spTransform(splimit,CRSobj=CRS("+init=epsg:2154"))
worldL93    <- spTransform(world,CRSobj=CRS("+init=epsg:2154"))
plot(worldL93,xlim=range(splimit_l93$lon),ylim=range(splimit_l93$lat),asp=1,
     main='NO2 colour and O3 circle size \n 2014-01-01 \n proj LAMBERT 93')
STFDF_jourL93 <- STFDF_jour
STFDF_jourL93 <- spTransform(STFDF_jourL93,CRSobj=CRS("+init=epsg:2154"))
plot(STFDF_jourL93[,"2014-01-01",'O3'],col=colours[findInterval(O3$O3,
    quantile(O3$O3,seq(0,1,length.out =10),na.rm=T), all.inside=TRUE)],
    cex= 0.4*as.numeric(cxNO2$NO2),scales=list(draw=TRUE),add=TRUE,pch=19)
```



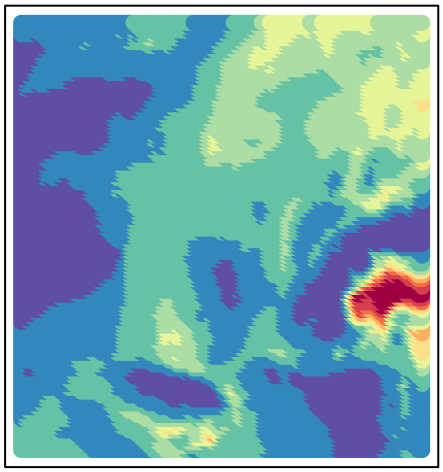
```
### The same, using spplot (easier and centered around our data)
spplot(STFDF_jour[,"2014-01-01",'O3'],cex=0.4*as.numeric(cxNO2$NO2),
       scales=list(draw=TRUE),pch=19,sp.layout =list(world,first=TRUE))
```




```
###################################

### Some examples of stplot
#### 1) Plot all locations, 4 day (by date), 1 variable (O3)
stplot(STFDF_jour[,"2014-01-01::2014-01-04",'O3'],col.regions=
       brewer.pal(10,"Spectral"),cuts=10)
```

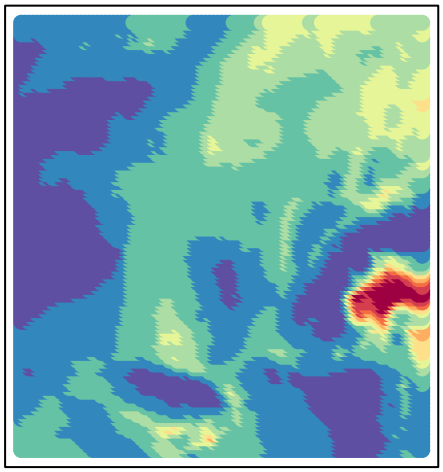



```
#### 2) Plot GRID data, 1 day (by date), 1 variable (O3)
load('data/CHM_jour.Rdata')
chim_O3_1j<-CHM_jour[,"2014-01-01",'O3']
spplot(chim_O3_1j,col.regions=brewer.pal(10,"Spectral"),cuts=10)
```



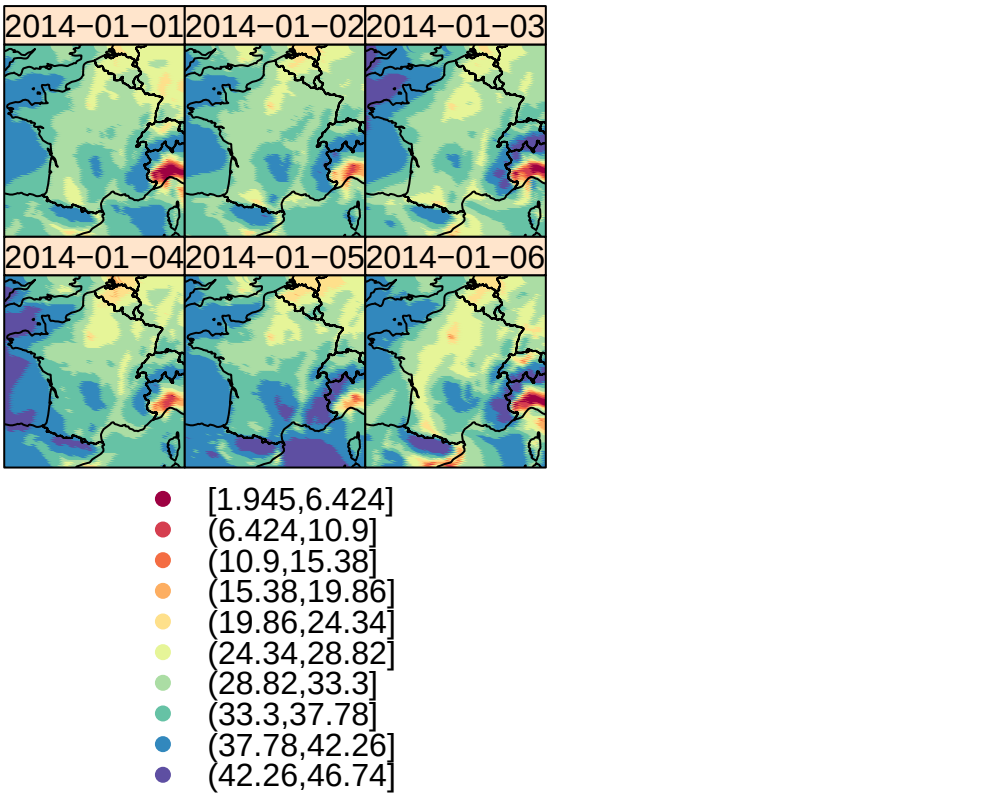
- [1.945,5.932]
- (5.932,9.919]
- (9.919,13.91]
- (13.91,17.89]
- (17.89,21.88]
- (21.88,25.87]
- (25.87,29.86]
- (29.86,33.84]
- (33.84,37.83]
- (37.83,41.82]

```
#### 3) with grid data as real gridded object
spplot(chim_03_1j,col.regions=brewer.pal(10,"Spectral"),cuts=10)
```

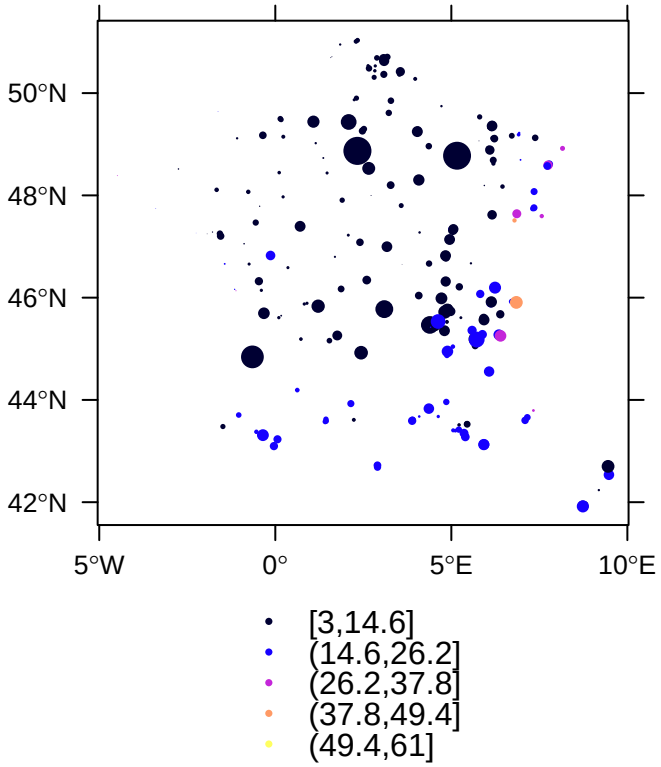


- [1.945,5.932]
- (5.932,9.919]
- (9.919,13.91]
- (13.91,17.89]
- (17.89,21.88]
- (21.88,25.87]
- (25.87,29.86]
- (29.86,33.84]
- (33.84,37.83]
- (37.83,41.82]

```
#### 4) with a STFDF we use gridded(STFDF@sp), 7 days
chim_O3_10j <- CHM_jour[,"2014-01-01::2014-01-7",'O3']
stplot(chim_O3_10j,col.regions=brewer.pal(10,"Spectral"),cuts=10,
       sp.layout=list(world,first=FALSE))
```




```
#### 5) add and plot new variable (ratio between NO2 and PM10)
NO2   <- attr(STFDF_jour[,"2014-01-01",'NO2'],'data')
PM10  <- attr(STFDF_jour[,"2014-01-01",'PM10'],'data')
ratio <- NO2$NO2/PM10$PM10
spplot(STFDF_jour[,"2014-01-01",'PM10'],cex=0.4*as.numeric(ratio),
       scales=list(draw=TRUE))
```



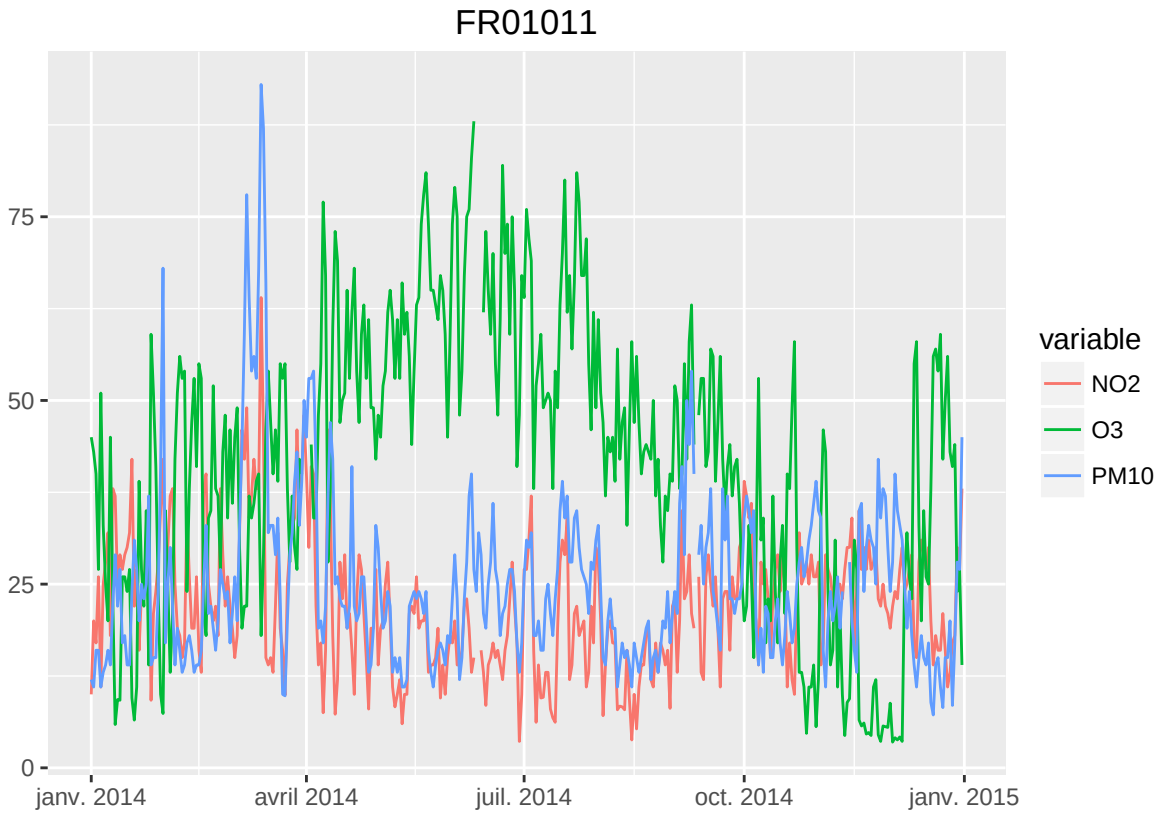
3.b) Static plots using klm (plotklm)

```
### May require an update of Google Earth
library(plotKML)
O3df    <- attr(STFDF_jour[,"2014-01-01","O3"],'data')
NO2df   <- attr(STFDF_jour[,"2014-01-01","NO2"],'data')
PM10df  <- attr(STFDF_jour[,"2014-01-01","PM10"],'data')
colours <- SAGA_pal[[1]]

### plotKML with colours for O3
plotKML(STFDF_jour[,"2014-01-01",'O3'],colours=colours[findInterval(O3df$O3,
        quantile(O3df$O3,seq(0,1,length.out =10),na.rm=T), all.inside=TRUE)])

### same for NO2 with colour_scale
plotKML(STFDF_jour[,"2014-01-01",'NO2'])
```

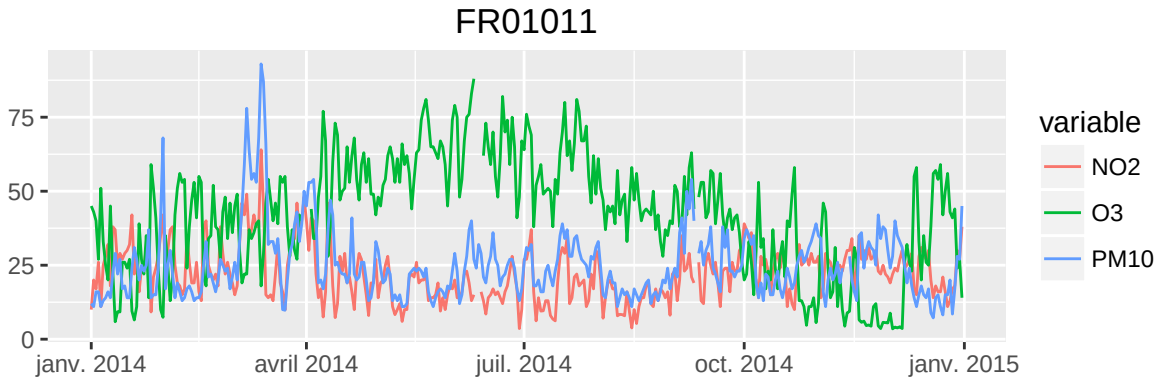
3.c) Time series plots

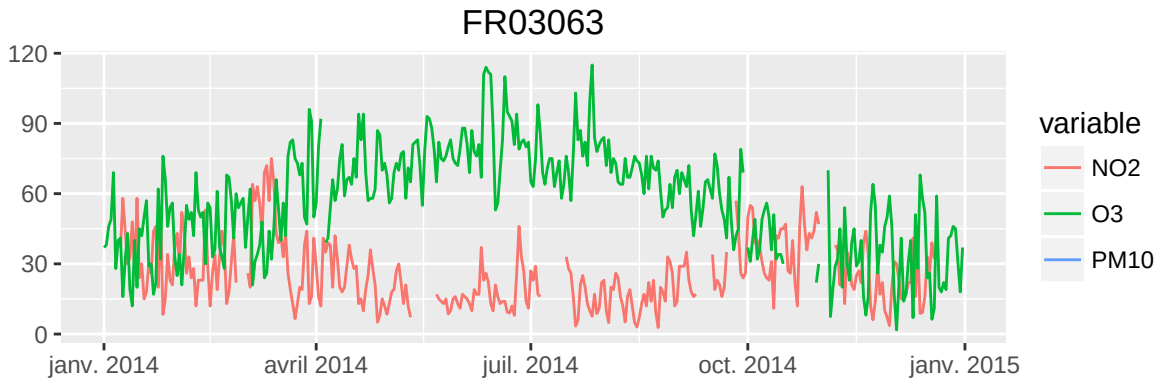
```
rm(list=ls())
library(tseries)
library(ggfortify)
library(ggplot2)
load('data/STFDF_jour.Rdata')
load('data/OBS_jour.Rdata')
```

```
### One time serie plot can represent the data corresponding to one or
### multiple locations as well as one or multiple variables.

#### 1) single location, multiple variables
IDs <- levels(OBS_jour$ID)
ts  <- STFDF_jour[which(IDs %in% c('FR01011')),,c("PM10","NO2","O3")]
colours <- brewer.pal(3,"Set1")
irr     <- irts(index(ts),as.matrix(ts[,1:3]))

p1 <- autoplot(irr, facets = FALSE)
p1 <- p1+ggtitle('FR01011')
p1
```

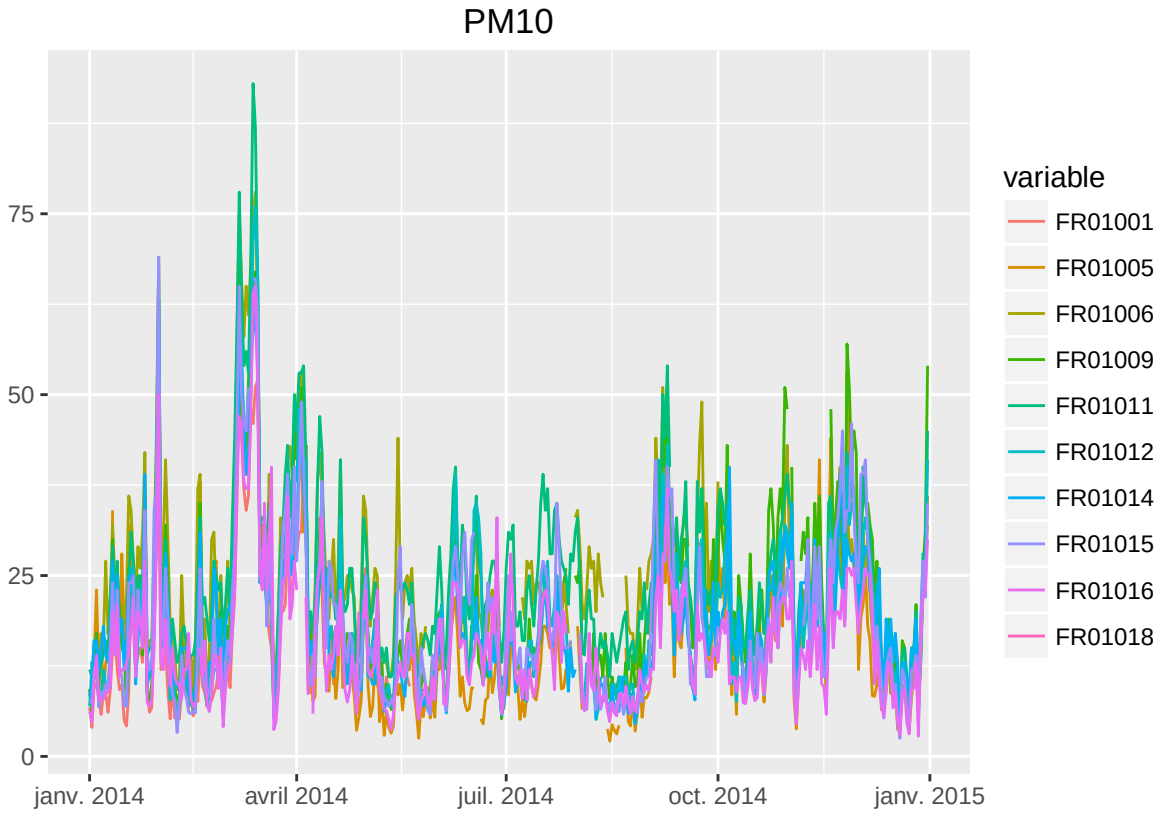



```
#### 2) Two locations, multiple variables
ts2  <- STFDF_jour[which(IDs %in% c('FR03063')),,c("PM10","NO2","O3")]
irr2 <- irts(index(ts2),as.matrix(ts2[,1:3]))
p2   <- autoplot(irr2, facets = FALSE)
p2   <- p2+ggtitle('FR03063')
new("ggmultiplot",plots=list(p1,p2),ncol=1)
```

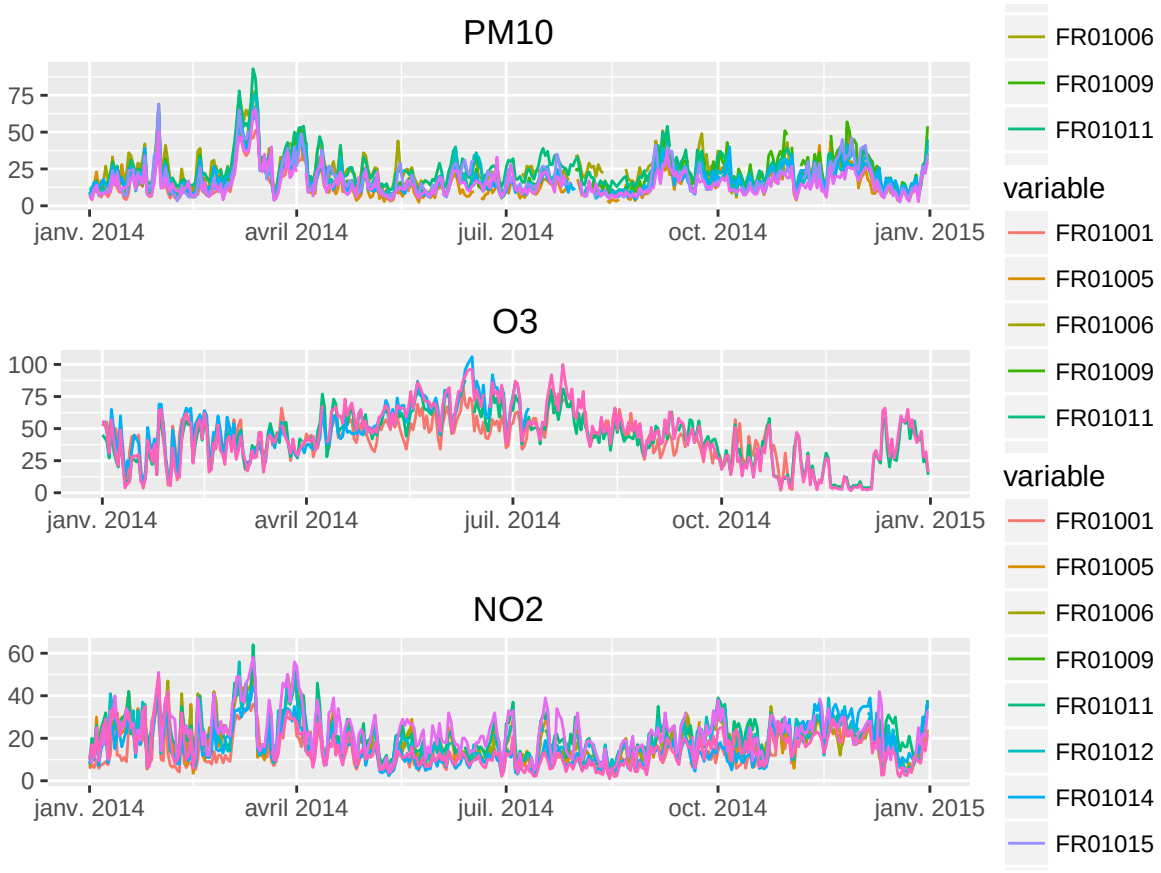




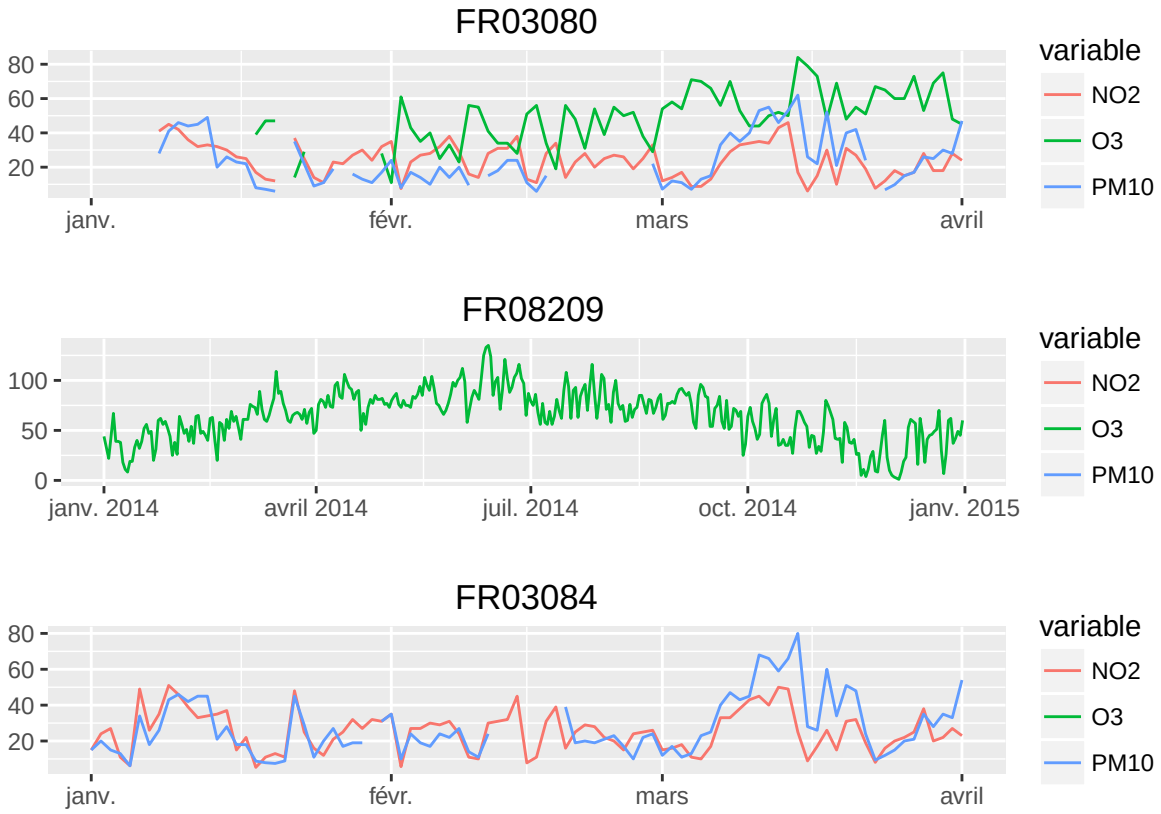

```
#### 3)  Multiple locations >3, single variable
##### PM10
init <- STFDF_jour[which(IDs==IDs[1]),,c("PM10")]
   a <- matrix(0,ncol=10,nrow=length(init[,1]))
for(i in 1:10 ){
  ID    <- IDs[i]
  a[,i] <- STFDF_jour[which(IDs==ID),,c("PM10")][,1]
}
colnames(a) <- IDs[1:10]
irr <- irts(index(init),a)
p3  <- autoplot(irr, facets = FALSE)+ggtitle('PM10')
p3
```




```
####  4) Multiple locations >3, multiples variables (grouped by variable)

#####  O3
init <- STFDF_jour[which(IDs==IDs[1]),,c("O3")]
a    <- matrix(0,ncol=10,nrow=length(init[,1]))
for(i in 1:10 ){
  ID    <- IDs[i]
  a[,i] <- STFDF_jour[which(IDs==ID),,c("O3")][,1]
}
colnames(a) <- IDs[1:10]
irr <- irts(index(init),a)
p4  <- autoplot(irr, facets = FALSE)+ggtitle('O3')

##### NO2
init <- STFDF_jour[which(IDs==IDs[1]),,c("NO2")]
a    <- matrix(0,ncol=10,nrow=length(init[,1]))
for(i in 1:10 ){
  ID    <- IDs[i]
  a[,i] <- STFDF_jour[which(IDs==ID),,c("NO2")][,1]
}
colnames(a) <- IDs[1:10]
irr <- irts(index(init),a)
p5  <- autoplot(irr, facets = FALSE)+ggtitle('NO2')

new("ggmultiplot",plots=list(p3,p4,p5),ncol=1)
```




```
####  5) As before, but grouping by location
IDs <- levels(OBS_jour$ID)
ts  <- STFDF_jour[which(IDs %in% c('FR03080')),"2014-01-01::2014-04-01",c("PM10","NO2","O3")]
colours <- brewer.pal(3,"Set1")
irr <- irts(index(ts),as.matrix(ts[,1:3]))
p1  <- autoplot(irr, facets = FALSE)
p1  <- p1+ggtitle('FR03080')

ts2  <- STFDF_jour[which(IDs %in% c('FR08209')),"2014-01-01::2015-04-01",c("PM10","NO2","O3")]
irr2 <- irts(index(ts2),as.matrix(ts2[,1:3]))
p2   <- autoplot(irr2, facets = FALSE)
p2   <- p2+ggtitle('FR08209')

ts2  <- STFDF_jour[which(IDs %in% c('FR03084')),"2014-01-01::2014-04-01",c("PM10","NO2","O3")]
irr2 <- irts(index(ts2),as.matrix(ts2[,1:3]))
p3   <- autoplot(irr2, facets = FALSE)
p3   <- p3+ggtitle('FR03084')

new("ggmultiplot",plots=list(p1,p2,p3),ncol=1)
```



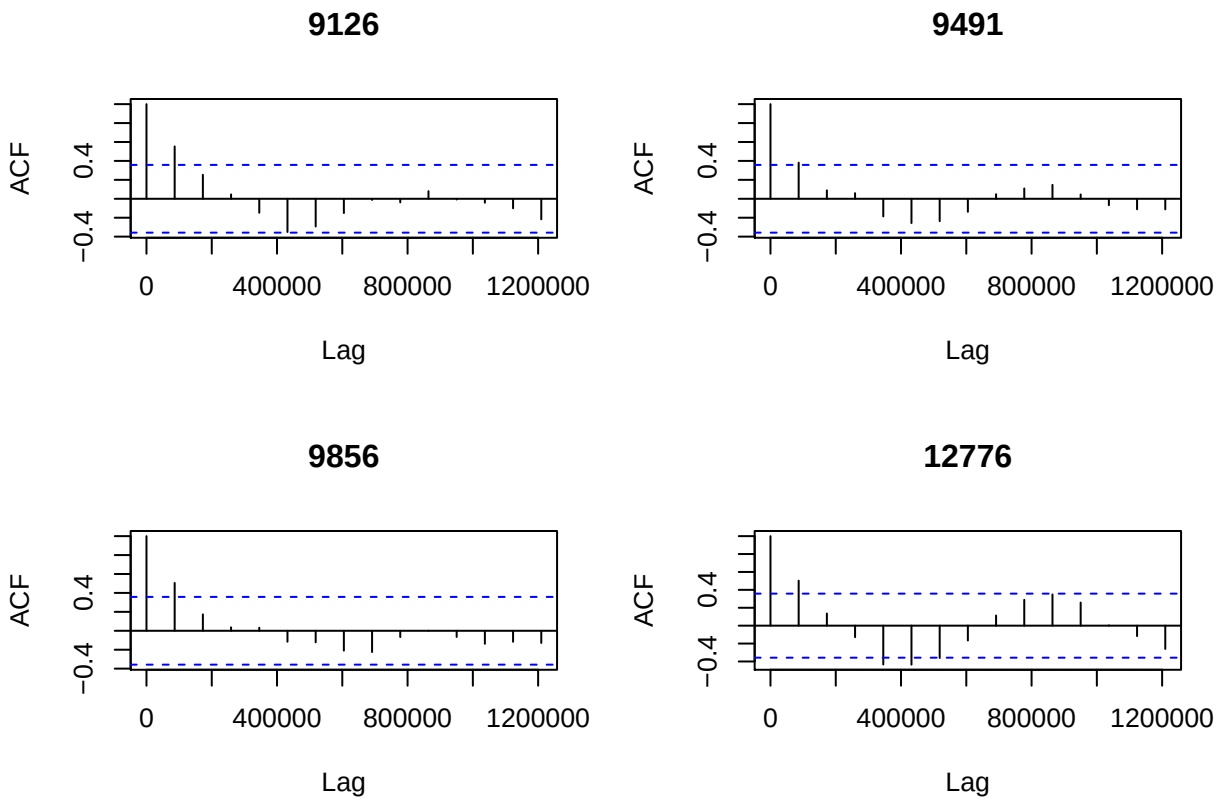
3.d) Animated plot using stplot() – RESULTS NOT DISPLAYED –

```
###  Animated plot -- NOT DISPLAYED --
### 10 days, two variables, with sp.layout, but refresh too slow
cxNO2_10j <- attr(STFDF_jour[,"2014-01-01::2014-01-10",'NO2'],'data')/
                  max(attr(STFDF_jour[,"2014-01-01::2014-01-10",'NO2'],'data'),na.rm=T) * 5
stplot(STFDF_jour[,"2014-01-01::2014-01-10",'O3'],cex=0.4* as.numeric(cxNO2_10j$NO2),
       col.regions=brewer.pal(10,"Spectral"),cuts=10,
       animate=0.2,do.repeat=F,sp.layout =list(world,first=TRUE))

### 10 days, two columns, without sp.layout: better refresh
cxNO2_4j <- attr(STFDF_jour[,"2014-01-01::2014-01-04",'NO2'],'data')
                /max(attr(STFDF_jour[,"2014-01-01::2014-01-04",'NO2'],'data'),na.rm=T) * 5
stplot(STFDF_jour[,"2014-01-01::2014-01-04",'O3'],cex=0.4* as.numeric(cxNO2_4j$NO2),
       col.regions=brewer.pal(10,"Spectral"),cuts=10, animate=0.2,do.repeat=F)

### 10 days, one variable for color one for size, with HTML output
### The animation is saved in mpeg format with high quality
### (requires installation of command line tool imagemagick)
stplot(STFDF_jour[,"2014-01-01::2014-01-10",'O3'],cex=0.4* as.numeric(cxNO2_10j$NO2),
       col.regions=brewer.pal(10,"Spectral"),cuts=10,sp.layout =world, animate=0.2,do.repeat=F)

for(i in c('01','02','03','04','05','06','07','08','09','10')){
 par(mar = c(3, 3, 2, 0.5), mgp = c(2, 0.5, 0), tcl = -0.3, cex.axis = 0.8
     ,cex.lab = 0.8, cex.main = 1)
 cxNO2_1j <- attr(STFDF_jour[,paste("2014-01-",i,sep=''),'NO2'],'data')
```

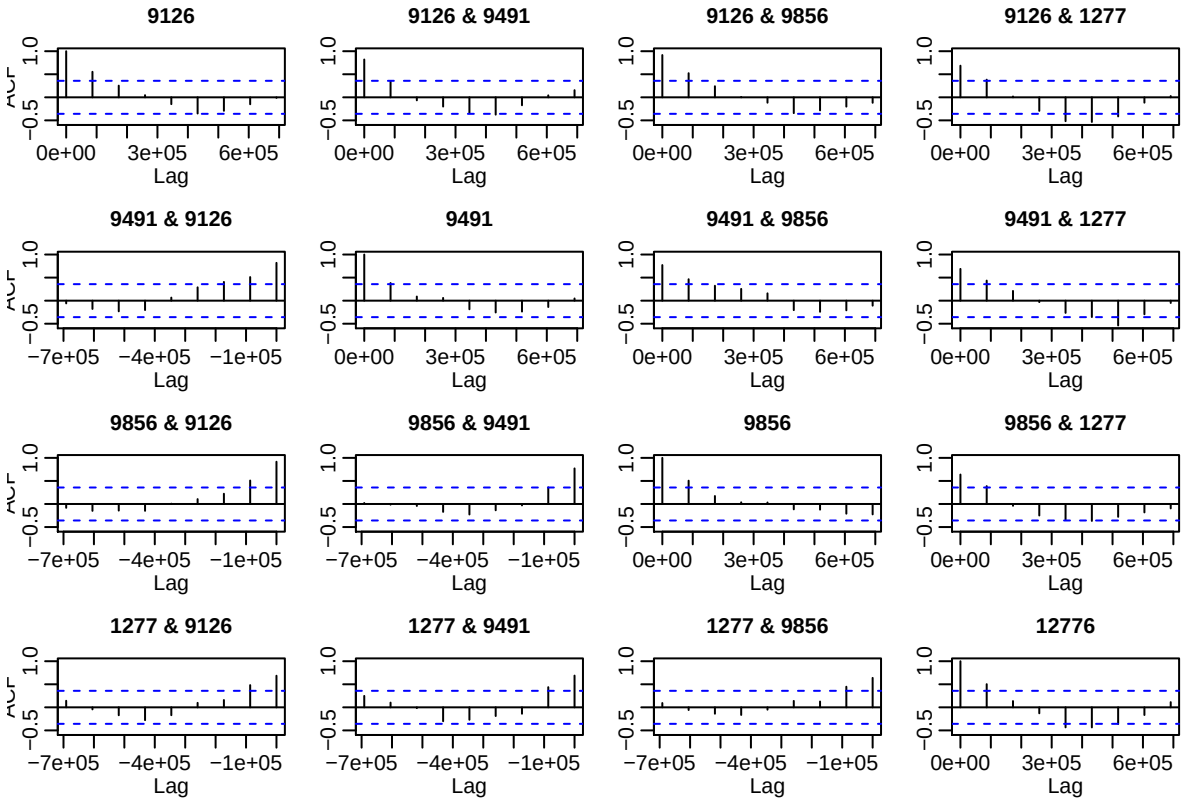
```
                /max(attr(STFDF_jour[,paste("2014-01-",i,sep=''),'NO2'],'data'),na.rm=T) * 5
 jpeg(paste('images/se',i,'.jpg',sep=''),width=5, height=5, units="in",res=600)
 cat(paste('se',i,'.jpg',sep=''),'\n')
 print(spplot(STFDF_jour[,paste("2014-01-",i,sep=''),'O3'],cex=0.4*as.numeric(cxNO2_1j$NO2),
            col.regions=brewer.pal(10,"Spectral"),cuts=10,sp.layout =world))
dev.off()
}
system('convert  -delay 25 -quality 100  images/*.jpg  animation10jours.mpeg')
system('open animation10jours.mpeg')


### Animated plotKML with colours NO2
library(plotKML)
ID <- STFDF_jour[,"2014-01-01",'ID']
plotKML(STFDF_jour[,"2014-01-01",'ID'],colour_scale=SAGA_pal[[1]],
labels='')
```

3.e) Plot of spatio-temporal dependencies

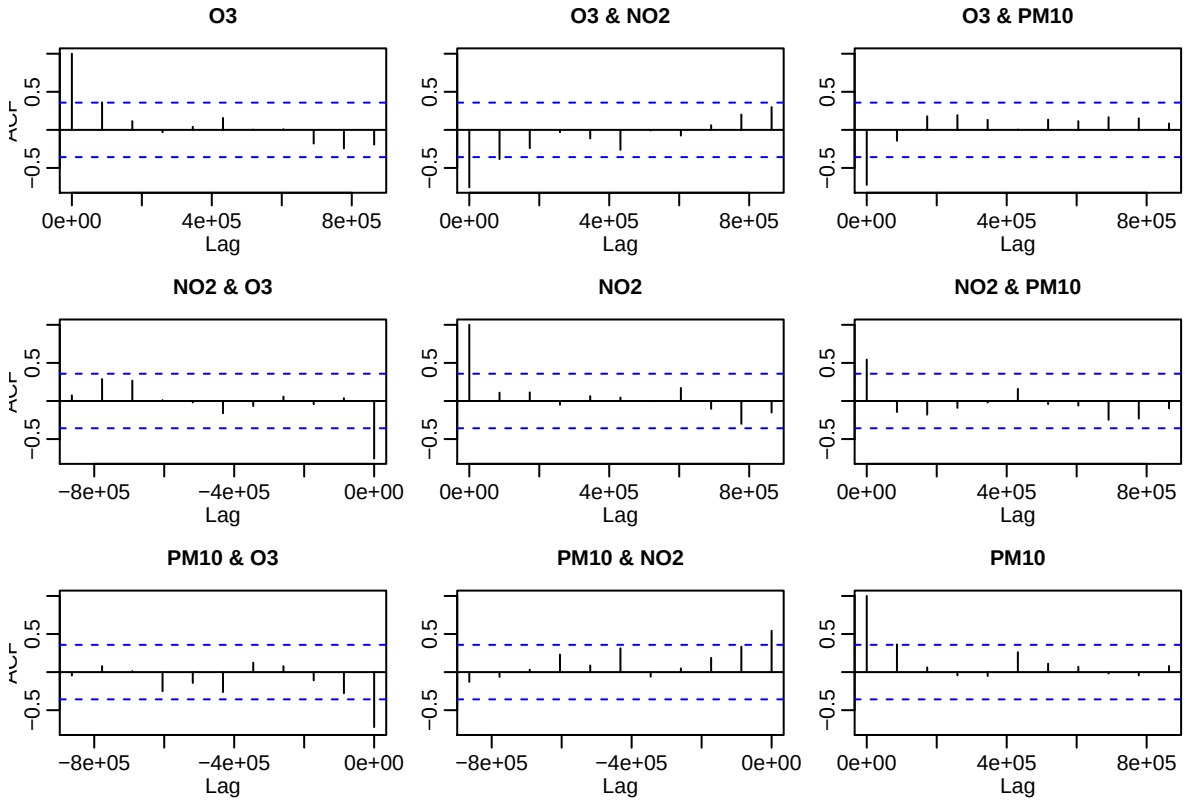
```
rm(list=ls())
library(gstat)
load('data/STFDF_jour.Rdata')


### Temporal auto-correlation
### choose station with no NA for O3 for the period 2014-01-01::2014-01-30
rn <- row.names(STFDF_jour@sp)[c(26:28,36)]

## Multiple stations, one variable
## lags are expressed in secondes, not days
par(mfrow=c(2,2))
for(i in rn) acf(STFDF_jour[i,"2014-01-01::2014-01-30","O3"]$O3,main=i)
```



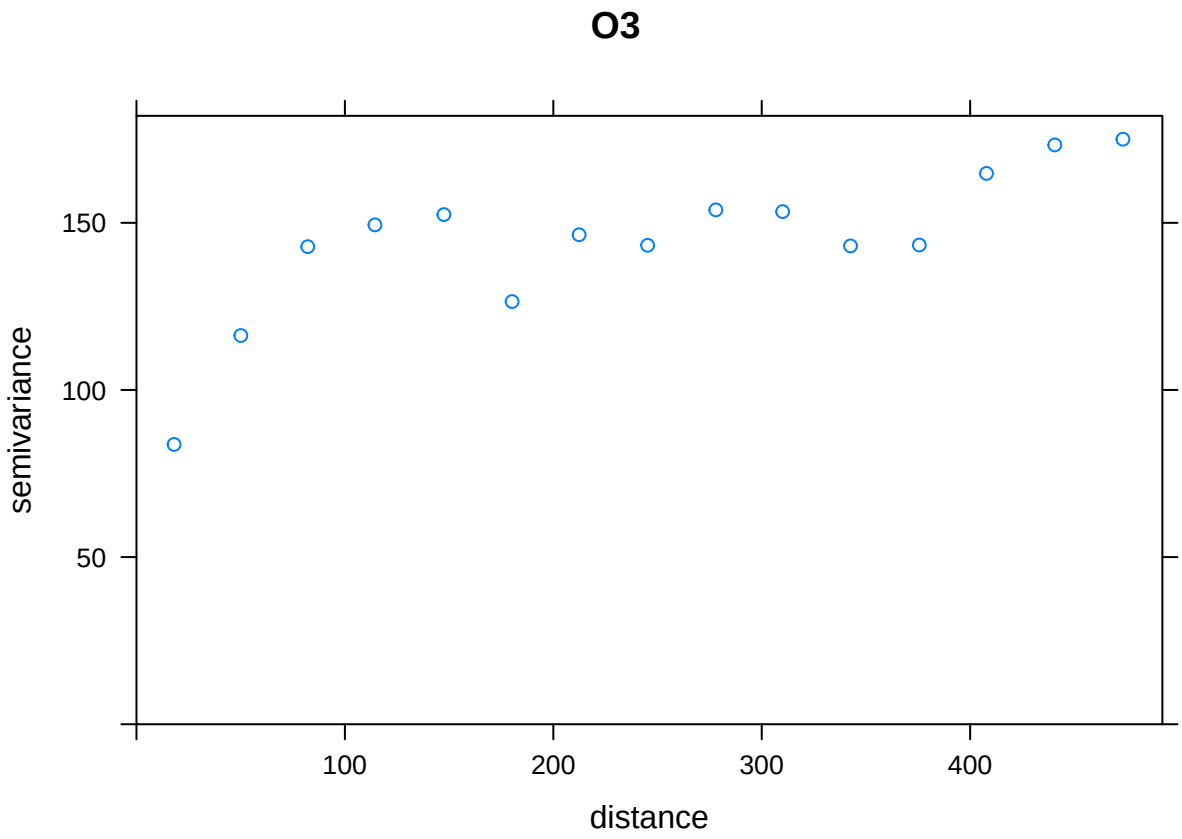

```
### Cross correlation

## Cross correlation between multiple stations and one variable
par(mfrow=c(1,1))

acf(as(STFDF_jour[rn,"2014-01-01::2014-01-30","O3"],"xts"))
```



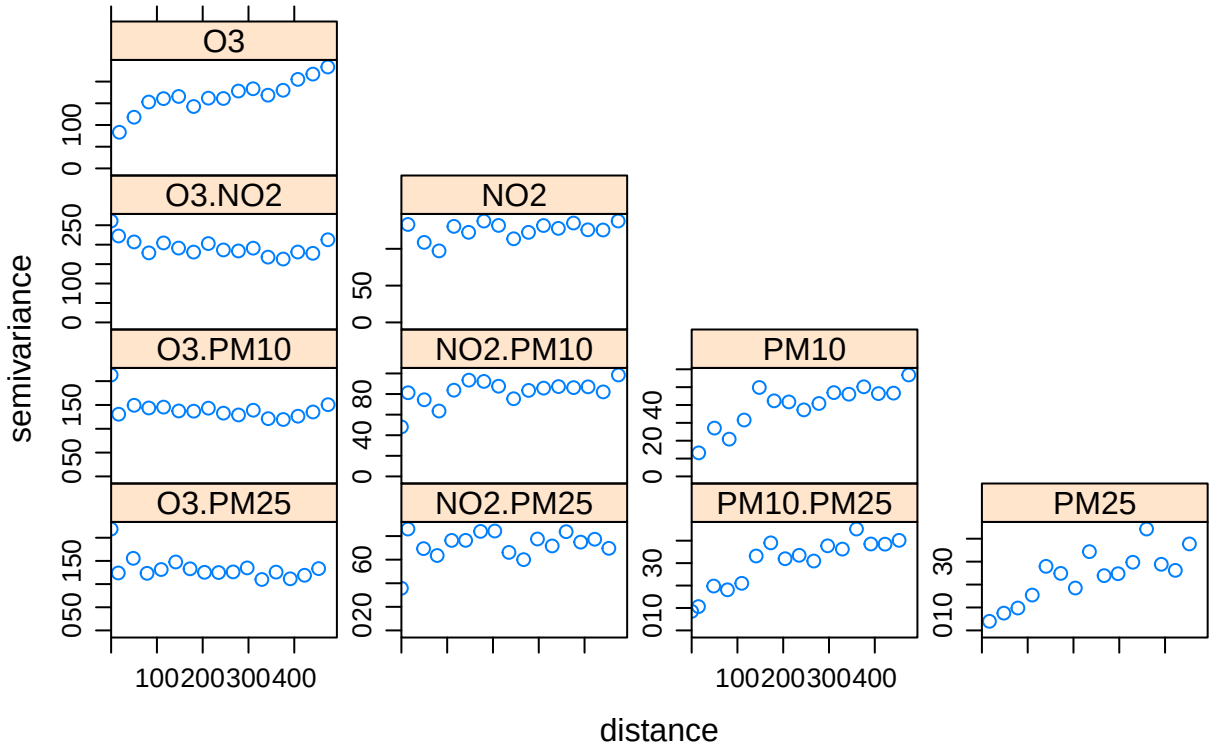

```
## Cross correlation between multiple stations and three variables

acf(STFDF_jour[5,"2014-01-01:::2014-01-30",c("O3","NO2","PM10")][,1:3])
```



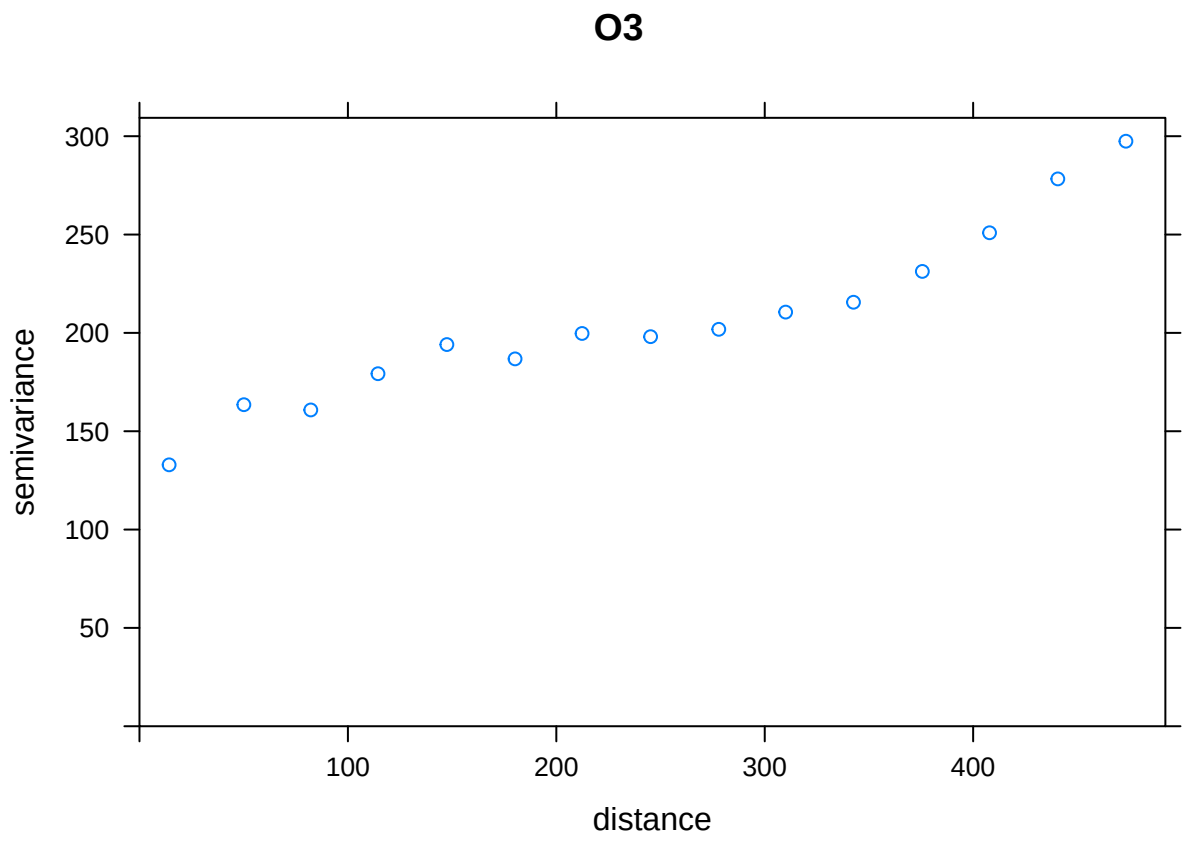

```
### Spatial auto-correlation

## Spatial auto-correlation for fixed date and one variable

library(gstat)
v <- variogram(O3~lat+long,STFDF_jour[!is.na(STFDF_jour[,"2014-01-01","O3"]$O3),"2014-01-01","O3"])
plot(v,main="O3")
```



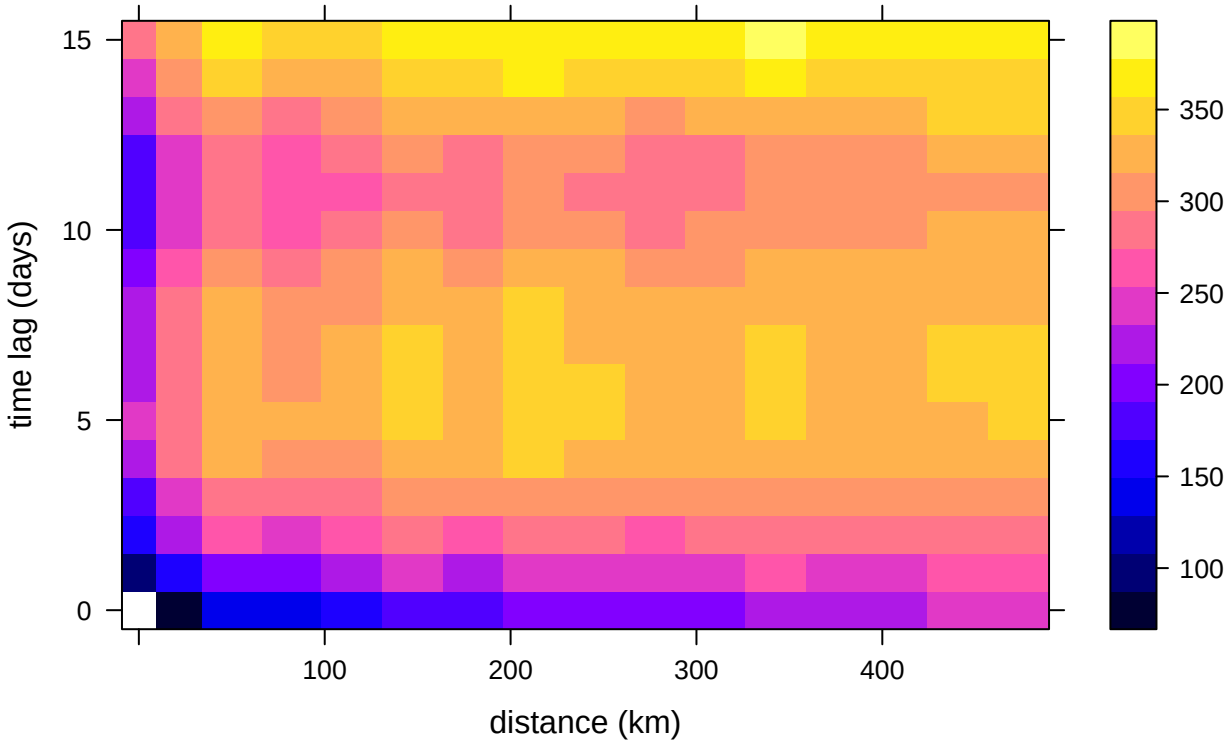

```
## Spatial auto-correlation for fixed date and multiple variables
g <- gstat(NULL, "O3", O3 ~ 1, data = STFDF_jour[!is.na(STFDF_jour[,"2014-01-01","O3"]$O3)
           ,"2014-01-01","O3"])
g <- gstat(g, "NO2", NO2 ~ 1, data = STFDF_jour[!is.na(STFDF_jour[,"2014-01-01","NO2"]$NO2)
            ,"2014-01-01","NO2"])
g <- gstat(g, "PM10", PM10 ~ 1, data = STFDF_jour[!is.na(STFDF_jour[,"2014-01-01","PM10"]$PM10)
            ,"2014-01-01","PM10"])
g <- gstat(g, "PM25", PM25 ~ 1, data = STFDF_jour[!is.na(STFDF_jour[,"2014-01-01","PM25"]$PM25)
            ,"2014-01-01","PM25"])

vm <- variogram(g)
plot(vm)
```



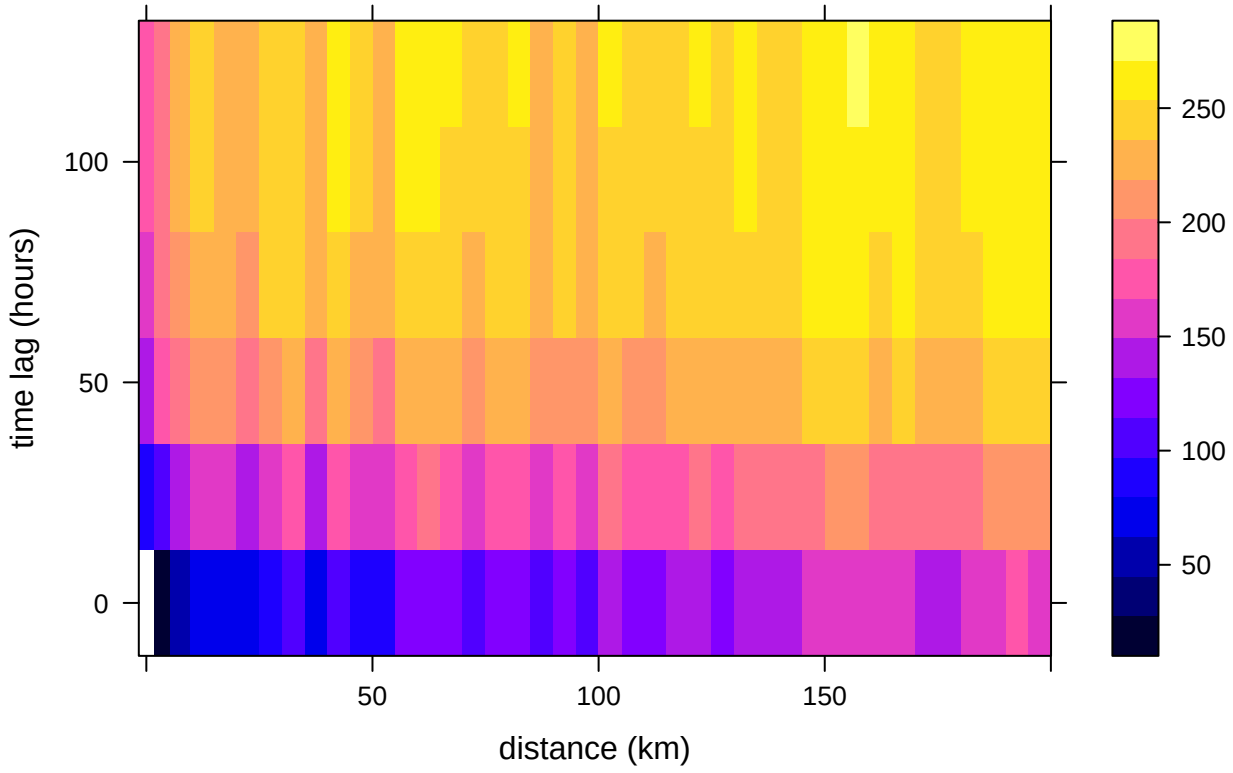

```
## mean variogram on multiple dates

lst <- lapply(1:5,function(i){x=STFDF_jour[,i,"O3"]; x$ti=i ; rownames(x@coords)=NULL;x})
pts <- do.call(rbind,lst)

vv <- variogram(O3~ti,pts[!is.na(pts$O3),])
plot(vv, main="O3")
```



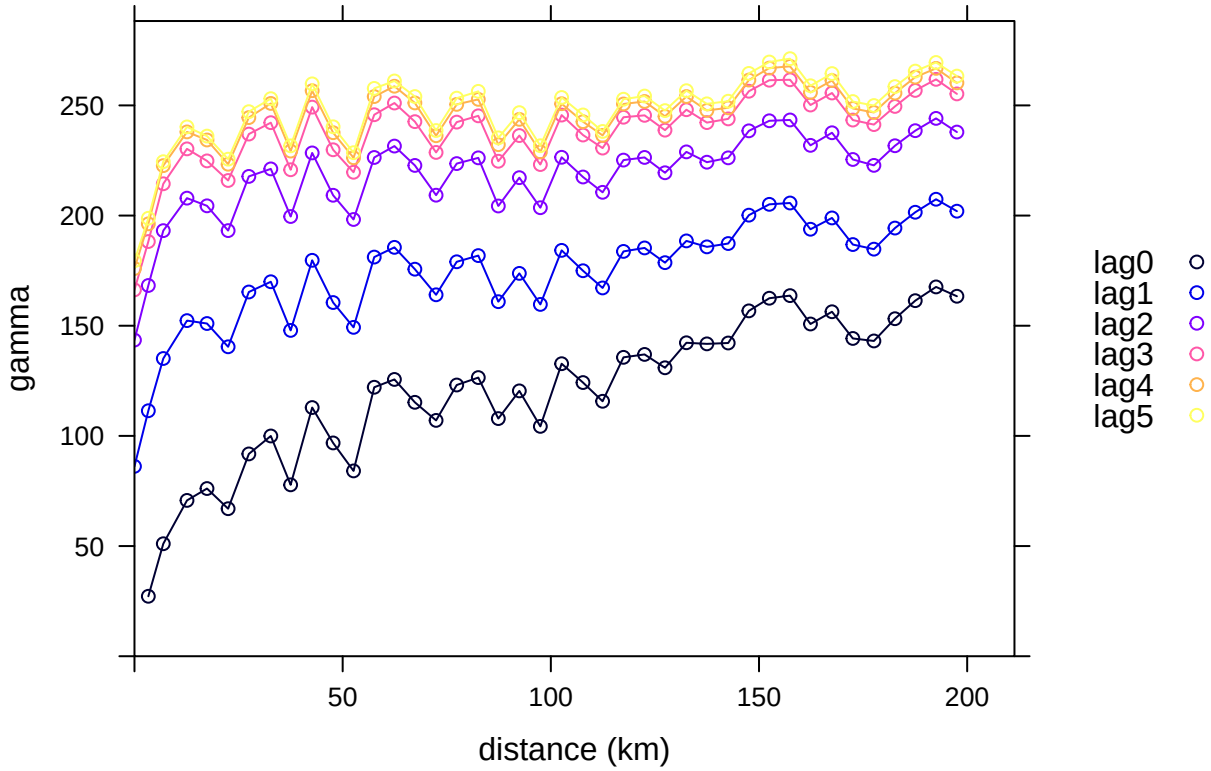

```
## Spatio-temporal auto-correlation

vst <- variogram(O3~lat+long,STFDF_jour[!is.na(STFDF_jour[,"2014-01-01::2014-01-31","O3"]$O3),
                                        "2014-01-01::2014-01-31","O3"])
plot(vst,main="O3")
```



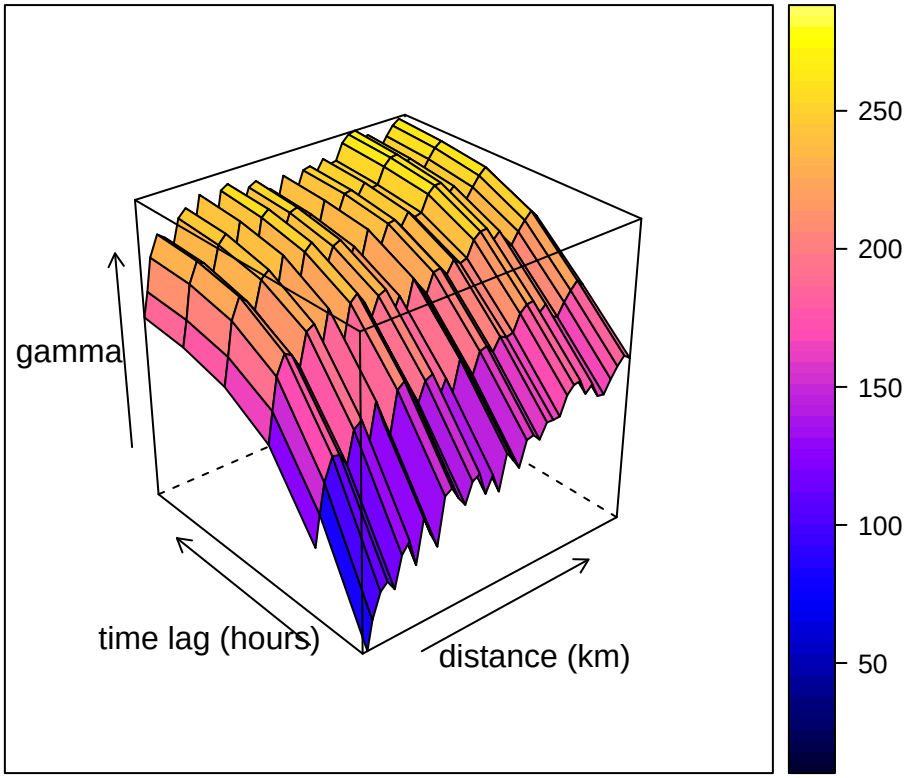

```
vvst <- variogram(O3~1,STFDF_jour[!is.na(STFDF_jour["2014-01-01::2014-12-31","O3"]$O3),
                                   "2014-01-01::2014-12-31","O3"],
                  width=5,cutoff=200,tlags=0:5)
plot(vvst)
```




```
plot(vvst,map=FALSE)
```



```
plot(vvst,wireframe=T)
```



3.f) Interactive plots with Shiny

```
setwd(paste(rep_base,'/data',sep=""))
library(raster)
library(tseries)
library(ggfortify)
library(gstat)
library(ggplot2)
library(shiny)
library(spacetime)
library(leaflet)
library(RColorBrewer)
library(sp)
library(rgeos)
library(rworldmap)
library(maptools)
library(ggmap)

load('STFDF_jour.Rdata')
ST2 <- STFDF_jour[,"2014-01-01::2014-01-30",c("ID","NO2","O3","PM10")]

### This Shiny application creates an interface that the user can navigate and that
### contains different objects,such as, maps, plots, summaries, etc, used to explore our data set.
```

```
### The user has the possibility to interact with the inteface by choosing, e.g.,
### the station to consider, the time interval or the variables to explore, and the
### resulting outputs reacts according to the choice

### The basic ingredients to construct a Shiny application are:
### 1) the UI, where the layout of the interface is defined,
### 2) the SERVER, which contains the "instructions" on how the UI should work.

### In the UI (user interface) we control:
### 1) the VISUAL ASPECT of the application (color, disposition of panels, etc.)
### 2) the INPUT variables: the objects that the user can handle and modify (db, dates, etc.)
### 3) the OUTPUT: what the app will show to the user (plots, maps, graphs, etc.)

### In the SERVER we:
### 1) explain what to do with input variables and how to generate the output
### 2) control the reactivity of the app (more complicated, but highly efficient)


runApp(list(
 ui = fluidPage(
### ui will contain the UI of the app.
### UI is of type "FluidPage": layout consisting of rows which in turn include columns.
   sidebarLayout(
### Create a layout with a sidebar and main area.
     sidebarPanel(
### Create a sidebar panel containing input controls that can in turn be passed to sidebarLayout
### sidebarPanel() contains the objects that the user can interact with, i.e. the inputs.
### To each input correspond a unique name, that can be called in the SERVER part by input$name
### Every time that an input is modified by the user, all functions in SERVER that make use of
### that input will react to the change (unless differently specified).
### Objects of different class can become inputs, such as numerical values, dataframes, files, etc.
### Different control widgets, listed below, are used to interact with the user:
###actionButton -> Action Button
###checkboxGroupInput   -> A group of check boxes
###checkboxInput    -> A single check box
###dateInput    -> A calendar to aid date selection
###dateRangeInput   -> A pair of calendars for selecting a date range
###fileInput    -> A file upload control wizard
###numericInput -> A field to enter numbers
###radioButtons -> A set of radio buttons
###selectInput  -> A box with choices to select from
###sliderInput  -> A slider bar
###submitButton -> A submit button
###textInput    -> A field to enter text

         selectInput('dataset', 'Choose Dataset', ls()[sapply(mget(ls(), .GlobalEnv),
                         function(x) 'STFDF' %in% is(x))]),
### Create a select list with all active STFDF objects. Choose one to start working.
         selectInput('vars', 'Variables', "",multiple=TRUE),
### Choose the variables of the chosen STFDF.
         dateRangeInput(inputId="tim_int",label="Choose date interval"),
### Set the time interval using two calendars.
         radioButtons("choiceID","Choice of locations using...",c(IDs = "byIDS",map = "bymap"),
```

```
                    selected=NULL),conditionalPanel(condition = "input.choiceID == 'byIDS'",
                        selectInput('IDs', 'Which station', "",multiple=TRUE)),
        conditionalPanel(condition = "input.choiceID == 'bymap'", leafletOutput("mymap")),
### Create a radio button to select the type of choice for the station:
### If Ids, it will visualize the list of all IDs amongst which the user can choose
### If map, it will show a leaflet map with marked dots in correspondence of the stations
### radioButtons() works in combination with conditionalPanel(), for the conditional choices
        actionButton("buttonUpdate","Update")
### creates a button that updates the choices made by the user and start the computations
### for generating the output. This functionality can be avoided in case that output generation
### is very fast, allowing hence an interactive and immediate control of the user on the outputs.
### We prefered to add it because our outputs are sometimes slow to compute.
### It is a way of controlling the reactivity of the application.
        ),
     mainPanel(tabsetPanel(
       tabPanel("Summary",textOutput("choices")),
       tabPanel("Cartography",plotOutput("plot_carto"),plotOutput("plot_carto2")),
       tabPanel("Plots",plotOutput("plot1")),
       tabPanel("Dependence Structure",plotOutput("dep1"),plotOutput("dep2"))
### mainPanel is the place where the outputs will be visualized. We divided in 4 tabs (tabPanel())
### Outputs are generated by the functions in the SERVER part.
### To each output correspond a unique name, that can be called in the SERVER part by output$name
### Outputs can take different forms, such as text, images, plots, maps, etc.
### htmlOutput
### imageOutput
### plotOutput
### tableOutput
### textOutput
### uiOutput
### verbatimTextOutput
        )
      ))),
 server = function(session,input, output){
### the SERVER function contains all the instruction on what to do once the inputs have been chosen

   triggerd.by <<- NULL
   data1=reactive({get(input$dataset)})
### The chosen dataset "input$dataset" is set as data1 (class STFDF)
   observeEvent(input$dataset,{
     df1<-attr(data1(),'data')
     dt1<-attr(data1(),'time')
     updateSelectInput(session, "vars",choices = names(df1))
     updateDateRangeInput(session,"tim_int",start =min(index(dt1)),end=max(index(dt1)),
                          min=min(index(dt1)),max=max(index(dt1)))
     updateSelectInput(session, "IDs",choices = levels(df1$ID))
### Once the dataset has been chosen, the widgets of "Which variable","time interval" and " available
### stations" are automatically updated to reflect the choice of the database
   })

   observeEvent({"input.choiceID == 'bymap'"},{
     ds1=as(data1(),"Spatial")
     output$mymap <- renderLeaflet({leaflet(ds1) %>%
         addTiles(urlTemplate = "http://{s}.tile.opentopomap.org/{z}/{x}/{y}.png") %>%
```

```
        setView(lng =mean(coordinates(ds1)[,1]),
                lat = mean(coordinates(ds1)[,2]), zoom = 8)%>%
        addCircles(color="red")})
### If the choice of the stations is made "by map", a leaflet map will appear in the sidebar
   })

   observeEvent(input$buttonUpdate,{
### All function contained in observeEvent(input$buttonUpdate{}) will execute only after the
### "Update" button has been clicked. It is an easy way of controlling the reactivity.
### Obviously, the outputs depends on the chosen inputs:
     nvars <- length(input$vars)
     time_interval <- input$tim_int
     int_in <- input$tim_int[1]
     int_en <- input$tim_int[2]
     int_date<-paste(int_in,int_en,sep="::")
     ndays <- as.numeric(int_en-int_in)

     if(input$choiceID == 'bymap'){
       ds1=as(data1()[,1:2],"Spatial")
       a<-matrix(unlist(input$mymap_bounds)[c(4,3,2,1)],2,2)
       b_poly <- as(extent(a), "SpatialPolygons")
       proj4string(b_poly)<-proj4string(ds1)
       inters_sp<-!is.na(over(ds1,b_poly))
       npoints <- sum((inters_sp==1)*1)
       d<-data1()[inters_sp,int_date,input$vars]
       d2<-data1()[inters_sp,int_date,c("ID",input$vars)]
     }
     if(input$choiceID == 'byIDS'){
       d<-data1()[attr(data1(),"data")$ID %in% input$IDs,int_date,input$vars]
       d2<-data1()[attr(data1(),"data")$ID %in% input$IDs,int_date,c("ID",input$vars)]
       npoints <- length(input$IDs)
       chosenID <- attr(data1(),"data")$ID %in% input$IDs
     }

     output$choices = renderText({
       paste("You have chosen ",nvars," variable(s).\n",
             "The time interval is of ",as.numeric(int_en-int_in)," days.\n",
             "You are considering ",npoints," points/stations\n",sep="")
     })
### The summary of the chosen subset is print using "renderText()".

     world<-getMap(resolution="low")

### output$plot_carto and output$plot_carto2 are cartographies plotted using
### the functions presented in Sections 3.a and 3.d. They are plotted using "renderPlot()".
### output$plot1 is a static plot as presented in Section 3.c for time series.
### output$dep1 and output$dep2 are dependency plots as explained in Section 3.e.

###Map: n stations, 1 day, 1 variable
     if (npoints>1 && ndays==0 && nvars==1 ){
       output$plot_carto <- renderPlot({spplot(d,col.regions=brewer.pal(10,"Spectral"),
                                               cuts=10,sp.layout=world)})
     }
```

```
###Map: n stations, 1 day, 2 variables
     if (npoints>1 && ndays==0 && nvars==2 ){
       output$plot_carto <- renderPlot({
         cx = attr(d,"data")[,input$vars[2]]/max(attr(d,"data")[,input$vars[2]],na.rm=T) * 5
         spplot(d[,input$vars[1]],cex=0.4*as.numeric(cx),scales=list(draw=TRUE),
                pch=19,sp.layout=world)
       })
       output$plot_carto2 <- renderPlot({
         cx = attr(d,"data")[,input$vars[1]]/max(attr(d,"data")[,input$vars[1]],na.rm=T) * 5
         spplot(d[,input$vars[2]],cex=0.4*as.numeric(cx),scales=list(draw=TRUE),
                pch=19,sp.layout=world)
       })
     }

###Animated Map: n stations, n days, 1 variable
###Time seriesPlot: n stations, n days, 1 variable
     if (npoints>1 && ndays>0 && nvars==1 ){
       output$plot_carto <- renderPlot({stplot(d,col.regions=brewer.pal(10,"Spectral"),
                                               cuts=10,sp.layout=world) })
       output$plot_carto2 <- renderPlot({stplot(d,col.regions=brewer.pal(10,"Spectral"),
                                      cuts=10,sp.layout=world, animate=0.2,do.repeat=F)})
       output$plot1 <- renderPlot({
         tim<-attr(d2,"time")
         dt<-attr(d2,"data")
         n<-dim(dt)[1]/length(tim)
         a<-split(dt,dt$ID)
         m<-matrix(0,ncol=n,nrow=length(tim))
         for (i in 1:n){
           if (!all(is.na(a[[i]][,2]))) m[,i]<-as.numeric(a[[i]][,2])
         }
         colnames(m)<-levels(d2$ID)
         ir<-irts(index(tim),m)
         p=autoplot(ir, facets = FALSE)+ggtitle(as.character(input$vars))
         p
       })

     }

###Animated Map: n stations, 1 day, 2 vars
###Time series Plot: n stations, 1 day, 2 vars
     if (npoints>1 && ndays>0 && nvars==2 ){
       output$plot_carto <- renderPlot({
         cx = attr(d,'data')[,input$vars[2]]/max(attr(d,'data')[,input$vars[2]],na.rm=T) * 5
         stplot(d[,,input$vars[1]],cex=0.4* as.numeric(cx),col.regions=brewer.pal(10,"Spectral"),
                cuts=10,sp.layout=world, animate=0.2,do.repeat=F)
       })

       output$plot_carto2 <- renderPlot({
         cx = attr(d,'data')[,input$vars[1]]/max(attr(d,'data')[,input$vars[1]],na.rm=T) * 5
         stplot(d[,,input$vars[2]],cex=0.4* as.numeric(cx),col.regions=brewer.pal(10,"Spectral"),
                cuts=10,sp.layout=world, animate=0.2,do.repeat=F)
       })
```

```
    }

### 1 station, multiple variables
    if(npoints==1 && ndays>0 && nvars>=2 ){
      output$plot1 <- renderPlot({
        colours=brewer.pal(3,"Set1")
        irr<-irts(index(d),as.matrix(d[,1:nvars]))
        p1<-autoplot(irr, facets = FALSE)
        p1})
    }

    if(npoints>1 && ndays>0 && nvars>=2 ){
      output$plot1 <- renderPlot({
        tim<-attr(d2,"time")
        dt<-attr(d2,"data")
        n<-dim(dt)[1]/length(tim)
        k<-length(input$vars)
        a<-split(dt,dt$ID)
        p=vector(mode="list",length = k)
        for (j in 1:k){
          m<-matrix(0,ncol=n,nrow=length(tim))
          for (i in 1:n){
            if (!all(is.na(a[[i]][,j+1]))) m[,i]<-as.numeric(a[[i]][,j+1])}
          colnames(m)<-levels(dt$ID)
          ir<-irts(index(tim),m)
          p[[j]]=autoplot(ir, facets = FALSE)+ggtitle(as.character(input$vars[j]))
        }
        new("ggmultiplot",plots=p,ncol=1)})
    }

    if (npoints>1 && ndays>3 && nvars==1){
      output$dep1 <- renderPlot({
        acf(as(d,"xts"),na.action = na.pass)
      })
    }

    if (npoints==1 && ndays>3 && nvars==1){
      output$dep1 <- renderPlot({
        acf(d[,input$vars],na.action = na.pass)
      })
    }

    if (ndays>3 && nvars>1 && npoints==1){
      output$dep1 <- renderPlot({
        acf(as(d,"xts"),na.action = na.pass)
      })
    }

  })
 }))
```

4.) Estimating a space-time covariance function

4.1) Data preparation

```
rm(list=ls())
DATA = 'data/'
OUT = DATA
library(rgdal)
library(sp)
library(spacetime)
######### reload saved data objects (if necessary)
load(paste0(DATA, 'OBS_jour.Rdata'))
load(paste0(DATA, 'CHM.Rdata'))
load(paste0(DATA, 'stations.Rdata'))
######### extract French background stations
IDs.bg   <- stations$station_european_code[stations$type_of_station=="Background"]
IDs.fr   <- stations$station_european_code[stations$country_name=="France"]
IDs.bgfr <- intersect(IDs.bg,IDs.fr)
idx.stations <- which(stations$station_european_code %in% IDs.bgfr)
coord    <- cbind(stations$station_longitude_deg,stations$station_latitude_deg)[idx.stations,]
######### define time window for data used for fitting
tstart   <- "2014-01-01"
tend     <- "2014-03-31"
OBS_jour <- OBS_jour[OBS_jour$ID %in% IDs.bgfr,]
OBS_sel  <- stConstruct(OBS_jour,space=c('long','lat'),time='date',
                        SpatialObj=SpatialPoints(OBS_jour[,c('long','lat')]))
OBS_sel  <- as(OBS_sel,"STFDF")
proj4string(OBS_sel)="+init=epsg:4326"
OBS_sel  <- OBS_sel[,paste0(tstart,"::",tend)]
######### transform to Lambert 93 coordinates system
OBS_sel@sp <- spTransform(OBS_sel@sp, CRS("+init=epsg:2154"))

######### interpolate gridded CHIMERE values to space-time observation points
library(fields)
CHM_sel  <- CHM[as.character(CHM$time)>=tstart&as.character(CHM$time)<=tend,]
grid.CHM <- unique(cbind(CHM_sel$lon,CHM_sel$lat))
CHM_mat  <- matrix(CHM_sel$PM10, nrow=nrow(grid.CHM))
dim(CHM_mat)
```

```
## [1] 11211    90
```

```
lon.grid <- sort(unique(CHM_sel$lon))
lat.grid <- sort(unique(CHM_sel$lat))
ind.sites.obs <- match(unique(OBS_sel@data$ID),IDs.bgfr)
fun <- function(i){
  tmp <- list(x=lon.grid,y=lat.grid,
              z=matrix(CHM_mat[,i],length(lon.grid),length(lat.grid)))
  interp.surface(tmp, coord[ind.sites.obs,])
}
CHM_sites <- sapply(1:ncol(CHM_mat),fun)
######### save as variable CHM
OBS_sel@data$CHM <- as.numeric(CHM_sites)
```

```
######### save PM10 residuals with respect to CHM as variable PM10res
OBS_sel@data$PM10res <- OBS_sel@data$PM10-as.numeric(CHM_sites)
```

4.2) Estimation with *CompRandFld*

```
library(CompRandFld)
library(spam)
######### data must follow the format t x d (with t observation times and d sites)
######### to obtain a set of stations and time points without missing data, we
######### proceed through an iterative removal of stations and time points with many
######### missing data
names(OBS_sel@data)
```

```
## [1] "ID"      "PM10"    "PM25"    "NO2"     "O3"      "CHM"     "PM10res"
```

```
data <- matrix(OBS_sel@data$PM10res, ncol = dim(OBS_sel)[1], nrow = dim(OBS_sel)[2],
    byrow = T)
dim(data)
```

```
## [1]  90 378
```

```
mean(is.na(data))
```

```
## [1] 0.4398001
```

```
space.keep <- 1:ncol(data)
time.keep <- 1:nrow(data)
propNA <- 0.9
toKeep <- apply(is.na(data), 1, mean) < propNA
time.keep <- time.keep[toKeep]
data.keep <- data[toKeep, ]
toKeep <- apply(is.na(data), 2, mean) < propNA
space.keep <- space.keep[toKeep]
data.keep <- data.keep[, toKeep]
dim(data.keep)
```

```
## [1]  90 224
```

```
propNA <- 0.2
toKeep <- apply(is.na(data.keep), 1, mean) < propNA
time.keep <- time.keep[toKeep]
data.keep <- data.keep[toKeep, ]
toKeep <- apply(is.na(data.keep), 2, mean) < propNA
space.keep <- space.keep[toKeep]
data.keep <- data.keep[, toKeep]
dim(data.keep)
```

```
## [1]  90 209
```

```
propNA <- 0.1
toKeep <- apply(is.na(data.keep), 1, mean) < propNA
time.keep <- time.keep[toKeep]
data.keep <- data.keep[toKeep, ]
toKeep <- apply(is.na(data.keep), 2, mean) < propNA
space.keep <- space.keep[toKeep]
data.keep <- data.keep[, toKeep]
dim(data.keep)
```

```
## [1]  90 190
```

```
propNA <- 0.05
toKeep <- apply(is.na(data.keep), 1, mean) < propNA
time.keep <- time.keep[toKeep]
data.keep <- data.keep[toKeep, ]
toKeep <- apply(is.na(data.keep), 2, mean) < propNA
space.keep <- space.keep[toKeep]
data.keep <- data.keep[, toKeep]
dim(data.keep)
```

```
## [1]  87 164
```

```
propNA <- 0.025
toKeep <- apply(is.na(data.keep), 1, mean) < propNA
time.keep <- time.keep[toKeep]
data.keep <- data.keep[toKeep, ]
toKeep <- apply(is.na(data.keep), 2, mean) < propNA
space.keep <- space.keep[toKeep]
data.keep <- data.keep[, toKeep]
dim(data.keep)
```

```
## [1]  78 113
```

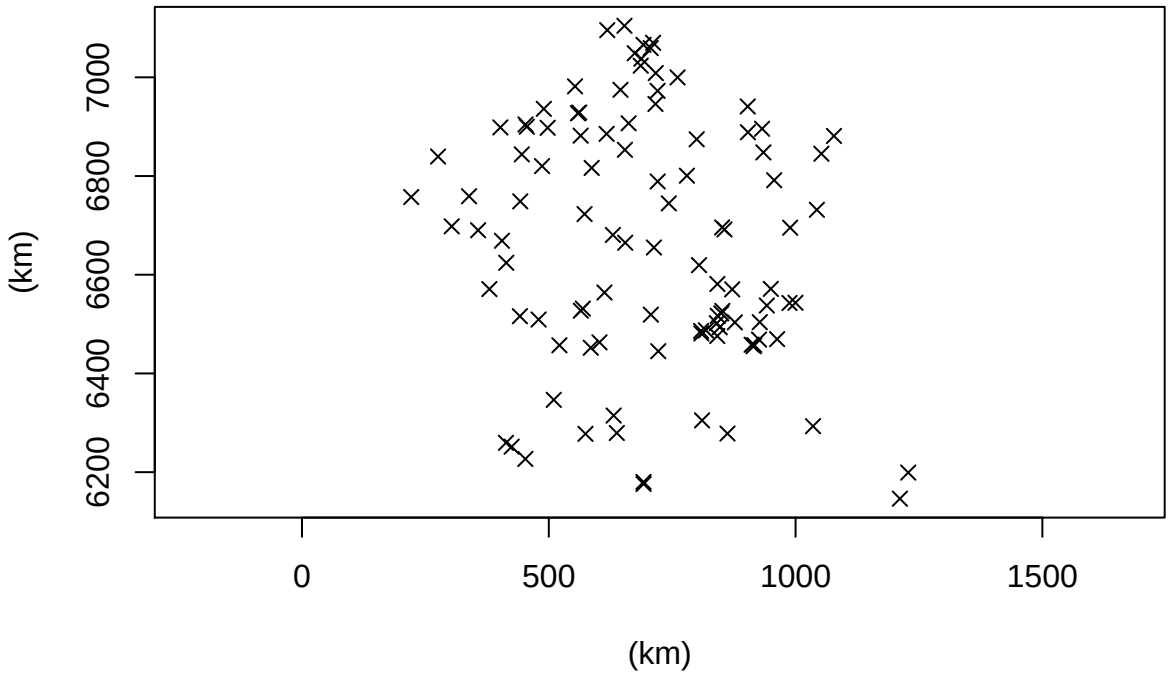
```
propNA <- 0.01
toKeep <- apply(is.na(data.keep), 1, mean) < propNA
time.keep <- time.keep[toKeep]
data.keep <- data.keep[toKeep, ]
toKeep <- apply(is.na(data.keep), 2, mean) < propNA
space.keep <- space.keep[toKeep]
data.keep <- data.keep[, toKeep]
dim(data.keep)
```

```
## [1]  70 103
```

```
propNA <- 10^(-10)
toKeep <- apply(is.na(data.keep), 1, mean) < propNA
time.keep <- time.keep[toKeep]
data.keep <- data.keep[toKeep, ]
toKeep <- apply(is.na(data.keep), 2, mean) < propNA
space.keep <- space.keep[toKeep]
data.keep <- data.keep[, toKeep]
dim(data.keep)
```

```
## [1]  70 103
```

```
coord.sel <- (OBS_sel@sp@coords/1000)[space.keep, ]  #Lambert 93 coordinates in  km
stations.fit <- stations[idx.stations[space.keep], ]
plot(coord.sel, asp = 1, xlab = "(km)", ylab = "(km)", pch = 4)
```




```
times <- (1:dim(OBS_sel)[2])[time.keep]
data <- data.keep
######### save selected data for later use
save(stations.fit, file = paste0(DATA, "stationsforfitting.Rdata"))
save(times, file = paste0(DATA, "timesforfits.Rdata"))
save(data, file = paste0(DATA, "data.Rdata"))

######### we assume that the PM10res data are stationary over space and time we
######### start by estimating the empirical mean and the empirical variance from
######### data
mean.est <- mean(data)
mean.est
```

```
## [1] 8.024326
```

```
var.est <- var(as.numeric(data))
var.est
```

```
## [1] 116.0013
```

```
######### now calculate an empirical space-time variogram
vgm.emp <- EVariogram(data = data, coordx = coord.sel, coordt = times, cloud = F,
    maxdist = 1000, maxtime = 12)
######### full likelihood estimation would be really hard and slow:
prod(dim(data))
```

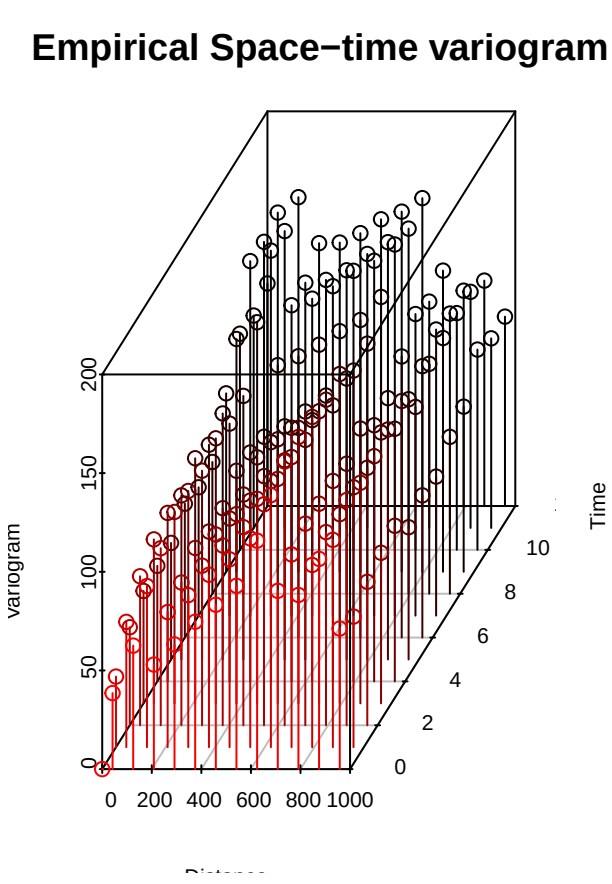
```
## [1] 7210
```

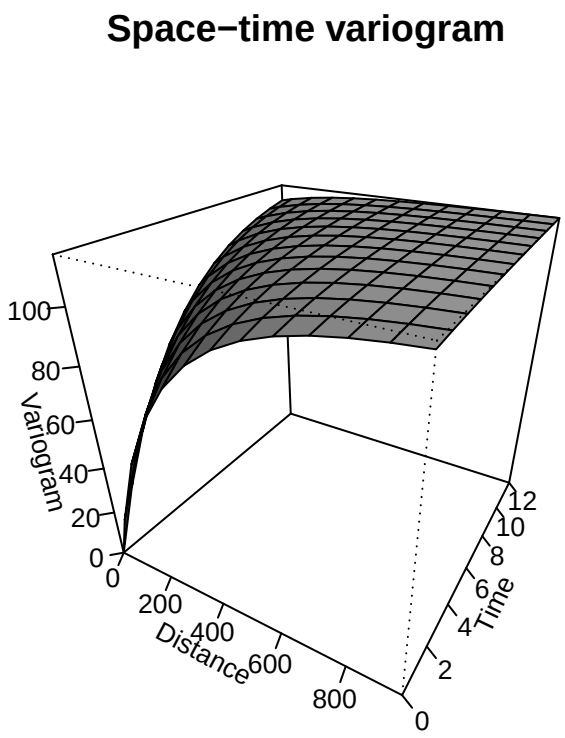
```
######### now we define some parametric models to estimate (with initial values and
######### some of the parameters fixed) space-time separable exponential model
######### without nugget
cormod1 <- "exp_exp"
fixed1 <- list(mean = mean.est, nugget = 0, sill = var.est)
start1 <- list(scale_s = 200, scale_t = 2)
# space-time separable exponential model with nugget
cormod2 <- "exp_exp"
fixed2 <- list(mean = mean.est, sill = var.est)
start2 <- list(scale_s = 200, scale_t = 2, nugget = 0)
# Gneiting model with powers fixed to 1, without nugget
cormod3 <- "gneiting"
fixed3 <- list(sill = var.est, mean = mean.est, nugget = 0, power_s = 1, power_t = 1)
start3 <- list(scale_s = 200, scale_t = 2, sep = 0.5)
# Gneiting model with powers fixed to 1, with nugget
cormod4 <- "gneiting"
fixed4 <- list(sill = var.est, mean = mean.est, power_s = 1, power_t = 1)
start4 <- list(scale_s = 200, scale_t = 2, sep = 0.5)
# Gneiting model with powers fixed to .5, without nugget
cormod5 <- "gneiting"
fixed5 <- list(sill = var.est, mean = mean.est, nugget = 0, power_s = 0.5, power_t = 0.5)
start5 <- list(scale_s = 200, scale_t = 2, sep = 0.5)
# Gneiting model with powers fixed to .5, with nugget
cormod6 <- "gneiting"
fixed6 <- list(sill = var.est, mean = mean.est, power_s = 0.5, power_t = 0.5)
start6 <- list(scale_s = 200, scale_t = 2, sep = 0.5)
# Gneiting model where powers are estimated, without nugget
cormod7 <- "gneiting"
fixed7 <- list(sill = var.est, mean = mean.est, nugget = 0)
start7 <- list(scale_s = 200, scale_t = 2, sep = 0.5, power_s = 0.5, power_t = 0.5)
# Gneiting model where powers are estimated, with nugget
cormod8 <- "gneiting"
fixed8 <- list(sill = var.est, mean = mean.est)
start8 <- list(scale_s = 200, scale_t = 2, sep = 0.5, power_s = 0.5, power_t = 0.5,
    nugget = 0)

######### pairwise likelihood fitting of the first model with space and time cut-off
######### distances
fit1PL <- FitComposite(data = data, coordx = coord.sel, coordt = times, maxdist = 300,
    maxtime = 4, corrmodel = cormod1, likelihood = "Marginal", type = "Pairwise",
    fixed = fixed1, start = start1)
fit1PL
```

```
##
## ###################################################################
## Maximum Composite-Likelihood Fitting of Gaussian Random Fields
##
## Setting: Marginal Composite-Likelihood
##
## Model associated to the likelihood objects: Gaussian
##
## Type of the likelihood objects: Pairwise
```

```
##
## Covariance model: exp_exp
## Number of spatial coordinates: 103
## Number of dependent temporal realisations: 70
## Number of replicates of the random field: 1
## Number of estimated parameters: 2
##
## Maximum log-Composite-Likelihood value: -6577607.64
##
## Estimated parameters:
## scale_s  scale_t
## 238.015    4.141
##
## ###########################################################################
```

```
######### visual comparison of empirical and fitted variogram
library(scatterplot3d)
Covariogram(fit1PL, vario = vgm.emp, show.vario = T, pch = 20)
```






```
######### weighted least squares fitting
fit1WLS <- WLeastSquare(data = data, coordx = coord.sel, coordt = times, maxdist = 300,
    maxtime = 4, corrmodel = cormod1, fixed = fixed1, start = start1, weighted = T)
fit1WLS
```

```
##
```

```
## ################################################################
## Results: Weighted Least Squares Fitting of Gaussian Random Fields.
##
## Model used from the Weighted Least Squares : Gaussian
##
## Covariance model: exp_exp
## Number of spatial coordinates: 103
## Number of dependent temporal realisations: 70
## Number of replicates of the random field: 1
## Number of estimated parameters: 2
## The value of the Weighted Least Squares at the minimum: 35788.26
## Number of spatial bins 14
## Number of temporal bins 5
## Min and max spatial distances: 2.690935 300
## Min and max temporal interval: 1 4
##
## Estimated parameters:
## scale_s  scale_t
## 306.903    3.131
##
## ################################################################
```

```
######### now we fit all models with weighted least squares or pairwise likelihood
fitsWLS <- fitsPL <- vector("list", 8)
for (i in 1:8) {
    fitsWLS[[i]] <- WLeastSquare(data = data, coordx = coord.sel, coordt = times,
        maxdist = 300, maxtime = 4, corrmodel = get(paste0("cormod", i)), fixed = get(paste0("fixed",
            i)), start = get(paste0("start", i)), weighted = T)
    fitsPL[[i]] <- FitComposite(data = data, coordx = coord.sel, coordt = times,
        maxdist = 300, maxtime = 4, corrmodel = get(paste0("cormod", i)), likelihood = "Marginal",
        type = "Pairwise", fixed = get(paste0("fixed", i)), start = get(paste0("start",
            i)), varest = T)
}
######### save lists of fit objects
save(fitsWLS, file = paste0(DATA, "fitsWLS.Rdata"))
save(fitsPL, file = paste0(DATA, "fitsPL.Rdata"))

######### which pairwise likelihood model is best in terms of the CLIC?
which.min(sapply(fitsPL, getElement, "clic"))
```

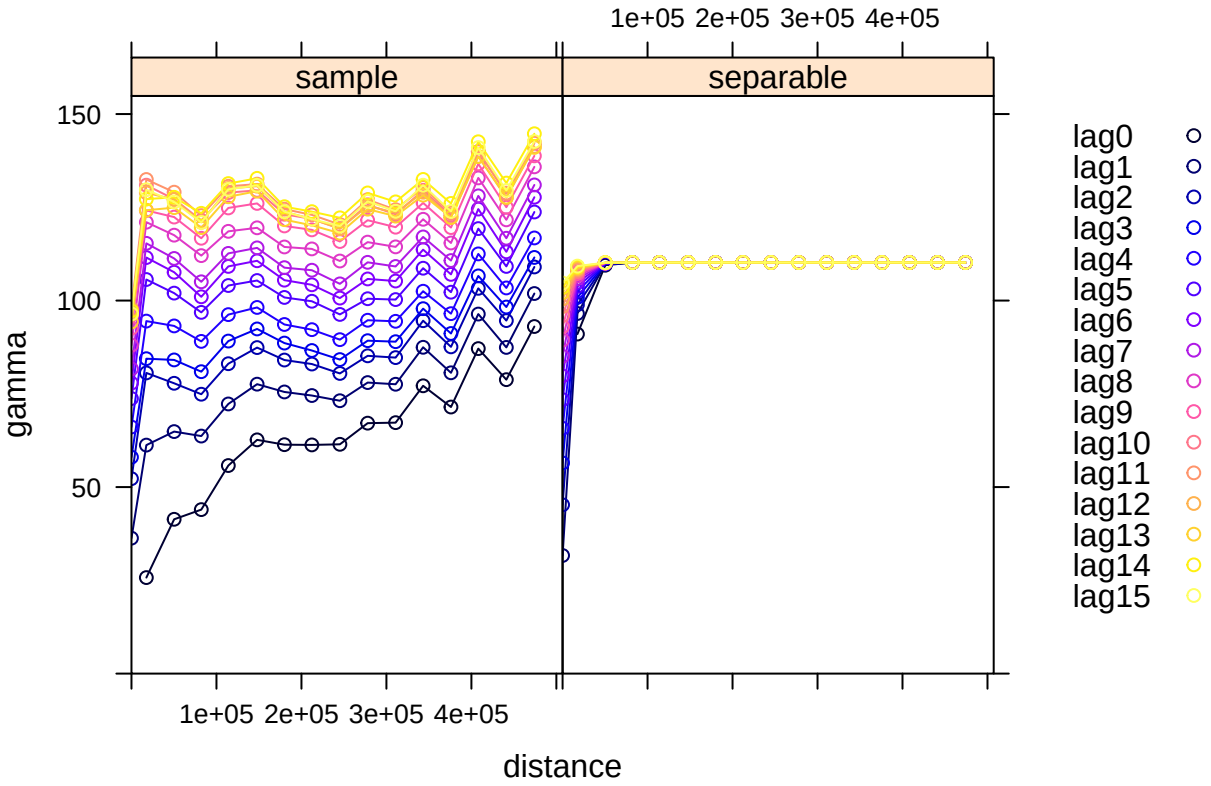
```
## [1] 2
```

```
######### test some hypotheses for embedded models
fit8 <- fitsPL[[8]]
fit7 = fitsPL[[7]]
fit5 = fitsPL[[5]]
HypoTest(fit8, fit7, fit5, statistic = "Wald")
```

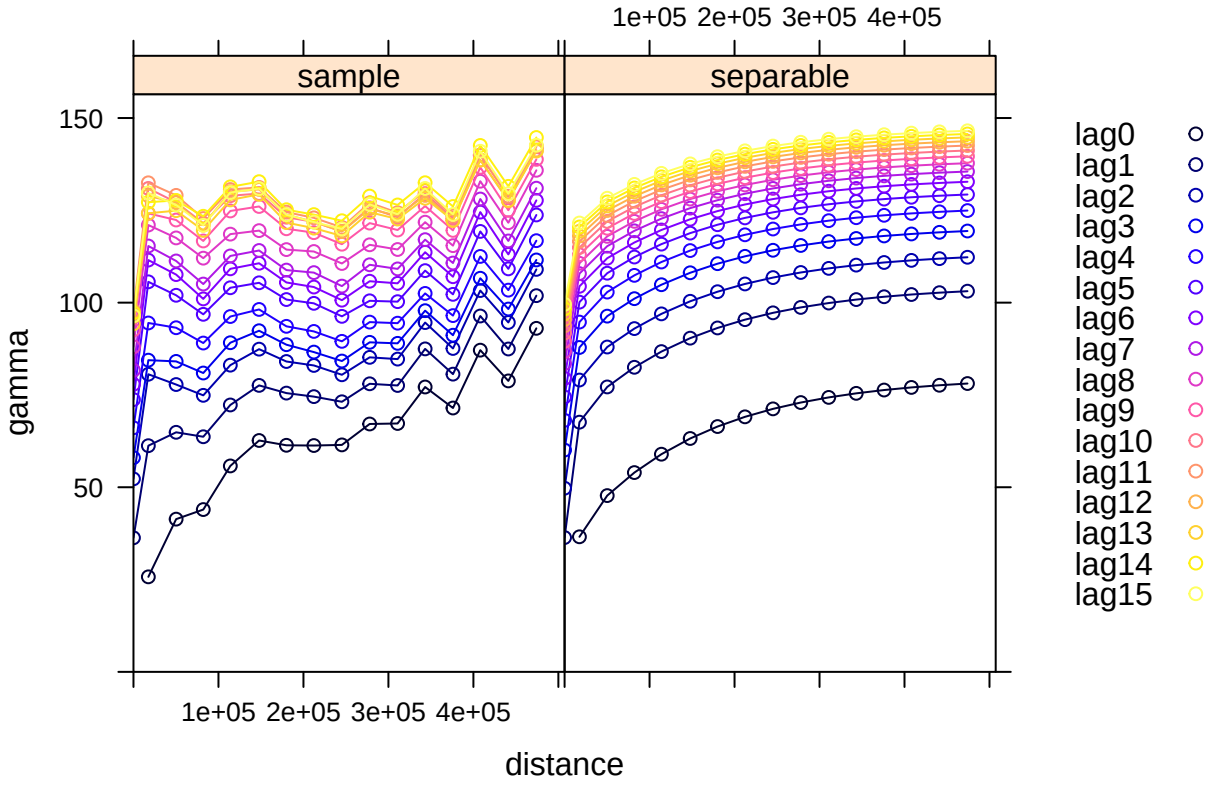
```
##      Num.Par Diff.Par Df      Chisq   Pr(>chisq)
## fit8       6       NA NA         NA           NA
## fit7       5        1  1  0.1769632 6.739961e-01
## fit5       3        2  2 27.7078933 9.622932e-07
```

4.3) Estimation with *gstat*

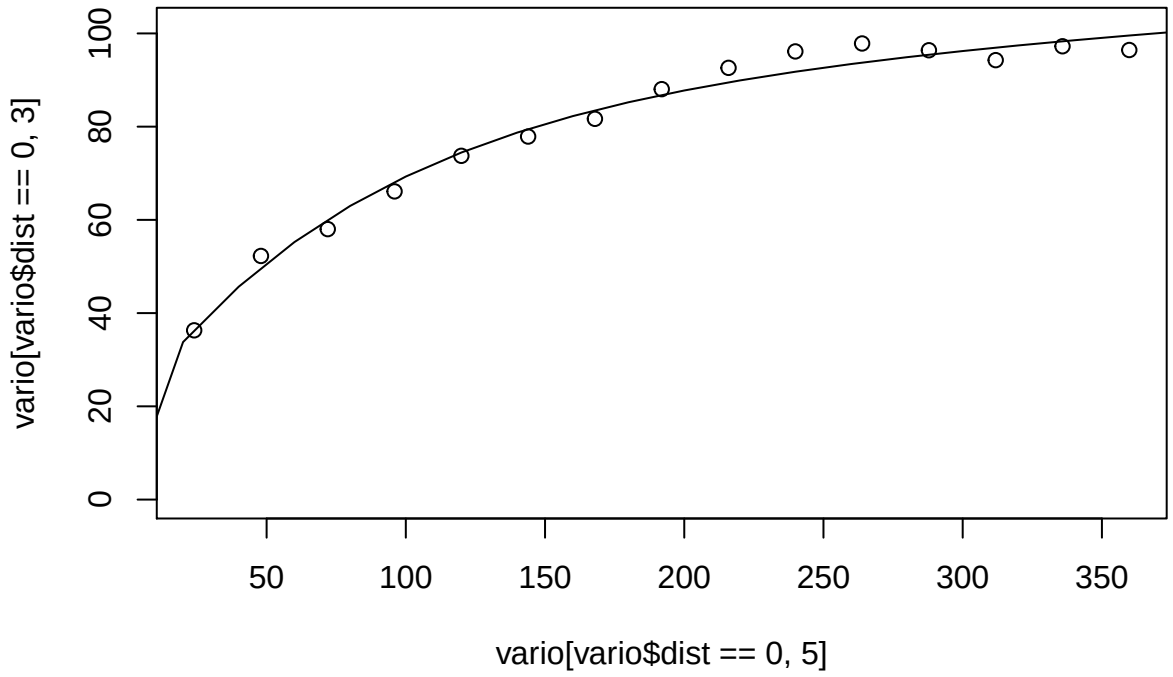
```
library(gstat)
######### We first define a function variot for plotting the marginal temporal
######### variogram at distance 0
variot <- function(model,vario=vario){
  tseq <- seq(0,400,20)
  grid <- expand.grid(seq(0,50000,10),tseq)
  dg   <- data.frame(spacelag=grid[,1],timelag=grid[,2])
  table<- variogramSurface(model,dg)
  nx   <- length(seq(0,50000,10))
  ny   <- length(seq(0,15,1))
  plot(vario[vario$dist==0,5],vario[vario$dist==0,3],ylim=range(c(0,vario[vario$dist==0,3],
       table[table[,1]==0,3]),na.rm=T))
  lines(tseq,table[table[,1]==0,3])
}

######### we calculate the empirical space-time variogram
vario <- variogramST(PM10res~1,data=OBS_sel)

######### separable model, least squares fit
separableModel <- vgmST("separable",space=vgm(0.9,"Exp",10000,0.1),
                        time=vgm(0.9,"Exp",3.5,0.1),sill=40)
separable_fit  <- fit.StVariogram(model=separableModel,object=vario)
plot(vario,separable_fit,all=T,map=F)
```



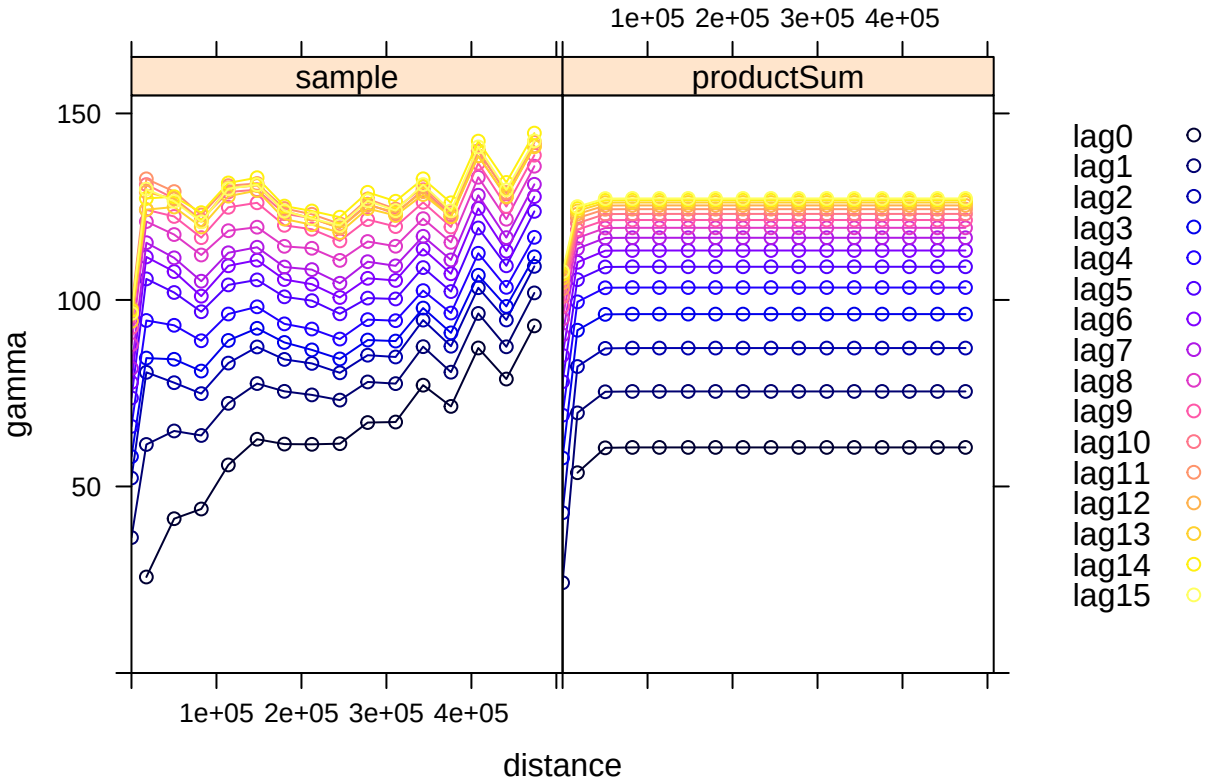
```
######### separable model, manual fit
sill <- 250
separable_man <- vgmST("separable",space=vgm(.18,"Exp",1.5e5,0.01,
                        add.to=vgm(.12,"Exp",8e3,.01)),time=vgm(80/sill,"Exp",1670,19/sill,
                        add.to=vgm(66/sill,"Exp",85,0)),sill=sill)
plot(vario,separable_man,all=T,map=F)
```



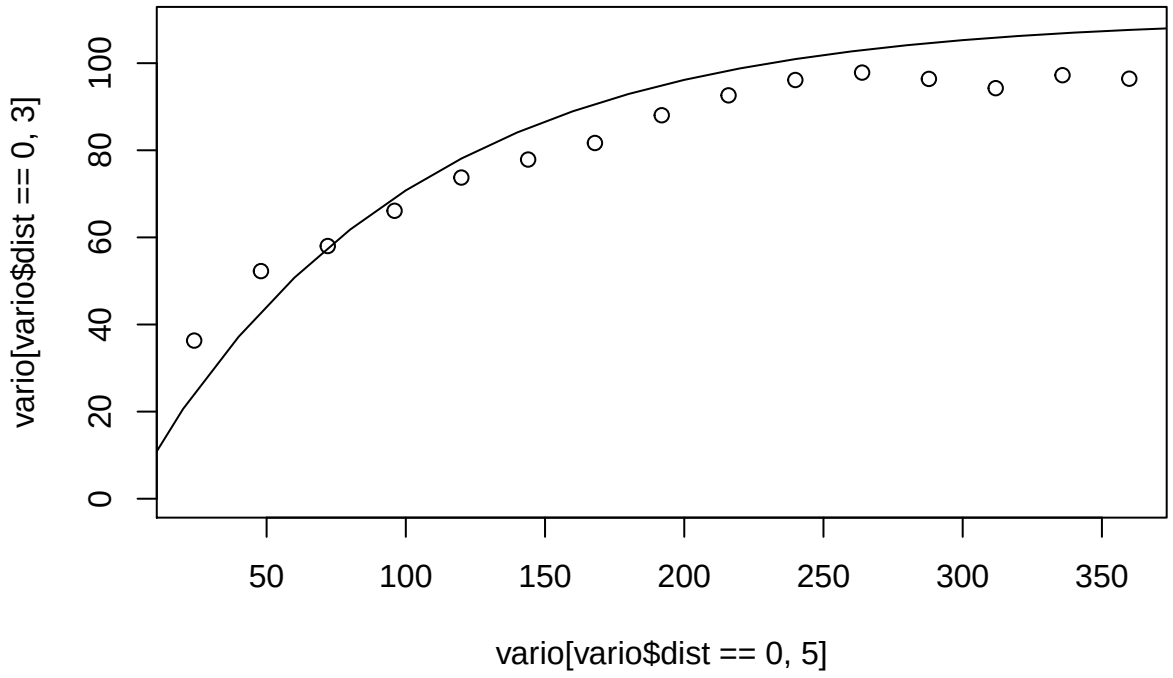

```
variot(separable_man,vario)
```



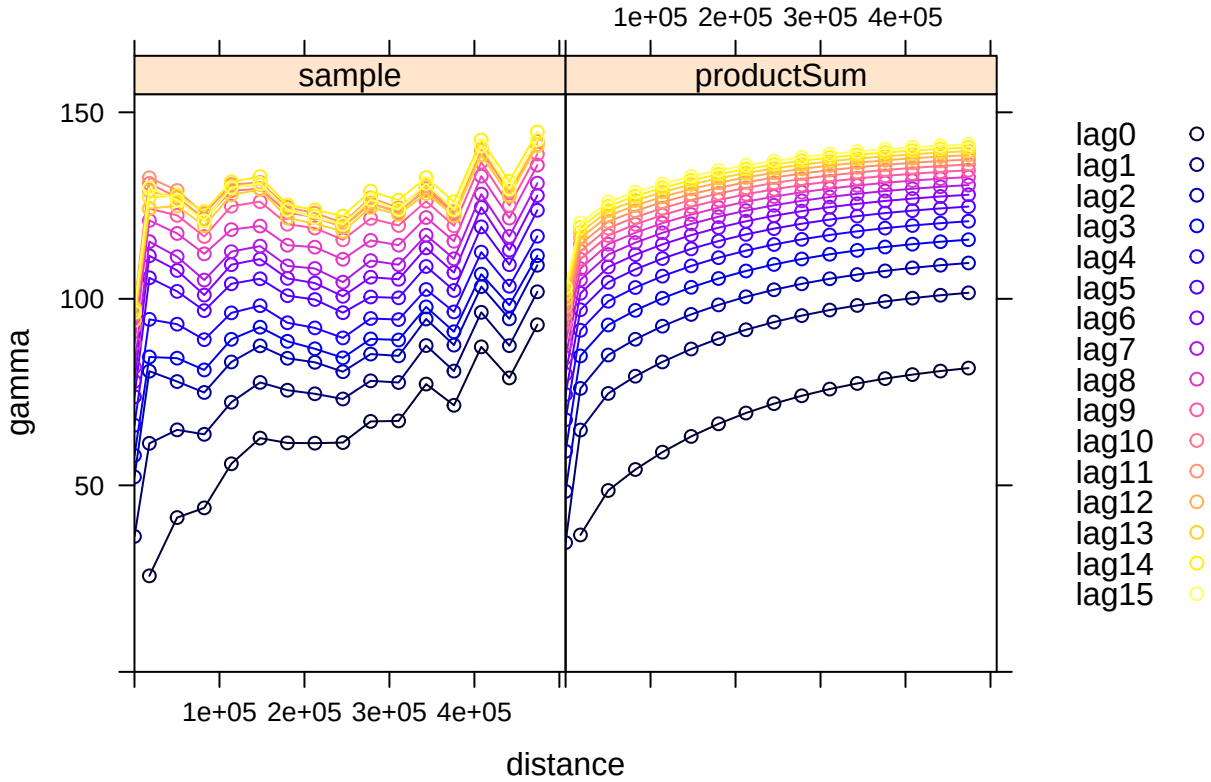

```
######### product sum model, least squares fit
ProductSum     <- vgmST("productSum",space =vgm(9,"Exp",8e3,1),time=vgm(8,"Exp",106,2),k=2)
ProductSum_fit <- fit.StVariogram(model=ProductSum,object=vario)
plot(vario,ProductSum_fit,all=T,map=F)
```



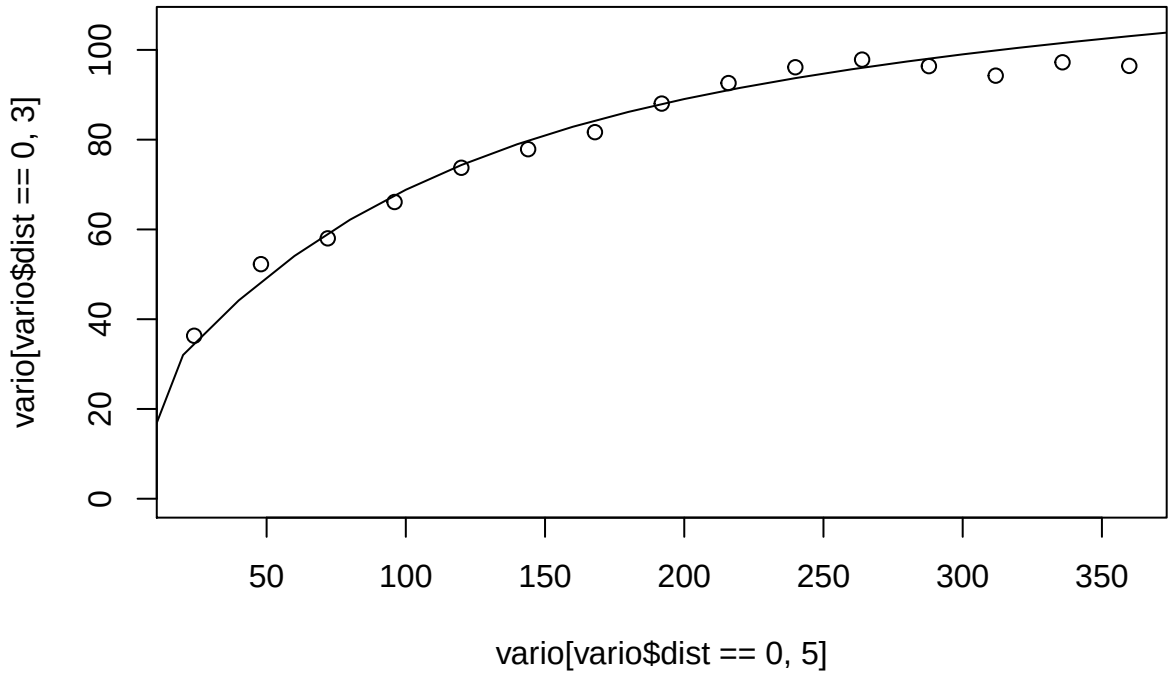
```
variot(ProductSum_fit,vario)
```



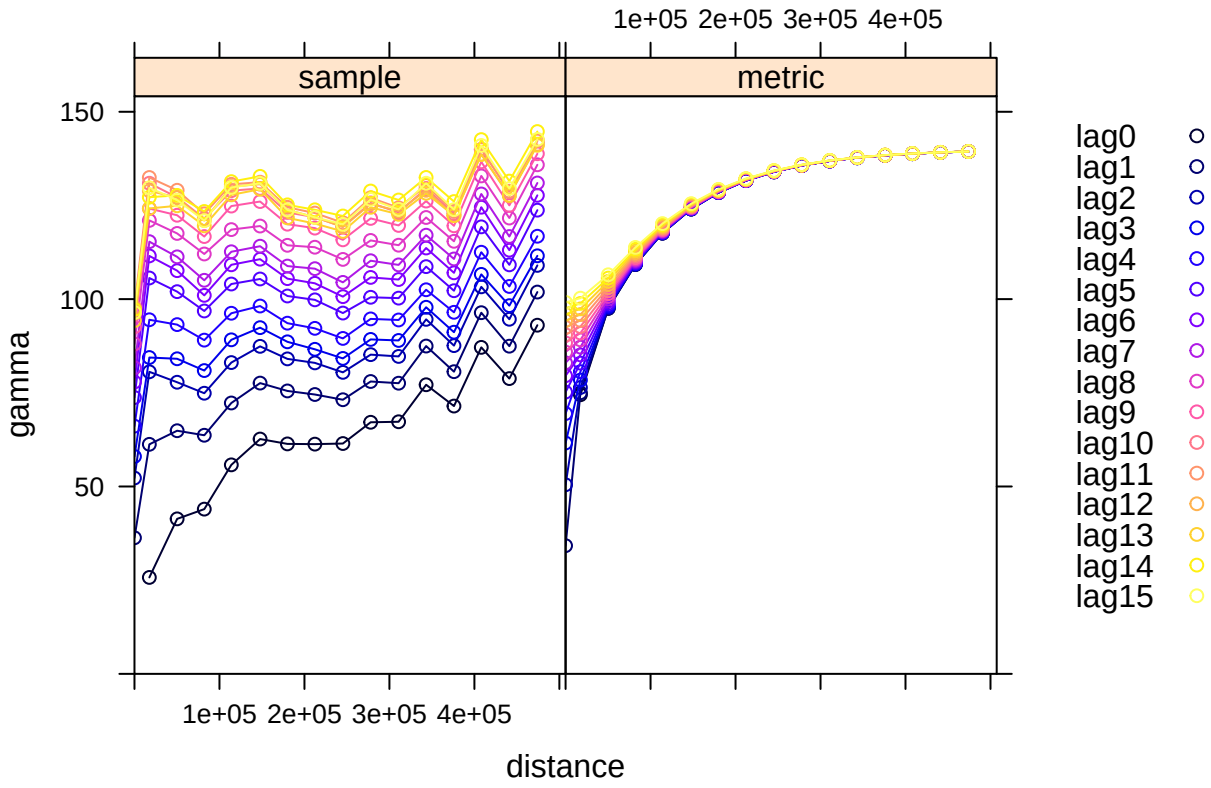

```
######### product sum model, manual fit
ProductSum_man <- vgmST("productSum",space=vgm(12.5,"Exp",2e5,0,add.to=vgm(10,"Exp",9e3,0)),
                        time=vgm(35,"Exp",800,9.5,add.to=vgm(36,"Exp",85,0)),k=0.035)
plot(vario,ProductSum_man,all=T,map=F)
```



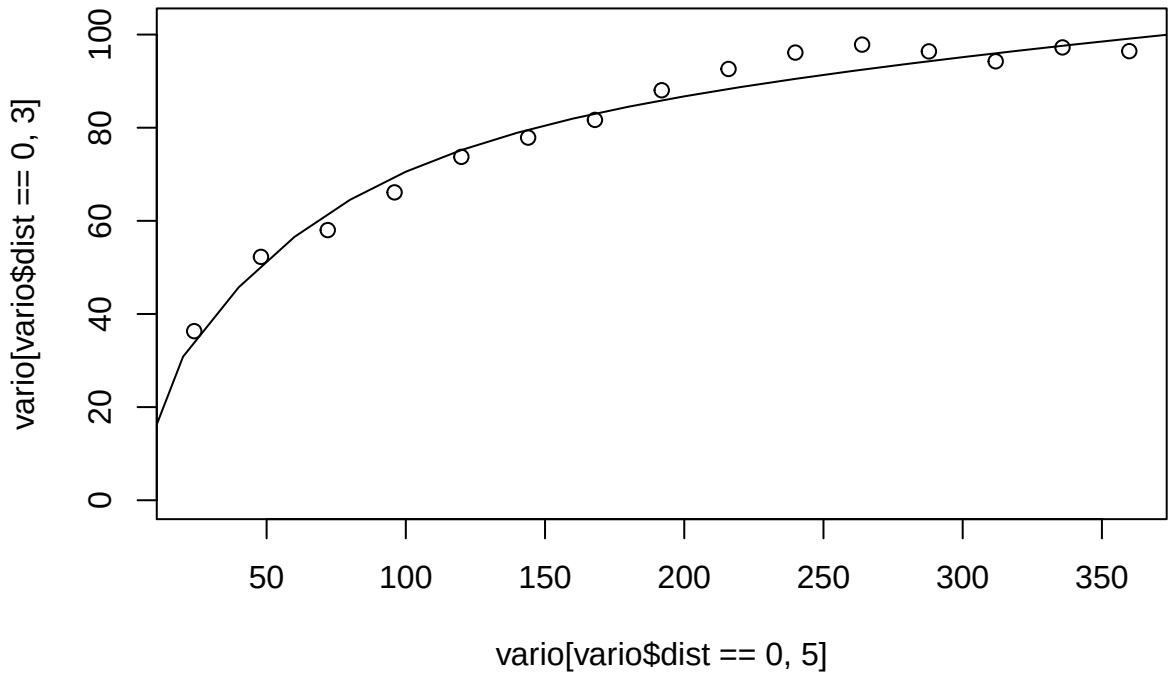
```
variot(ProductSum_man,vario)
```



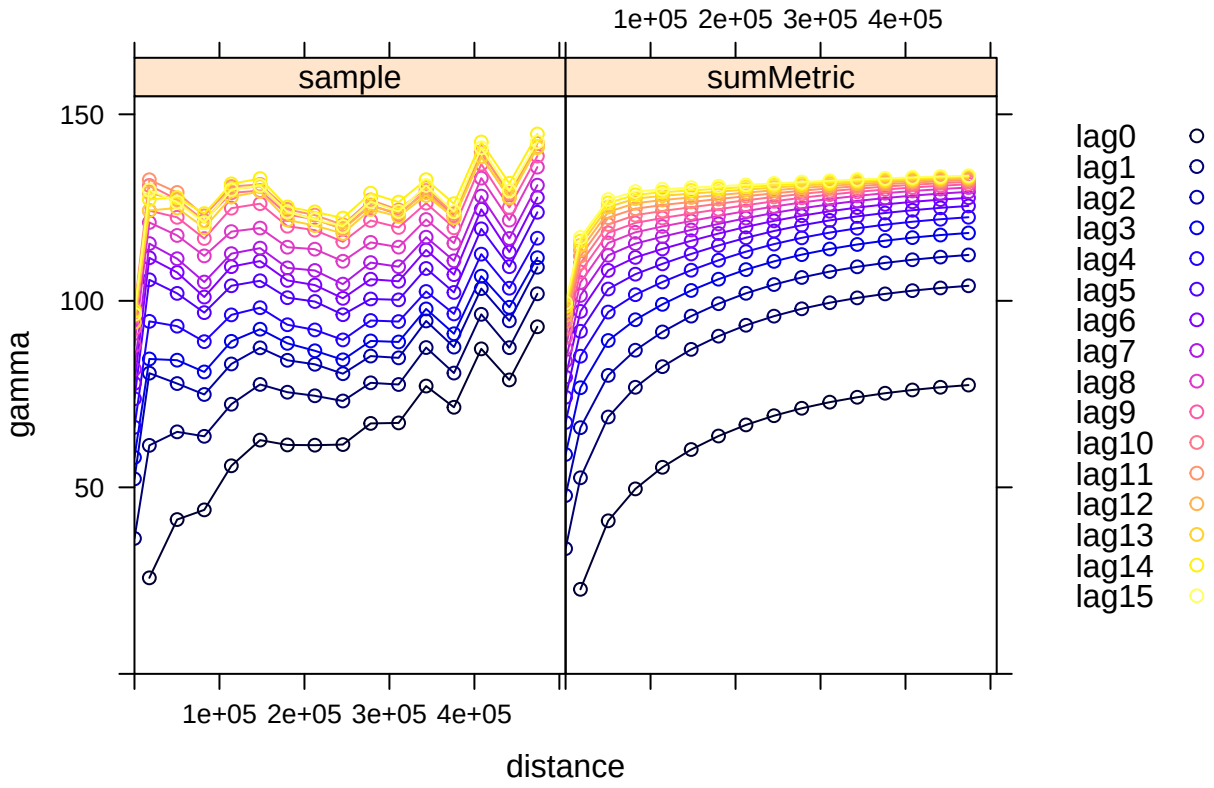

```
######### metric model, manual fit
Metric <- vgmST("metric",joint=vgm(60,"Exp",8e3,10,add.to=vgm(70,"Exp",1e5,0,stAni=1000)),
                stAni=150);model=Metric
plot(vario,Metric,map=F,all=T)
```



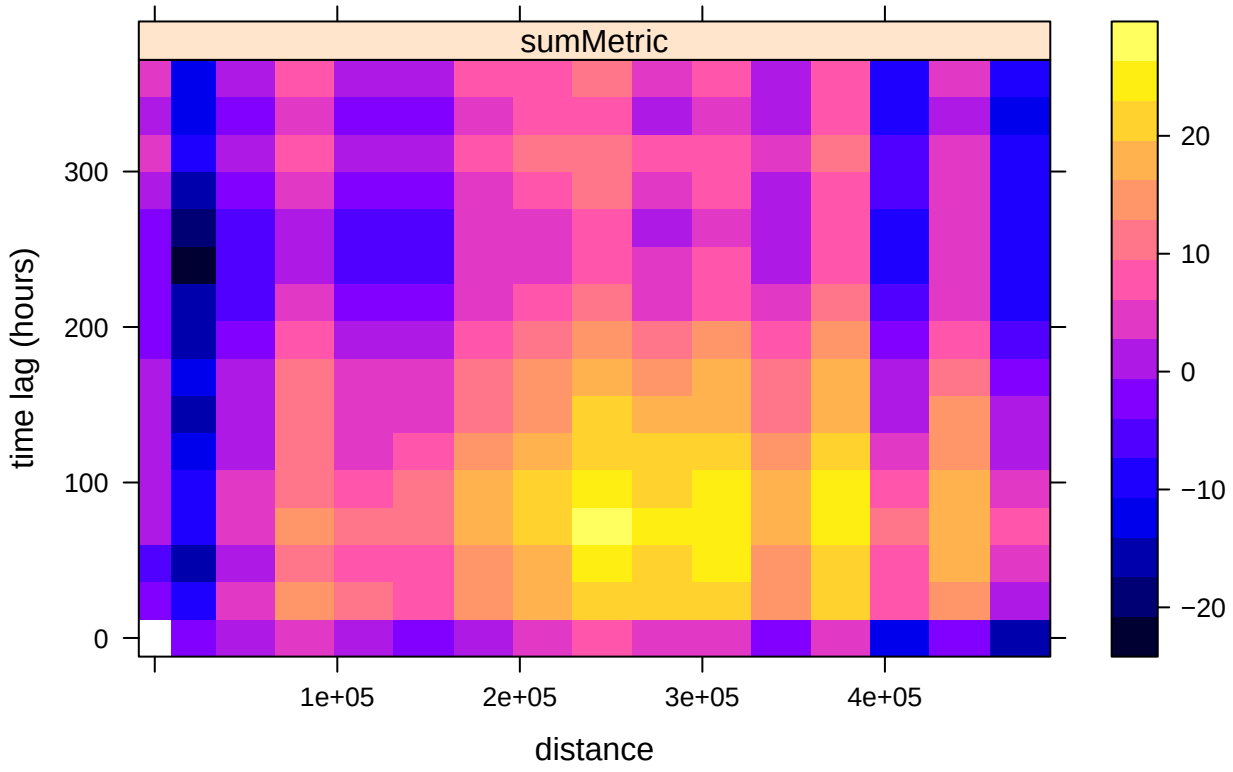
```
variot(Metric,vario)
```



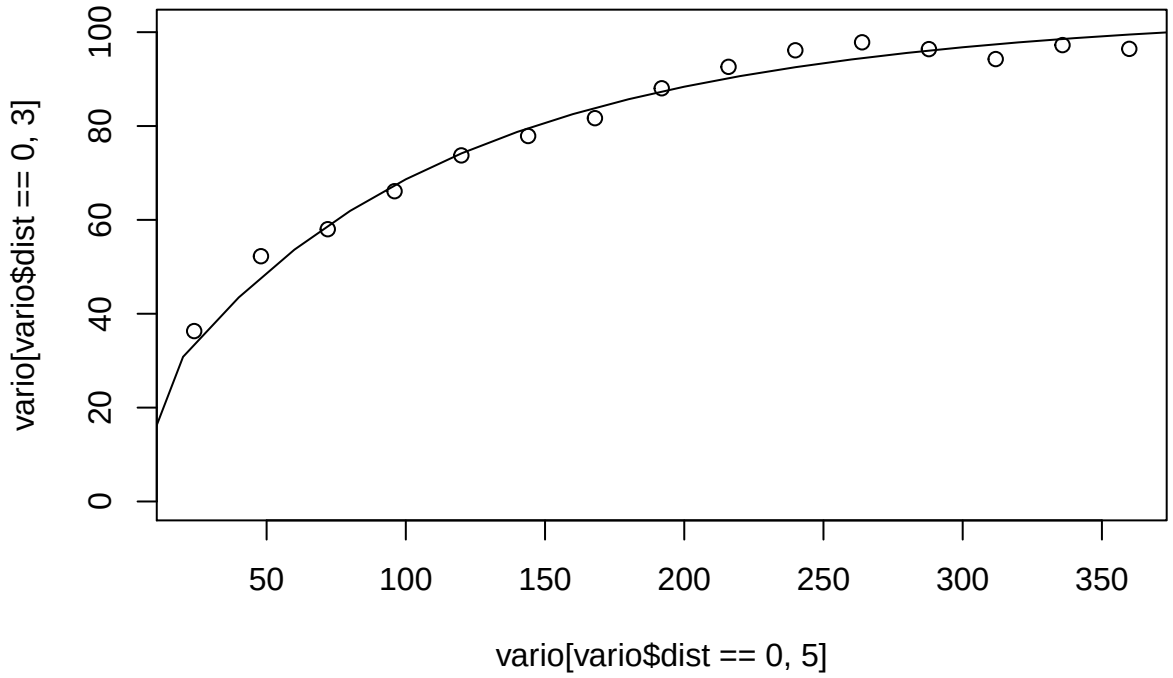

```
######### sum-metric model, manual fit
SumMetric <- vgmST("sumMetric",space=vgm(30,"Exp",20000),time=vgm(40,"Exp",70,15),
                   joint=vgm(50,"Exp",1.6e5,0),stAni=1000)
plot(vario,SumMetric,map=F,all=T)
```



```
plot(vario,SumMetric,map=T,diff=T)
```




```
variot(SumMetric,vario)
```



```
######### create list of all models fitted with gstat
models <- list(separable_fit,separable_man,ProductSum_fit,ProductSum_man,Metric,SumMetric)
######### save list of all models
save(models,file=paste0(DATA,"fits_gstat.Rdata"))
```

5) Prediction

5.1) Data Preparation

```
library(CompRandFld)
library(fields)
library(gstat)
library(rgdal)
library(sp)
library(spam)
library(spacetime)
library(RColorBrewer)
library(raster)


DATA = "data/"
load(paste0(DATA, "data.Rdata"))
load(paste0(DATA, "stationsforfitting.Rdata"))
load(paste0(DATA, "timesforfits.Rdata"))
load(paste0(DATA, "CHM.Rdata"))


### Getting the CHIMERE grid coordinates
coord <- cbind(CHM$lon, CHM$lat)
coordchm <- expand.grid(unique(CHM$lon), unique(CHM$lat))
coord.spchm <- SpatialPoints(coordchm, proj4string = CRS("+init=epsg:4326"))
# transform to Lambert 93 coordinate projection for approximately correct
# spatial geodesic distances
coord.spchm <- spTransform(coord.spchm, CRSobj = CRS("+init=epsg:2154"))

loc <- coord.spchm@coords/1000
loc_to_pred <- loc  # prediction
ns <- ncol(data)  # number of stations
nT <- nrow(data)
lT <- 2  # depth of the past used for prediction
mean.est <- mean(data)
mean.est
```

```
## [1] 8.024326
```

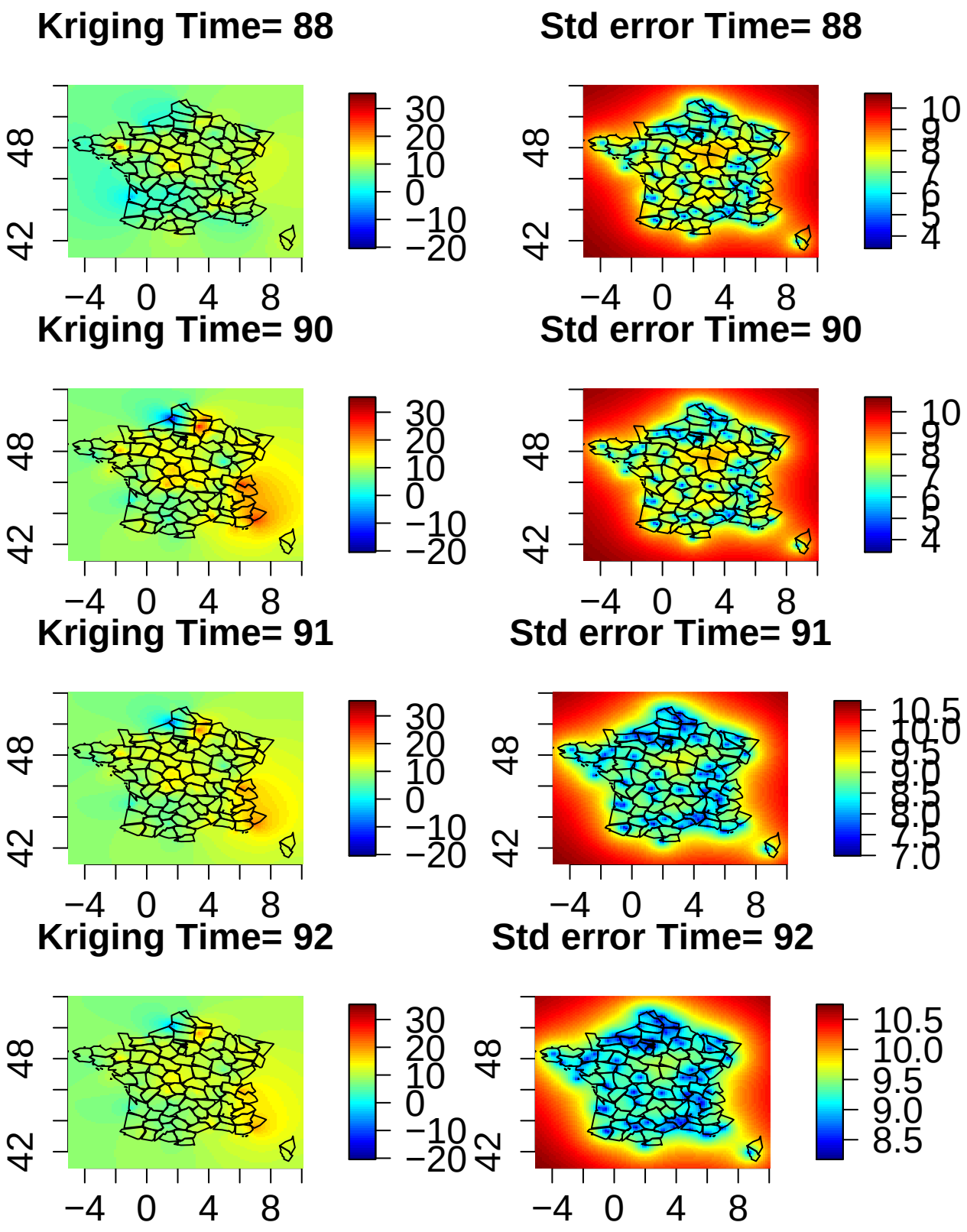
```
var.est <- var(as.numeric(data))
var.est
```

```
## [1] 116.0013
```

```
### Getting the stations coordinates
coord.sel <- cbind(stations.fit$station_longitude_deg, stations.fit$station_latitude_deg)
coord.sp <- SpatialPoints(coord.sel, proj4string = CRS("+init=epsg:4326"))
# transform to Lambert 93 coordinate projection for approximately correct
# spatial geodesic distances
coord.spt <- spTransform(coord.sp, CRSobj = CRS("+init=epsg:2154"))

coords <- coord.spt@coords/1000  # coordinates of the selected stations
coord.sel <- coords
times <- c(88:90)  # three last observation dates

dobs <- data[(nT - lT):nT, ]  # observed data locations
tt <- c(88, 90:92)
times_to_pred <- tt  # prediction times
```

5.2) A) Kriging with *CompRandFld*

```
load(paste0(DATA, "fitsPL.Rdata"))  # load the fitted models
nummod <- 8
corrmodel <- fitsPL[[nummod]]$corrmodel
param <- as.vector(fitsPL[[nummod]]$param)
param <- list(nugget = param[1], power_s = param[2], power_t = param[3], scale_s = param[4],
    scale_t = param[5], sep = param[6], sill = var.est, mean = mean.est)
pr <- Kri(loc = loc_to_pred, time = times_to_pred, coordx = coords, coordt = times,
    corrmodel = corrmodel, param = param, data = dobs)

### Plots of the predicted residuals with *image.plot*

grx <- unique(CHM$lon)
gry <- unique(CHM$lat)
rat <- diff(range(grx))/diff(range(gry))
for (i in 1:4) {
    par(pin = c(1.6, 1.6/rat))
    image.plot(grx, gry, matrix(pr$pred[i, ], nrow = length(grx)), col = tim.colors(64),
        zlim = c(-20, 35), main = paste("Kriging Time=", tt[i]), ylab = "",
        xlab = "", cex.axis = 1.5, axis.args = list(cex.axis = 1.5), asp = 1,
        cex.main = 1.5, bty = "n")
    map("france", add = TRUE)
    # namefig <- paste0('krisres',i,'.pdf') dev.copy2pdf(file=namefig)
    # system(paste0('pdfcrop ',namefig,' ',namefig)) # linux with pdfcrop only
    par(pin = c(1.6, 1.6/rat))
    image.plot(grx, gry, matrix(sqrt(pr$varpred[i, ]), nrow = length(grx)),
        col = tim.colors(64), main = paste("Std error Time=", tt[i]), ylab = "",
        xlab = "", cex.axis = 1.5, axis.args = list(cex.axis = 1.5), asp = 1,
        cex.main = 1.5, bty = "n")
    map("france", add = TRUE)
    # namefig <- paste0('kristdres',i,'.pdf') dev.copy2pdf(file=namefig)
    # system(paste0('pdfcrop ',namefig,' ',namefig)) # linux with pdfcrop only

}
```

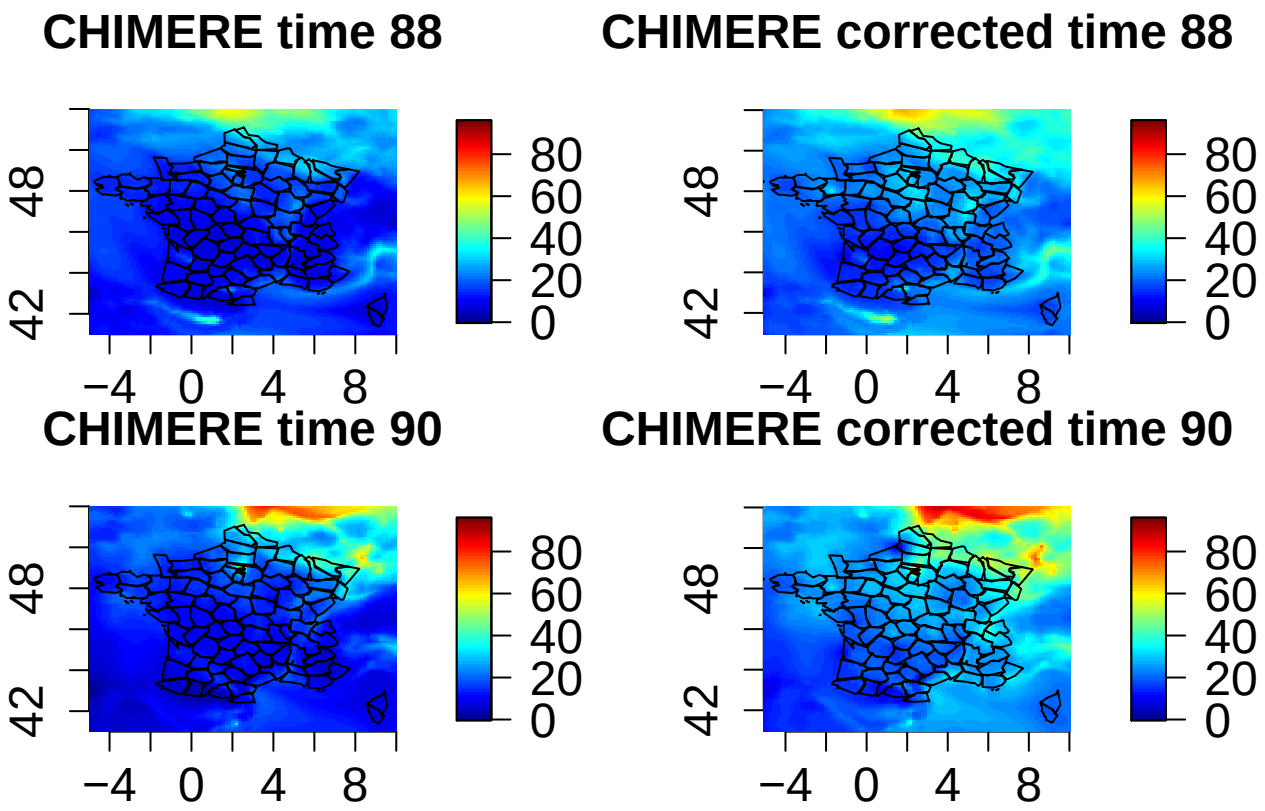


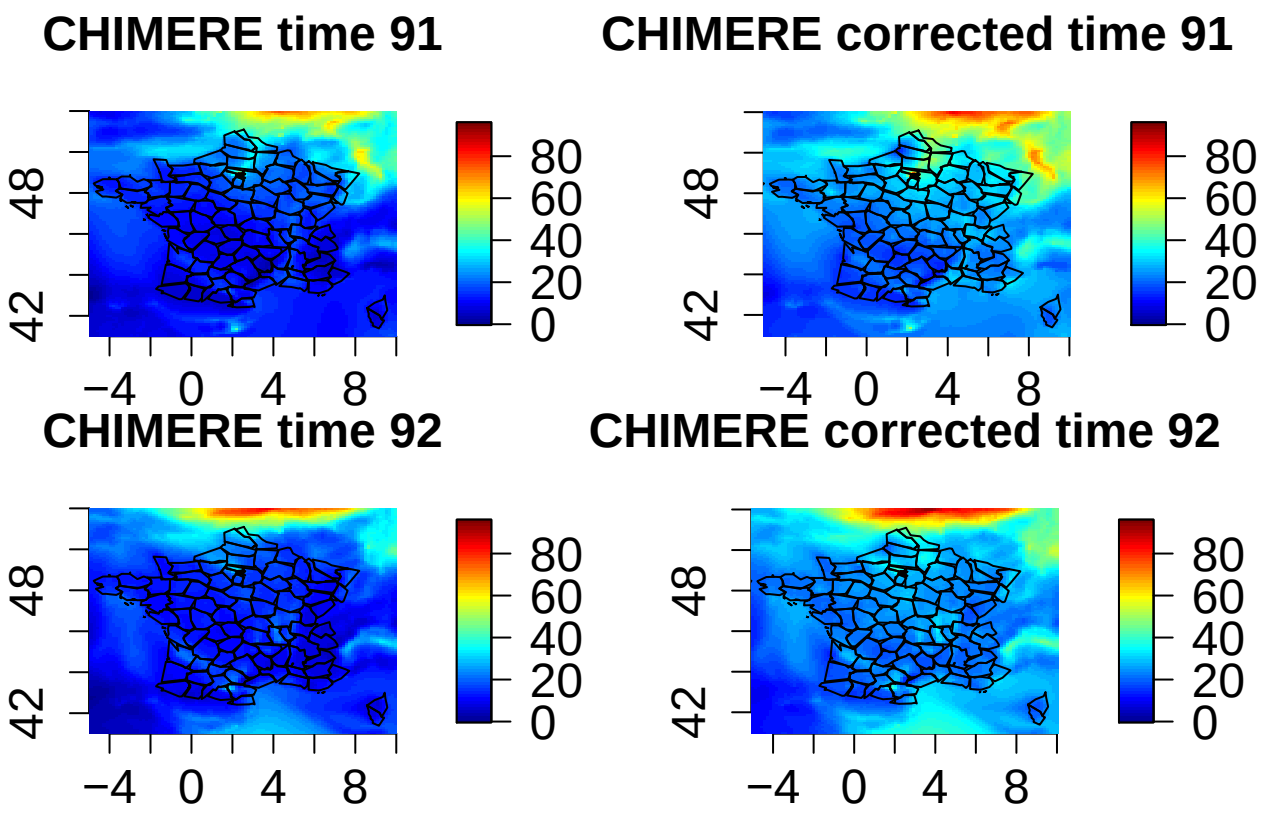

```
par(mfrow = c(1, 1))
```

```
### Plots of CHIMERE and corrected CHIMERE predictions
CHM_mat <- matrix(CHM$PM10, nrow = nrow(loc))
dim(CHM_mat)
```

```
## [1] 11211   365
```

```
zlim <- range(c(CHM_mat[, tt], pr$pred[, ] + t(CHM_mat[, tt])))
for (i in 1:4) {
    par(pin = c(1.6, 1.6/rat))
    image.plot(grx, gry, matrix(CHM_mat[, tt[i]], nrow = length(grx)), col = tim.colors(64),
        zlim = zlim, xlab = "", main = paste0("CHIMERE time ", tt[i]), ylab = "",
        xlim = range(grx), ylim = range(gry), cex.axis = 1.5, axis.args = list(cex.axis = 1.5),
        asp = 1, cex.main = 1.5, bty = "n")
    map("france", add = TRUE)
    # namefig <- paste0('predpm10',tt[i],'chm.pdf') dev.copy2pdf(file=namefig)
    # system(paste0('pdfcrop ',namefig,' ',namefig)) # linux with pdfcrop only

    par(pin = c(1.6, 1.6/rat))
    image.plot(grx, gry, matrix(pr$pred[i, ] + CHM_mat[, tt[i]], nrow = length(grx)),
        col = tim.colors(64), zlim = zlim, main = paste0("CHIMERE corrected time ",
            tt[i]), ylab = "", xlab = "", cex.axis = 1.5, axis.args = list(cex.axis = 1.5),
        asp = 1, cex.main = 1.5, bty = "n")
    map("france", add = TRUE)
    # namefig <- paste0('predpm10',tt[i],'.pdf') dev.copy2pdf(file=namefig)
    # system(paste0('pdfcrop ',namefig,' ',namefig)) # linux with pdfcrop only
    # title(main = paste('Kriging Time='
    # ,tt[i]),outer=TRUE,line=-1,cex.main=1.5)
}
```






```
par(mfrow = c(1, 1))

### Plots with the gstat tools
load(paste0(DATA, "STFDF_jour.Rdata"))
pts <- expand.grid(grx, gry)
pts <- SpatialPoints(pts)
temps <- STFDF_jour@time[tt]
mydata2 <- as.data.frame(cbind(as.double(CHM_mat[, tt]), as.double(CHM_mat[,
    tt] + t(pr$pred))))
names(mydata2) <- c("CHIMERE", "CHIMERE corrected")
CHMcor = STFDF(sp = pts, time = temps, data = mydata2)
library(rworldmap)
library(maptools)
library(ggmap)
world <- getMap(resolution = "low")
proj4string(world)
```

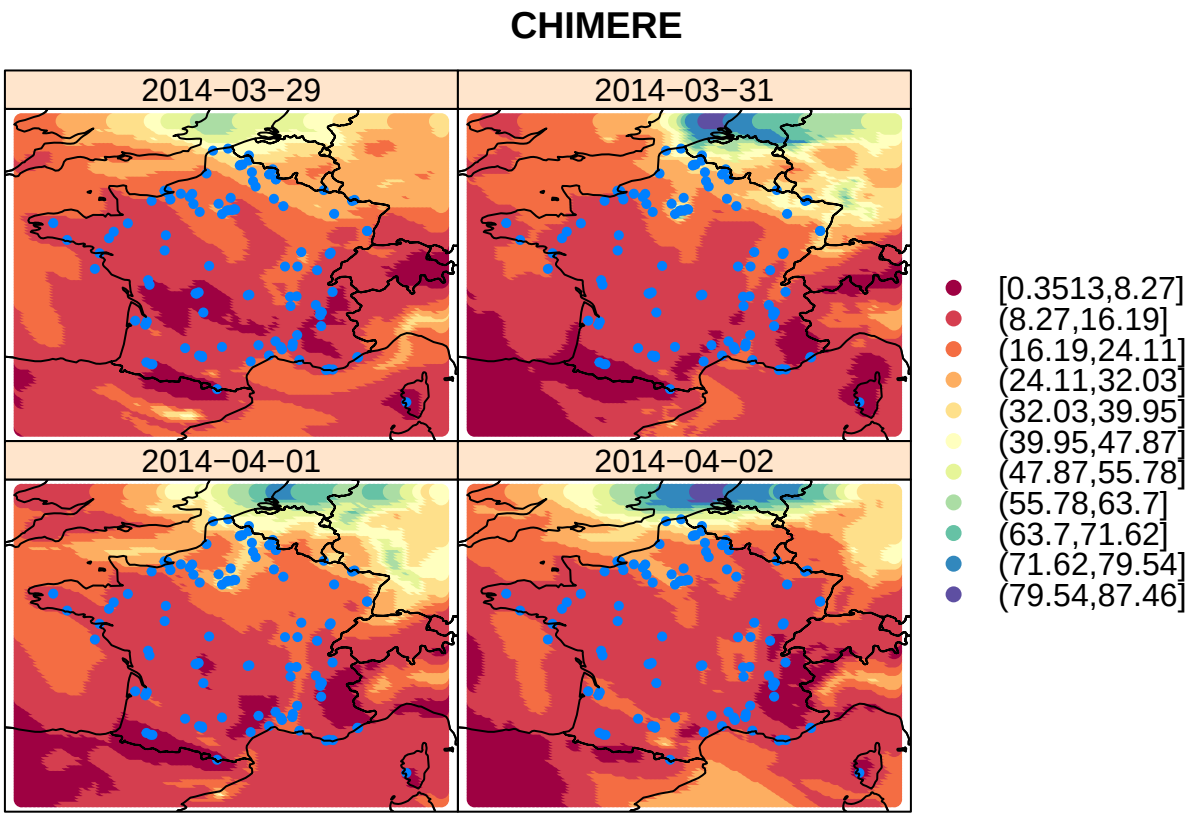
```
## [1] "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"
```

```
europe.limits <- geocode(c("Spain", "Greece", "Ireland", "Norway", "Ukraine"))
```

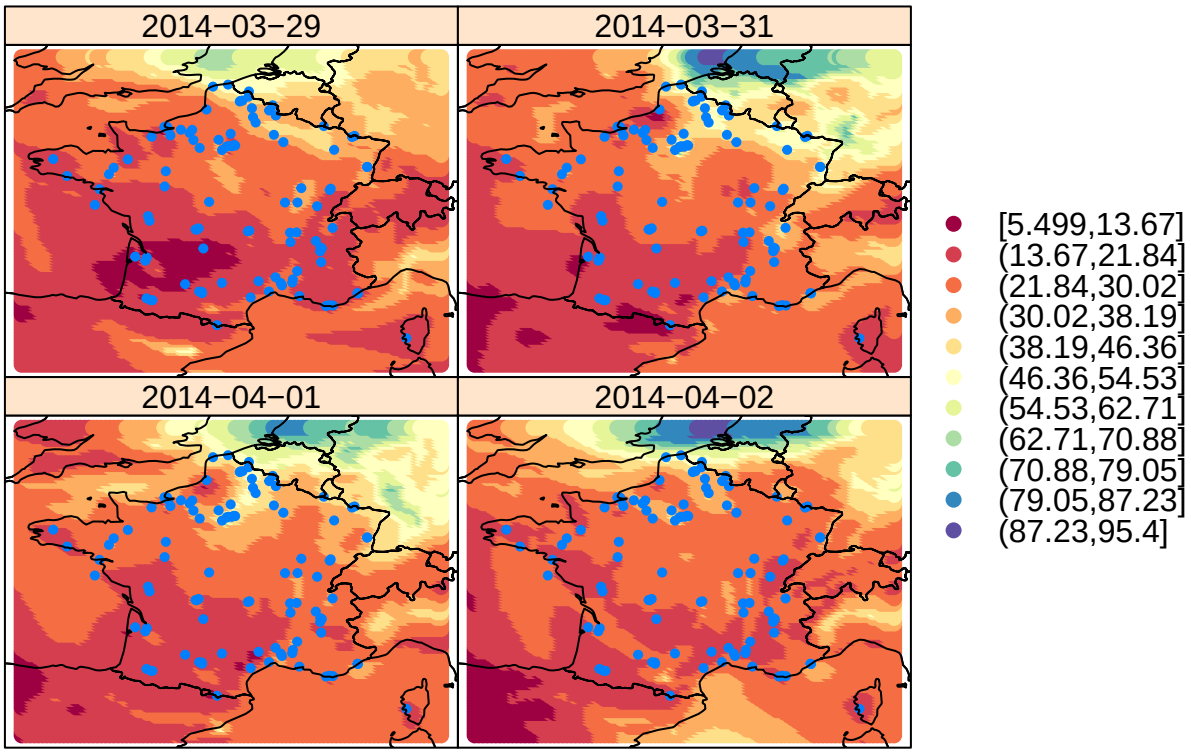
```
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Spain&sensor=false

## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Greece&sensor=false

## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Ireland&sensor=false
```

```
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Norway&sensor=false

## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Ukraine&sensor=false
```

```
lvls <- seq(0, 100, 10)


stplot(CHMcor[, 1:4, 1], col.regions = brewer.pal(11, "Spectral"), at = lvls,
    cuts = 11, sp.layout = list(coord.sp, world, cex = 0.5, pch = 19, first = FALSE),
    aspect = "iso", lattice.options = list(key = list(cex.title = 1.5)))
```



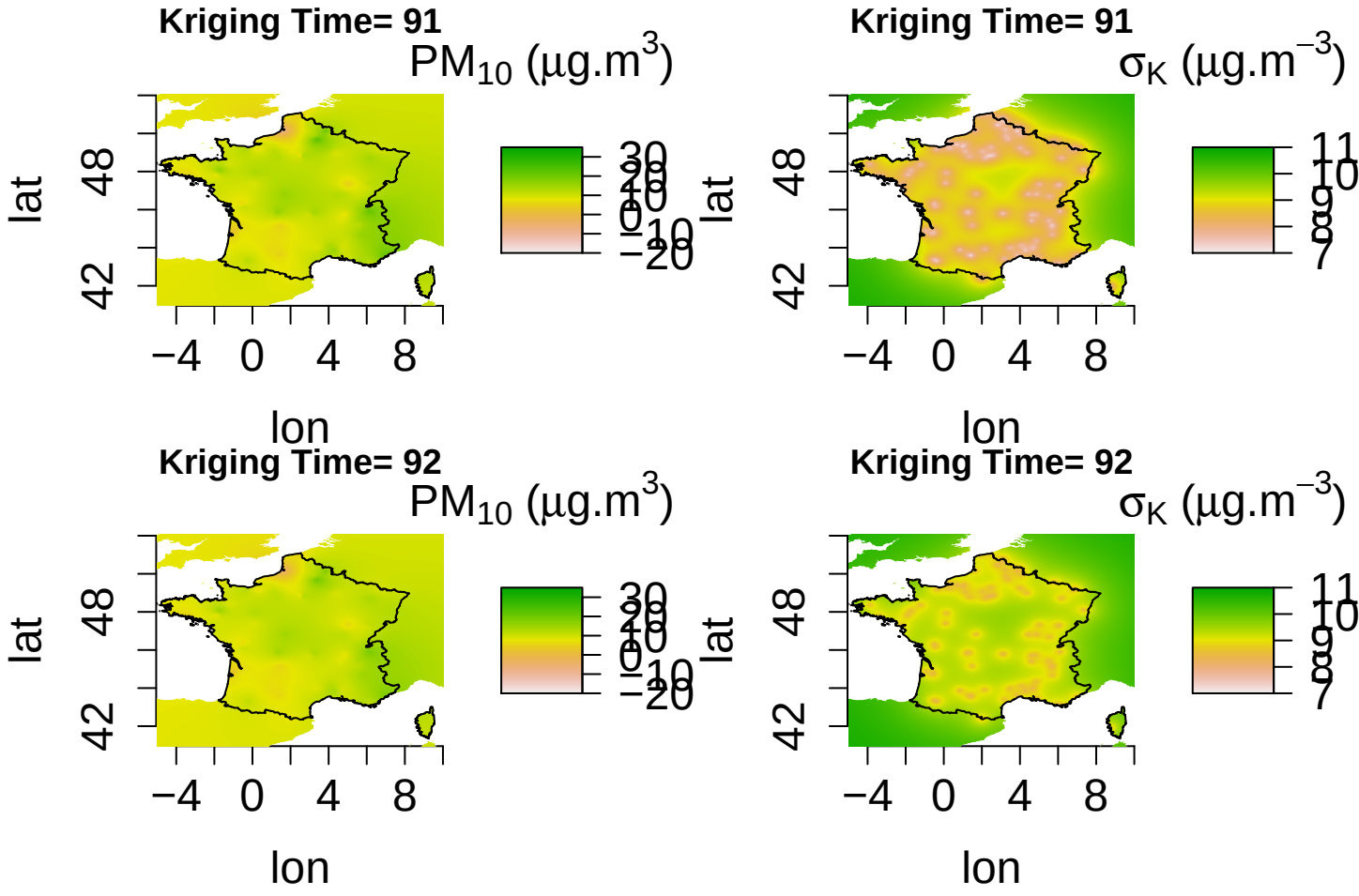

```
# namefig <- 'chimstplot2.pdf' dev.copy2pdf(file=namefig)
# system(paste0('pdfcrop ',namefig,' ',namefig)) # linux with pdfcrop only

stplot(CHMcor[, 1:4, 2], col.regions = brewer.pal(11, "Spectral"), at = lvls,
    cuts = 11, sp.layout = list(coord.sp, world, cex = 0.5, pch = 19, first = FALSE))
```




```
# namefig <- 'chimcorrstplot2.pdf' dev.copy2pdf(file=namefig)
# system(paste0('pdfcrop ',namefig,' ',namefig)) # linux with pdfcrop only

### Plots with raster tools

europe = readOGR("maps", p4s = NULL, layer = "carto_eur")


## OGR data source with driver: ESRI Shapefile
## Source: "maps", layer: "carto_eur"
## with 75 features
## It has 32 fields
```

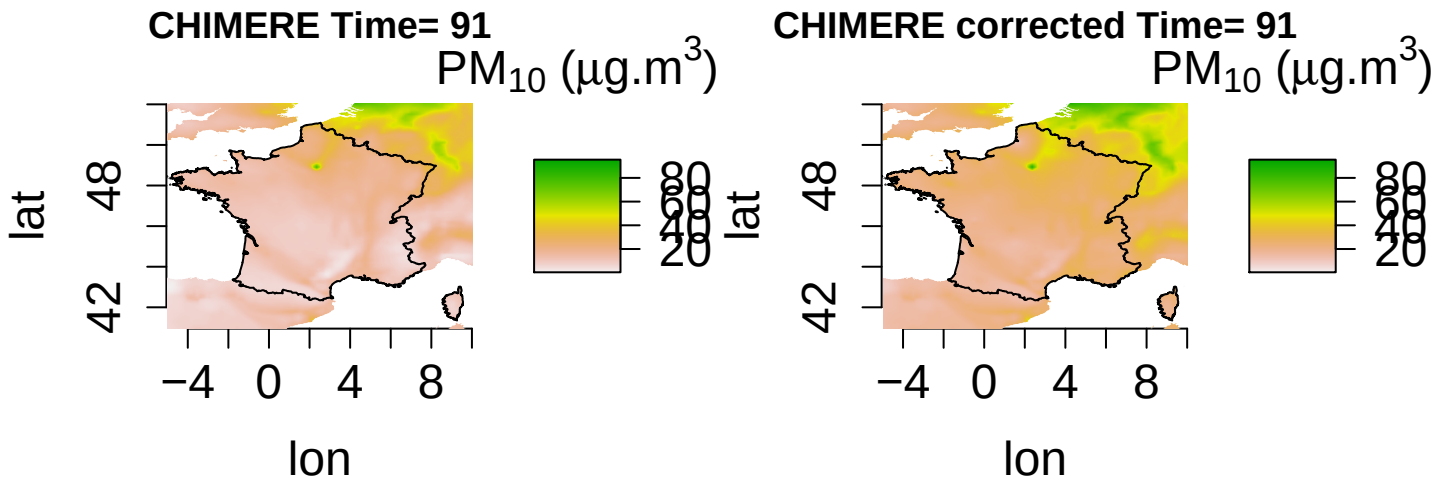
```
grx <- unique(CHM$lon)
gry <- unique(CHM$lat)

for (i in 3:4) {
    ras <- raster(list(x = grx, y = gry, z = matrix(pr$pred[i, ], nrow = length(grx)),
        col = tim.colors(64)))
    ras <- disaggregate(ras, fact = c(5, 5), method = "bilinear")
    ras <- rasterize(europe, ras, mask = T)
    plot(ras, xlab = "lon", ylab = "lat", xlim = c(-5, 10), zlim = c(-20, 35),
        legend.width = 3, cex.axis = 1.5, axis.args = list(cex.axis = 1.5),
        cex.lab = 1.5, asp = 1, main = paste("Kriging Time=", tt[i]), legend.args = list(text = expressi
            paste(PM[10], " (", mu, "g.", m^3, ")", sep = "")), line = 1.5,
            cex = 1.5), bty = "n", box = F)
    plot(europe[europe$COUNTRY == "France", ], xlim = range(grx), ylim = range(gry),
        add = T)
```

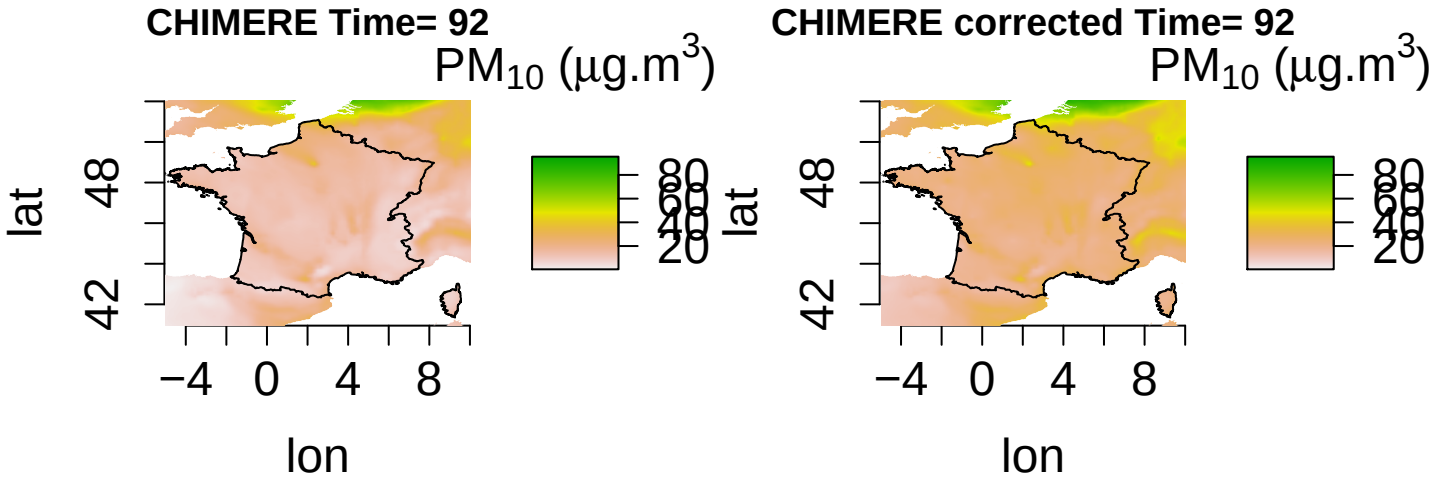
```
    # namefig <- paste0('krires',i,'ras.pdf') dev.copy2pdf(file=namefig)
    # system(paste0('pdfcrop ',namefig,' ',namefig)) # linux with pdfcrop only

    ras_sd <- raster(list(x = grx, y = gry, z = matrix(sqrt(pr$varpred[i, ]),
        nrow = length(grx)), col = tim.colors(64)))
    ras_sd <- disaggregate(ras_sd, fact = c(5, 5), method = "bilinear")
    ras_sd <- rasterize(europe, ras_sd, mask = T)
    plot(ras_sd, xlab = "lon", ylab = "lat", xlim = c(-5, 10), zlim = c(7, 11),
        legend.width = 3, cex.axis = 1.5, axis.args = list(cex.axis = 1.5),
        cex.lab = 1.5, asp = 1, main = paste("Kriging Time=", tt[i]), legend.args = list(text = expressi
            paste(sigma[K], " (", mu, "g.", m^-3, ")", sep = ""), line = 1.5,
            cex = 1.5), bty = "n", box = F)
    plot(europe[europe$COUNTRY == "France", ], xlim = range(grx), ylim = range(gry),
        add = T)
    # namefig <- paste0('kristdres',i,'ras.pdf') dev.copy2pdf(file=namefig)
    # system(paste0('pdfcrop ',namefig,' ',namefig)) # linux with pdfcrop only
}
```




```
# Comparing CHIMERE and corrected CHIMERE
zlim <- range(c(CHM_mat[, tt], pr$pred[, ] + t(CHM_mat[, tt])))
for (i in 3:4) {
    ras <- raster(list(x = grx, y = gry, z = matrix(CHM_mat[, tt[i]], nrow = length(grx)),
        col = tim.colors(64)))
```

```
    ras <- disaggregate(ras, fact = c(5, 5), method = "bilinear")
    ras <- rasterize(europe, ras, mask = T)
    plot(ras, xlab = "lon", ylab = "lat", xlim = c(-5, 10), zlim = zlim, legend.width = 3,
        cex.axis = 1.5, axis.args = list(cex.axis = 1.5), cex.lab = 1.5, asp = 1,
        main = paste("CHIMERE Time=", tt[i]), legend.args = list(text = expression("     ",
            paste(PM[10], " (", mu, "g.", m^3, ")", sep = "")), line = 1.5,
            cex = 1.5), bty = "n", box = F)
    plot(europe[europe$COUNTRY == "France", ], xlim = range(grx), ylim = range(gry),
        add = T)
    # namefig <- paste0('predpm10',tt[i],'chmras.pdf')
    # dev.copy2pdf(file=namefig) system(paste0('pdfcrop ',namefig,' ',namefig))
    # # linux with pdfcrop only

    ras_sd <- raster(list(x = grx, y = gry, z = matrix(pr$pred[i, ] + CHM_mat[,
        tt[i]], nrow = length(grx)), col = tim.colors(64)))
    ras_sd <- disaggregate(ras_sd, fact = c(5, 5), method = "bilinear")
    ras_sd <- rasterize(europe, ras_sd, mask = T)
    plot(ras_sd, xlab = "lon", ylab = "lat", xlim = c(-5, 10), zlim = zlim,
        legend.width = 3, cex.axis = 1.5, axis.args = list(cex.axis = 1.5),
        cex.lab = 1.5, asp = 1, main = paste("CHIMERE corrected Time=", tt[i]),
        legend.args = list(text = expression("     ", paste(PM[10], " (", mu,
            "g.", m^3, ")", sep = "")), line = 1.5, cex = 1.5), bty = "n", box = F)
    plot(europe[europe$COUNTRY == "France", ], xlim = range(grx), ylim = range(gry),
        add = T)
    # namefig <- paste0('predpm10',tt[i],'ras.pdf') dev.copy2pdf(file=namefig)
    # system(paste0('pdfcrop ',namefig,' ',namefig)) # linux with pdfcrop only
}
```





5.2) B) Cross-validation with *CompRandFld*

```
tt <- 91
times_to_pred <- tt
times <- c(89, 90)
err <- matrix(NA, ns, 8)
for (nummod in 1:8) {
    corrmodel <- fitsPL[[nummod]]$corrmodel
    if (nummod == 1) {
        param <- as.vector(fitsPL[[nummod]]$param)
        param <- list(scale_s = param[1], scale_t = param[2], sill = var.est,
            mean = mean.est)
    } else if (nummod == 2) {
        param <- as.vector(fitsPL[[nummod]]$param)
        param <- list(nugget = param[1], scale_s = param[2], scale_t = param[3],
            sill = var.est, mean = mean.est)
    } else if (nummod == 3) {
        param <- as.vector(fitsPL[[nummod]]$param)
        param <- list(scale_s = param[1], power_s = 1, power_t = 1, scale_t = param[2],
            sep = param[3], sill = var.est, mean = mean.est)
    } else if (nummod == 4) {
        param <- as.vector(fitsPL[[nummod]]$param)
        param <- list(nugget = param[1], scale_s = param[2], power_s = 1, power_t = 1,
            scale_t = param[3], sep = param[4], sill = var.est, mean = mean.est)
    } else if (nummod == 5) {
        param <- as.vector(fitsPL[[nummod]]$param)
        param <- list(nugget = 0, scale_s = param[1], power_s = 0.5, power_t = 0.5,
            scale_t = param[2], sep = param[3], sill = var.est, mean = mean.est)
    } else if (nummod == 6) {
        param <- as.vector(fitsPL[[nummod]]$param)
        param <- list(nugget = param[1], scale_s = param[2], power_s = 0.5,
            power_t = 0.5, scale_t = param[3], sep = param[4], sill = var.est,
            mean = mean.est)
    } else if (nummod == 7) {
        param <- as.vector(fitsPL[[nummod]]$param)
        param <- list(power_s = param[1], power_t = param[2], scale_s = param[3],
            scale_t = param[4], sep = param[5], sill = var.est, mean = mean.est)
    } else if (nummod == 8) {
        param <- as.vector(fitsPL[[nummod]]$param)
```

```
        param <- list(nugget = param[1], power_s = param[2], power_t = param[3],
            scale_s = param[4], scale_t = param[5], sep = param[6], sill = var.est,
            mean = mean.est)
    }
    for (ista in 1:ns) {
        loc_to_pred <- coord.sel[ista, ]
        coords <- coord.sel[-ista, ]
        dobs <- data[(nT - 2):(nT - 1), -ista]

        prvc <- Kri(loc = loc_to_pred, time = times_to_pred, coordx = coords,
            coordt = times, corrmodel = corrmodel, param = param, data = dobs)
        err[ista, nummod] = data[nT, ista] - prvc$pred
    }
}
RMSE <- sqrt(apply(err^2, 2, mean))
RMSE
```

```
## [1] 9.701158 9.132365 9.319150 9.025487 9.211886 9.215222 9.222762 9.169025
```

5.2) C) External validation with *CompRandFld*

```
######### reloading saved data objects (if necessary)
load(paste0(DATA, "OBS_jour.Rdata"))
load(paste0(DATA, "CHM.Rdata"))
load(paste0(DATA, "stations.Rdata"))
######### extract French background stations
IDs.bg <- stations$station_european_code[stations$type_of_station == "Background"]
IDs.fr <- stations$station_european_code[stations$country_name == "France"]
IDs.bgfr <- intersect(IDs.bg, IDs.fr)
idx.stations <- which(stations$station_european_code %in% IDs.bgfr)
coord <- cbind(stations$station_longitude_deg, stations$station_latitude_deg)[idx.stations,
    ]
######### defining time window for data used for validation
tstart <- "2014-04-01"
tend <- "2014-04-02"  # at least two dates !
OBS_jour <- OBS_jour[OBS_jour$ID %in% IDs.bgfr, ]
OBS_sel2 <- stConstruct(OBS_jour, space = c("long", "lat"), time = "date", SpatialObj = SpatialPoints(O
    c("long", "lat")]))
OBS_sel2 <- as(OBS_sel2, "STFDF")
proj4string(OBS_sel2) <- "+init=epsg:4326"
OBS_sel2 <- OBS_sel2[, paste0(tstart, "::", tend)]
######### transforming to Lambert 93 coordinates system
OBS_sel2@sp <- spTransform(OBS_sel2@sp, CRS("+init=epsg:2154"))

######### interpolating gridded CHIMERE values to space-time observation points
library(fields)
CHM_sel <- CHM[as.character(CHM$time) >= tstart & as.character(CHM$time) <=
    tend, ]
grid.CHM <- unique(cbind(CHM_sel$lon, CHM_sel$lat))
CHM_mat2 <- matrix(CHM_sel$PM10, nrow = nrow(grid.CHM))
dim(CHM_mat2)
```

```
## [1] 11211     2
```

```
lon.grid <- sort(unique(CHM_sel$lon))
lat.grid <- sort(unique(CHM_sel$lat))
ind.sites.obs <- match(unique(OBS_sel2@data$ID), IDs.bgfr)
fun <- function(i) {
    tmp <- list(x = lon.grid, y = lat.grid, z = matrix(CHM_mat2[, i], length(lon.grid),
        length(lat.grid)))
    interp.surface(tmp, coord[ind.sites.obs, ])
}
CHM_sites <- sapply(1:ncol(CHM_mat2), fun)
######### saving as variable CHM
OBS_sel2@data$CHM <- as.numeric(CHM_sites)


tt <- 91
times_to_pred <- tt
times <- c(89, 90)
RMSE_ext <- NULL
val <- (OBS_sel2[, 1, "PM10"])$PM10
indexnona <- which(!is.na(val))
CHMval <- (OBS_sel2[, 1, "CHM"])$CHM[indexnona]
val <- val[indexnona]
for (nummod in 1:8) {
    corrmodel <- fitsPL[[nummod]]$corrmodel
    if (nummod == 1) {
        param <- as.vector(fitsPL[[nummod]]$param)
        param <- list(scale_s = param[1], scale_t = param[2], sill = var.est,
            mean = mean.est)
    } else if (nummod == 2) {
        param <- as.vector(fitsPL[[nummod]]$param)
        param <- list(nugget = param[1], scale_s = param[2], scale_t = param[3],
            sill = var.est, mean = mean.est)
    } else if (nummod == 3) {
        param <- as.vector(fitsPL[[nummod]]$param)
        param <- list(scale_s = param[1], power_s = 1, power_t = 1, scale_t = param[2],
            sep = param[3], sill = var.est, mean = mean.est)
    } else if (nummod == 4) {
        param <- as.vector(fitsPL[[nummod]]$param)
        param <- list(nugget = param[1], scale_s = param[2], power_s = 1, power_t = 1,
            scale_t = param[3], sep = param[4], sill = var.est, mean = mean.est)
    } else if (nummod == 5) {
        param <- as.vector(fitsPL[[nummod]]$param)
        param <- list(nugget = 0, scale_s = param[1], power_s = 0.5, power_t = 0.5,
            scale_t = param[2], sep = param[3], sill = var.est, mean = mean.est)
    } else if (nummod == 6) {
        param <- as.vector(fitsPL[[nummod]]$param)
        param <- list(nugget = param[1], scale_s = param[2], power_s = 0.5,
            power_t = 0.5, scale_t = param[3], sep = param[4], sill = var.est,
            mean = mean.est)
    } else if (nummod == 7) {
        param <- as.vector(fitsPL[[nummod]]$param)
        param <- list(power_s = param[1], power_t = param[2], scale_s = param[3],
            scale_t = param[4], sep = param[5], sill = var.est, mean = mean.est)
    } else if (nummod == 8) {
```

```
        param <- as.vector(fitsPL[[nummod]]$param)
        param <- list(nugget = param[1], power_s = param[2], power_t = param[3],
            scale_s = param[4], scale_t = param[5], sep = param[6], sill = var.est,
            mean = mean.est)
    }
    loc_to_pred <- OBS_sel2@sp@coords/1000
    coords <- coord.sel
    dobs <- data[(nT - 1):(nT), ]

    prvc <- Kri(loc = loc_to_pred, time = times_to_pred, coordx = coords, coordt = times,
        corrmodel = corrmodel, param = param, data = dobs)
    RMSE_ext <- c(RMSE_ext, sqrt(mean((val - prvc$pred[indexnona] - CHMval)^2)))
}
RMSE_ext
```

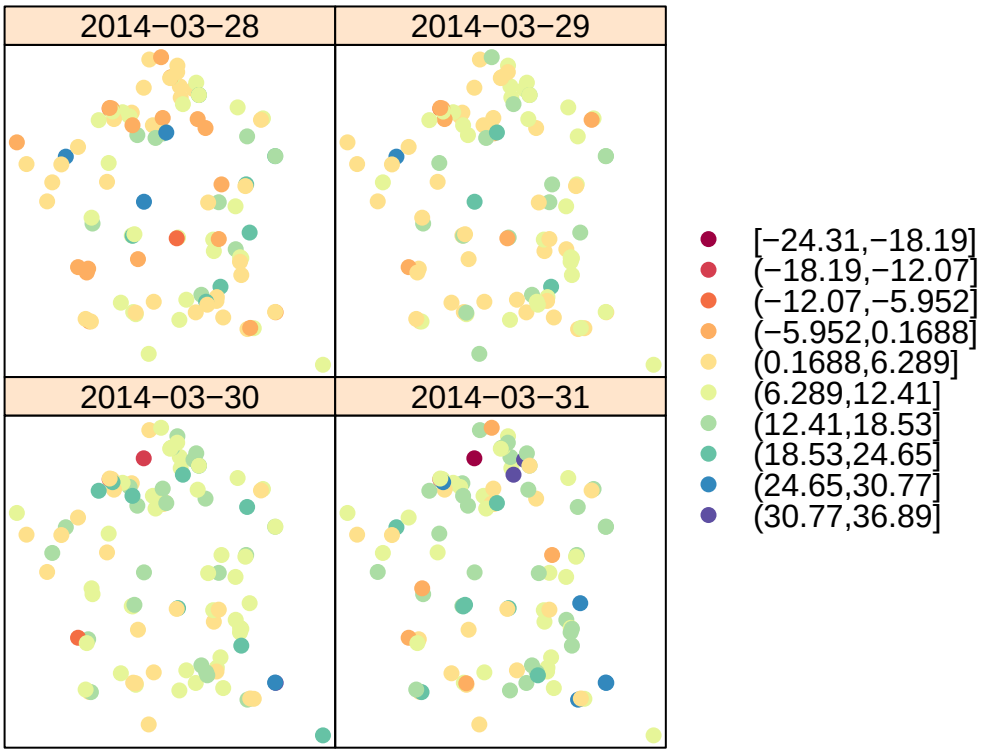
```
## [1] 12.62653 12.13389 12.14231 12.00067 12.51909 12.45865 12.41649 12.28487
```

```
### error made by CHIMERE
sqrt(mean((val - CHMval)^2))
```

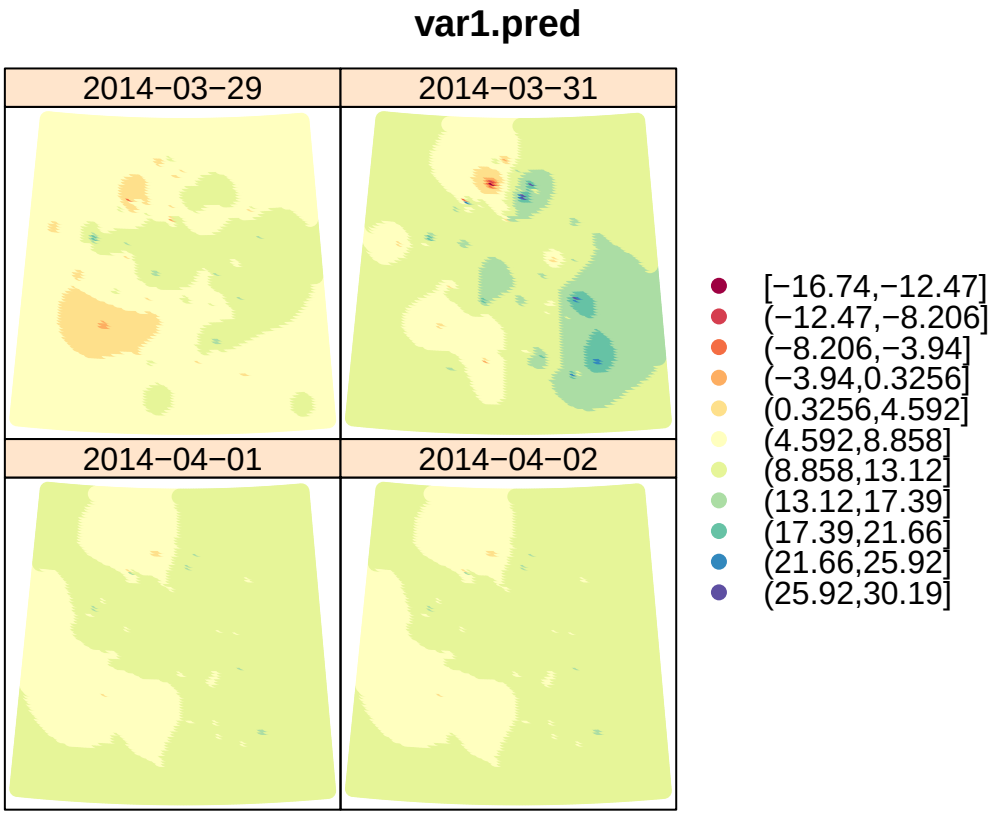
```
## [1] 19.3351
```

5.3) A) Kriging with *gstat*

```
### Kriging on selected data, no NA allowed
load(paste0(DATA, "data.Rdata"))
load(paste0(DATA, "stationsforfitting.Rdata"))
load(paste0(DATA, "timesforfits.Rdata"))
load(paste0(DATA, "STFDF_jour.Rdata"))
load(paste0(DATA, "fits_gstat.Rdata"))

### We use times from STDF_jour to ensure the correct time format
temps <- STFDF_jour@time[times]
data2 <- as.data.frame(as.vector(t(data)))
names(data2) <- "PM10res"
REScor <- STFDF(sp = coord.spt, time = temps, data = data2)
stplot(REScor[, "2014-03-28::2014-03-31", "PM10res"], col.regions = brewer.pal(10,
    "Spectral"), cuts = 10)
```

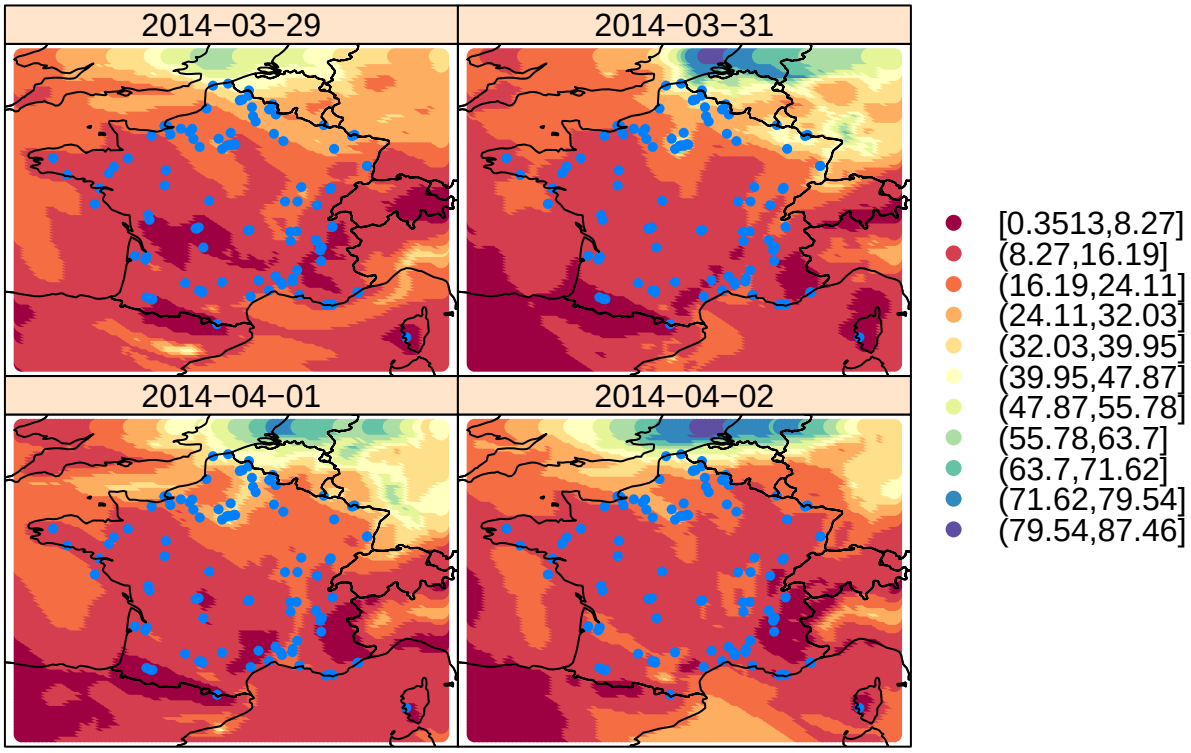



```
# Prediction dates
tt <- c(88, 90:92)
nummod <- 4  # Product sum manual
dates <- STFDF_jour@time[tt]
predGrid <- STF(coord.spchm, dates)
krig <- krigeST(PM10res ~ 1, data = REScor[, seq(as.Date("2014-03-29"), as.Date("2014-03-31"),
    by = "day")], newdata = predGrid, modelList = models[[nummod]])

stplot(krig, col.regions = brewer.pal(11, "Spectral"), cuts = 11)
```

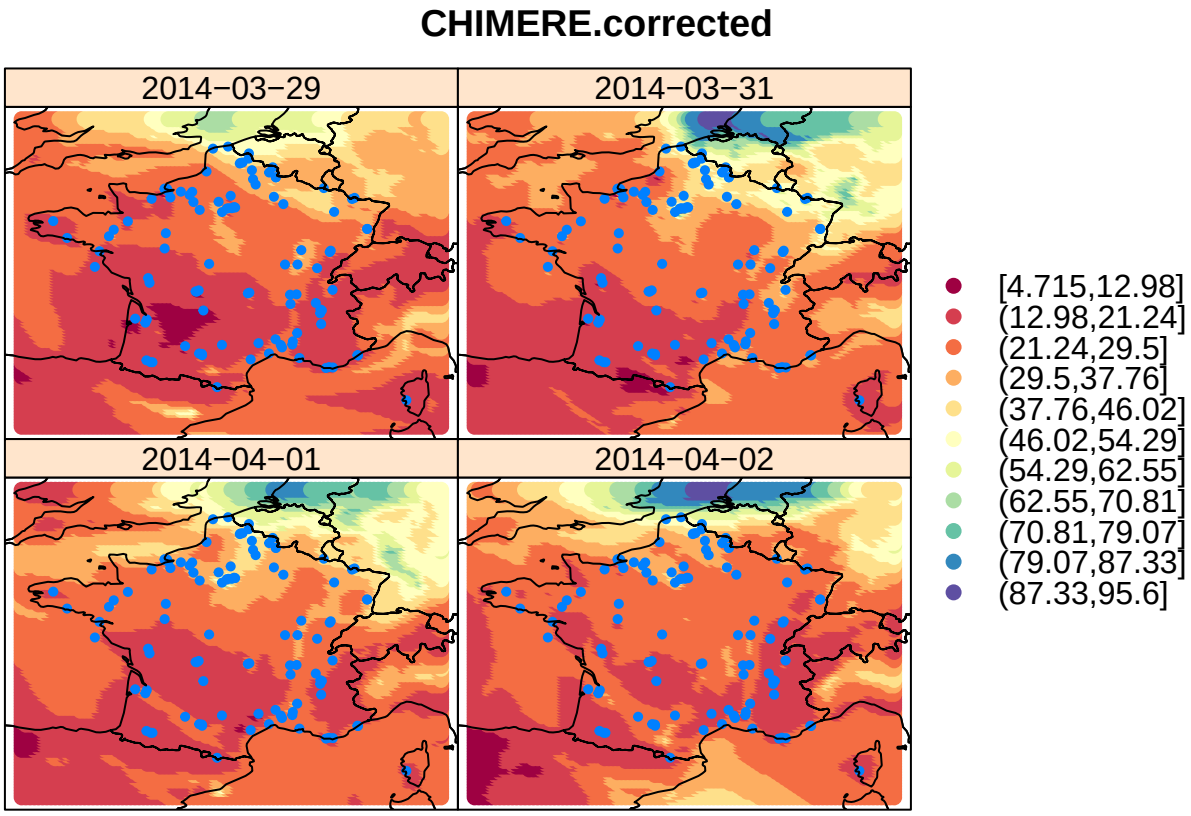



```
temps <- STFDF_jour@time[tt]
mydata2 <- as.data.frame(cbind(as.double(CHM_mat[, tt]), as.double(CHM_mat[,
    tt] + krig@data[[1]])))
names(mydata2) <- c("CHIMERE", "CHIMERE corrected")
CHMcorgs <- STFDF(sp = pts, time = temps, data = mydata2)

stplot(CHMcorgs[, 1:4, 1], col.regions = brewer.pal(11, "Spectral"), at = lvls,
    cuts = 11, sp.layout = list(coord.sp, world, cex = 0.5, pch = 19, first = FALSE))
```

CHIMERE



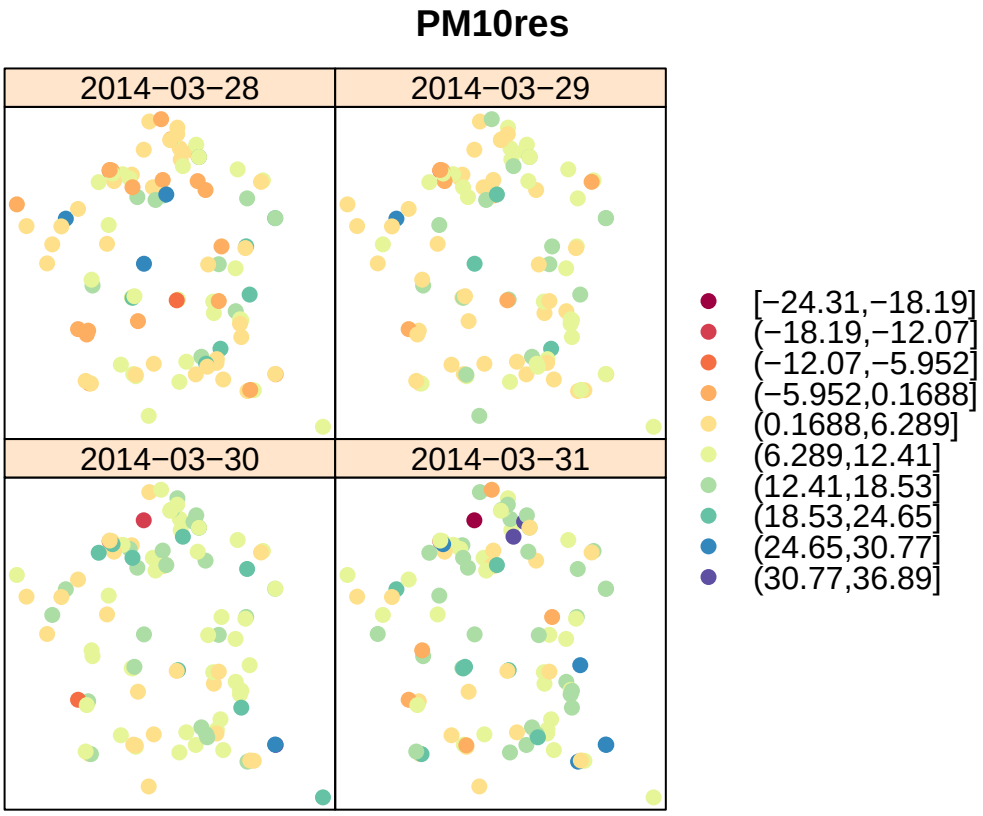

```
# namefig = 'chimgsstplot.pdf' dev.copy2pdf(file=namefig)
# system(paste0('pdfcrop ',namefig,' ',namefig)) # linux with pdfcrop only

stplot(CHMcorgs[, 1:4, 2], col.regions = brewer.pal(11, "Spectral"), at = lvls,
    cuts = 11, sp.layout = list(coord.sp, world, cex = 0.5, pch = 19, first = FALSE))
```




```
# namefig = 'chimcorrgsstplotpsman.pdf' dev.copy2pdf(file=namefig)
# system(paste0('pdfcrop ',namefig,' ',namefig)) # linux with pdfcrop only
```

5.3) B) Cross Validation with *gstat*

```
temps <- STFDF_jour@time[times]
data2 <- as.data.frame(as.vector(t(data)))
names(data2) <- "PM10res"
REScor <- STFDF(sp = coord.spt, time = temps, data = data2)
stplot(REScor[, "2014-03-28::2014-03-31", "PM10res"], col.regions = brewer.pal(10,
    "Spectral"), cuts = 10)
```




```
tt <- 89:90
dates <- STFDF_jour@time[tt]
times <- c(69, 70)
err <- matrix(NA, ns, 6)
for (nummod in 1:6) {
    corrmodel <- models[[nummod]]
    for (ista in 1:ns) {
        krig = krigeST(PM10res ~ 1, data = REScor[-ista, times], newdata = STF(coord.spt[ista],
            dates), modelList = corrmodel)
        err[ista, nummod] = data[nT, ista] - krig@data[2, ]
    }
}
RMSE <- sqrt(apply(err^2, 2, mean))
RMSE
```

```
## [1]  9.175435  9.429810  9.076592  9.535824  9.389305 10.135178
```

5.3) C) External validation with *gstat*

```
tt <- 91:92
dates <- STFDF_jour@time[tt]
times <- c(69, 70)
RMSE_extgs <- NULL
val <- (OBS_sel2[, 1, "PM10"])$PM10
indexnona <- which(!is.na(val))
CHMval <- (OBS_sel2[, 1, "CHM"])$CHM[indexnona]
val <- val[indexnona]
```

```
for (nummod in 1:6) {
    corrmodel <- models[[nummod]]
    krig <- krigeST(PM10res ~ 1, data = REScor[, times], newdata <- STF(OBS_sel2@sp,
        dates), modelList = corrmodel)
    RMSE_extgs <- c(RMSE_extgs, sqrt(mean((val - (krig[, 1, 1])$var1.pred[indexnona] -
        CHMval)^2)))
}
RMSE_extgs
```

```
## [1] 12.50706 13.87683 12.44715 13.42448 14.12144 13.34911
```

```
# error made by CHIMERE
sqrt(mean((val - CHMval)^2))
```

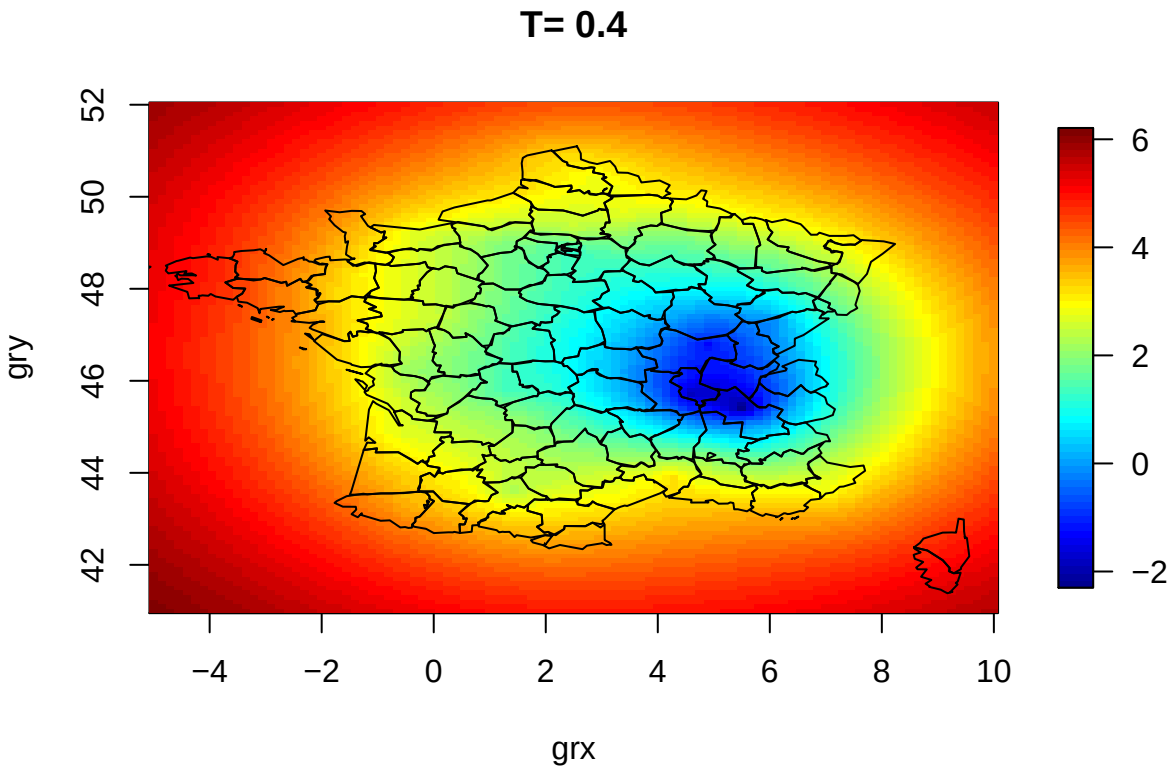
```
## [1] 19.3351
```

5.4) Miscellaneous: kriging with RandomFields

```
library(RandomFields)
load(paste0(DATA, "timesforfits.Rdata"))

times <- seq(0.1, 0.3, 0.1)  # last three observation dates
nTp <- length(times)
# Put the data as (x_i,y_i,T_i,z_i)
donnees <- cbind(rep(coord.sel[, 1], nTp), rep(coord.sel[, 2], nTp), as.vector(sapply(times,
    rep, ns)), as.vector(t(data[(nT - nTp + 1):nT, ])))
colnames(donnees) <- c("x", "y", "T", "z")

# Define the Gneiting covariance model Note : no 'sep' parameter, value 1
model <- RMnsst(phi = RMstable(scale = param$scale_s, alpha = param$power_s),
    psi = RMstable(scale = param$scale_t, alpha = param$power_t), delta = 2,
    var = param$sill) + RMnugget(param$nugget) + RMtrend(mean = mean.est)

# Define the target
loc <- coord.spchm@coords/1000
loc_to_pred <- loc

tt <- 0.4  # consecutive times only

zint <- RFinterpolate(model, x = loc, T = tt, data = donnees)
zint
```

```
## Object of class 'RFspatialPointsDataFrame'
## 'data.frame':    11211 obs. of  1 variable:
##  $ T: num  6.14 6.11 6.09 6.06 6.03 ...
```

```
zlim <- range(zint$T)
Z <- matrix(zint$T, nrow = length(grx))
image.plot(grx, gry, Z, main = paste("T=", tt), zlim = zlim)
map("france", add = TRUE)
```




```
# Strange results ...
image.plot(grx, gry, matrix(zint$T + CHM_mat[, 91], nrow = length(grx)), main = paste("T=",
    tt))
map("france", add = TRUE)
```