Happy R - Investigations Spatiales et Cartographiques

Poursuite de la présentation de Jessica du 17/11/17

Eric & Isabelle

23 mars 2018

Introduction

Les données spatiales et R

Les données spatiales se présentes sous la forme de:

- `vecteurs` (points, vecteurs, lignes, etc)
- `raster` (pixels).

R permet de lire, d'écrire et de manipuler des données géoréférencées, puis d'en faire l'analyse statistique. Nous avons repérés deux packages de base :

- `sp` - classe de base pour gérér des données spatiales, en particulier vectorisées, mais aussi grilles.
- `raster` - classes et outils pour manipuler des données sous forme raster.

+ des outils, divers et variés, en particulier:

- `rgdal`: Interface R pour la library C/C++ de stat spatiales `gdal` (Geospatial Data Abstraction Library) pour lire et écrire des données spatialisées
- `rgeos`: Interface R de la library `geos` (Geometry Engine Open Source) avec opérations de type ensemblistes

Bien d'autres packages spatiaux sur le CRAN!, voir - Spatial task view

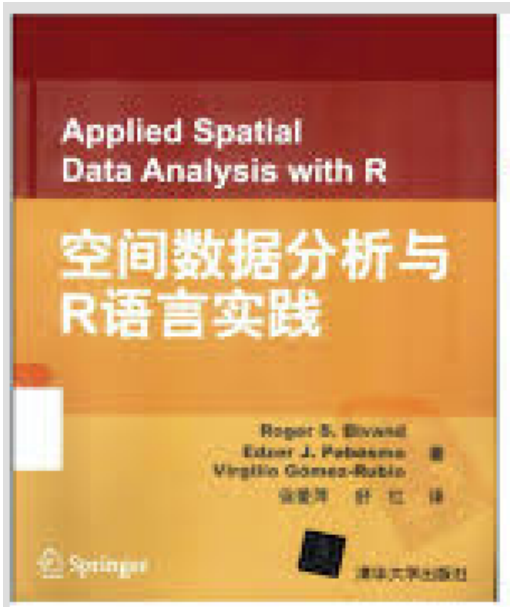
Que fait-on aujourd'hui ?

On étudie les packages **sp** puis **gstat**. . . puis **spacetime**

Il est possible de bricoler avec l'emploi des commandes uniquement de base de R, telles que data.frame, image, etc. . . .**MAIS** pourquoi utiliser un package?

Accepter de perdre un peu de liberté indisciplinée ?

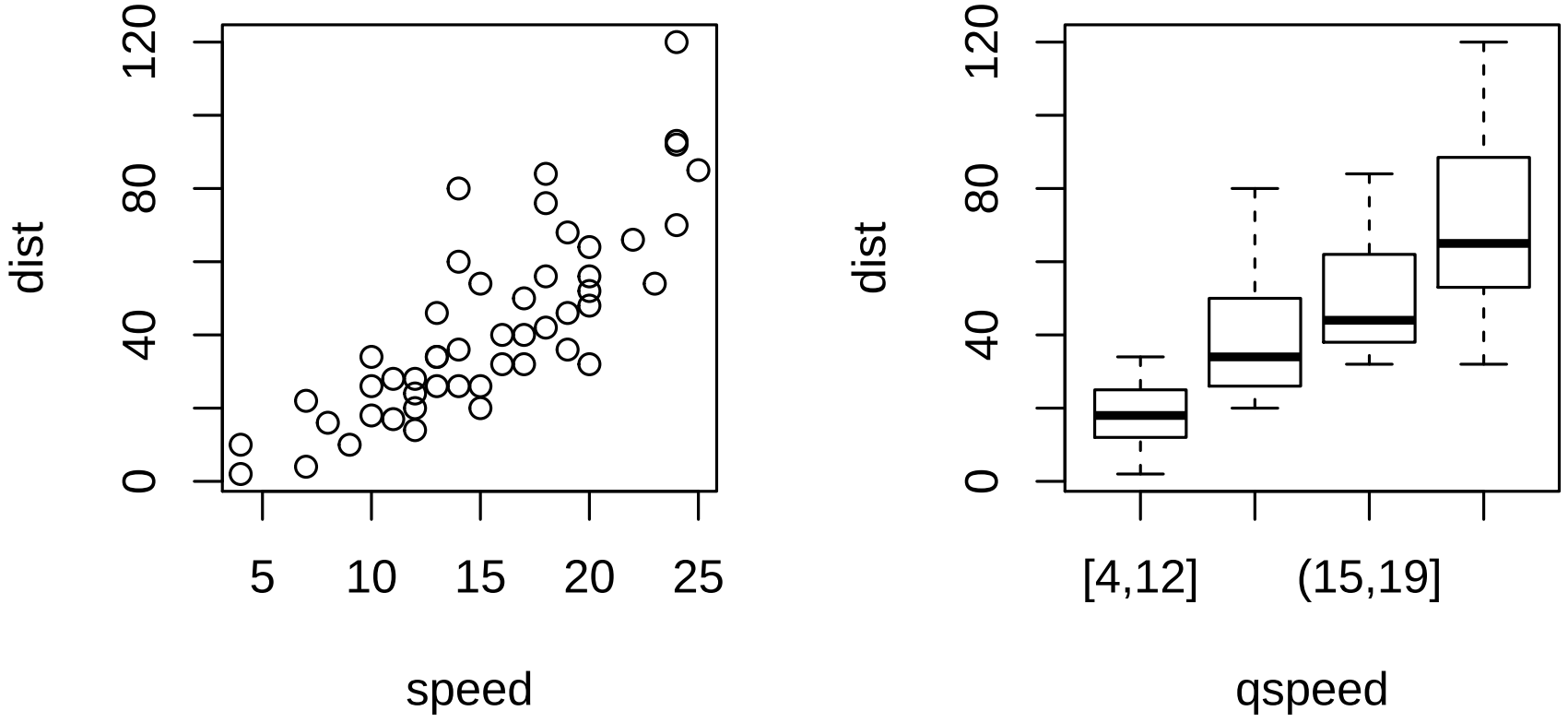
- pour travailler et partager avec d'autres collègues,
- pour récupérer le travail d'autres collègues.
- bénéficier des fondations et des extensions de **sp**



Présentation inspirée des exemples de ce bouquin et du travail réalisé avec le réseau RESSTE

R n'est pas simplement la calculette du statisticien

```
cars$qspeed <- cut(cars$speed, breaks=quantile(cars$speed),
                   include.lowest=TRUE)
par(mfrow=c(1,2))
plot(dist ~ speed, data=cars)
plot(dist ~ qspeed, data=cars)
```




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

Le package sp définit des classes d'objet explicitement spatialisés

```
library(sp)
getClass("Spatial")
## Class "Spatial" [package "sp"]
##
## Slots:
##
## Name:         bbox proj4string
## Class:      matrix         CRS
##
## Known Subclasses:
## Class "SpatialPoints", directly
## Class "SpatialMultiPoints", directly
## Class "SpatialGrid", directly
## Class "SpatialLines", directly
## Class "SpatialPolygons", directly
## Class "SpatialPointsDataFrame", by class "SpatialPoints", distance 2
## Class "SpatialPixels", by class "SpatialPoints", distance 2
## Class "SpatialMultiPointsDataFrame", by class "SpatialMultiPoints", distance
## Class "SpatialGridDataFrame", by class "SpatialGrid", distance 2
## Class "SpatialLinesDataFrame", by class "SpatialLines", distance 2
## Class "SpatialPixelsDataFrame", by class "SpatialPoints", distance 3
## Class "SpatialPolygonsDataFrame", by class "SpatialPolygons", distance 2
```

Le package sp définit des classes d'objet explicitement spatialisés

- Points avec ou sans data : *SpatialPoints* et *SpatialPointsDataFrame*
- Pixels avec ou sans data : *SpatialPixels* et *SpatialPixelsDataFrame*
- Grilles avec ou sans data : *SpatialGrid* et *SpatialGridDataFrame*
- Lignes avec ou sans data : *SpatialGrid* et *SpatialGridDataFrame*
- Polygones : *SpatialPolygons* et *SpatialPolygonsDataFrame*

Spatial et SpatialPoints

La classe de base: Spatial

Elle ne possède que 2 *slots*

- un *bounding box* en longitude/latitude
- un objet de la classe *CRS* (coordinate reference system), par défaut mise à CRS(as.character(NA)) . Voir plus tard ce qu'est un CRS.

```
library(sp)
m <- matrix(c(-2,15,10,40), ncol=2, dimnames=list(NULL, c("min", "max")))
crs <- CRS(projargs=as.character(NA))
crs
## CRS arguments: NA
S <- Spatial(bbox=m, proj4string=crs)
S
## An object of class "Spatial"
## Slot "bbox":
##      min max
## [1,]  -2  10
## [2,]  15  40
##
## Slot "proj4string":
## CRS arguments: NA
```

Pour la suite de l'exposé, on s'appuie sur le jeux de données MEUSE

```
# check for example data
data(meuse) # mesures de métaux lourds dans la meuse
str(meuse) # 155 observations et 14 variables
## 'data.frame':    155 obs. of  14 variables:
##  $ x       : num  181072 181025 181165 181298 181307 ...
##  $ y       : num  333611 333558 333537 333484 333330 ...
##  $ cadmium: num  11.7 8.6 6.5 2.6 2.8 3 3.2 2.8 2.4 1.6 ...
##  $ copper : num  85 81 68 81 48 61 31 29 37 24 ...
##  $ lead    : num  299 277 199 116 117 137 132 150 133 80 ...
##  $ zinc    : num  1022 1141 640 257 269 ...
##  $ elev    : num  7.91 6.98 7.8 7.66 7.48 ...
##  $ dist    : num  0.00136 0.01222 0.10303 0.19009 0.27709 ...
##  $ om      : num  13.6 14 13 8 8.7 7.8 9.2 9.5 10.6 6.3 ...
##  $ ffreq   : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
##  $ soil    : Factor w/ 3 levels "1","2","3": 1 1 1 2 2 2 2 1 1 2 ...
##  $ lime    : Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 1 1 1 ...
##  $ landuse: Factor w/ 15 levels "Aa","Ab","Ag",..: 4 4 4 11 4 11 4 2 2 15 ...
##  $ dist.m : num  50 30 150 270 380 470 240 120 240 420 ...
```

Spatial points

On utilise les 2 colonnes du data.frame meuse renseignant les coordonnées géographiques pour en faire un objet *SpatialPoints*

```
# check for example data
coords <- SpatialPoints(meuse[, c("x", "y")])
summary(coords)
## Object of class SpatialPoints
## Coordinates:
##      min    max
## x 178605 181390
## y 329714 333611
## Is projected: NA
## proj4string : [NA]
## Number of points: 155
```

Exercice à vous de jouer

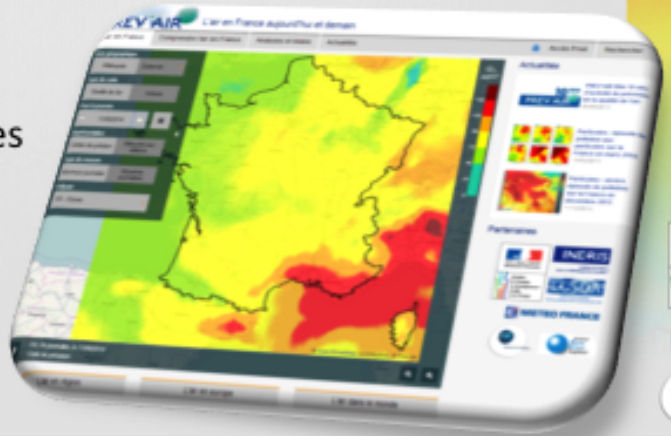
Merci à Maxime Beauchamp et Laure Malherbe de l'Ineris

Objectifs du système PREV 'AIR

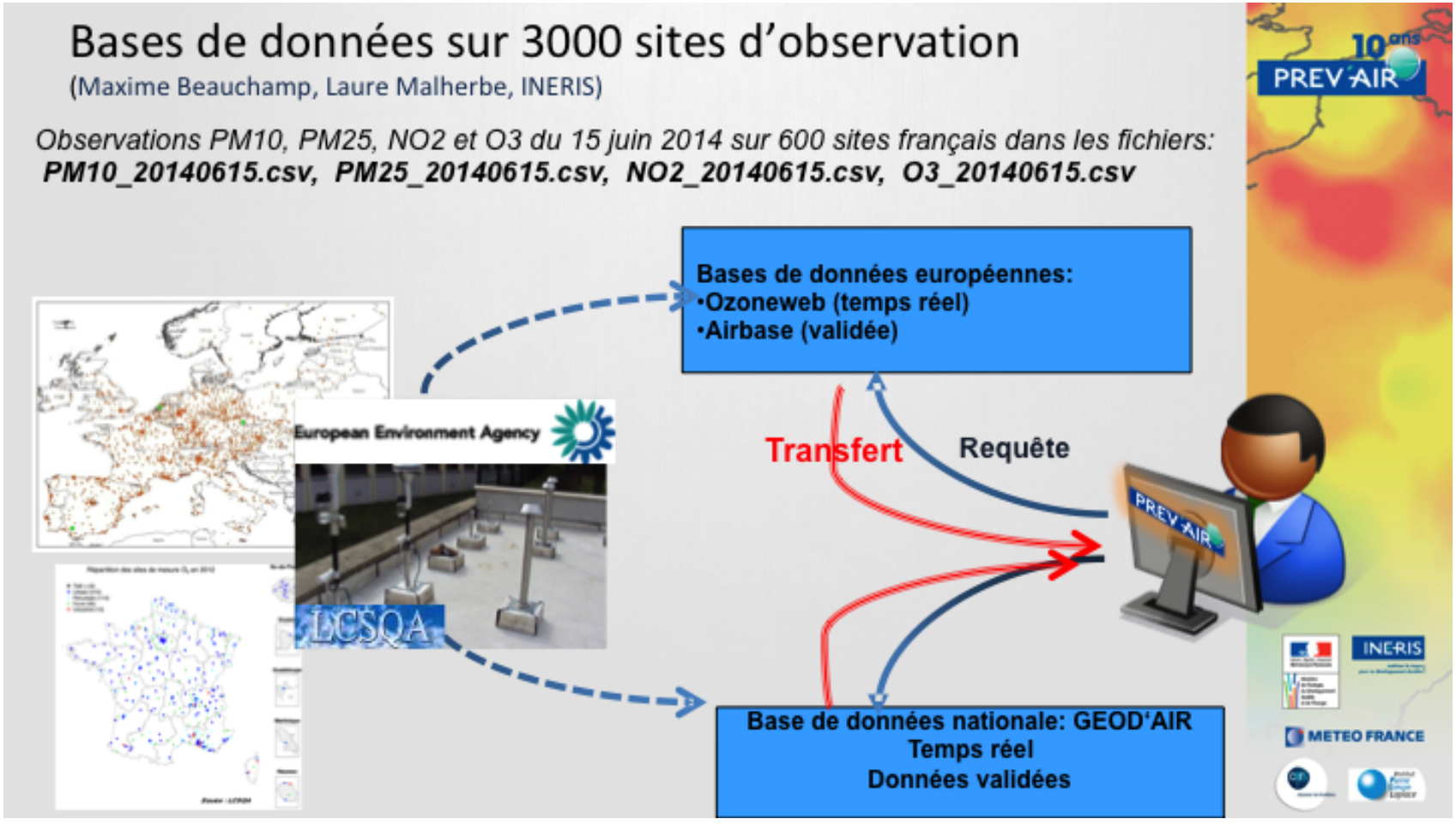
(Maxime Beauchamp, Laure Malherbe, INERIS)

- **Suivi et Prévision :**
  - Polluants réglementés: O_3, NO_2 , PM_{10}, $PM_{2.5}$
  - Échelles globale / européenne / nationale
  - 3 jours d 'échéance

  L'information du public et les mesures d'urgence ne sont plus uniquement déclenchées en fonction des observations et mais également des prévisions.
- **Surveillance** dans les régions peu ou pas couvertes par des stations de mesure fixes
- **Aide à la gestion de la qualité de l'air** en cas d'épisode de pollution.
  Recherche des causes des épisodes



Exercice à vous de jouer



Exercice à vous de jouer

Présentation du système PREV 'AIR -> CHIMERE

(Maxime Beauchamp, Laure Malherbe, INERIS)

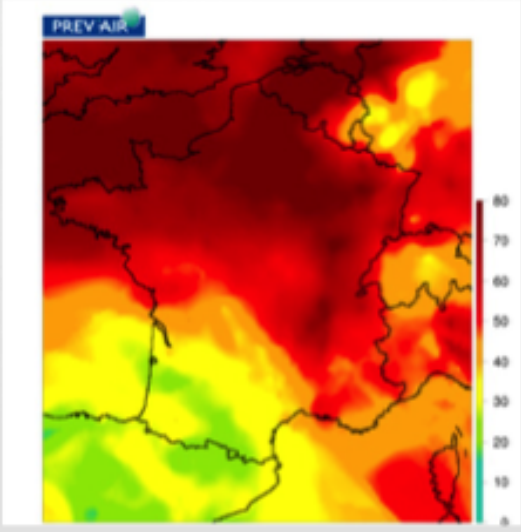
Modélisation dynamique des espèces chimiques (polluants) dans la troposphère

+

Post-traitement statistique par rapport aux observations des jours précédents

=

Carte CHIMERE



PM_{10} : carte analysée du 12 mars 2014

Le modèle CHIMERE a tourné sur le domaine France, dit AFM, à 10km de résolution le 15 juin 2014: fichier ***chim.20140615.csv***

Exercice 1

Lire les stations de suivi de la pollution atmosphérique en PM10, PM25 , NO2 et O3 dans le fichier **AirBase_v7_stations.csv**.

Extraire leur coordonnées, jeter les doublons et sélectionner les stations de France métropolitaine. En faire un objet *SpatialPoints* et les représenter. On pourra extraire un fond de carte de la France par les commandes R suivantes:

```
library(cshapes)
## Loading required package: maptools
## Checking rgeos availability: TRUE
## Loading required package: plyr
cs<-cshp()
row.names(cs)=paste(as.character(cs$CNTRY_NAME),1:244)
numfrance=grep('France',row.names(cs)) #=106
cowcodefrance=cs$COWCODE[106]
france <- cs[cs$COWCODE==cowcodefrance,]
```

Spatial points data frame

Retour à Meuse. On ajoute aux coordonnées géographiques les données pour en faire un objet *SpatialPointsDataFrame*. Ca se comporte comme un dataframe!

```
meuse_sp <- SpatialPointsDataFrame(coords, meuse)
names(meuse_sp)
##  [1] "x"       "y"       "cadmium" "copper"  "lead"    "zinc"
##  [7] "elev"    "dist"    "om"      "ffreq"   "soil"    "lime"
## [13] "landuse" "dist.m"
is(meuse_sp)
## [1] "SpatialPointsDataFrame" "SpatialPoints"
## [3] "Spatial"
is(meuse)
## [1] "data.frame" "list"       "oldClass"   "vector"
#str(meuse_sp)
```

On aurait pu faire plus direct:

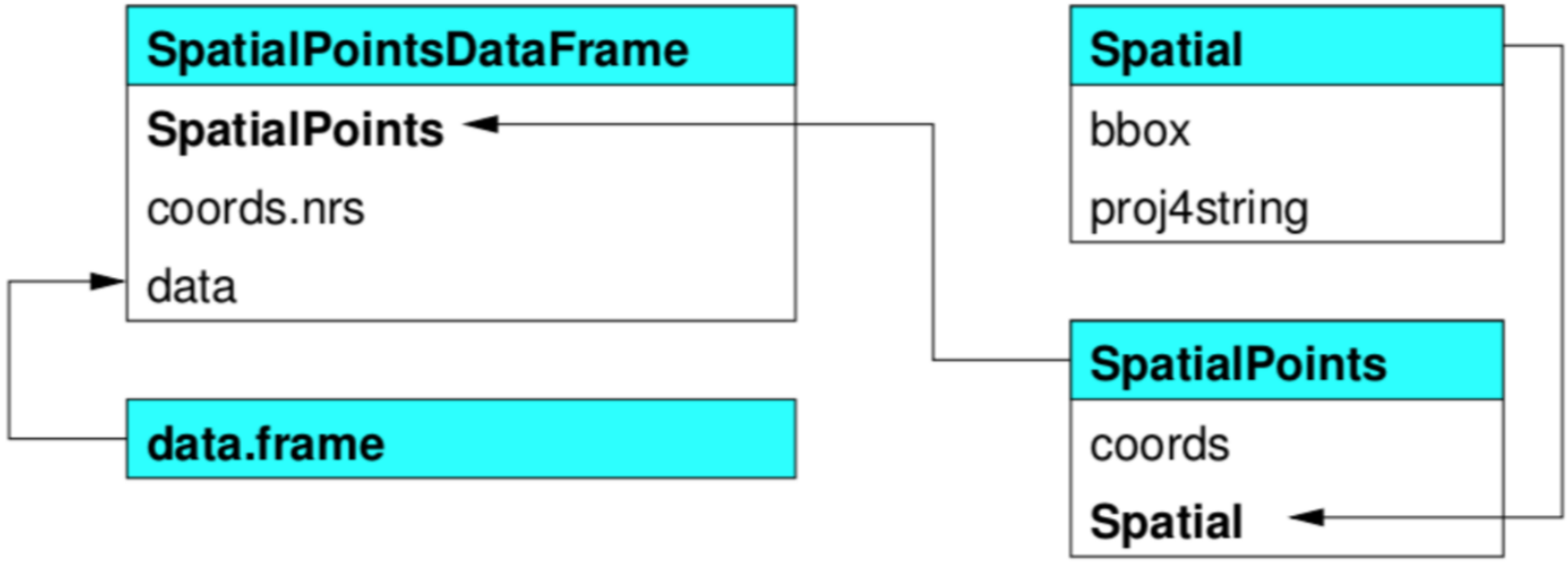
```
meuse_sp2 <- meuse
coordinates(meuse_sp2) <- ~x+y
# str(meuse_sp2)
is(meuse_sp2)
## [1] "SpatialPointsDataFrame" "SpatialPoints"
## [3] "Spatial"
```

Exercice 2

Lire les données de pollution atmosphérique du 15 juin 2014 en PM10, PM25 , NO2 et O3 pour les stations actives de France métropolitaine dans les fichiers **PM10_20140615.csv**, **PM25_20140615.csv**, **NO2_20140615.csv** et **03_20140615.csv** . Les données manquantes sont codées -999.

On pourra utiliser les packages *tidyR* et *data.table* pour avoir recours à la méthode *merge*. Créer un objet *SpatialPointsDataFrame* contenant les enregistrements géolocalisés de PM10, PM25 , NO2 et O3 pour les stations actives au 15 juin 2014 dont on conservera le type et l'aire.

Recap: ASDAR page 35



SpatialGrids & SpatialPixels

Spatial grids & pixels

2 representations pour les data sur une grille régulière rectangualire, orientée N-S et E-W: *SpatialPixels* et *SpatialGrid*

- SpatialPixels comme SpatialPoints, mais avec coordonnées equi-espacées
- SpatialPixelsDataFrame = SpatialPixels+data
- SpatialGridDataFrame = grille entiere et NA où manquantes

```
 data(meuse.grid)
 coords <- SpatialPixels(SpatialPoints(meuse.grid[, c("x","y")]))
meuse_spx <- SpatialPixelsDataFrame(coords, meuse.grid)
names(meuse_spx)
## [1] "x"      "y"      "part.a" "part.b" "dist"   "soil"   "ffreq"
slot(meuse_spx, "grid")
##                         x      y
## cellcentre.offset 178460 329620
## cellsize              40     40
## cells.dim             78    104
object.size(meuse_spx)
## 565952 bytes
dim(slot(meuse_spx, "data"))
## [1] 3103    7
```

Spatial grids

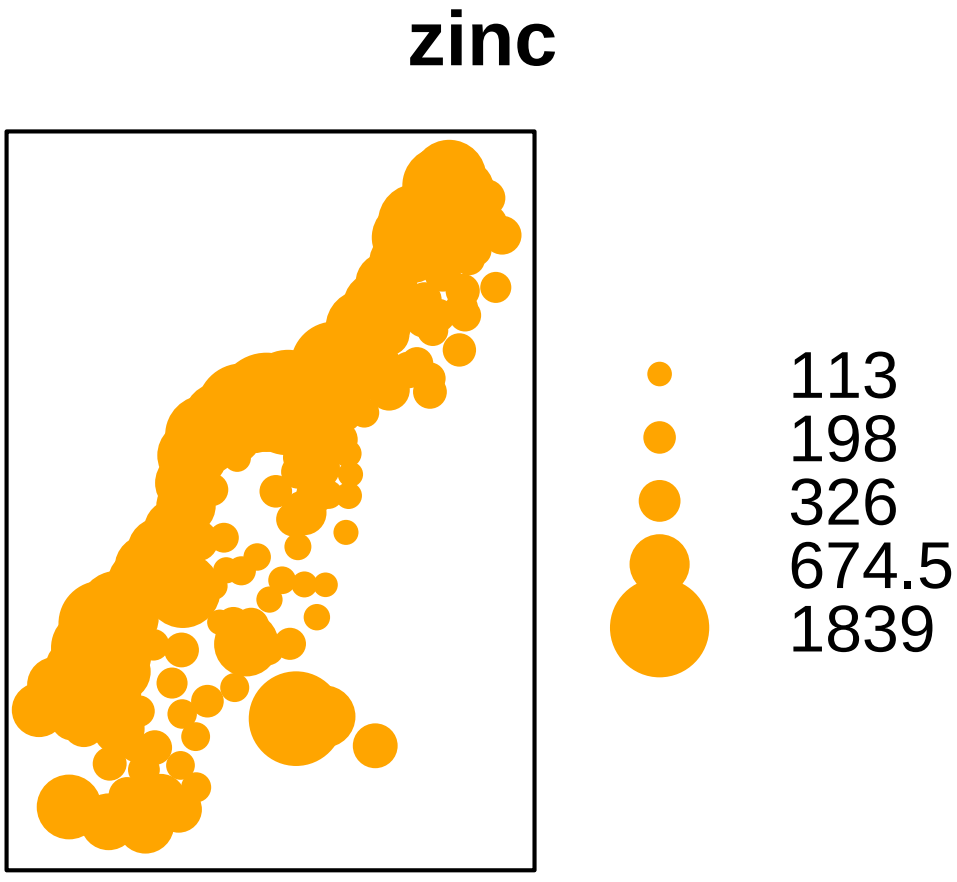
On peut convertir un *SpatialPixels* en un *SpatialGrid*, ce qui économise de la place de stockage:

```
meuse_sg <- meuse_spx
fullgrid(meuse_sg) <- TRUE
slot(meuse_sg, "grid")
##                          x      y
## cellcentre.offset 178460 329620
## cellsize              40     40
## cells.dim             78    104
class(slot(meuse_sg, "grid"))
## [1] "GridTopology"
## attr(,"package")
## [1] "sp"
object.size(meuse_sg)
## 395808 bytes
dim(slot(meuse_sg, "data"))
## [1] 8112    7
par(mfrow=c(1,1))
```

Spatial grids

Interpolons par “inverse distance weighting” un objet *SpatialPointsDataFrame*

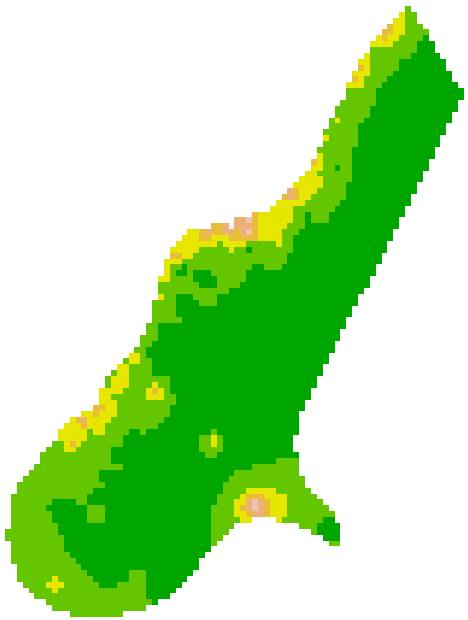
```
library(gstat)
meuse_sg$zincIDW <- idw(zinc~1,meuse_sp, meuse_sg)$var1.pred
## [inverse distance weighted interpolation]
#interpolation par un poids inverse à la distance
    bubble(meuse_sp, "zinc",col = c("white", "orange"))
```



Spatial grids

L'interpolation par "inverse distance weighting" d'un objet *SpatialPointsDataFrame* crée un objet *SpatialGridDataFrame*:

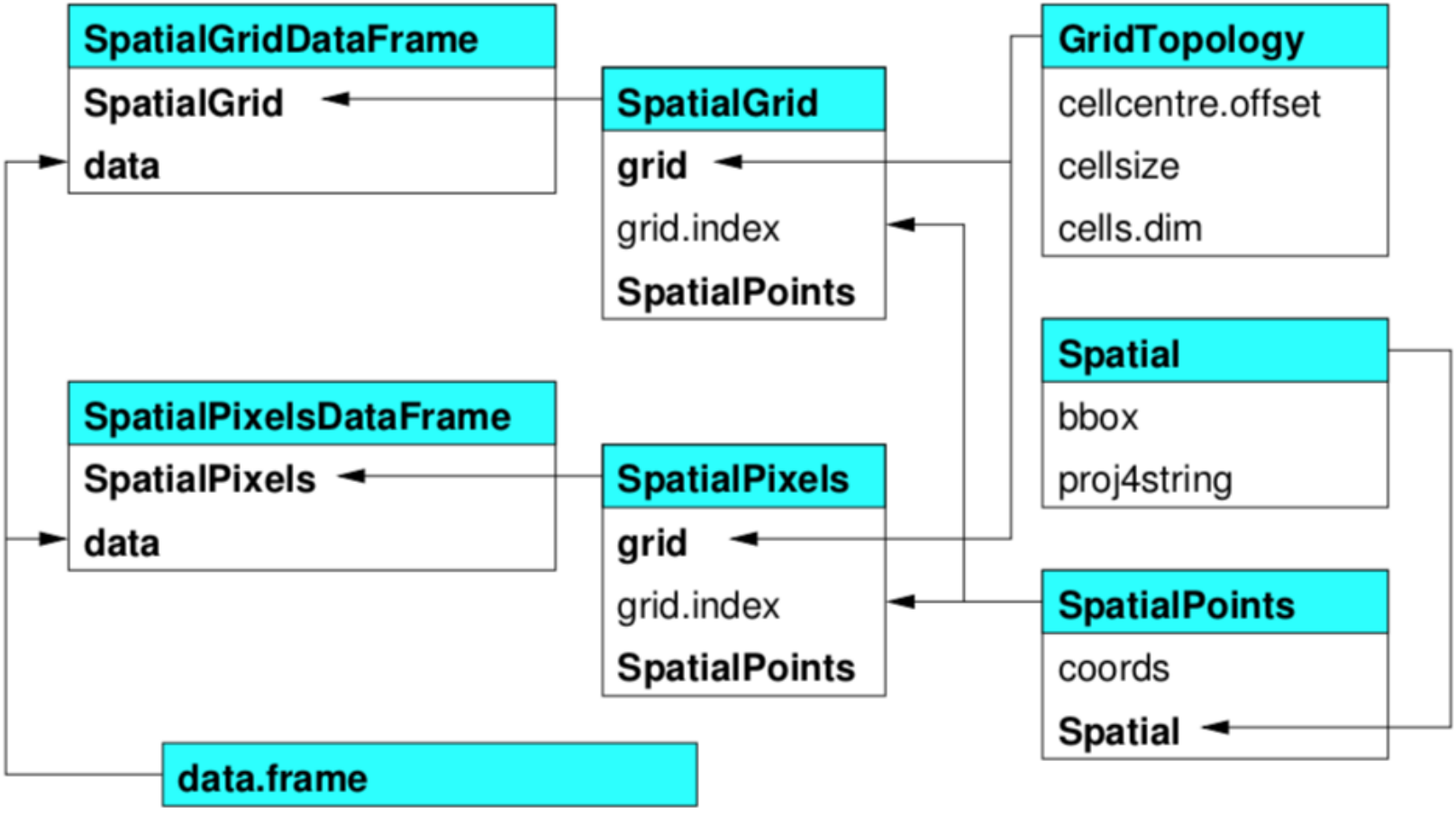
```
    image(meuse_sg,"zincIDW", col= terrain.colors(7))
```



Exercice 3

- Lire le champ de pollution atmosphérique du 15 juin 2014 en PM10, PM25 , NO2 et O3 prévues par le code numérique CHIMERE sur une grille régulière recouvrant la France métropolitaine dans le fichier **chim.20140615.csv**. Vérifier que l'écartement entre les points est régulier et le corriger si nécessaire. Faire une représentation graphique du champ de prévisions de PM25 du 15 juin 2014 sur laquelle on pourra tracer le fond de carte des contours de la France métropolitaine. Créer un objet *SpatialPixelDataFrame* . Quelle est sa taille? Le transformer en un objet *SpatialGridDataFrame*. A-t-on économisé de l'espace de stockage?
- Interpoler par "inverse distance weighting" (en supposant qu'un écart d'un degré de latitude correspond à celui d'un degré de longiture) les observations de PM25 (un objet *SpatialPointsDataFrame*) sur la grille des previsions Chimère (un objet *SpatialGridDataFrame*). Représenter les erreurs de previsions.

Recap ASDAR page 52



SpatialLines & SpatialPolygones

Polygons

- Un objet *Line* = ensemble de coordonnées 2D sans ordre. Un *Polygon* = *Line* fermée. *Lines*= liste de *Line*.
- Un objet *SpatialLines* = ensemble de listes de coordonnées 2D avec ordre. Idem pour *SpatialPolygons*
- *SpatialLinesDataFrame* et *SpatialPolygonsDataFrame* =+ data.frame dont les lignes associées aux coordonnées.

Polygons et leur famille

Les données meuse du package **sp** contiennent les coordonnées des bords de la rivière. On les transforme en un objet *SpatialPolygons* :

```
data(meuse.riv)
str(meuse.riv)
##  num [1:176, 1:2] 182004 182137 182252 182314 182332 ...
meuse.riv_poly <- SpatialPolygons(list(Polygons(list(Polygon(meuse.riv)), ID = '
summary(meuse.riv_poly )
## Object of class SpatialPolygons
## Coordinates:
##        min      max
## x 178304.0 182331.5
## y 325698.5 337684.8
## Is projected: NA
## proj4string : [NA]
```

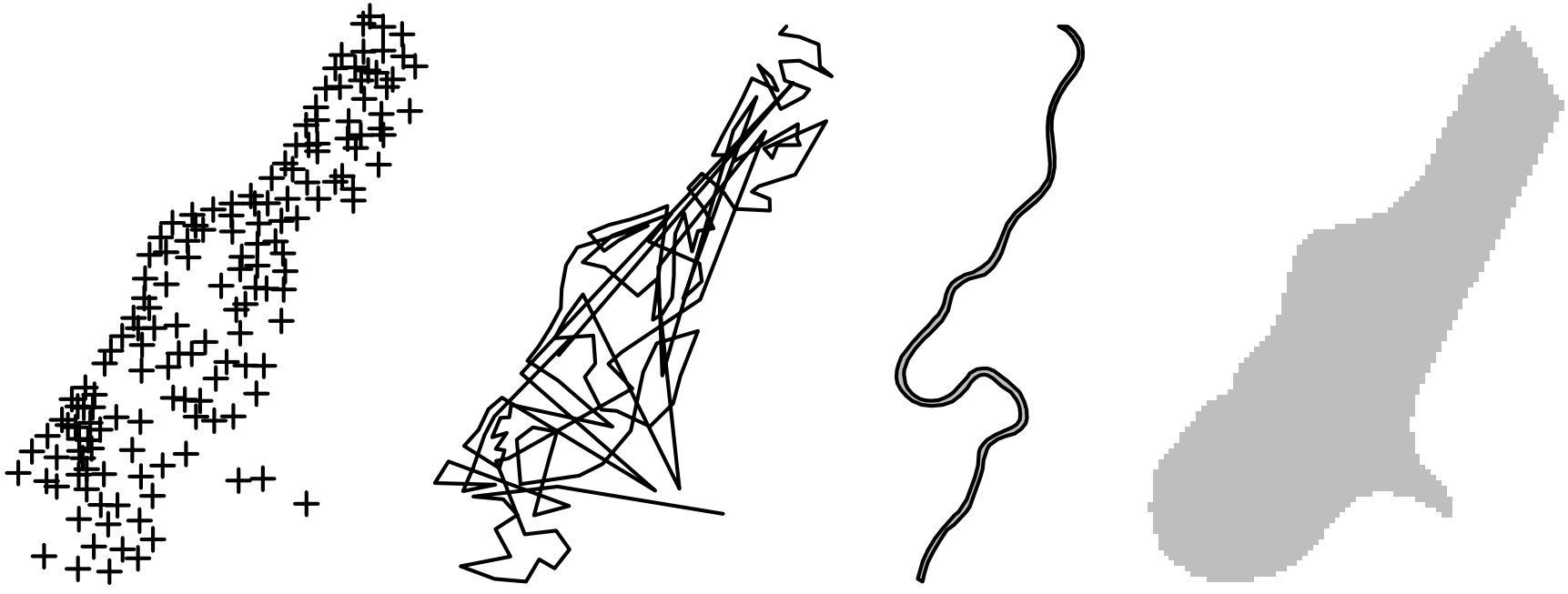
SpatialPolygons + SpatialPoints + SpatialPixels

Le tout ensemble:

```
layout(matrix(1:4, 1, 4, byrow = TRUE))
par(mar = c(0,0,1,0))
#plot(meuse_sp, cex = 0.6)
is(meuse_sp)
## [1] "SpatialPointsDataFrame" "SpatialPoints"
## [3] "Spatial"

cc = coordinates(meuse_sp)
meuse.sl = SpatialLines(list(Lines(list(Line(cc)), "mess")))
#plot(meuse.sl)
is(meuse.sl)
## [1] "SpatialLines" "Spatial"

#plot(meuse.riv_poly, col = "grey")
is(meuse.riv_poly)
## [1] "SpatialPolygons" "Spatial"

#image(meuse_spx, col = "grey")
is(meuse_spx)
## [1] "SpatialPixelsDataFrame" "SpatialPixels"
## [3] "SpatialPointsDataFrame" "SpatialPoints"
## [5] "Spatial"
```

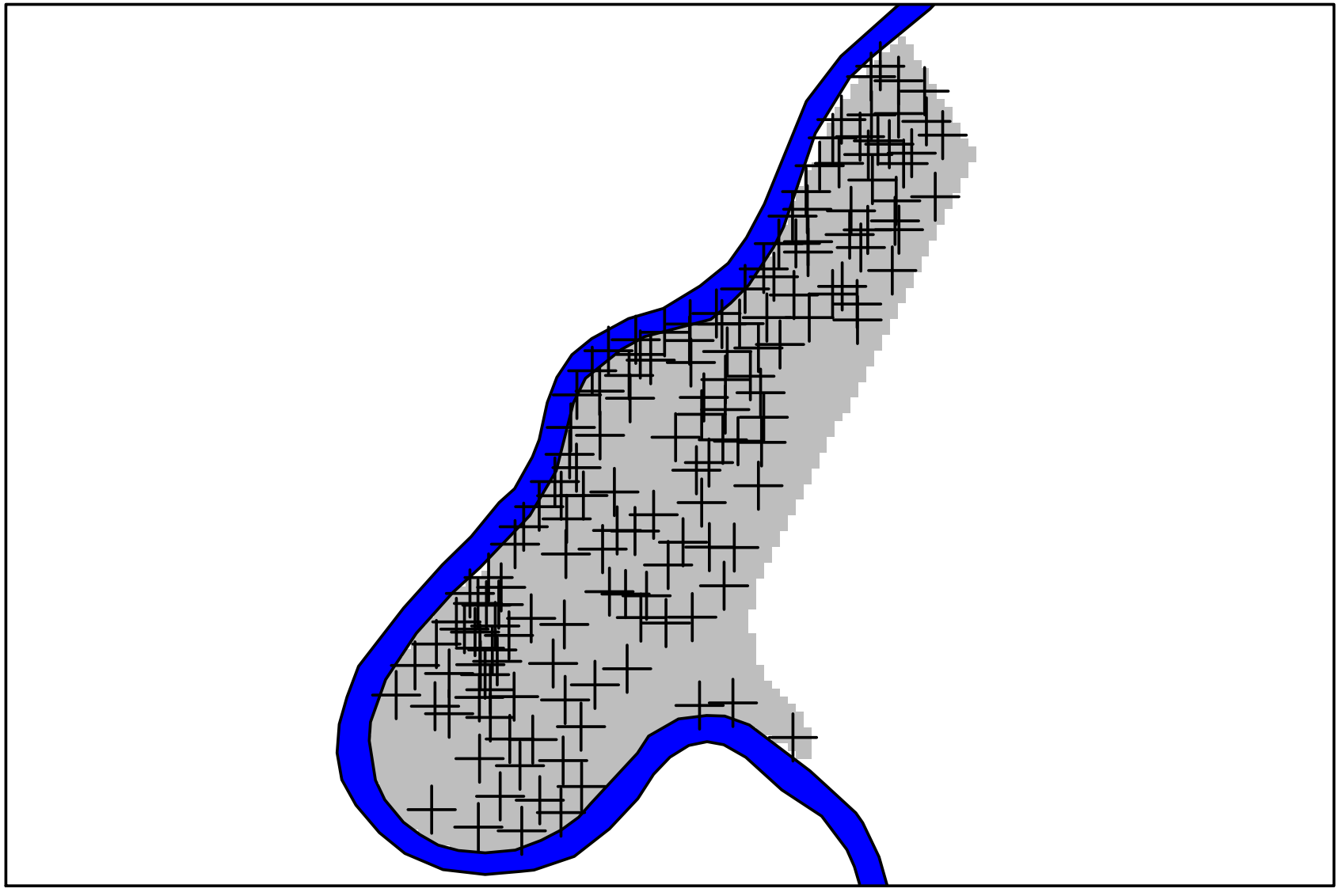
Le tout ensemble:

points **lines** **polygons** **grid**



SpatialPoints + SpatialPolygons + SpatialPixels

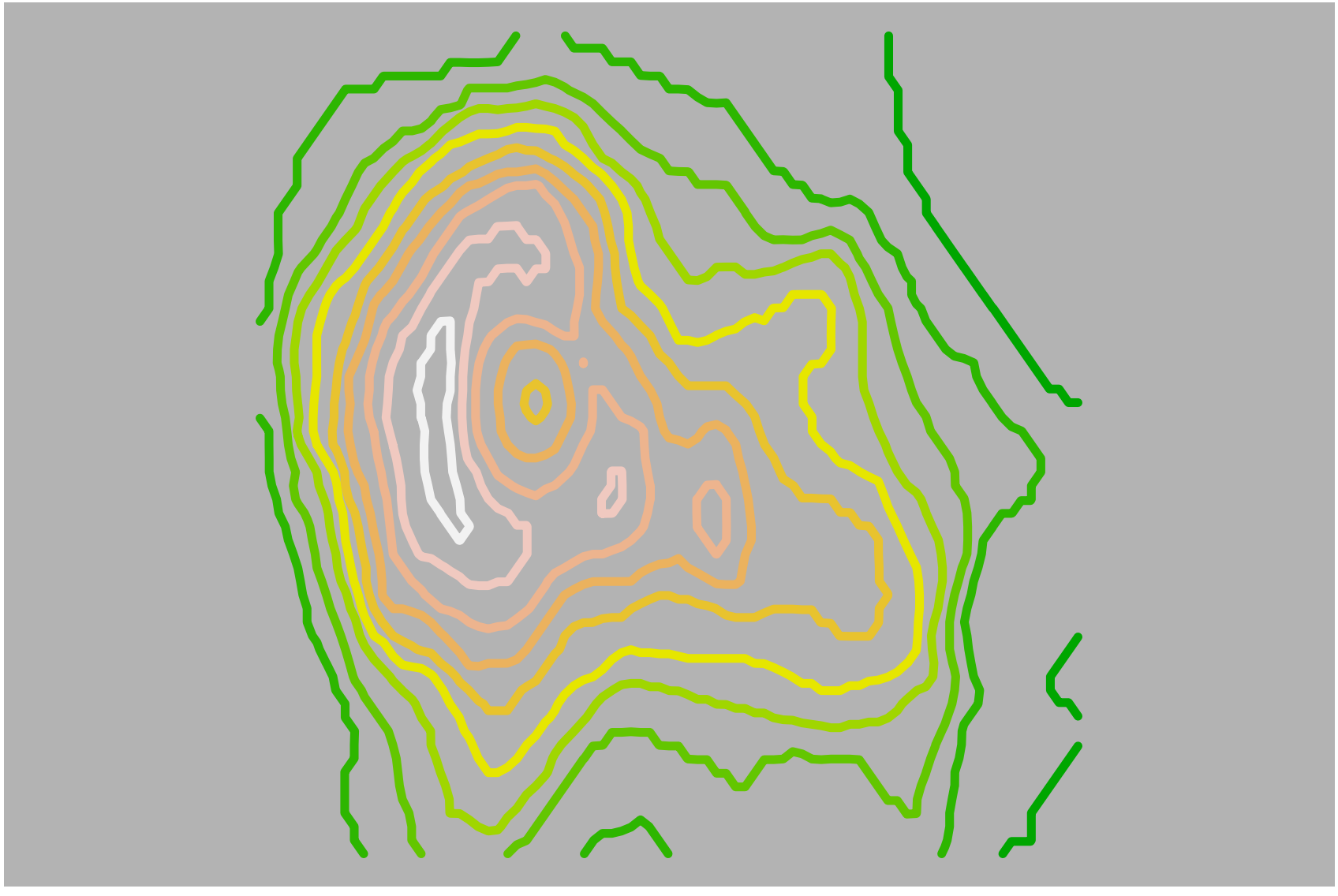
Le tout ensemble:



Spatial Lines et *ContourLines2SLDF*

```
data(volcano)
#?ContourLines2SLDF
volcano_sl <- ContourLines2SLDF(contourLines(volcano,nlevels=10))
sapply(slot(volcano_sl, "lines"), function(x) length(slot(x, "Lines")))
##  [1] 3 4 1 1 1 2 2 3 2 1
volcano_sl$level
##  [1] 100 110 120 130 140 150 160 170 180 190
## Levels: 100 110 120 130 140 150 160 170 180 190


col <- terrain.colors(nlevels(volcano_sl$level))
# plot(volcano_sl, bg = "grey70",
# col = col[as.numeric(volcano_sl$level)], lwd = 3)
```

Spatial Lines et *ContourLines2SLDF*



Exercice 4

- Quelle est la nature de l'objet *france* obtenu à partir du package *Cshapes* dans l'exercice 1 ?
- Créer des lignes de niveaux pour les previsions Chimère de PM25. Les transformer en un objet de la famille *SpatialPolygonsDataFrame*. Représenter ces lignes de niveaux sur celles obtenues pour l'interpolation ("IDW" ou "krigeage") obtenue à partir des observations de PM25.

Recap ASDAR page 40