Séminaire d'introduction à R

séance 7 - Cartographie

Joël Gombin

CURAPP - UPJV

20 avril 2012

Plan

Introduction : notions de base de cartographie

Carte raster

Carte composée de pixels : par exemple une photo satellite. L'information est représentée par la couleur du pixel, lui-même porteur d'une information géographique (coordonnées `x` et `y`).

Carte vectorielle

Carte composée d'éléments : points, lignes, courbes, polygones... Ces éléments constituent en eux-même une information géographique (ils ont des coordonnées), à laquelle on peut ajouter une information supplémentaire (par exemple en coloriant un polygone).

Notions de base de cartographie

Si on s'en tient aux cartes vectorielles, seules traitées ici, pour faire une carte il faut :

- un fond de carte (dans un format vectoriel : une simple image ne suffit pas) (généralement associé à une projection, permettant le passage d'une sphère à un plan. E.g. : Mercator).
- des données se rapportant aux objets de cette carte
- un identifiant commun à ces deux ensembles.

Plan

La cartographie dans R

Dans R, les cartes sont des objets dotés d'une structure complexe. Il existe plusieurs approches, mais la plus répandue consiste à utiliser la classe d'objets `SpatialPolygonsDataFrame`. C'est le package `sp` qui définit cette classe, qui contient aussi bien des données géographiques (coordonnées des éléments) que des données sur ces objets.
Une bonne pratique consiste à séparer les donnnées portant sur les éléments (coordonnées mais aussi identifiants, par exemple) et celles qu'on souhaite cartographier.
Par ailleurs, le package `ggplot2` propose également une fonction pour représenter des cartes raster ou choroplèthes, mais il utilise un autre format de données (celui du package `maps`). Il semble toutefois que le choix de la projection soit plus aisé.

Les outils du package `rgrs`

Dans le cadre de cette introduction, je vais présenter essentiellement les outils du package `rgrs` développé par Julien Barnier. Ils sont à la fois faciles à utiliser et relativement puissants. Une fois la logique assimilée, il sera aisé d'aller plus loin si nécessaire.

```
library(rgrs)
library(sp)
```

Plan

Importer des données géographiques

Il existe de nombreux formats de données cartographiques, mais le plus répandu est le `shapefile`. Une carte est contenue dans un ensemble de fichiers portant tous le même nom mais dotés d'extensions différentes (`.shp`, `.dbf`, `.prj`...).

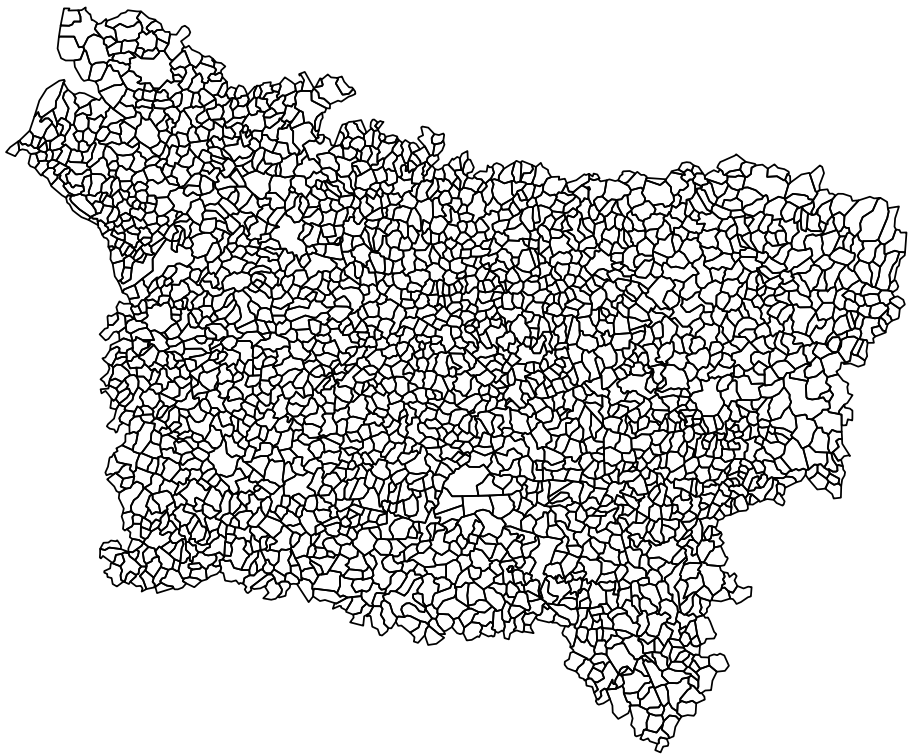
```
library(rgdal)


## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 1.7.3, released 2010/11/10 Path to GDAL shared
## files: /usr/share/gdal/1.7 Loaded PROJ.4 runtime: Rel. 4.7.1, 23 September
## 2009, [PJ_VERSION: 470] Path to PROJ.4 shared files: (autodetected)


picardie <- readOGR("/media/Data/Dropbox/Thèse/séminaire R/séance 7",
    "picardie")


## OGR data source with driver: ESRI Shapefile
## Source: "/media/Data/Dropbox/Thèse/séminaire R/séance 7", layer: "picardie"
## with 2291 features and 18 fields
## Feature type: wkbPolygon with 2 dimensions


picardie$STATUT <- as.factor(iconv(picardie$STATUT, from = "latin1",
    to = "utf8"))
picardie$codeINSEE <- as.character(picardie$INSEE_COM)
```

Plan

Avoir des informations sur une carte et la représenter

```
summary(picardie)
```

```
require(sp)
plot(picardie)
```



Manipuler une carte

Une carte est donc un objet de type `SpatialPolygonsDataFrame`. Il contient plusieurs ensemble de données ; généralement, le seul qu'on manipule directement est le slot `@data`, qui contient les données sur les objets géographiques.

```
names(picardie@data)
```

```
##  [1] "ID_GEOFLA"  "CODE_COMM"  "INSEE_COM"  "NOM_COMM"   "STATUT"
##  [6] "X_CHF_LIEU" "Y_CHF_LIEU" "X_CENTROID" "Y_CENTROID" "Z_MOYEN"
## [11] "SUPERFICIE" "POPULATION" "CODE_CANT"  "CODE_ARR"   "CODE_DEPT"
## [16] "NOM_DEPT"   "CODE_REG"   "NOM_REGION" "codeINSEE"
```

```
summary(picardie@data$NOM_DEPT)
```

```
## AISNE  OISE SOMME
##   816   693   782
```

Manipuler une carte

De plus, on manipule une carte comme n'importe quel dataframe :

```
somme <- picardie[picardie@data$CODE_DEPT == "80", ]
```

Le package sp offre de plus des outils pour facilement manipuler plusieurs couches cartographiques (par exemple étudier l'intersection de plusieurs ensembles d'éléments géographiques).
Enfin, on sauve et charge cet objet comme n'importe quel autre :

```
save(picardie, file = "picardie.Rdata")
load("picardie.Rdata")
```

Agréger des éléments cartographiques

On a parfois besoin de regrouper entre eux des polygones. Par exemple, de ma carte communale, je veux tirer une carte des départements.

```
require(gpclib)
```

```
## Loading required package: gpclib
```

```
## General Polygon Clipper Library for R (version 1.5-1) Type 'class ?
## gpc.poly' for help
```

```
gpclibPermit()
```

```
## Error: impossible de trouver la fonction "gpclibPermit"
```

```
require(maptools)
```

```
## Loading required package: maptools
```

```
## Loading required package: foreign
```

```
## Loading required package: lattice
```

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

```
dpts <- unionSpatialPolygons(picardie, IDs = picardie@data$CODE_DEPT)
```

```
## Loading required package: rgeos
```

```
## Loading required package: stringr
```

Plan

Représenter des proportions : les cartes choroplèthes

L'intérêt d'une carte n'est généralement pas de simplement représenter des objets géographiques mais d'y associer de l'information. Il y a de multiples manières de le faire ; on se concentrera ici sur les cartes choroplèthes, dans lesquelles les polygones sont coloriées en fonction de la variable d'intérêt.
On commence par vérifier qu'on a bien un identidiant commun à nos données et à notre carte :

```
load("/media/Data/Dropbox/Thèse/séminaire R/séance 1/mini_picardie.Rdata")
head(mini_picardie$CODE_COMMU)
```

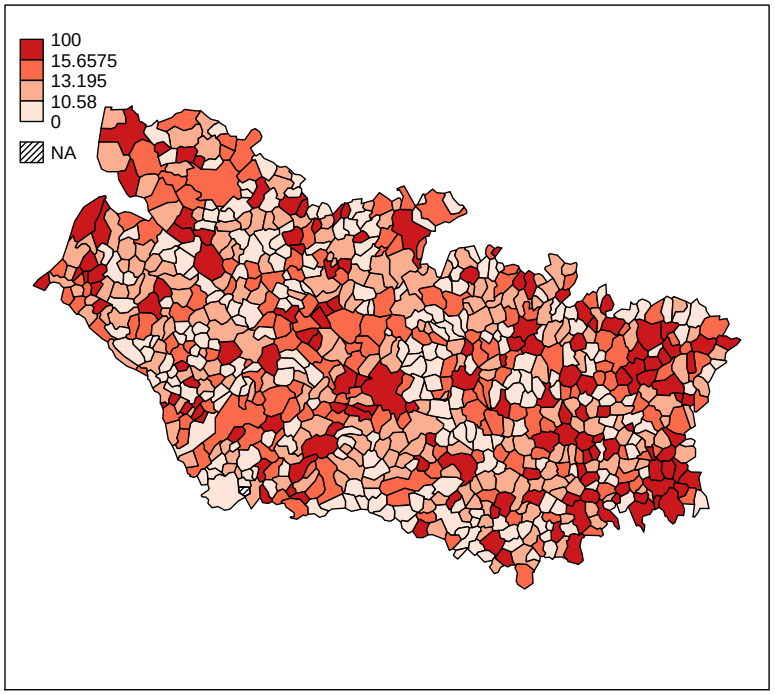
```
## [1] 001 002 003 004 005 006
## 908 Levels:     001 002 003 004 005 006 007 008 009 010 011 012 013 ... 909
```

```
# Non, donc on recode :
mini_picardie$codeINSEE <- paste(as.character(mini_picardie$CODE_D_PAR),
    as.character(mini_picardie$CODE_COMMU), sep = "")
```

Représenter des proportions : les cartes choroplèthes

Par facilité, on ne va travailler que sur la Somme. On utilise la fonction `carte.prop` du package `rgrs` :

```
somme <- picardie[picardie@data$CODE_DEPT == "80", ]
carte.prop(somme, mini_picardie, varname = "AbsIns", sp.key = "codeINSEE",
    data.key = "codeINSEE", at = quantile(mini_picardie$AbsIns, c(0, 0.25, 0.5,
        0.75, 1)))
```



Paramétrer une carte choroplèthe

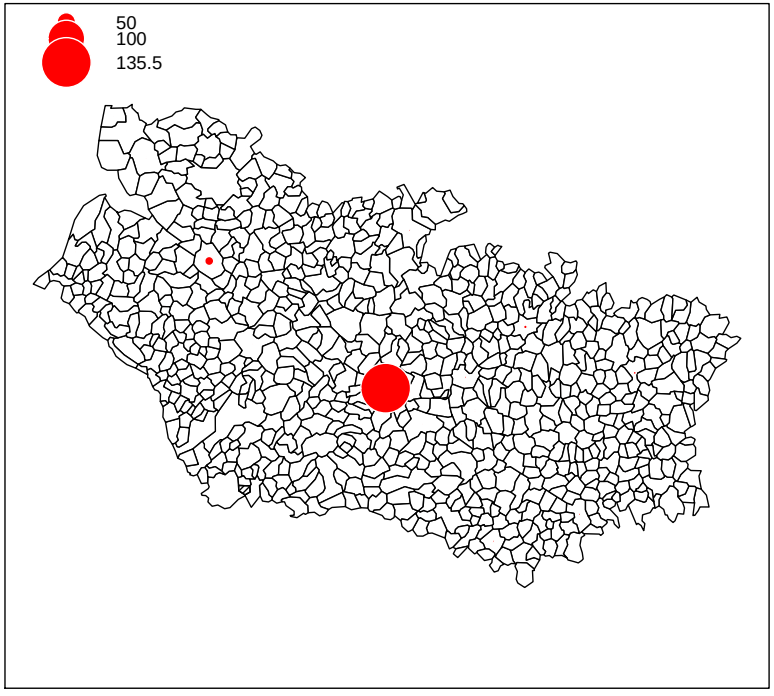
La fonction `carte.prop` prend une série d'arguments, que vous pouvez tester.
Par ailleurs, pour le choix des couleurs, on peut s'aider du site `http://www.colorbrewer2.org`. Les arguments `palette.pos` et `palette.neg` de `carte.prop` prennent les noms proposés sur le site.

Plan

Représenter des effectifs

Admettons que nous voulions représenter les effectifs des communes de la Somme.

```
carte.eff(somme, mini_picardie, varname = "Population", sp.key = "codeINSEE",
    data.key = "codeINSEE")
```

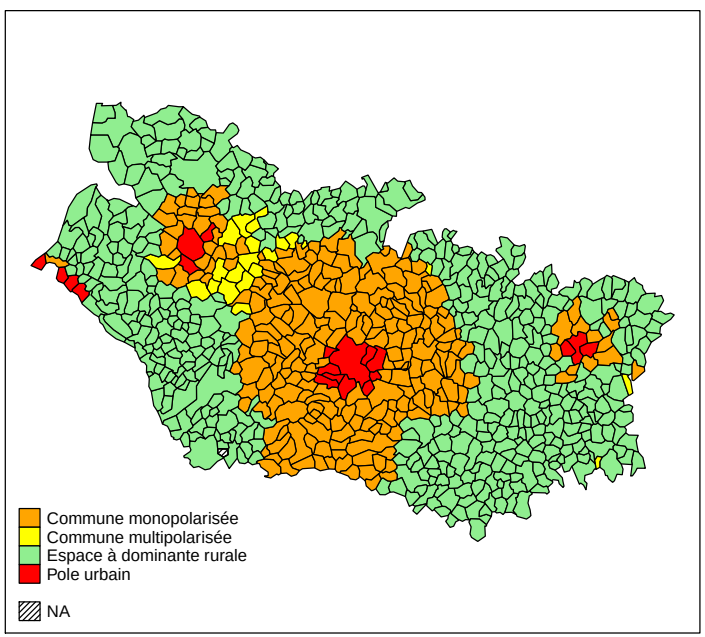


Plan

Représenter une variable catégorielle

La syntaxe est toujours très proche, on utilise cette fois-ci la fonction `carte.qual` :

```
pal <- c("orange", "yellow", "light green", "red")
carte.qual(somme, mini_picardie, varname = "type_urbain", sp.key = "codeINSEE",
    data.key = "codeINSEE", palette = pal, posleg = "bottomleft", main = "Le découpage en ZAUER en Picardie",
    sub = "source : INSEE.")
```

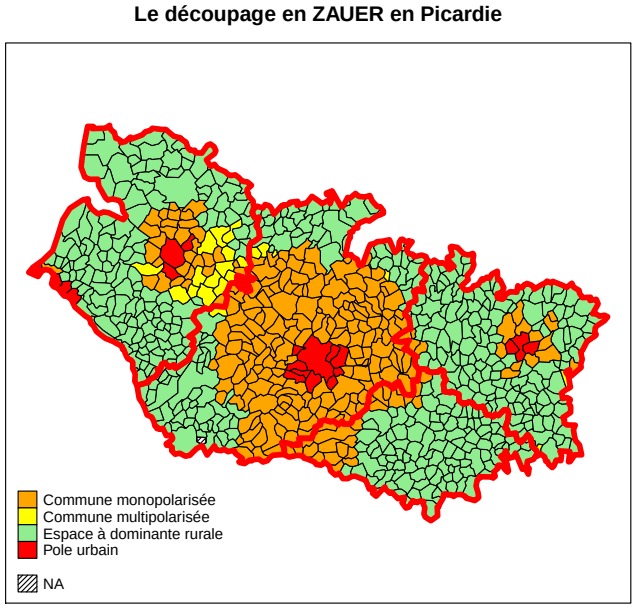


Plan

Rajouter des bordures

Imaginons que nous voulions rajouter les découpages des arrondissements. Il nous faut d'abord créer des éléments correspondants, et ensuite les ajouter.

```
arrdts <- unionSpatialPolygons(somme, IDs = somme@data$CODE_ARR)
carte.qual(somme, mini_picardie, varname = "type_urbain", sp.key = "codeINSEE",
    data.key = "codeINSEE", palette = pal, posleg = "bottomleft", main = "Le découpage en ZAUER en Picardie",
    sub = "source : INSEE.")
plot(arrdts, lwd = 5, border = "red", add = TRUE)
```

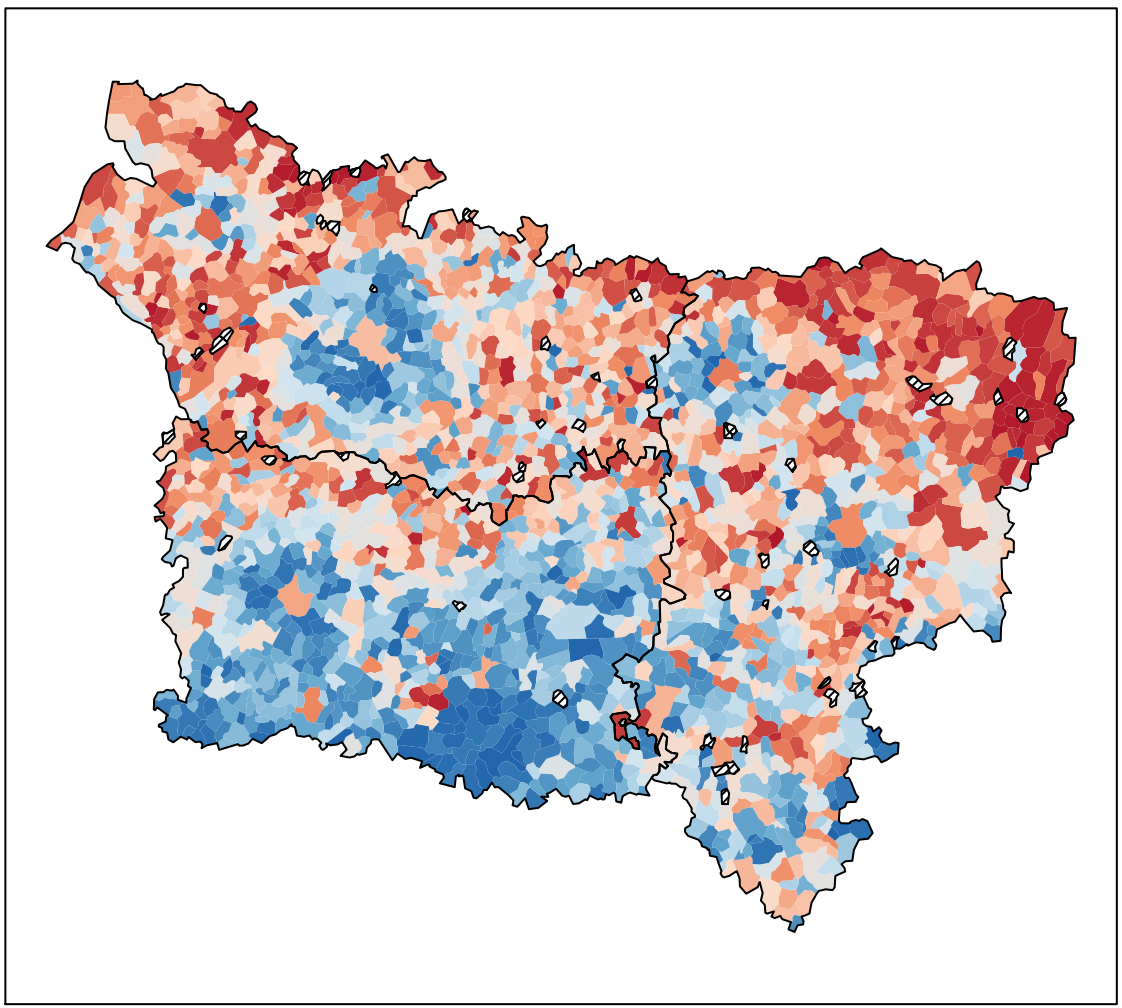


Faire une carte en coloration continue

Le "truc" consiste à passer autant de breaks, et de couleurs, à la fonction que de valeurs différentes de la variable continue représentée. Pour cela on choisit les couleurs qu'on veut pour sa palette (palette séquentielle ou divergente), puis on crée une palette de `n` valeurs à partir d'un certain nombre de points. On passe `pal(n)` en argument de `carte.prop`.

```
library(RColorBrewer)
colors <- brewer.pal(6, "RdBu")
pal <- colorRampPalette(colors)
carte.prop(picardie, mini_picardie, "revenu.fiscal.moyen", sp.key = "codeINSEE",
    data.key = "codeINSEE", at = as.integer(levels(as.factor(mini_picardie$revenu.fiscal.moyen))),
    border = "transparent", palette = pal(length(levels(as.factor(mini_picardie$revenu.fiscal.moyen)))),
    posleg = "none", main = "Le revenu fiscal moyen des ménages par communes")
plot(dpts, add = T)
```

Faire une carte en coloration continue



Conclusion

La cartographie dans R est assez simple, en tout cas pour des tâches simples, et surtout offre à la fois beaucoup de souplesse tout en utilisant une syntaxe commune avec la manipulation de données.
De plus, en scriptant, on peut produire très rapidement de très nombreuses cartes, soit de différents territoires, soit de différentes variables.
Pour la route, le blog d'un géographe anglais produisant de magnifiques cartes sous R : http://spatialanalysis.co.uk/.