

Cartographie avec R

Timothée Giraud & Hugues Pécout

2018-12-13

Contents

Introduction	5
1 Les données spatiales	7
1.1 Le package <code>sf</code>	7
1.2 Les projections	10
1.3 Opérations de géotraitement	10
1.4 Le package <code>raster</code>	13
2 Cartographie thématique	15
2.1 Le package <code>cartography</code>	15
2.2 Palettes de couleurs	18
2.3 Discrétilisations	20
2.4 Combinaisons	21
2.5 Labels	22
2.6 Les données OSM	23
2.7 Cartographie interactive	25
2.8 Géocodage d'adresses	25
2.9 Création de cartons	25
3 Cartographie thématique avancée	27
3.1 Les anamorphoses	27
3.2 Les grilles régulières	32
3.3 Les discontinuités	36
3.4 Les lissages	38
3.5 3D	41

Introduction

Toute carte est issue d'un processus complexe, de choix, de selections, d'opérations statistiques ou géomatiques. Certains auteurs énoncent que les cartes sont subjectives (Brunet) et d'autres auteurs disent carrément qu'elles mentent (Monmonnier). Toute carte résulte des choix de son auteur. Dans une démarche scientifique, ces choix doivent être traçables, partageables et soumis à la discussion scientifique et cela est difficilement faisable quand ces cartes sont réalisées dans un environnement "clic-bouton".

La réalisation des cartes dans le langage R permet de tracer toutes les opérations nécessaires à une réalisation cartographique de qualité. Réaliser des cartes dans ce langage unique permet, en diffusant le code source en même temps que les cartes, de jouer "cartes sur table". Cela permet de détailler les choix qui ont été faits et s'exposer à la controverse scientifique. Cela permet aussi de travailler à plusieurs sur une carte, en associant des compétences complémentaires (sémiologie graphique, statistique, géomatique, etc.) et de faciliter la mise à jour de documents déjà réalisés.

Ce document se compose de trois parties permettant d'appréhender la création de cartes thématiques avec R.

- Les données spatiales
- Cartographie thématique
- Cartographie thématique avancée

Ce document mobilise un certain nombre de packages dédiés à l'import, la manipulation, la transformation et l'affichage de données spatiales. Les principaux packages sont `sf`, `cartography`, `mapview`, `raster`, `SpatialPosition`, `spatstats` mais d'autres pourront être nécessaires ponctuellement.

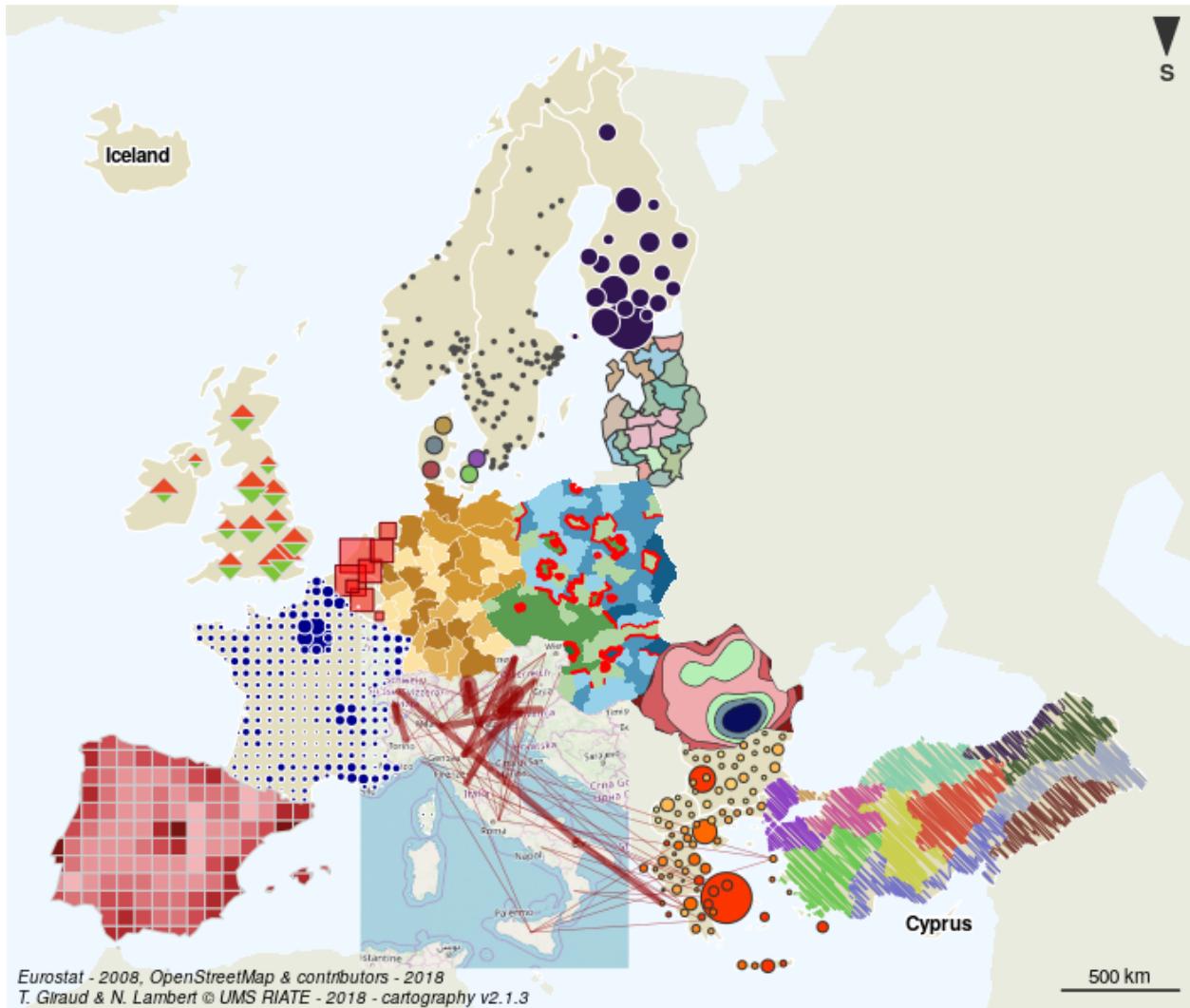


Figure 1:

Chapter 1

Les données spatiales

Il est possible d'importer, de manipuler, de traiter, d'afficher et d'exporter des données spatiales avec R. La grande majorité des opérations de géotraitement sont disponible dans R grace au package **sf**. Il devient alors possible d'utiliser R comme un SIG.

1.1 Le package **sf**

Historique

Historiquement trois packages permettait d'importer, de manipuler et de transformer les données spatiales. Le package **rgdal** qui est une interface entre R et les librairies GDAL (Geospatial Data Abstraction Library) et PROJ4 permet d'importer et d'exporter les données spatiales (les shapefiles par exemple) et aussi de gérer les projections cartographiques

Le package **sp** fournit des classes et méthodes pour les données spatiales dans R. C'est grâce à ce package que l'on peut afficher des fond de cartes, inspecter une table attributaire etc.

Le package **rgeos** donne accès à la librairie d'opérations spatiales GEOS (Geometry Engine - Open Source) et rend donc disponible les opérations SIG classiques : calcul de surface ou de périmètre, calcul de distances, agrégations spatiales, zones tampons, intersections etc.

La suite

Le package **sf** a été publié fin 2016 par Edzer Pebesma (auteur de **sp**). Son objectif est de combiner dans les fonctionnalités de **sp**, **rgeos** et **rgdal** dans un package unique plus ergonomique. Ce package propose des objets plus simples (suivant le standard simple feature) dont la manipulation est plus aisée. Une attention particulière a été portée à la compatibilité du package avec la syntaxe *pipe* et les opérateurs du **tidyverse**.

Aujourd'hui, les principaux développements dans l'écosystème spatial de R se détachent progressivement des 3 anciens (**sp**, **rgdal**, **rgeos**) pour se reposer sur **sf**. Dans ce document nous utiliserons **sf** tant que cela est possible, c'est à dire la plupart du temps.

1.1.1 Format des objets spatiaux **sf**

```
knitr::include_graphics("img/sf.png")
```

```
## Simple feature collection with 100 features and 6 fields
## geometry type: MULTIPOLYGON
## dimension: XY
## bbox: xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965
## epsg (SRID): 4267
## proj4string: +proj=longlat +datum=NAD27 +no_defs
## precision: double (default; no precision model)
## First 3 features:
##   BIR74 SID74 NWBIR74 BIR79 SID79 NWBIR79
## 1 1091     1      10    1364     0      19 MULTIPOLYGON((( -81.47275543...
## 2 487      0      10    542      3      12 MULTIPOLYGON((( -81.23989105...
## 3 3188     5      208   3616     6      260 MULTIPOLYGON((( -80.45634460...
```

Simple feature

Simple feature geometry list-column (sfc)

Simple feature geometry list-column (sfc)

Figure 1.1:

```
## Simple feature collection with 100 features and 6 fields
## geometry type: MULTIPOLYGON
## dimension: XY
## bbox: xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965
## epsg (SRID): 4267
## proj4string: +proj=longlat +datum=NAD27 +no_defs
## precision: double (default; no precision model)
## First 3 features:
##   BIR74 SID74 NWBIR74 BIR79 SID79 NWBIR79
## 1 1091     1      10    1364     0      19
## 2 487      0      10    542      3      12
## 3 3188     5      208   3616     6      260
```

Simple feature

Simple feature

Les objets sf sont des data.frame dont l'une des colonne contient des géométrie. Cette colonne est de la classe sfc (simple feature column) et chaque individu de la colonne est un sfg (simple feature geometry). Ce format très pratique dans la mesure où les données et les géométries sont intrinsèquement liées dans un même objet.

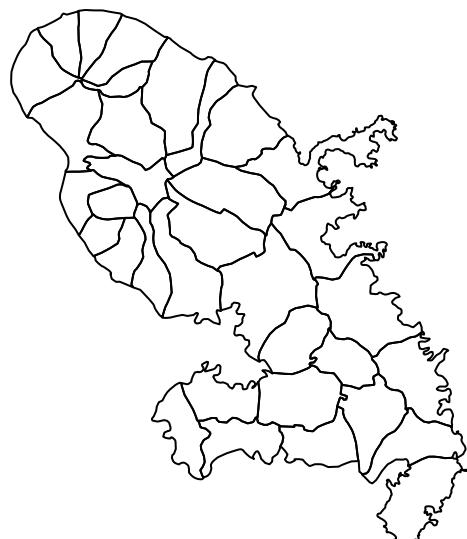
1.1.2 Import / Export

```
library(sf)
mtq <- st_read("data/martinique.shp")
```

```
Reading layer `martinique' from data source `/home/tim/Documents/prz/carto_avec_r/data/martinique.shp'
Simple feature collection with 34 features and 23 fields
geometry type:  POLYGON
dimension:      XY
bbox:           xmin: 690574.4 ymin: 1592426 xmax: 736126.5 ymax: 1645660
epsg (SRID):   32620
proj4string:   +proj=utm +zone=20 +datum=WGS84 +units=m +no_defs
```

1.1.3 Affichage de données

```
plot(st_geometry(mtq))
```



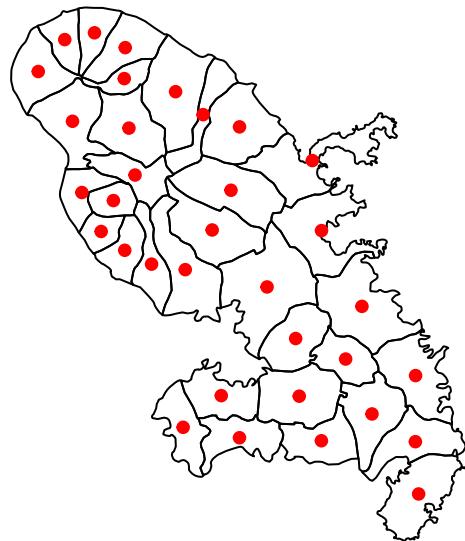
1.1.4 Joindre des données

1.2 Les projections

1.3 Opérations de géotraitements

1.3.1 Extraire des centroides

```
mtq_c <- st_centroid(mtq)
plot(st_geometry(mtq))
plot(st_geometry(mtq_c), add=TRUE, cex=1.2, col="red", pch=20)
```



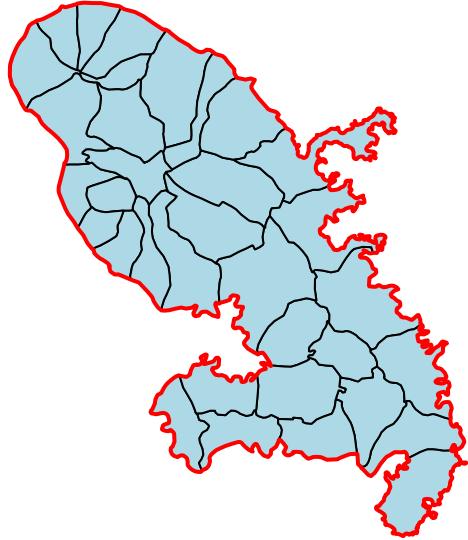
1.3.2 Créer une matrice de distances

```
mat <- st_distance(x=mtq_c, y=mtq_c)
mat[1:5,1:5]
```

```
Units: [m]
      [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 0.000 35297.56 3091.501 12131.617 17136.310
[2,] 35297.557 0.00 38332.602 25518.913 18605.249
[3,] 3091.501 38332.60 0.000 15094.702 20226.198
[4,] 12131.617 25518.91 15094.702 0.000 7177.011
[5,] 17136.310 18605.25 20226.198 7177.011 0.000
```

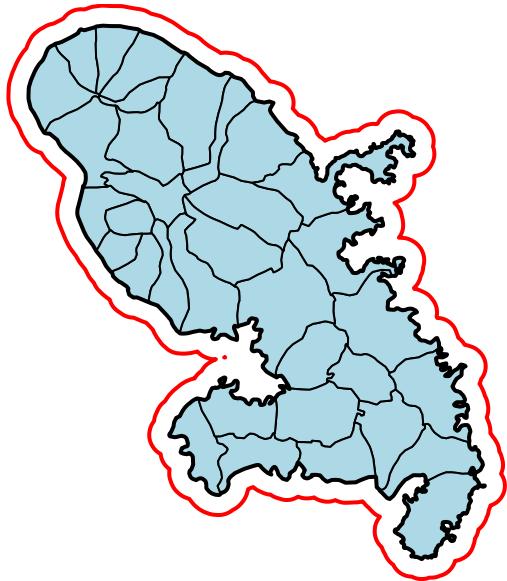
1.3.3 Agréger des polygones

```
mtq_u <- st_union(mtq)
plot(st_geometry(mtq), col="lightblue")
plot(st_geometry(mtq_u), add=T, lwd=2, border = "red")
```



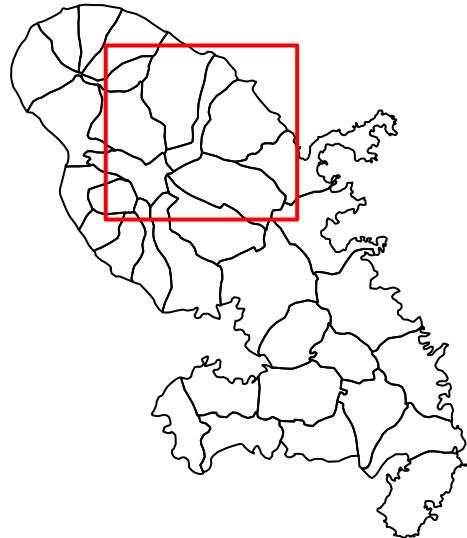
1.3.4 Construire une zone tampon

```
mtq_b <- st_buffer(x = mtq_u, dist = 2000)
plot(st_geometry(mtq), col="lightblue")
plot(st_geometry(mtq_u), add=T, lwd=2)
plot(st_geometry(mtq_b), add=T, lwd=2, border = "red")
```

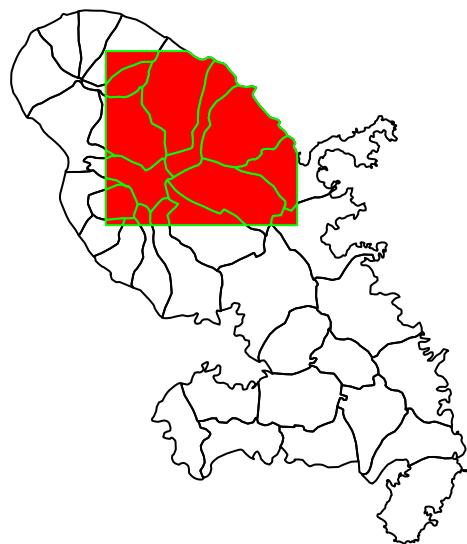


1.3.5 Réaliser une intersection

```
m <- rbind(c(700015,1624212), c(700015,1641586), c(719127,1641586), c(719127,1624212), c(700015,1624212))
p <- st_sf(st_sfc(st_polygon(list(m))), crs = st_crs(mtq))
plot(st_geometry(mtq))
plot(p, border="red", lwd=2, add=T)
```



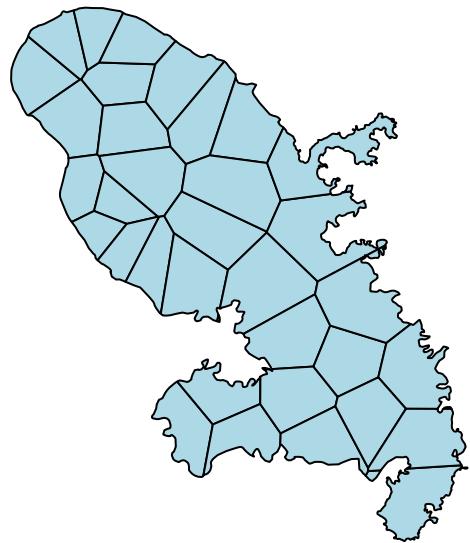
```
mtq_z <- st_intersection(x = mtq, y = p)
plot(st_geometry(mtq))
plot(st_geometry(mtq_z), col="red", border="green", add=T)
```



1.3.6 Construire des polygones de Voronoi

google: “st_voronoi R sf” (<https://github.com/r-spatial/sf/issues/474> & <https://stackoverflow.com/questions/45719790/create-voronoi-polygon-with-simple-feature-in-r>)

```
mtq_v <- st_voronoi(x = st_union(mtq_c))
mtq_v <- st_intersection(st_cast(mtq_v), st_union(mtq))
mtq_v <- st_join(x = st_sf(mtq_v), y = mtq_c, join=st_intersects)
mtq_v <- st_cast(mtq_v, "MULTIPOLYGON")
plot(st_geometry(mtq_v), col='lightblue')
```



1.4 Le package raster

Chapter 2

Cartographie thématique

2.1 Le package `cartography`

Le package `cartography` permet de créer et intégrer des cartes thématiques dans sa chaîne de traitements en R. Il permet des représentations cartographiques tels que les cartes en symboles proportionnels, des cartes choroplèthes, des typologies, des cartes de flux ou des cartes de discontinuités. Il offre également des fonctions qui permettent d'améliorer la réalisation de la carte, comme des palettes de couleur, des éléments d'habillage (échelle, flèche du nord, titre, légende...), d'y rattacher des labels ou d'accéder à des APIs cartographiques.

Pour utiliser aisément ce package, plusieurs sources d'intérêts peuvent être consultées :

- La documentation du package qui documente toutes les fonctions du package, accessible directement dans R Studio. Pour cela, vous pouvez taper simplement :

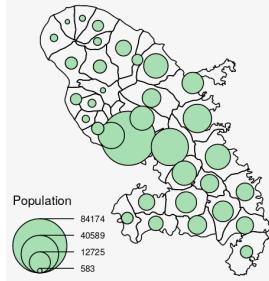
```
?cartography
```

- La cheat sheet de `cartography`, qui résume les principales fonctions du package de façon synthétique.

Thematic maps with cartography :: CHEAT SHEET

Use cartography with spatial objects from sf or sp packages to create thematic maps.

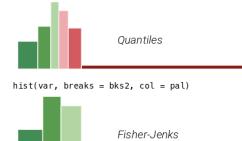
```
library(cartography)
library(sf)
mtq <- st_read("martinique.shp")
plot(st_geometry(mtq))
propSymbolsLayer(x = mtq, var = "P13_POP",
  legend.title.txt = "Population",
  col = "#ad7fb4")
```



Classification

Available methods are: quantile, equal, q5, fisher-jenks, mean-sd, sd, geometric progression.

```
bks1 <- getBreaks(v = var, nclass = 6,
  method = "quantile")
bks2 <- getBreaks(v = var, nclass = 6,
  method = "fisher-jenks")
pal <- carto.pal("green.pal", 3, "wine.pal", 3)
```



hist(var, breaks = bks2, col = pal)

hist(var, breaks = bks1, col = pal)

Symbology

In most functions the x argument should be an sf object. sp objects are handled through spdf and df arguments.

Choropleth
choroLayer(x = mtq, var = "myvar",
method = "hexagonal", nclass = 6)

Typology
typLayer(x = mtq, var = "myvar")

Proportional Symbols
propSymbolsLayer(x = mtq, var = "myvar",
inches = 0.1, symbols = "circle")

Colorized Proportional Symbols (relative data)
propSymbolsChoroLayer(x = mtq, var = "myvar",
var2 = "myvar2")

Colorized Proportional Symbols (qualitative data)
propSymbolsTypeLayer(x = mtq, var = "myvar",
var2 = "myvar2")

Double Proportional Symbols
propTrianglesLayer(x = mtq, var1 = "myvar",
var2 = "myvar2")

OpenStreetMap Basemap (see rosm package)
tile <- getTiles(x = mtq, type = "osm")
tilesLayer(tiles)

Isopleth (see SpatialPosition package)
smoothLayer(x = mtq, var = "myvar",
typefct = "exponential", span = 500,
beta = 2)

Discontinuities
disLayer(x = mtq.borders, df = mtq,
var = "myvar", threshold = 0.5)

Flows
flowLinkLayer(x = mtq_link, df = mtq_df,
var = "flj")

Dot Density
dotDensityLayer(x = mtq, var = "myvar")

Labels
labelLayer(x = mtq, txt = "myvar",
halo = TRUE, overlap = FALSE)

Transformations

Polygons to Grid
mtq_grid <- getGridLayer(x = mtq, cellsize = 3.6e+07,
type = "hexagonal", var = "myvar")
Grids layers can be used by
choroLayer() or propSymbolsLayer()

Points to Links
mtq_link <- getLinkLayer(x = mtq, df = link)
mtq_link

Polygons to Borders
mtq_border <- getBorders(x = mtq)

Polygons to Pencil Lines
mtq_pen <- getPencilLayer(x = mtq)

Borders layers can be used by
disLayer() function

Links layers can be used by
"LinkLayer()

Legends

legendChoro()

legendChoro(pos = "topleft",
title.txt = "legendChoro",
nbreaks = 6, labels = c("68, 88, 108, 128, 148, 168"),
col = carto.pal("green.pal", 6),
nodata = TRUE, nodata.txt = "No Data")

legendType()

legendType(title.txt = "legendType()",
categ = c("type 1", "type 2", "type 3"),
nodata = FALSE)

legendCirclesSymbol()

legendCirclesSymbol(var = c(10,100),
col = c('peru', 'skyblue', 'gray7'),
nbreaks = 10, labels = c("10", "100", "1000"),
inches = 0.3)

See also legendSquaresSymbols(), legendBarsSymbols(),
legendGradLines(), legendPencilLines() and legendPopTriangles().

Map Layout

North Arrow:
north\$pos = "topright")
Scale Bar:
barScale(size = 5)

Full Layout:
layoutLayer(
title = "Martinique",
tabtitle = TRUE,
author = "Author",
source = "Source",
north = TRUE,
scale = 5)

Figure Dimensions:
Get figure dimensions based on the dimension ratio of a spatial object,
figure margins and output resolution.

f.dim <- getFigDim(x = sf_obj, width = 500,

mar = c(0,0,0,0),

par(mar = c(0,0,0,0))

plot(sf_obj, col = "#7f7fcf")

dev.off()

default

a / b == f.dim[1] / f.dim[2]

f.dim[2] / f.dim[1]

controlled ratio

Color Palettes

carto.pal(pal1 = "blue.pal", n1 = 5,

pal2 = "sand.pal", n2 = 3)

display.carto.all(n = 8)

blue.pal

orange.pal

red.pal

brown.pal

purple.pal

green.pal

wine.pal

pink.pal

grey.pal

turquoise.pal

sand.pal

taupe.pal

kaki.pal

harro.pal

pastel.pal

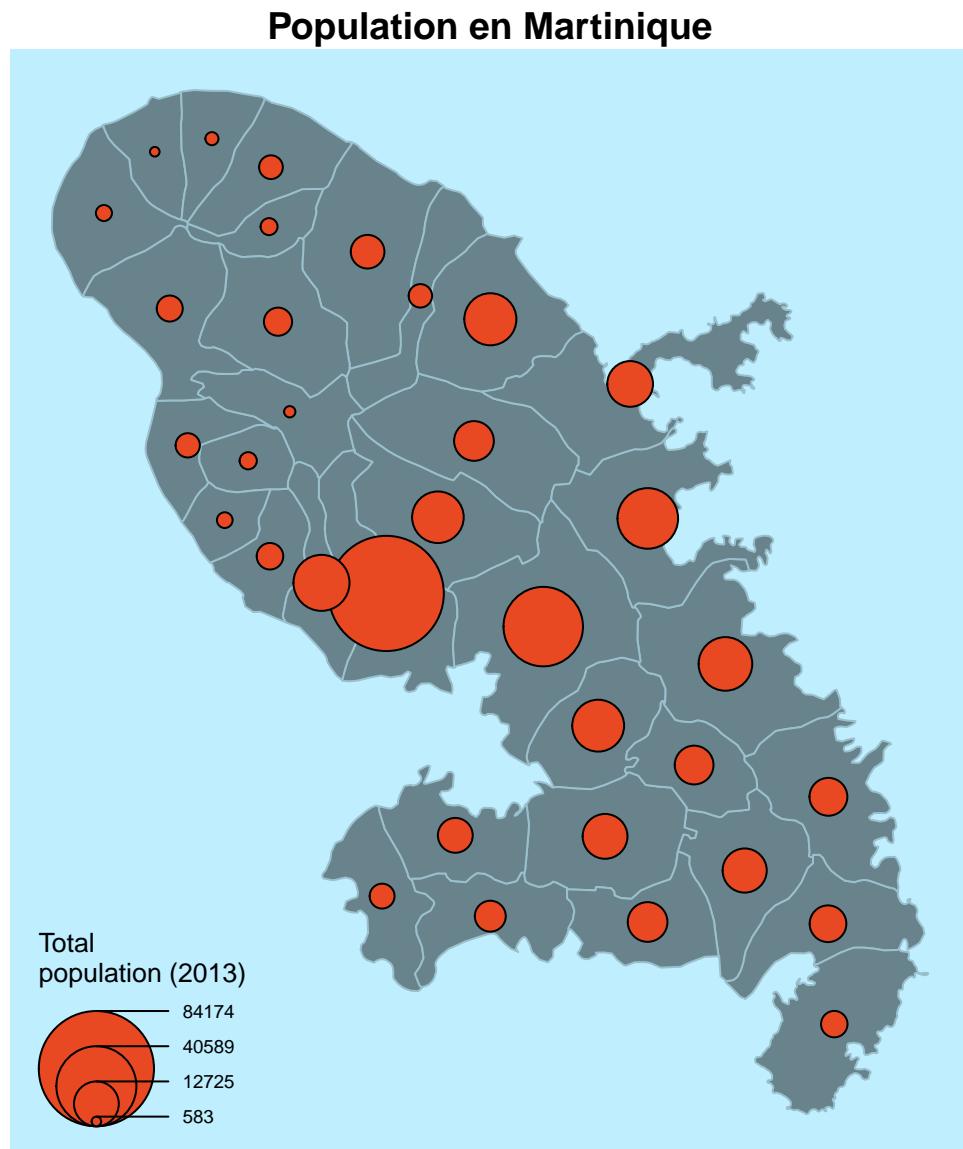
multi.pal

- La vignette associée au package, qui présente des réalisations issues de ce package, elle aussi accessible directement dans R.

- Le blog R Géomatique, maintenu par l'auteur de *cartography* qui met à disposition ressources et exemples d'intérêt liés au package et à la représentation d'information spatiale sous R.

2.1.1 Symboles proportionnels

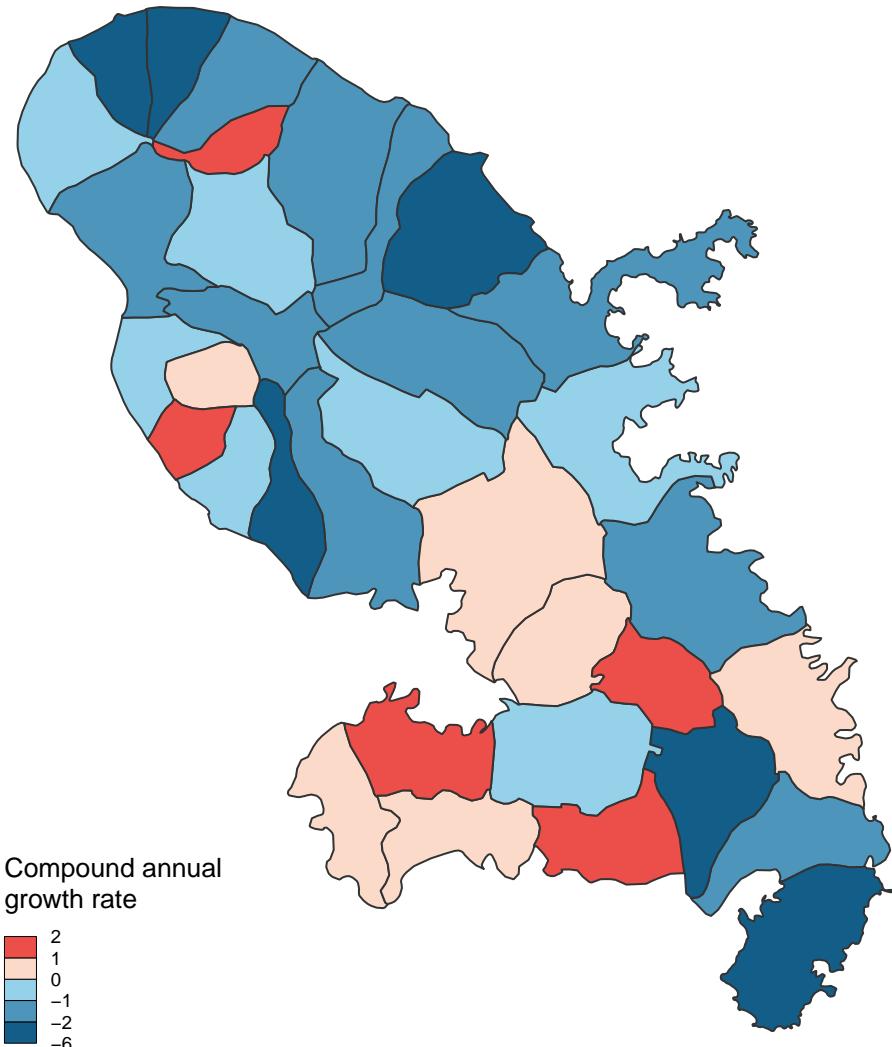
```
library(cartography)
library(sf)
# Import des données
mtq <- st_read(system.file("shape/martinique.shp", package="cartography"))
# Communes
plot(st_geometry(mtq), col = "lightblue4", border = "lightblue3",
  bg = "lightblue1")
# Symboles proportionnels
propSymbolsLayer(x = mtq, var = "P13_POP",
  legend.title.txt = "Total\npopulation (2013)")
# Titre
title(main = "Population en Martinique")
```



2.1.2 Carte choroplète

```
mtq$cagr <- (((mtq$P13_POP / mtq$P08_POP)^(1/4)) - 1) * 100
choroLayer(x = mtq, var = "cagr", breaks = c(-6.14, -2, -1, 0, 1, 2),
            col = c("#135D89", "#4D95BA", "#96D1EA", "#FCDACA", "#EC4E49"),
            legend.title.txt = "Compound annual\ngrowth rate")
title(main = "Evolution de la population")
```

Evolution de la population

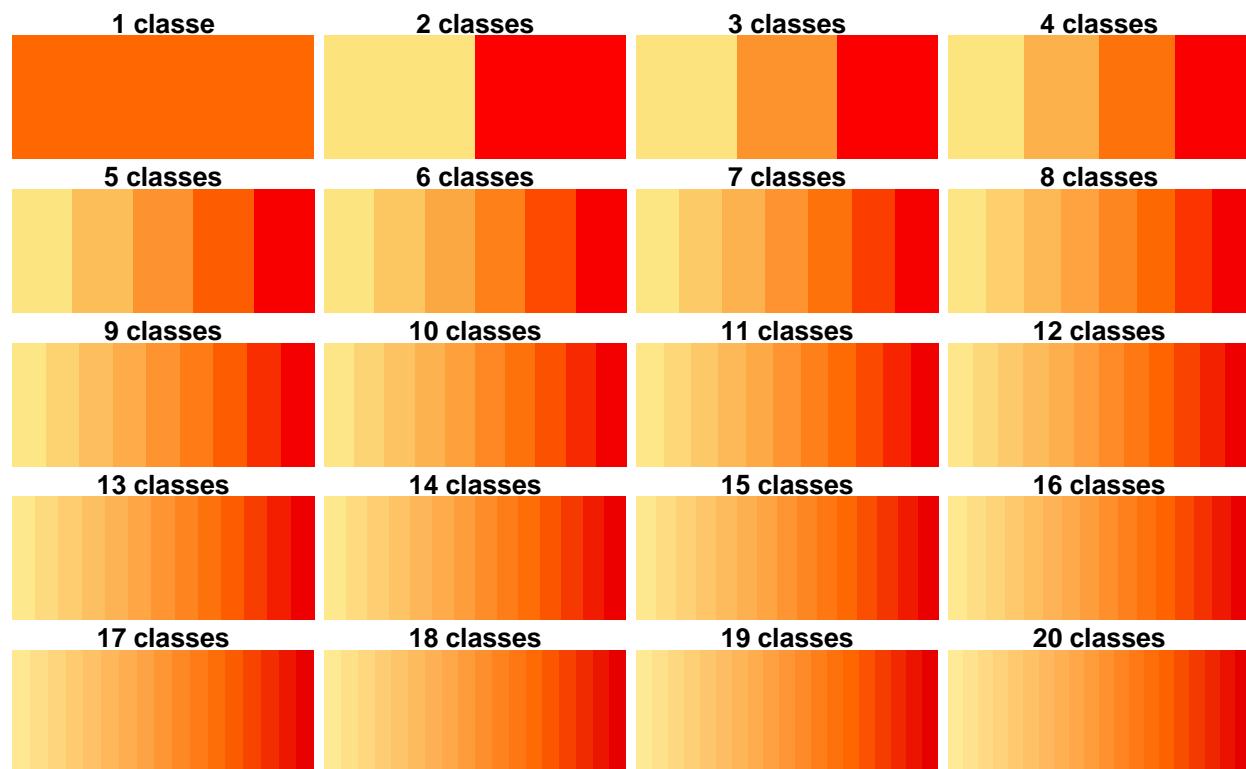


2.2 Palettes de couleurs

```
display.carto.all(20)
```

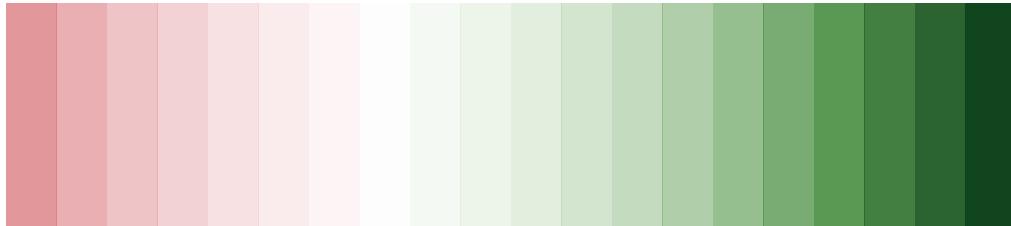


```
display.carto.pal("orange.pal")
```



```
mypal <- carto.pal(pal1 = "wine.pal", n1 = 7, pal2 = "green.pal", n2 = 12,
                     middle = TRUE, transparency = TRUE)
k <- length(mypal)
```

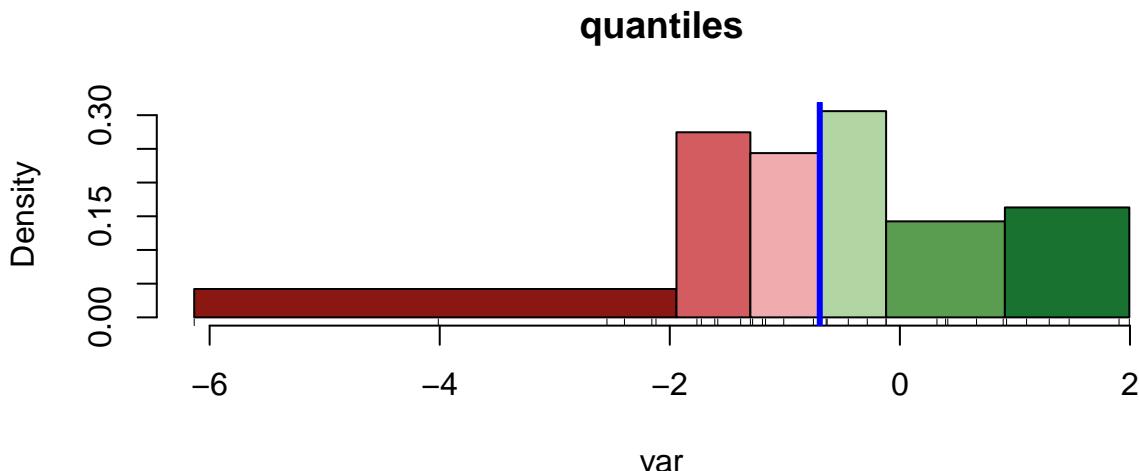
```
image(1:k, 1, as.matrix(1:k), col=mypal, xlab = paste(k, " classes", sep=""),
      ylab = "", xaxt = "n", yaxt = "n", bty = "n")
```



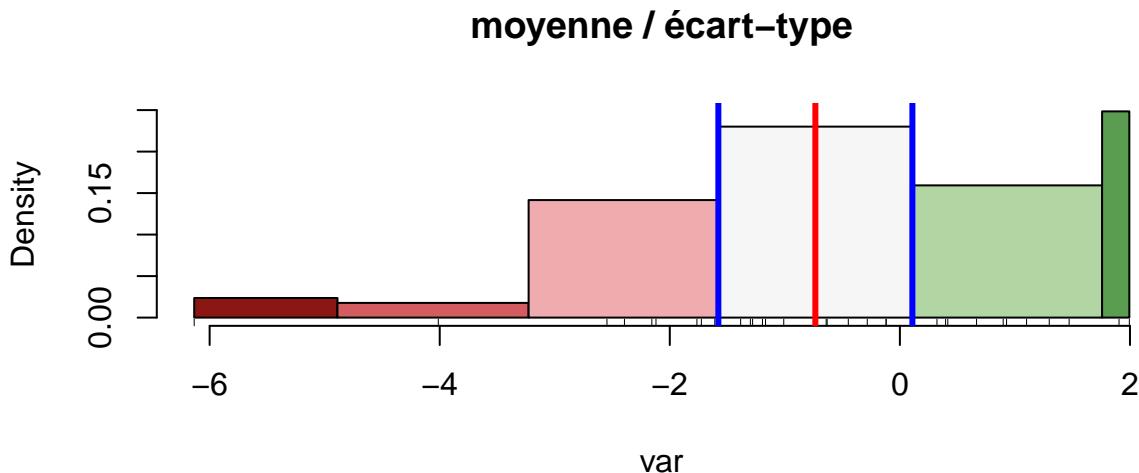
20 classes

2.3 Discrétilisations

```
var <- mtq$cagr
moy <- mean(var)
med <- median(var)
std <- sd(var)
# Quantile intervals
breaks <- getBreaks(v = var, nclass = 6, method = "quantile")
hist(var, probability = TRUE, breaks = breaks, main="quantiles",
     col = carto.pal(pal1 = "wine.pal", 3, "green.pal", 3))
rug(var)
abline(v = med, col = "blue", lwd = 3)
```



```
# Mean and standard deviation (msd)
breaks <- getBreaks(v = var, method = "msd", k = 1, middle = TRUE)
hist(var, probability = TRUE, breaks = breaks, main="moyenne / écart-type",
     col = carto.pal(pal1 = "wine.pal", 3 , "green.pal", 2, middle = TRUE))
rug(var)
abline(v = moy, col = "red", lwd = 3)
abline(v = moy + 0.5 * std, col = "blue", lwd = 3)
abline(v = moy - 0.5 * std, col = "blue", lwd = 3)
```



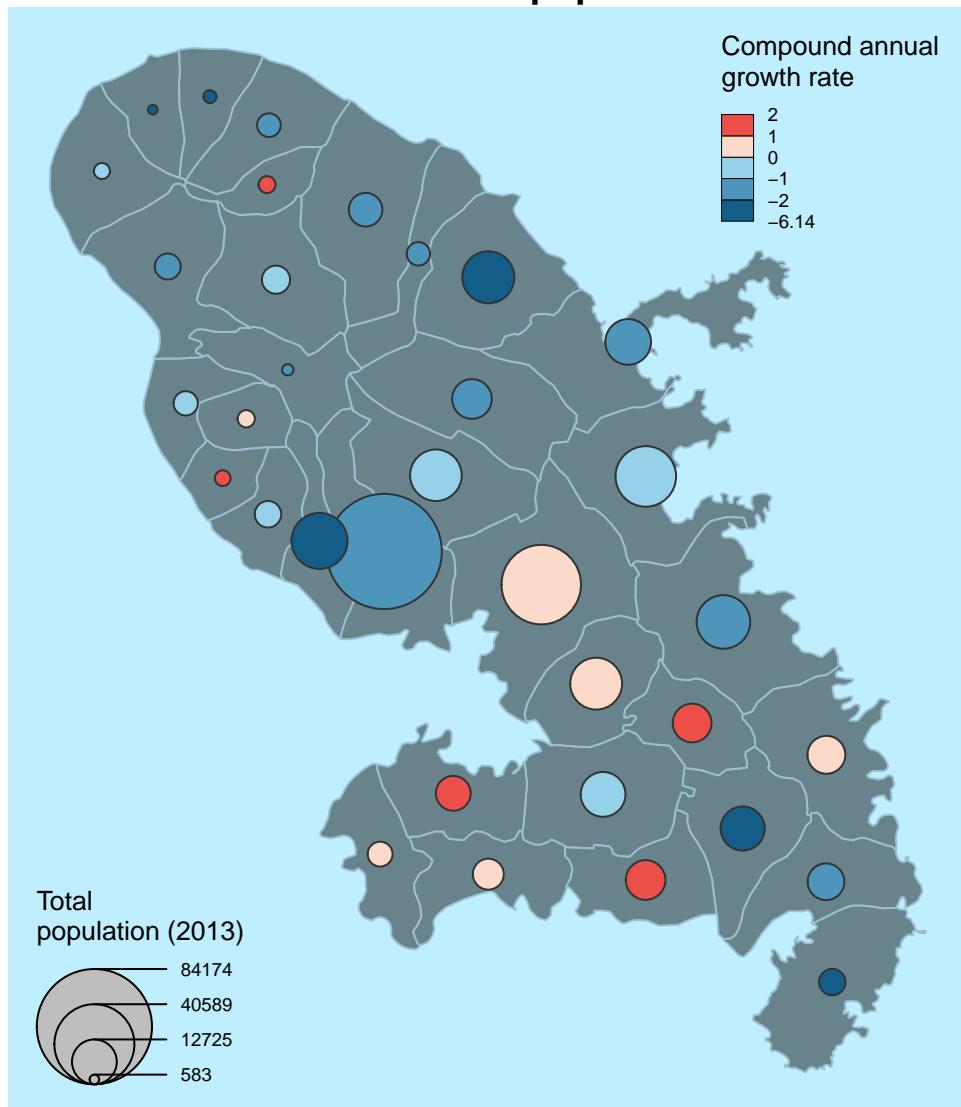
2.4 Combinaisons

```

plot(st_geometry(mtq), col = "lightblue4",
      border = "lightblue3", bg = "lightblue1")
propSymbolsChoroLayer(x = mtq, var= "P13_POP",
                      legend.var.title.txt = "Total\npopulation (2013)",
                      var2 = "cagr", legend.var.pos = "bottomleft",
                      breaks = c(-6.14,-2,-1,0,1,2),
                      col = c("#135D89", "#4D95BA", "#96D1EA", "#FCDACA", "#EC4E49"),
                      legend.var2.title.txt = "Compound annual\ngrowth rate")
# Title
title(main = "Evolution de la population")

```

Evolution de la population



2.5 Labels

```
plot(st_geometry(mtq), col = "darkseagreen3", border = "darkseagreen4",
      bg = "#A6CAE0")
labelLayer(x = mtq, txt = "LIBGEO", col= "black", cex = 0.7, font = 4,
           halo = TRUE, bg = "white", r = 0.1, overlap = FALSE,
           show.lines = FALSE)
```



2.6 Les données OSM

OpenStreetMap (OSM) est un projet de cartographie participative qui a pour but de constituer une base de données géographiques libre à l'échelle mondiale. OpenStreetMap vous permet de voir, modifier et utiliser des données géographiques dans le Monde entier. En résumé, c'est comme Google Maps, mais en mieux...

2.6.1 Données vectorielles

```
library("osmdata")
## Data (c) OpenStreetMap contributors, ODbL 1.0. http://www.openstreetmap.org/copyright
prj <- st_crs(Paris)
```

```

bbox <- st_bbox(st_transform(Paris,4326))

q <- opq(bbox = bbox , timeout = 2000) %>% add_osm_feature(key = 'man_made', value = 'surveillance')
cameras <- osmdata_sf(q)$osm_points
cameras <- st_transform(cameras, prj)

cameras$ok <- st_intersects(st_geometry(cameras), st_geometry(Paris), sparse = FALSE)
cameras <- cameras[cameras$ok == T,]

Paname <- getTiles(x = Paris, type ="cartodark", crop = TRUE)
tilesLayer(Paname)
par(mar = c(0.5,0.5,1.5,0.5))
plot(st_geometry(Paris), lwd=1, border="white", lty=2, add=T)
plot(st_geometry(cameras), pch=20, col="red", add=T)

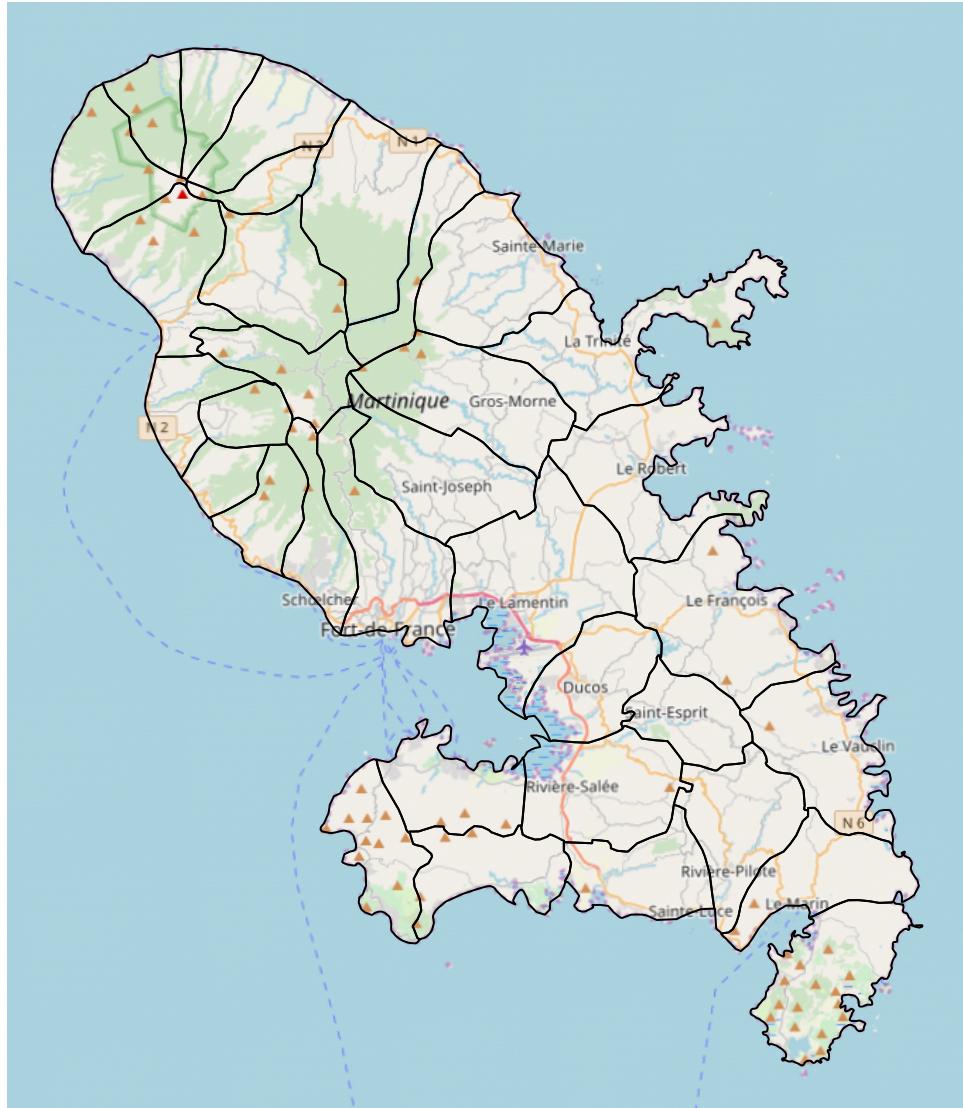
```

2.6.2 Données raster

```

tiles <- getTiles(x = mtq, type = "osm", crop=T, zoom = 11)
tilesLayer(tiles)
plot(st_geometry(mtz), add=T)

```



2.7 Cartographie interactive

leaflet / mapview

2.8 Géocodage d'adresses

2.9 Créditation de cartons

Chapter 3

Cartographie thématique avancée

3.1 Les anamorphoses

Voir : Les anamorphoses cartographiques
Nicolas Lambert, 2015

“L’anamorphose classique est une représentation des États (ou de mailles quelconques) par **des rectangles ou des polygones quelconques** en fonction d’une **quantité** qui leur est rattaché.”

“On s’efforce de **garder l’arrangement général** des mailles ou la silhouette du continent.”

Brunet, R., Ferras, R., & Théry, H. (1993). Les mots de la géographie: dictionnaire critique (No. 03) 911 BRU).

3.1.1 Les cartogrammes de Dorling

La taille des cercles est proportionnelle à une variable.

La position des cercles est définie selon les positions de départ.

Dorling, Daniel (1996): Area Cartograms: Their Use and Creation, Concepts and Techniques in Modern Geography (CATMOG), 59

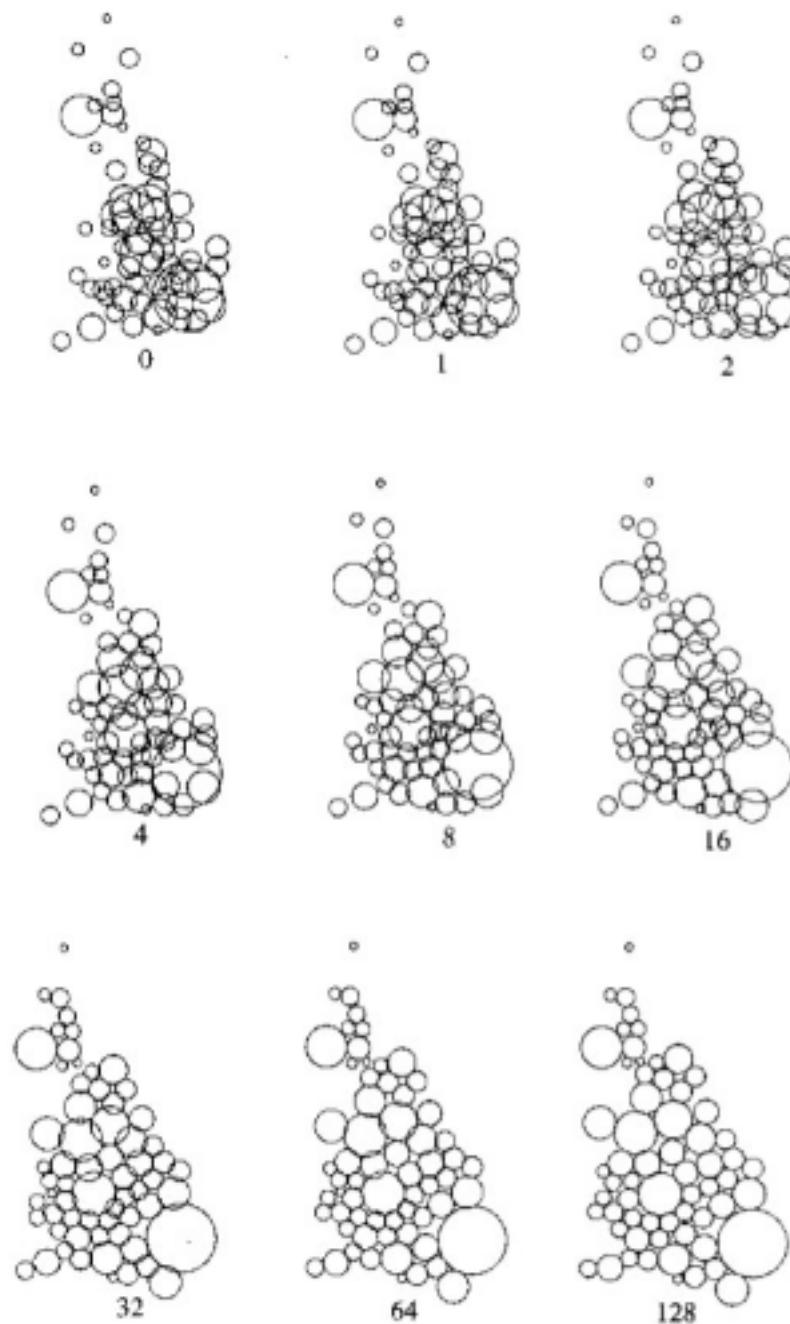
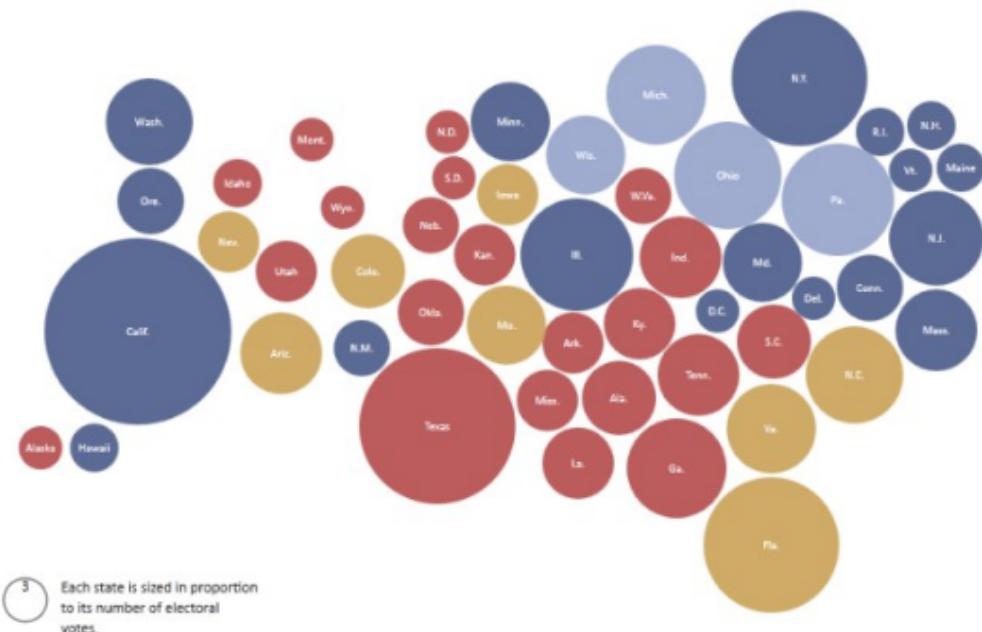
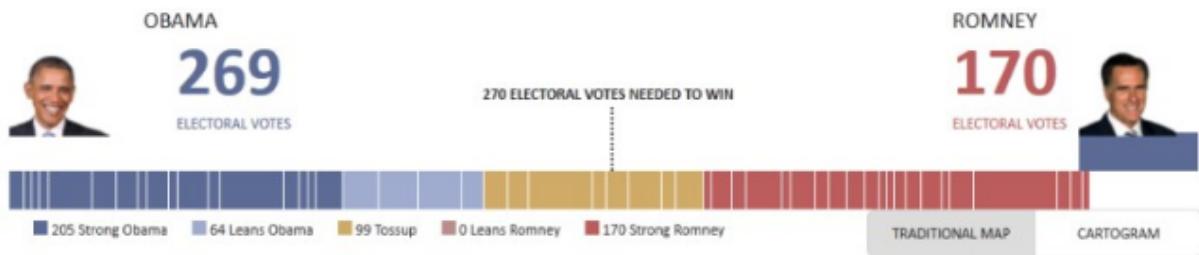


Figure 16 : British counties transformed after 128 iterations of the circular algorithm (Author).

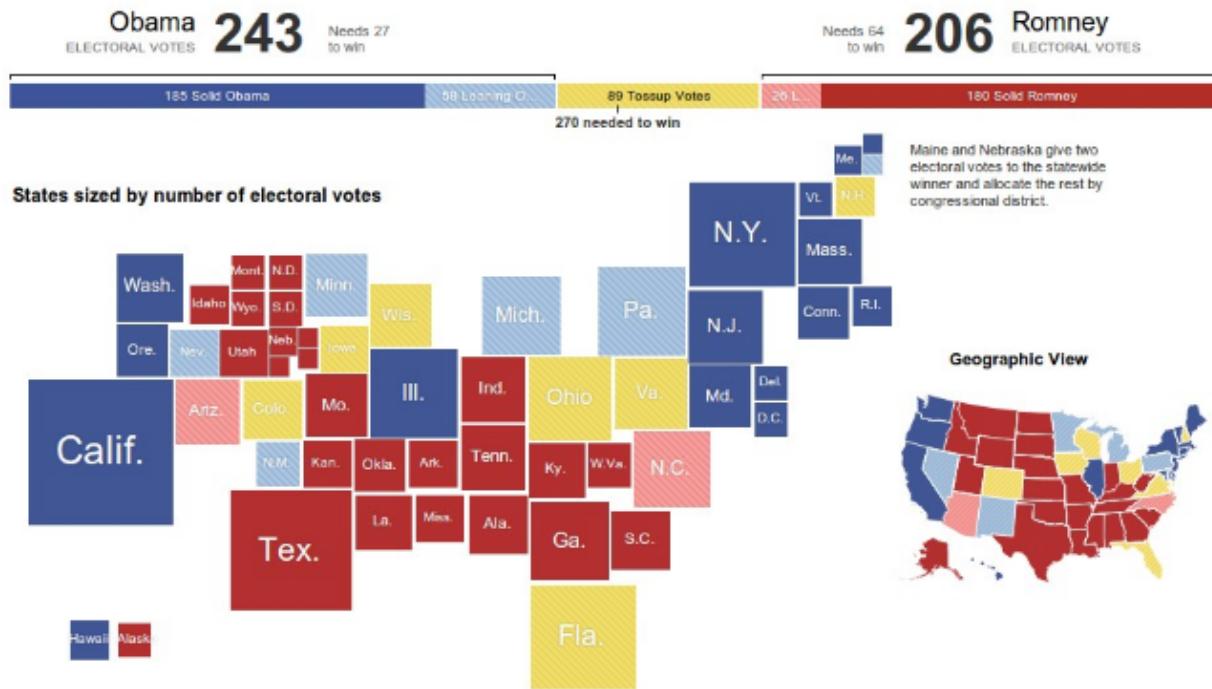
Figure 3.1:

3.1.2 Le principe

3.1.3 Exemple



<http://elections.huffingtonpost.com/2012/romney-vs-obama-electoral-map#map>



<http://elections.nytimes.com/2012/ratings/electoral-map>

3.1.4 Précautions d'emploi

- On identifie assez mal l'espace
- On peut nommer les cercles pour se repérer
- On peut s'aider de la couleur pour faire des clusters et mieux identifier les blocks géographiques
- + La perception de la quantité est très bonne.
- Les tailles de cercles sont vraiment comparables

3.1.5 Les cartogrammes non continus

La taille des polygones est proportionnelle à une variable.

L'agencement des polygones les uns par rapport aux autres est conservée.

La forme des polygones est ressemblante.

3.1.6 Exemple

3.1.7 Précautions d'emploi

- Non contigu, la topologie est perdue.
- + La conservation de la forme des polygones est optimisée.

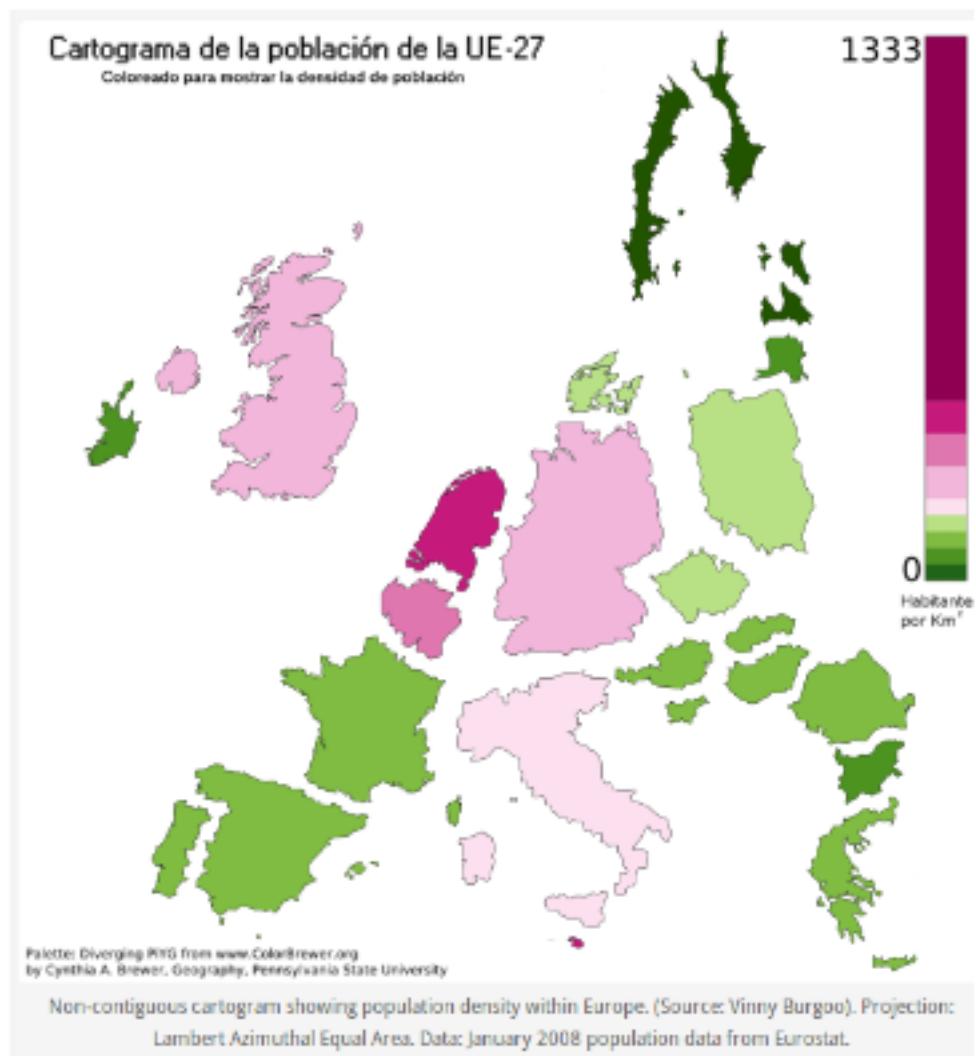


Figure 3.2:

3.1.8 Les cartogrammes continus

La taille des polygones est proportionnelle à une variable.

L'agencement des polygones les uns par rapport aux autres est conservée.

Pour conserver la contiguité, la forme des polygones est fortement transformée.

3.1.9 Exemple

3.1.10 Précautions d'emploi

- Par rapport aux anamorphoses non contigues, la forme des polygones est fortement distordue.

- + C'est une "vraie carte de géographie" : la topologie et la contiguité sont conservées.

3.1.11 Interêts des anamorphoses

Représentation cartographique perçue comme **innovante** (même si la méthode date de 40 ans)

Image très généralisée qui rend bien compte des **quantités** et des **gradiants**.

Une vraie image de **communication** : provoque, suscite l'intérêt, véhicule un message fort, interpelle.

3.1.12 Faiblesses des anamorphoses

Perte des **repères visuels** (difficile de retrouver son pays, ou sa région sur la carte).

Ne permet pas de connaître les **situations locales**.

Demande un **effort de lecture**.

Gestion des données manquantes

3.2 Les grilles régulières

La méthode du carroyage consiste à découper l'espace géographique en un maillage formé de carrés réguliers dans une projection donnée. La donnée est répartie sur ce quadrillage régulier au prorata de la surface représentée. Le quadrillage permet ainsi de s'affranchir des mailles administratives.

La fonction `getGridLayer` du package `cartography` permet de construire ces grilles régulières.

3.2.1 Exemples

```
knitr:::include_graphics(c("img/pregrid.png", "img/grid.png"))
```

WorldMapper : <http://www.worldmapper.org/>

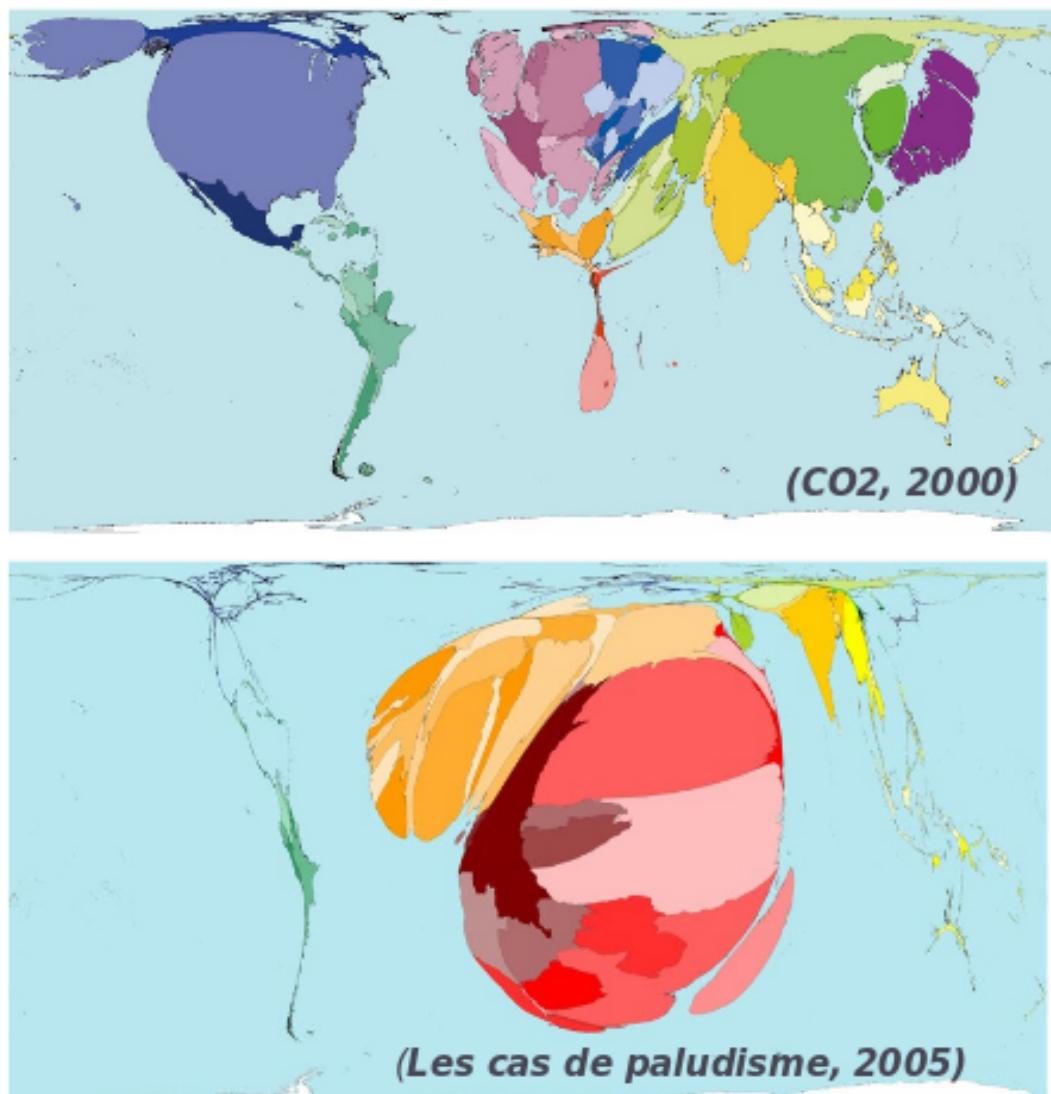


Figure 3.3:

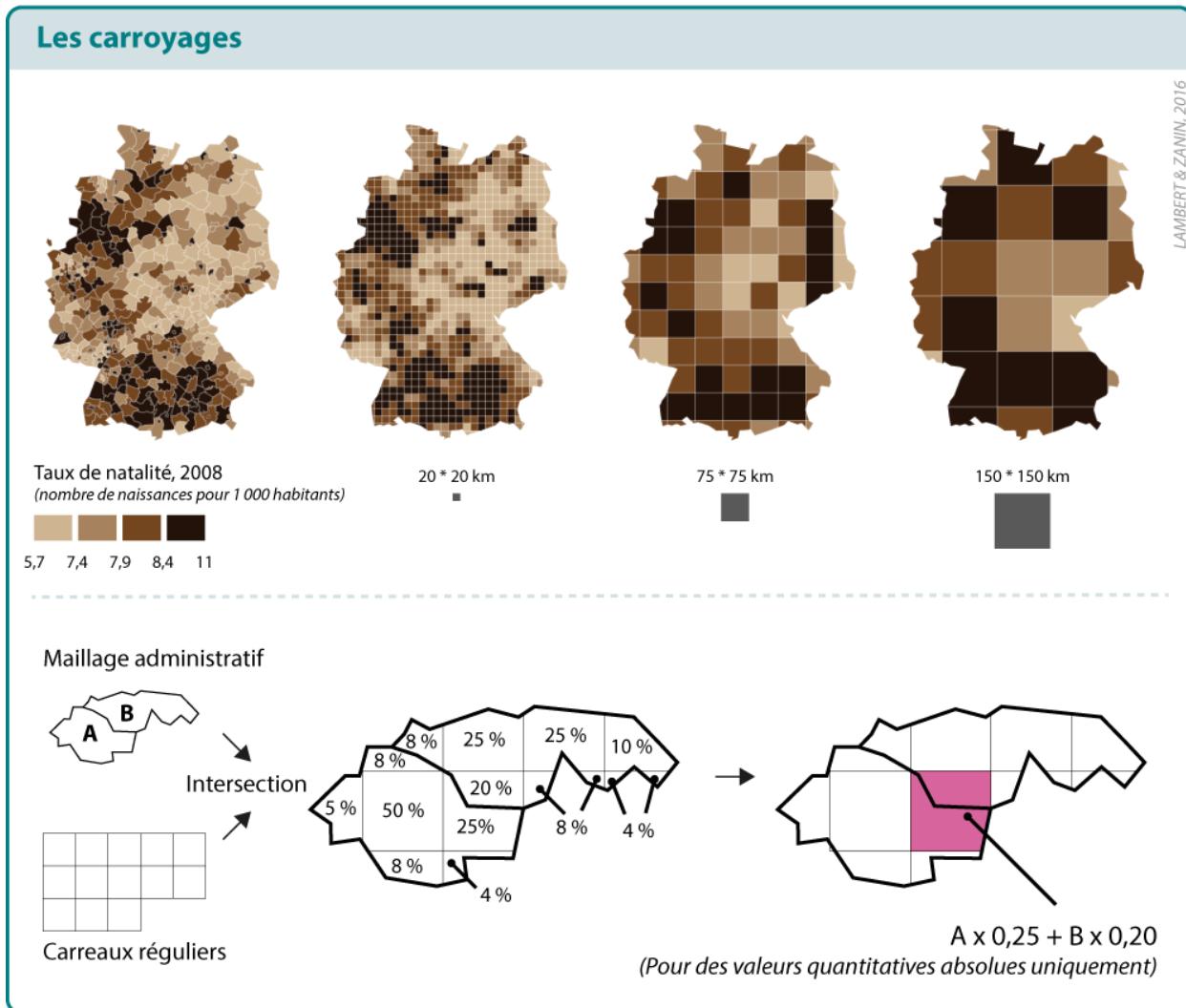
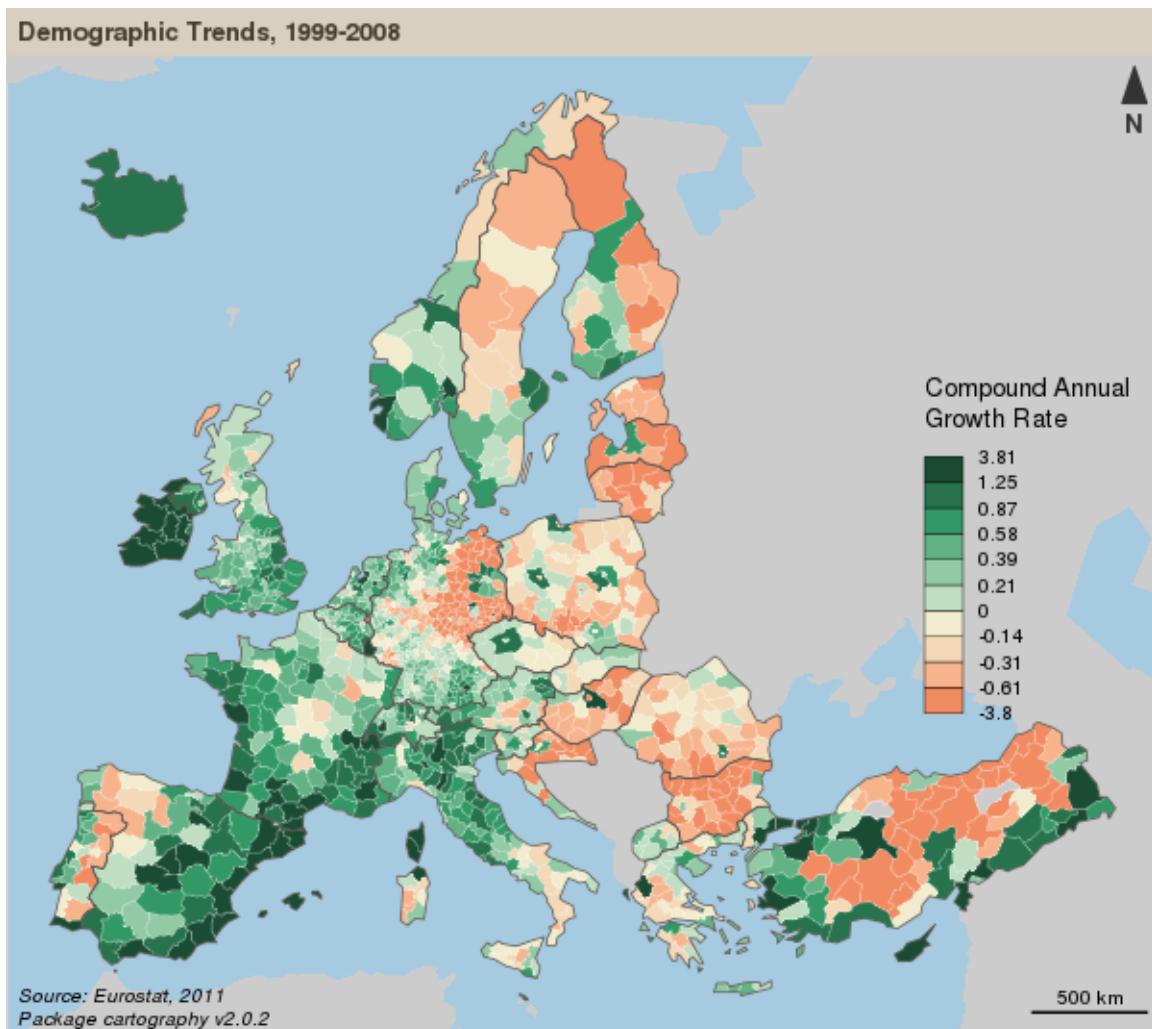
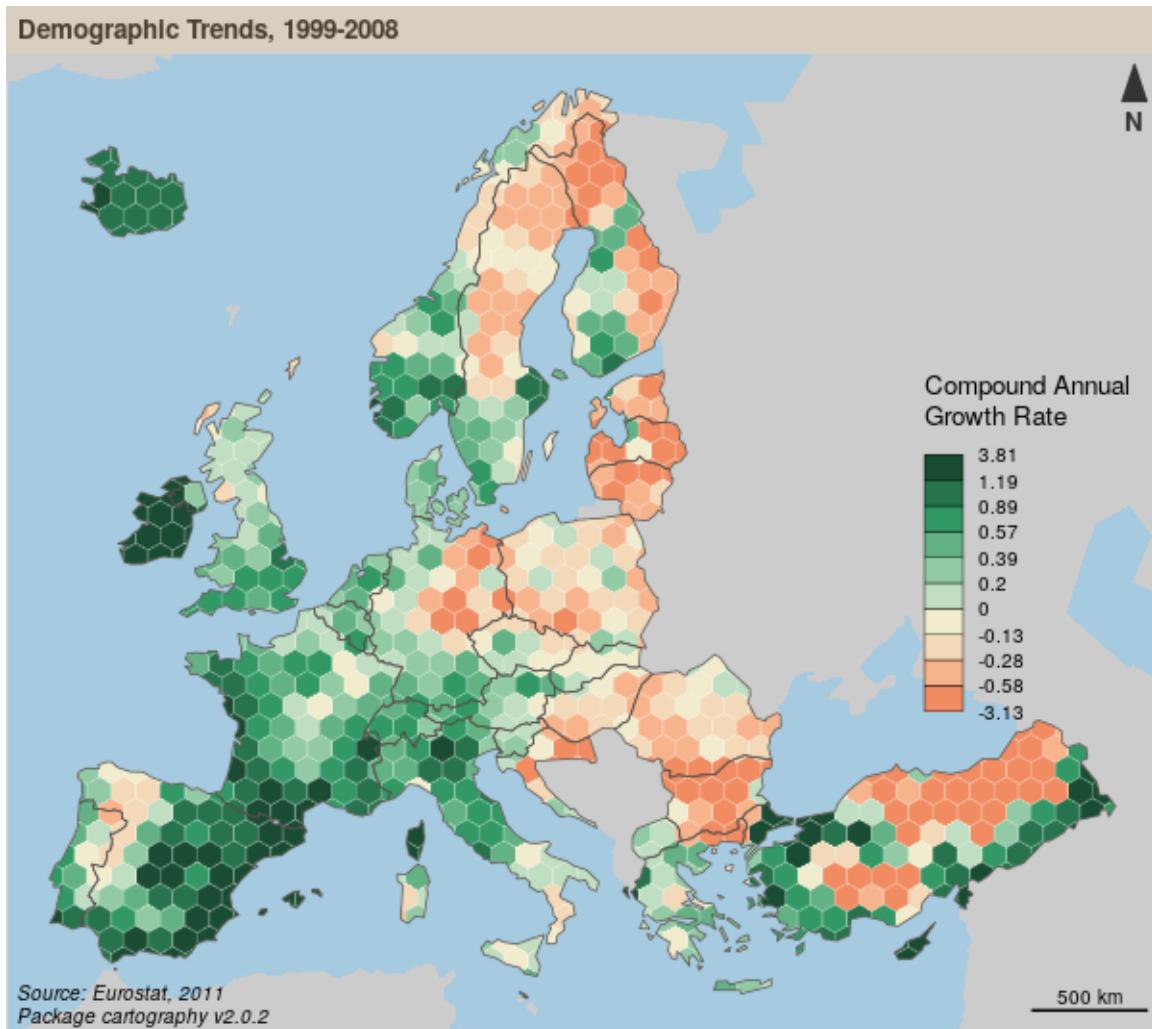


Figure 3.4:





3.2.2 Précautions d'emploi

- Perte de précision, maillage sans signification.
- La version simple (au prorata de la surface), implique une équirépartition du phénomène dans chaque unités.
- + Permet la comparaison de maillages différents, à plusieurs dates ou de différentes sources.

3.3 Les discontinuités

Ce type de représentation permet de souligner cartographiquement les discontinuités territoriales d'un phénomène.

L'accent est porté sur ce qui distingue des territoires.

Pour chaque frontière nous calculons le rapport ou la différence des valeurs des polygones de part et d'autre. Puis nous représentons la frontière par un trait d'autant plus épais que la différence est forte.

Il est souvent bénéfique de coupler ce type de représentation à une représentation choroplèthe (pour comprendre le sens des discontinuités).

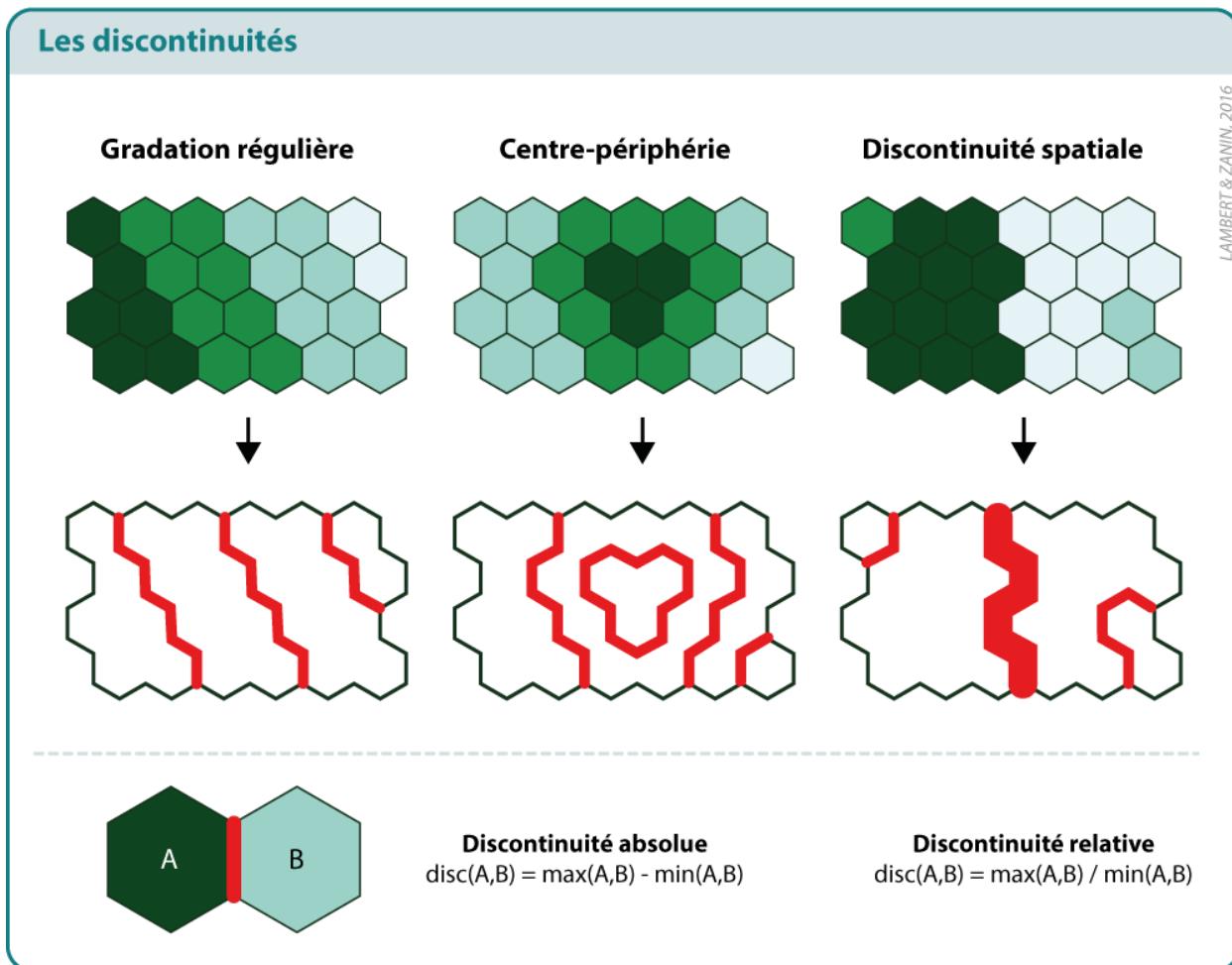


Figure 3.5:

3.3.1 Exemples

3.3.2 Précautions d'emploi

- Ces cartes ne sont pas évidentes à paramétriser.

Le choix des critères (seuil, type de différences...) va fortement influencer la représentation.
En fonction du maillage la lisibilité peut être faible.

+ Représentation très puissante pour montrer les inégalités.

3.4 Les lissages

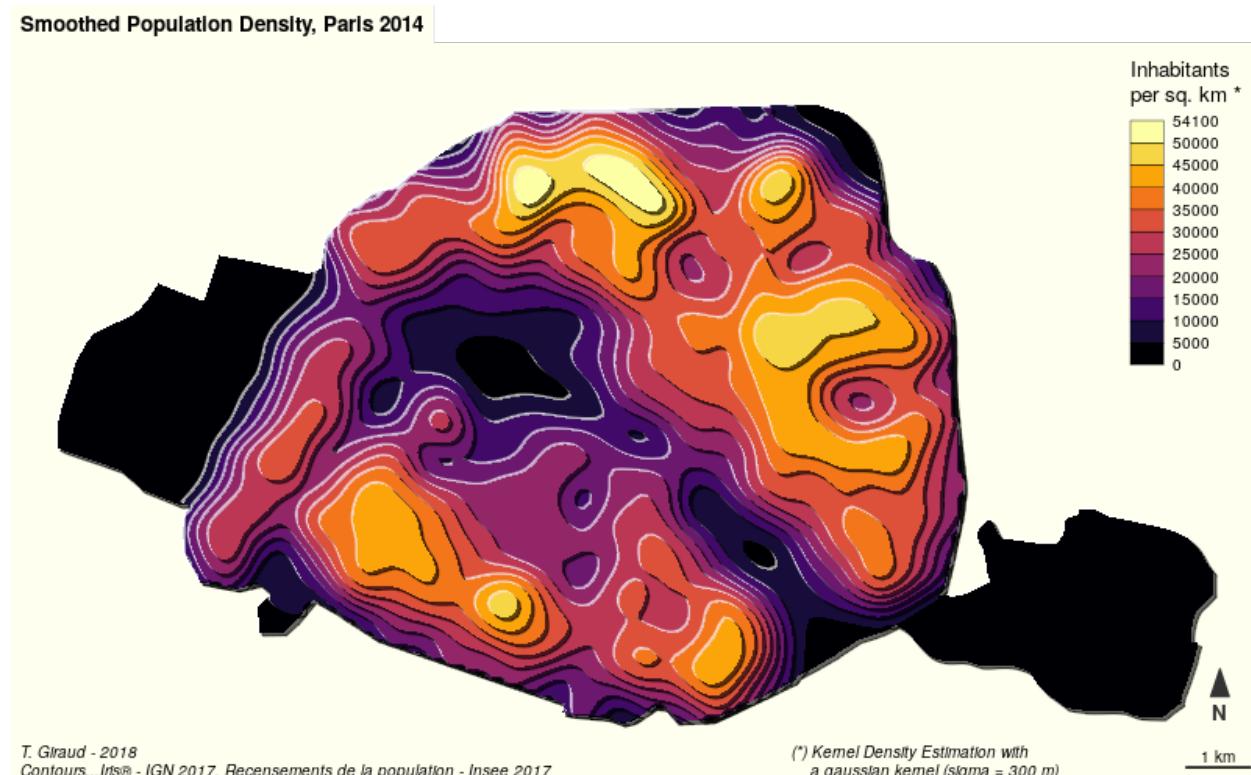
L'idée principale du lissage est de filtrer l'information pour révéler des structures spatiales sous-jacentes.

C'est un ensemble de méthodes qui consistent à affecter aux points que l'on observe une valeur prenant en compte les valeurs de leur voisinage.

Il existe plusieurs méthodes de lissage (kde, potentiels...) plus ou moins paramétrables.

Cette méthode permet de passer représentations ponctuelles à une représentation continu

3.4.1 Exemples



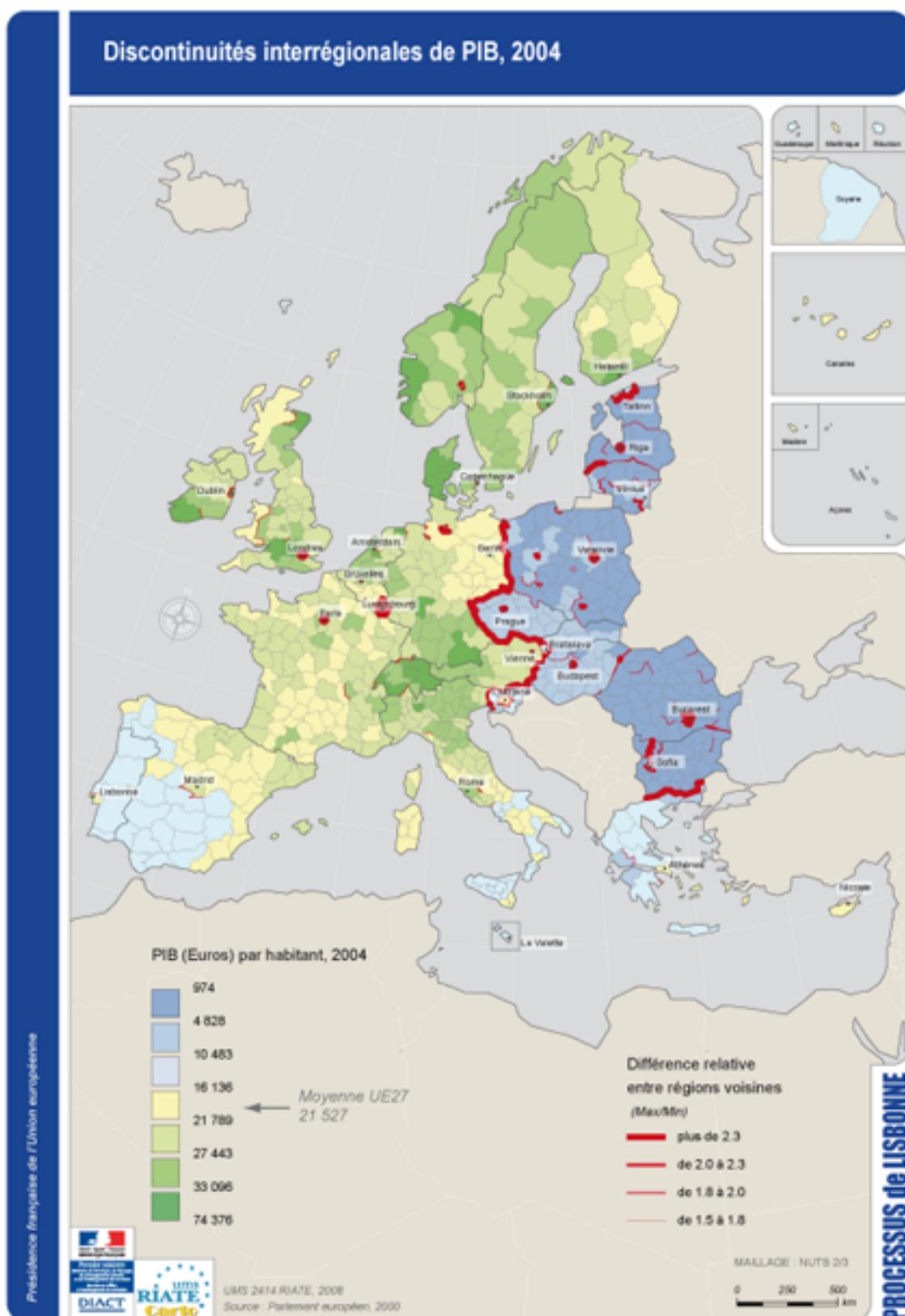


Figure 3.6:

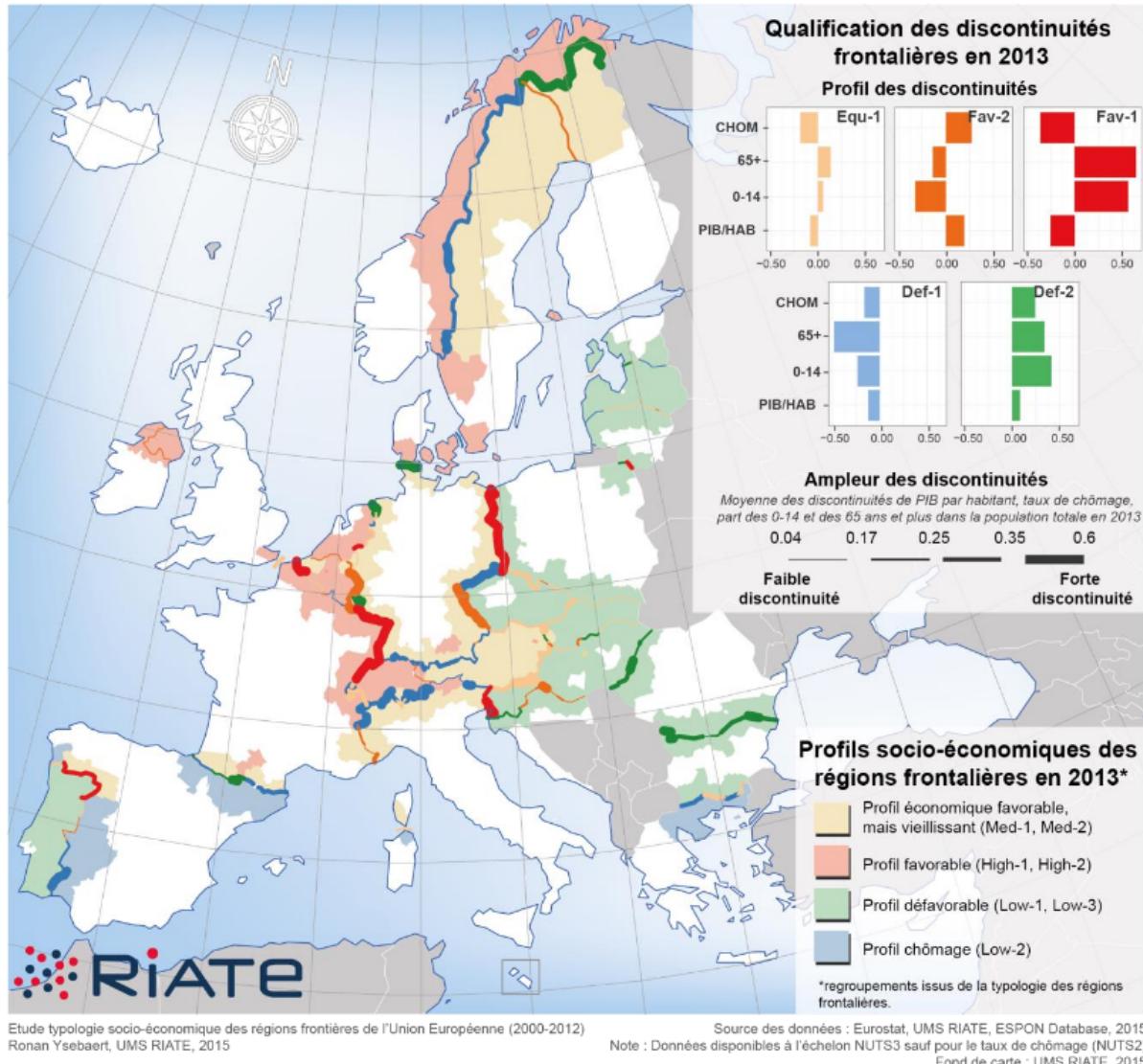
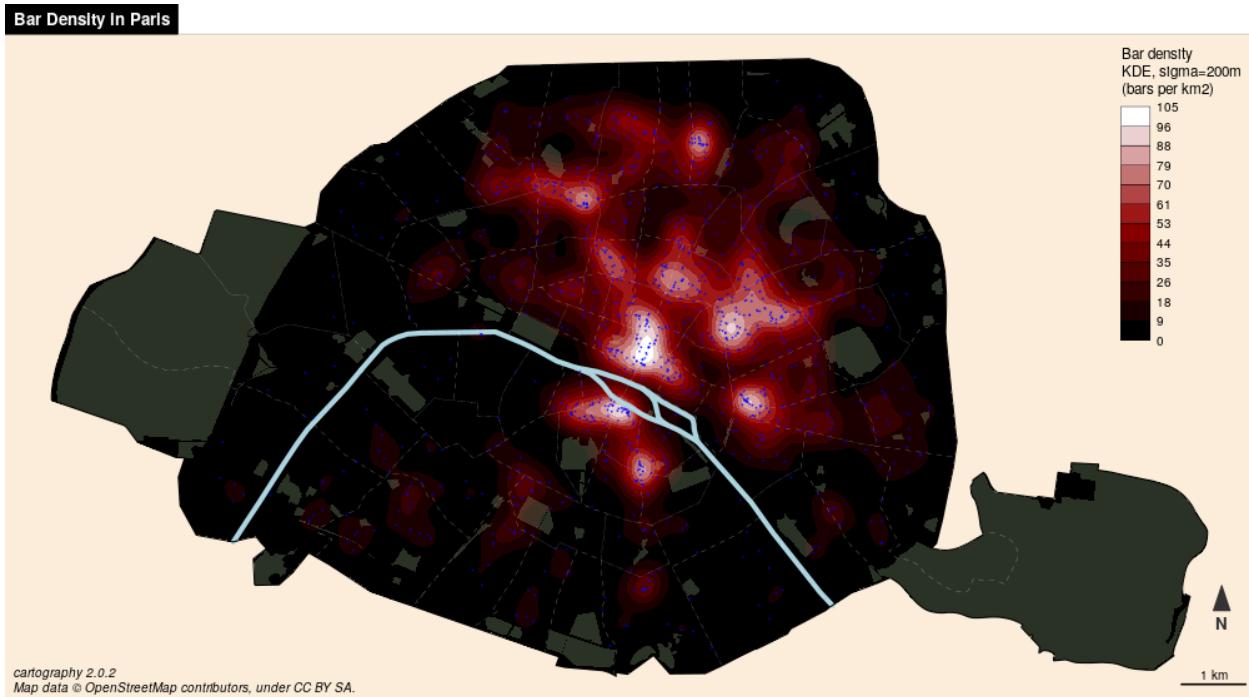


Figure 3.7:



3.4.2 Précautions d'emploi

- Il est difficile de paramétrier correctement les fonctions de lissages.
Elles doivent s'appuyer sur des hypothèses de comportement dans l'espace.
La compréhension par un public large n'est pas évidente, il faut alors simplifier les légendes, la présentation de la méthode.
- + Permet de faire ressortir des phénomènes spatiaux sous-jacents invisibles directement.
Les cartes produites attirent l'oeil par leur originalité.
Cette méthode permet de passer d'une représentation ponctuelle ou discontinue (dans un maillage) à une représentation continue s'affranchissant des maillages existants.

3.4.3 KDE

3.4.4 Stewart

Vignette du package SpatialPosition

3.5 3D

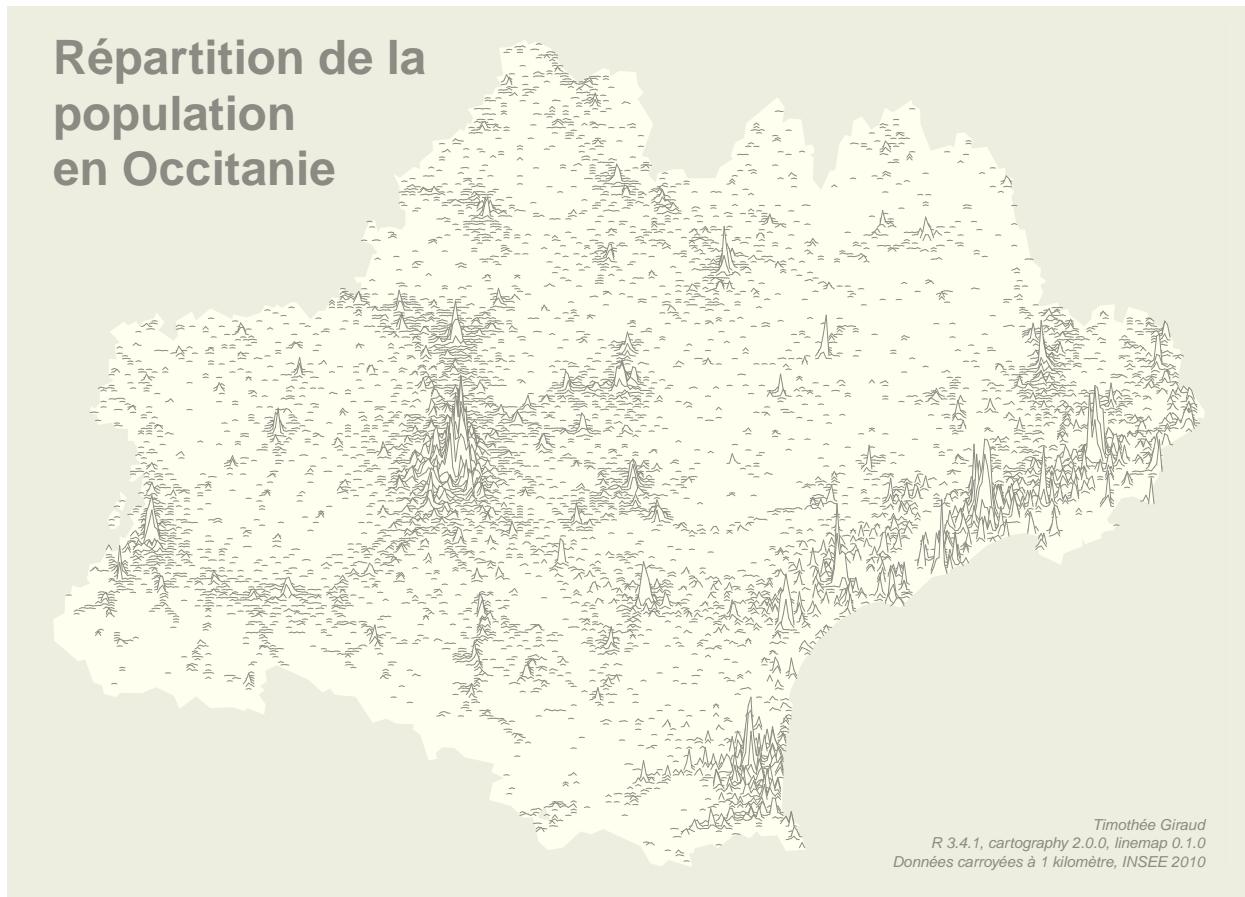
3.5.1 linemap

```
library(linemap)
library(sf)
data("popOcc")
data("occitanie")
opar <- par(mar=c(0,0,0,0), bg = "ivory2")
```

```

bb <- st_bbox(occitanie)
plot(st_geometry(occitanie), col="ivory1", border = NA)
linemap(x = popOcc, var = "pop", k = 2.5, threshold = 50,
        col = "ivory1", border = "ivory4", lwd = 0.6, add = TRUE)
text(x = bb[1], y = bb[4], adj = c(0,1),
     labels = "Répartition de la population en Occitanie",
     col = "ivory4", font = 2, cex = 1.8)
# add sources
mapsources <-"Timothée Giraud\nR 3.4.1, cartography 2.0.0, linemap 0.1.0\nDonnées carroyées à 1 kilomètre"
text(x = bb[3], y = bb[2], labels = mapsources,
     col = "ivory4", font = 3, adj = c(1,0), cex = 0.6 )

```



3.5.2 rayshader