Fiche TD avec le logiciel R : PointsGPS

Comment poser des points GPS sur une carte ?

Jean R. Lobry & Samuel Venner & Marie-Claude Venner

Table des matières

1 Introduction du cas pratique

Le cas pratique que nous allons utiliser ici est le suivant : pour construire le diaporama d'une présentation orale[1], nous avons besoin de produire une série de cartes représentant la géolocalisation de sites de suivi de la fructification de chênes et de hêtres en France métropolitaine. On veut afficher les points progressivement en suivant les modalités d'une variable qualitative indicatrice des partenaires en charge de la gestion des sites.

2 Installation des paquets nécessaires

2.1 Mac et Windows

Nous allons utiliser ici principalement les paquets R de cartographie `terra` [5] et `geodata` [6]. Le paquet `geodata` dépend du paquet `terra` qui lui-même dépend de ressources externes à R dont `GDAL`[2], `GEOS`[3] et `PROJ`[4]. Sous Windows et macOS les paquets `terra` et `geodata` sont disponibles sur les miroirs du CRAN sous la forme de fichiers exécutables avec en principe tout ce qu'il faut pour fonctionner après une installation standard.

2.2 Ubuntu

Sous Linux les paquets sont disponibles sous la forme de fichiers source et il va falloir installer les ressources externes. Les instructions les plus à jour d'installation sont sur la première page du site github de `terra`[5]. Pour Unbuntu la recommandation est d'utiliser les fichiers exécutables fournis par `r2u`[6]. Les étapes à suivre sont les suivantes :

[1] Démarrage du PEPR `REGE-ADAPT` : *Forest regeneration and the adaptation of forest socio-ecosystems to climate change*
[2] `https://gdal.org/`
[3] `https://libgeos.org/`
[4] `https://proj.org/`
[5] `https://github.com/rspatial/terra`
[6] `https://eddelbuettel.github.io/r2u/`

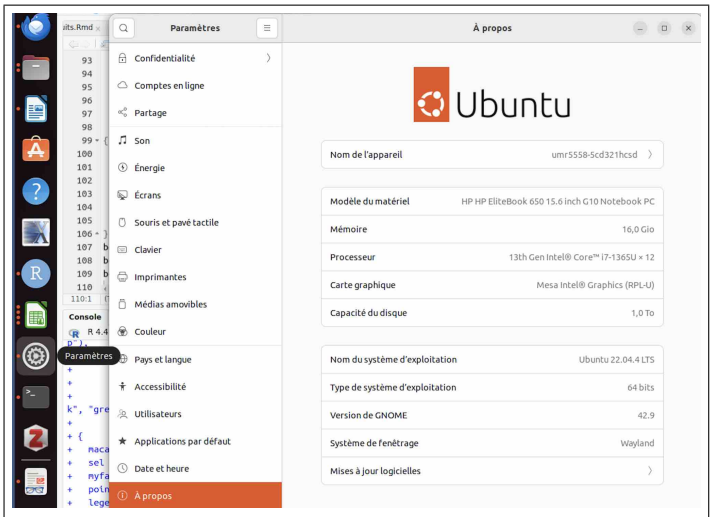


Figure 1 : Détermination de la version d'Ubuntu utilisée.

1° Commencez par déterminer quelle version d'Ubuntu tourne sur votre ordinateur : dans « Paramètres » sélectionnez l'onglet « À propos » comme dans la figure 1 page 3.

2° Dans un navigateur, allez sur le site de `r2u` dans la section « Usage and Setup » à `https://eddelbuettel.github.io/r2u/#usage-and-setup`. Cliquez sur le lien correspondant à votre version d'Unbuntu, par exemple `add_cranapt_jammy.sh` pour Ubuntu 22.04.

3° Dans la nouvelle fenêtre, cliquez sur le bouton permettant de télécharger le fichier (voir la figure 2 page 4).

4° Ouvrez une fenêtre de terminal (figure 3 page 4) et placez-vous dans le dossier où le fichier a été téléchargé avec :

```
cd ~/Téléchargements
```

5° Lancez l'exécution du script avec :

```
sudo bash add_cranapt_jammy.sh
```

6° Lancez R (ou `RStudio`) et installez les paquets `terra` et `geodata` comme vous avez l'habitude de le faire.

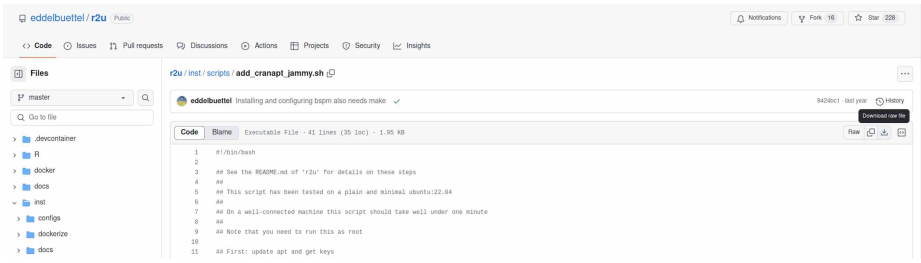


Figure 2 : Le bouton permettant de télécharger le fichier du script est en haut à droite.

Figure 3 : Les commnades pour exécuter le script dans un terminal.

Figure 4 : Les données du fond de carte sont stockées dans un fichier au format `XDR` [10] dans le dossier `gadm`.

3 Obtenir un fond de carte

3.1 Récupération des données

Le paquet `geodata` [6] permet de récupérer facilement un fond de carte plus ou moins détaillé selon la valeur de l'argument `level` (figure 5 page 9). On utilise ici le niveau des régions parce que l'on va vouloir exclure la Corse afin de gagner de la place pour la représentation graphique étant donné que nous n'avons pas de points sur l'Île-de-Beauté.

```
library(geodata)
fra <- gadm(country = "FRA", level = 1, path = ".")
```

L'argument `country` est le nom du pays dont on souhaite récupérer les données ou en abrégé son cade ISO à trois caractères. La fonction `country_codes()` permet d'accéder facilement à ces informations :

```
country_codes(query = "France")[ , c(1:2, 6)]
```

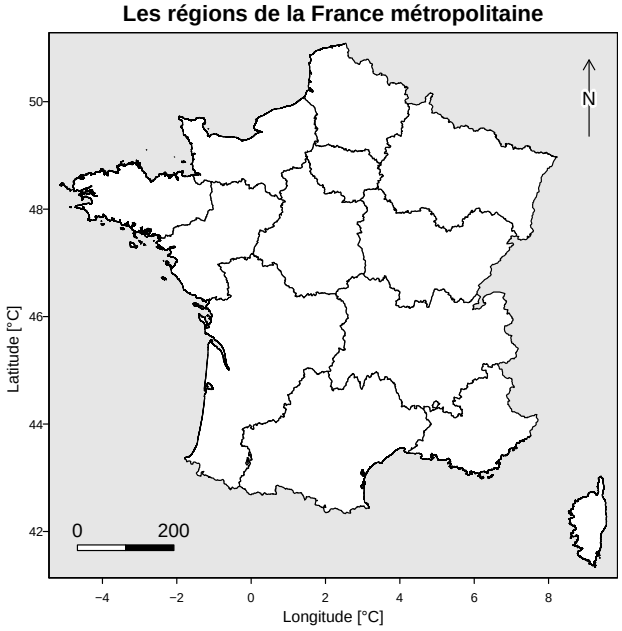
	NAME	ISO3	NAME_LOCAL
51	Clipperton Island	XCL	Clipperton Island
79	France	FRA	France
80	French Guiana	GUF	Guyane
81	French Polynesia	PYF	Polynésie Française
82	French Southern Territories	ATF	Terres Australes et Antarctiques Françaises
92	Guadeloupe	GLP	Guadeloupe
141	Martinique	MTQ	Martinique
144	Mayotte	MYT	Mayotte
159	New Caledonia	NCL	Nouvelle-Calédonie
186	Réunion	REU	Réunion|Ile de la Réunion
190	Saint-Barthélemy	BLM	Saint-Barthélemy
191	Saint-Martin	MAF	Saint-Martin
195	Saint Pierre and Miquelon	SPM	Saint-Pierre et Miquelon
252	Wallis and Futuna	WLF	Wallis et Futuna

L'argument `path` de la fonction sert à indiquer où l'on souhaite stocker les données sur disque. Par exemple dans notre cas c'est dans le répertoire courant, et la fonction va créer un dossier `gadm` contenant les données téléchargées (figure 4 page 5) dans un fichier au format `XDR` [10], donc très rapide à lire sous R. Si les données sont déjà présentes sur le disque, la fonction `gadm()` ne cherchera pas à les importer.

```
fra
class       : SpatVector
geometry    : polygons
dimensions  : 13, 11  (geometries, attributes)
extent      : -5.143751, 9.560416, 41.33375, 51.0894  (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat WGS 84 (EPSG:4326)
names       :   GID_1 GID_0 COUNTRY          NAME_1 VARNAME_1 NL_NAME_1 TYPE_1
type        :   <chr> <chr>   <chr>           <chr>     <chr>     <chr>  <chr>
values      : FRA.1_1   FRA  France Auvergne-Rhône~        NA        NA Région
              FRA.2_1   FRA  France Bourgogne-Fran~        NA        NA Région
              FRA.3_1   FRA  France        Bretagne        NA        NA Région
ENGTYPE_1  CC_1 HASC_1  ISO_1
    <chr> <chr>  <chr>  <chr>
   Region    NA  FR.AR     NA
   Region    NA  FR.BF     NA
   Region    NA  FR.BT FR-BRE
```

L'objet `fra` est un objet de type `SpatVector` qui définit les 13 polygones correspondants aux régions en utilisant des coordonnées GPS (`EPSG:4326`). La fonction `plot()` du paquet `terra` [5] nous permet de représenter facilement ces données. On complète en ajoutant une barre d'échelle et une flèche vers le nord.

```
library(terra)
macarte <- function(sv, ...){
  plot(sv, las = 1, xlab = "Longitude [°C]", ylab = "Latitude [°C]",
       background = grey(0.9), col = "white", ...)
  sbar(200, type = "bar")
  north()
}
macarte(fra, main = "Les régions de la France métropolitaine")
```



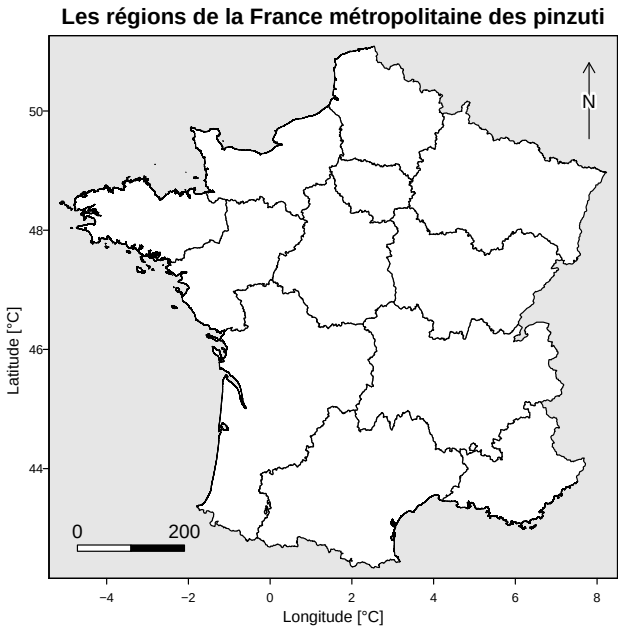
3.2 Sélection des données

Comme il n'y a pas de site à représenter en Corse, on décide de l'enlever pour gagner de la place pour la représentation graphique. Notez que cette opération avec les objets de type `SpatVector` se fait exactement avec la même syntaxe que nous aurions utilisée avec des objets de type `data.frame`.

```
head(fra[ , 1:5])
```

```
   GID_1 GID_0 COUNTRY                  NAME_1 VARNAME_1
1 FRA.1_1   FRA  France    Auvergne-Rhône-Alpes      <NA>
2 FRA.2_1   FRA  France Bourgogne-Franche-Comté      <NA>
3 FRA.3_1   FRA  France               Bretagne      <NA>
4 FRA.4_1   FRA  France     Centre-Val de Loire      <NA>
5 FRA.5_1   FRA  France                  Corse   Corsica
6 FRA.6_1   FRA  France            Grand Est      <NA>
```

```
frasc <- fra[fra$NAME_1 != "Corse", ]
macarte(frasc, main = "Les régions de la France métropolitaine des pinzuti")
```

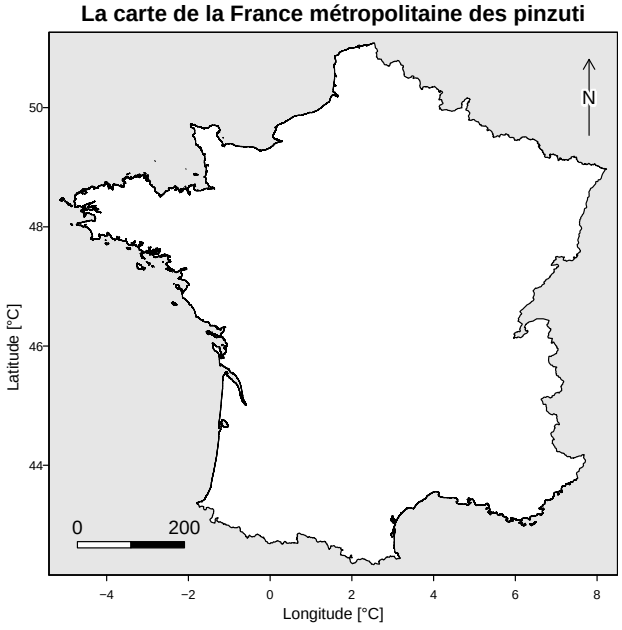


La représentation des régions ne nous intéresse plus, on agrège tous les polygones des régions en une seule entité avec la fonction `aggregate()`.

```
fdchr <- aggregate(frasc)
nrow(geom(fdchr))
```

```
[1] 189277
```

```
macarte(fdchr, main = "La carte de la France métropolitaine des pinzuti")
```



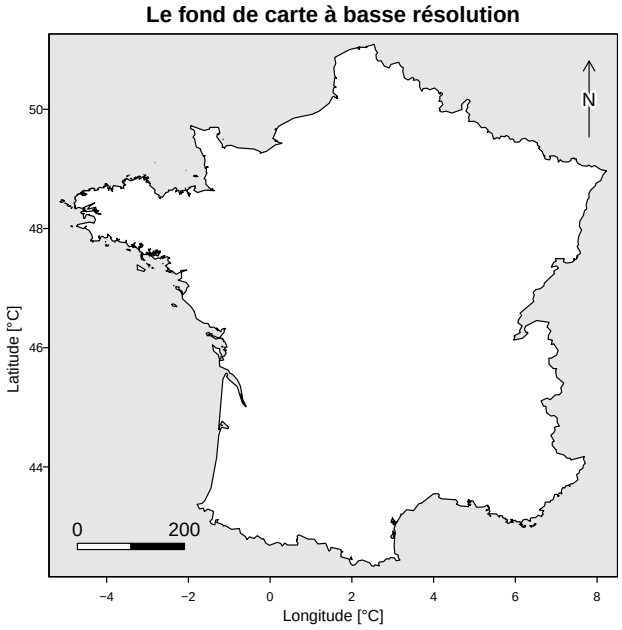
3.3 Ajustement de la résolution

Comme nous visons ici à produire une illustration pour une présentation orale, la résolution de notre fond de carte est inutilement élevée (189277 points !). On simplifie en demandant à ce qu'il y ait au moins une seconde d'arc entre les points avec la fonction `simplifyGeom()`.

```
fdc <- simplifyGeom(fdchr, 1/60)
nrow(geom(fdc))
```

```
[1] 2643
```

```
macarte(fdc, main = "Le fond de carte à basse résolution")
```



Il n'y a plus que 2643 points, c'est bien plus raisonnable, notre fond de carte est prêt. Il ne reste plus qu'à poser des points d'intérêt dessus.

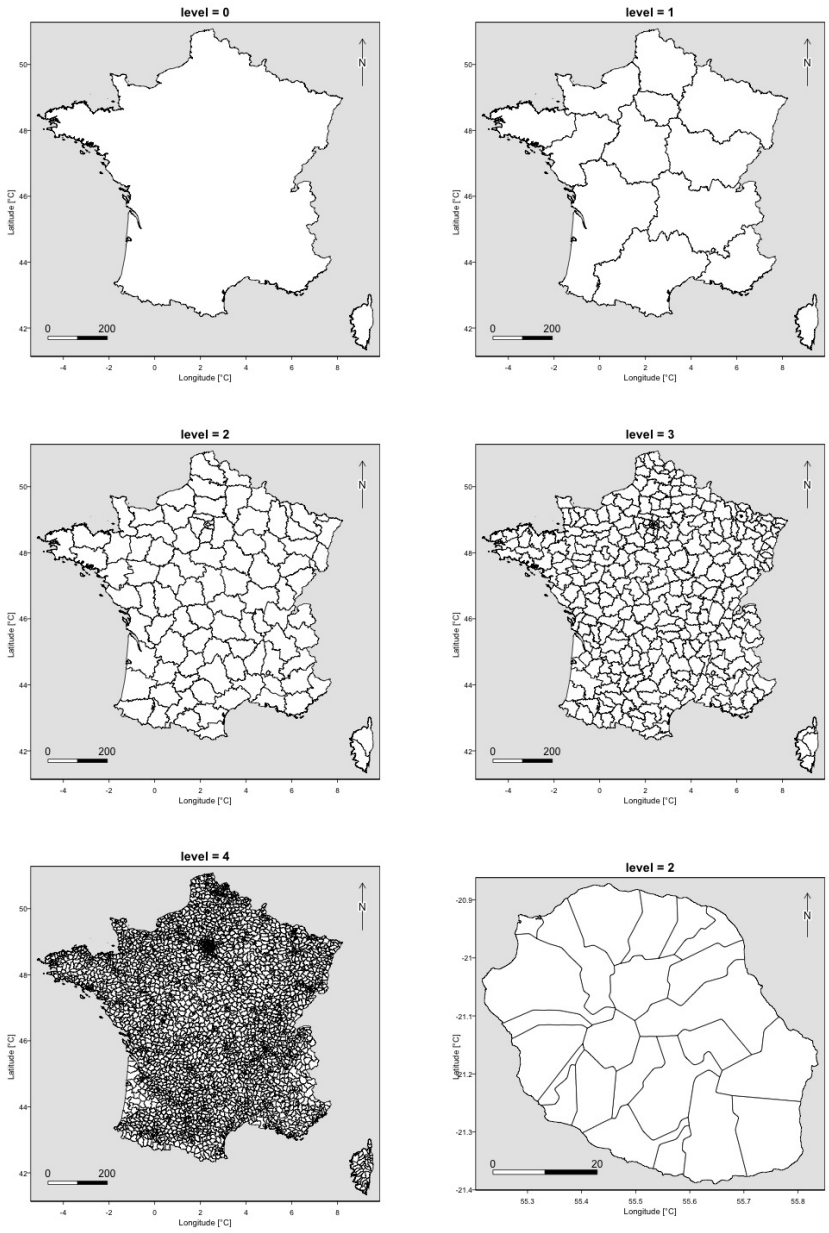


Figure 5 : La fonction `gadm()` du paquet `geodata` [6] permet d'obtenir les contours administratifs avec plus ou moins de résolution : le pays, les régions, les départements, les cantons et les communes. Les données sont issues du site GADM (`https://gadm.org/`). En bas à droite, la carte de l'île de la Réunion.

4 Poser des points

4.1 Import et représentation

Les coordonnées GPS des points d'intérêt sont disponibles dans un fichier de type tableur (figure 6 page 11). On commence par récupérer ce fichier en local :

```
ficname <- "PointSuiviFructification.ods"

chmin <- "https://pbil.univ-lyon1.fr/R/donnees/PointsGPS/"
path <- paste0(chmin, ficname)
download.file(path, destfile = ficname)
```

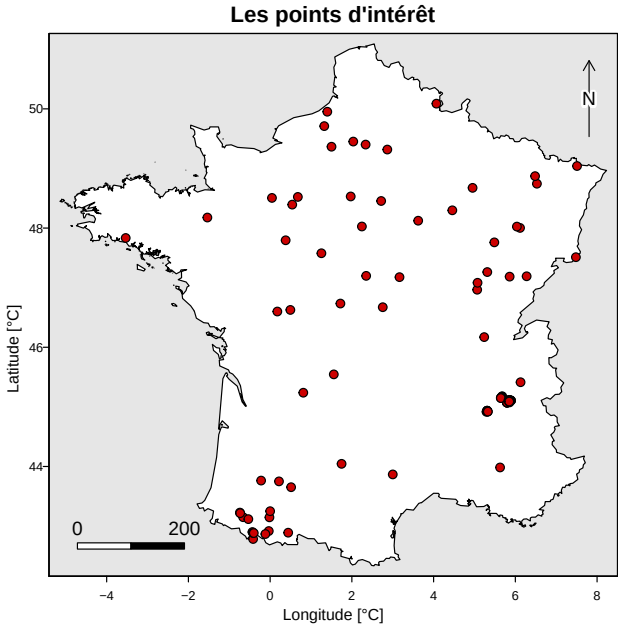
Nous importons ensuite les données du fichier dans R avec la fonction `read_ods()` du paquet `readODS` [9] :

```
library(readODS)
dta <- read_ods(ficname, as_tibble = FALSE, strings_as_factors = TRUE)
head(dta)
```

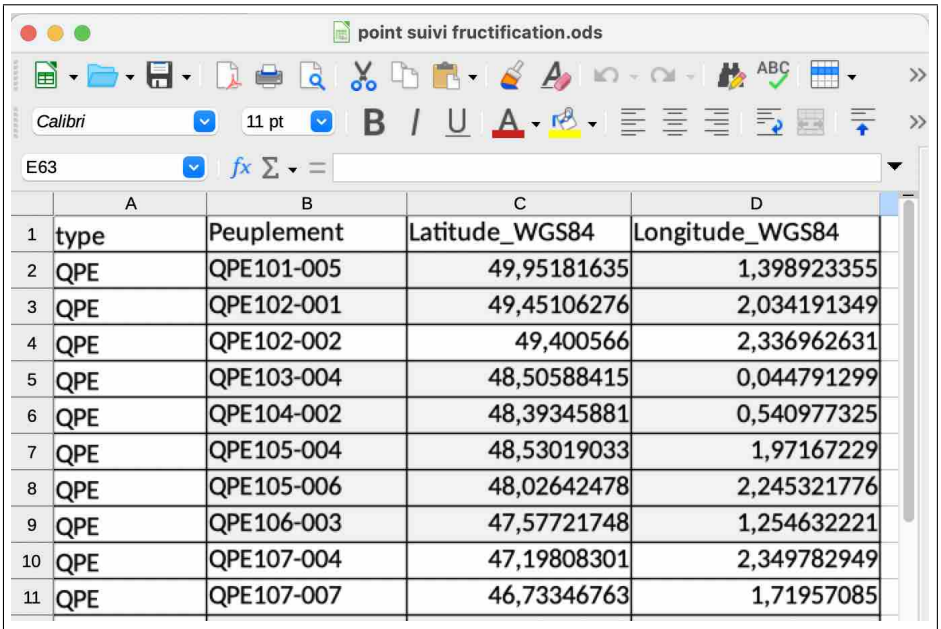
```
  type Peuplement Latitude_WGS84 Longitude_WGS84
1  QPE QPE101-005       49.95182       1.3989234
2  QPE QPE102-001       49.45106       2.0341913
3  QPE QPE102-002       49.40057       2.3369626
4  QPE QPE103-004       48.50588       0.0447913
5  QPE QPE104-002       48.39346       0.5409773
6  QPE QPE105-004       48.53019       1.9716723
```

On récupère les coordonnées des points en précisant bien que ce sont des coordonnées GPS avec `EPSG:4326`. Il suffit de les rajouter à notre fond de carte, et le tour est joué.

```
pts <- with(dta, vect(cbind(Longitude_WGS84, Latitude_WGS84), crs = "EPSG:4326"))
macarte(fdc, main = "Les points d'intérêt")
points(pts, pch = 21, cex = 1, bg = "red3")
```



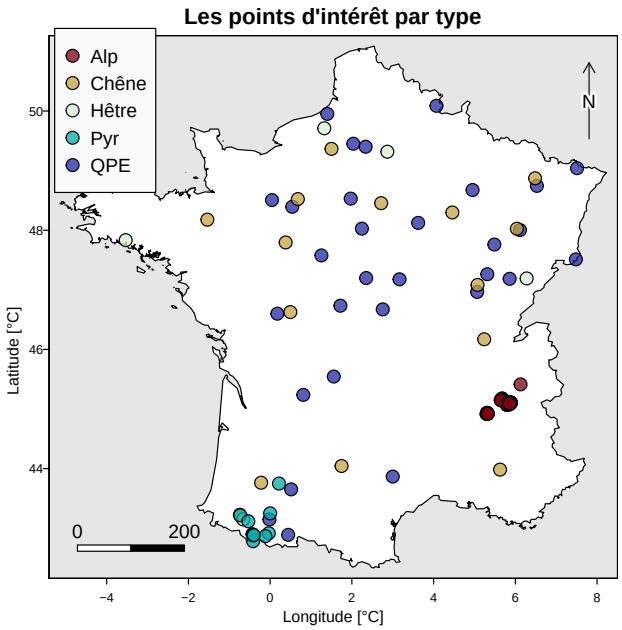
On peut raffiner en utilisant des couleurs différentes selon les types.



	A	B	C	D
1	type	Peuplement	Latitude_WGS84	Longitude_WGS84
2	QPE	QPE101-005	49,95181635	1,398923355
3	QPE	QPE102-001	49,45106276	2,034191349
4	QPE	QPE102-002	49,400566	2,336962631
5	QPE	QPE103-004	48,50588415	0,044791299
6	QPE	QPE104-002	48,39345881	0,540977325
7	QPE	QPE105-004	48,53019033	1,97167229
8	QPE	QPE105-006	48,02642478	2,245321776
9	QPE	QPE106-003	47,57721748	1,254632221
10	QPE	QPE107-004	47,19808301	2,349782949
11	QPE	QPE107-007	46,73346763	1,71957085

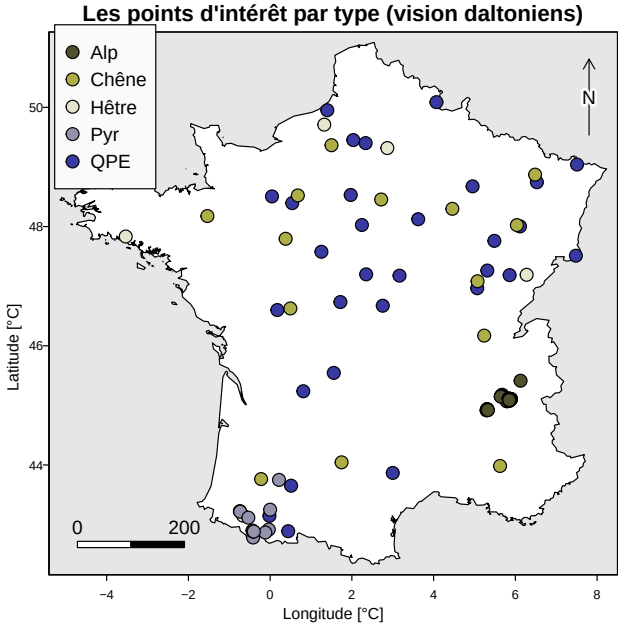
FIGURE 6 : Les coordonnées GPS des points d'intérêt dans un tableur LibreOffice.

```
types <- levels(dta$type)
mypal <- hcl.colors(length(types), "Roma", alpha = 0.75)
mycols <- mypal[as.integer(dta$type)]
macarte(fdc, main = "Les points d'intérêt par type")
points(pts, pch = 21, cex = 1.5, bg = mycols)
legend("topleft", inset = 0.01, legend = types, pch = 21, pt.bg = mypal,
       xpd = NA, bg = grey(0.98), pt.cex = 1.5)
```



On teste si les couleurs utilisées sont bien discriminées par les daltoniens.

```
library(dichromat)
macarte(fdc, main = "Les points d'intérêt par type (vision daltoniens)")
points(pts, pch = 21, cex = 1.5, bg = dichromat(mycols))
legend("topleft", inset = 0.01, legend = types, pch = 21, pt.bg = dichromat(mypal),
       xpd = NA, bg = grey(0.98), pt.cex = 1.5)
```

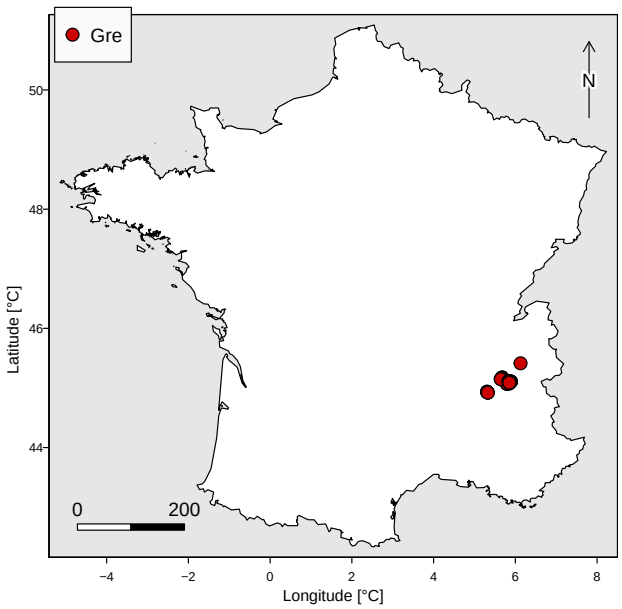


Avec cinq modalités, il n'est pas évident de trouver une palette qui soit discriminante par les daltoniens. On aura tout intérêt à moduler également la forme des points pour faciliter la lecture. C'est ce que nous allons faire dans la section suivante.

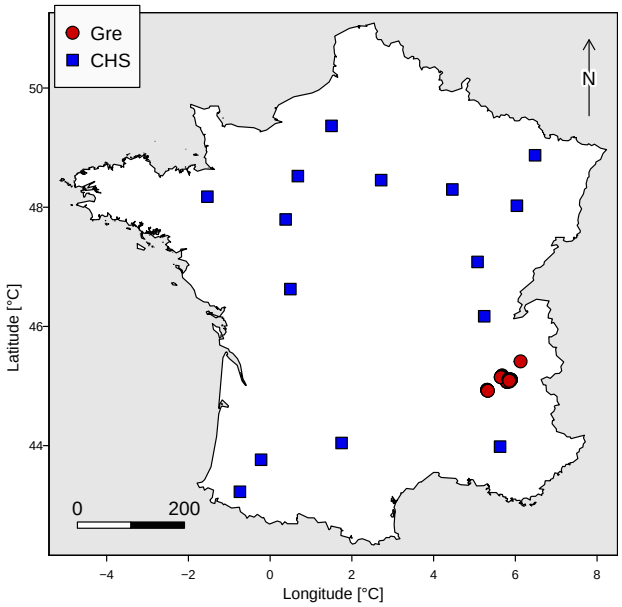
4.2 Apparition progressive de séries de points

On veut afficher les points progressivement en suivant les modalités du facteur `type` indicateur des partenaires en charge de la gestion des sites. On définit pour cela la fonction `progressif()` dont le premier paramètre, `n`, contrôle le nombre de modalités à afficher. Le paramètre `ordre` contrôle l'ordre dans lequel on veut faire apparaître les modalités, le paramètre `lab` les noms que l'on veut faire figurer dans la légende, le paramètre `mycol` la couleur des points à utiliser et le paramètre `mypch` la forme des points à utiliser.

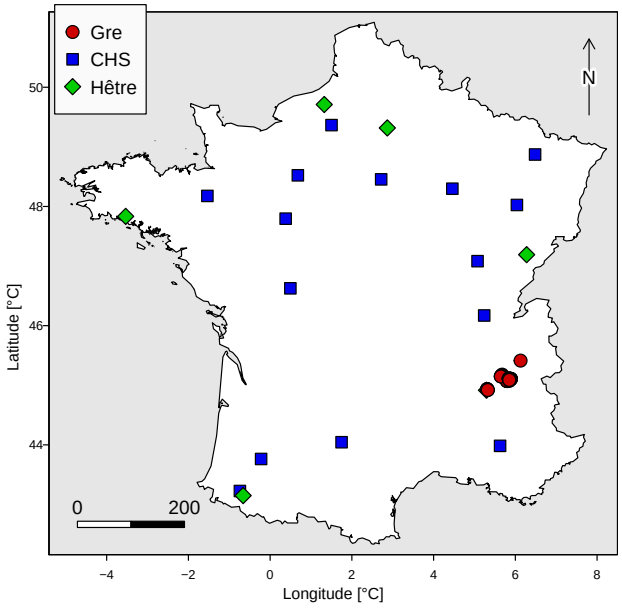
```
pts$type <- dta$type
progressif <- function(n = 1,
                       ordre = c("Alp", "Chêne", "Hêtre", "Pyr", "QPE"),
                       lab   = c("Gre", "CHS",   "Hêtre", "Pyr", "G&P"),
                       mycol = c("red3", "blue2", "green3", "yellow", "black"),
                       mypch = 21:25)
{
  macarte(fdc)
  sel <- pts[which(pts$type %in% ordre[1:n]), ]
  myfac <- factor(sel$type, levels = ordre[1:n])
  points(sel, pch = mypch[myfac], bg = mycol[myfac], cex = 1.5)
  legend("topleft", inset = 0.01, legend = lab[1:n], pch = mypch[1:n],
         pt.bg = mycol[1:n], xpd = NA, bg = grey(0.98), pt.cex = 1.5)
}
progressif(1)
```



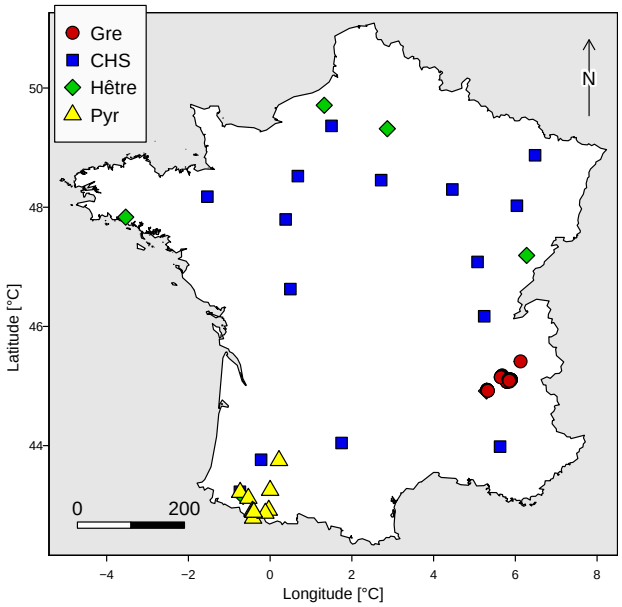

```
progressif(2)
```



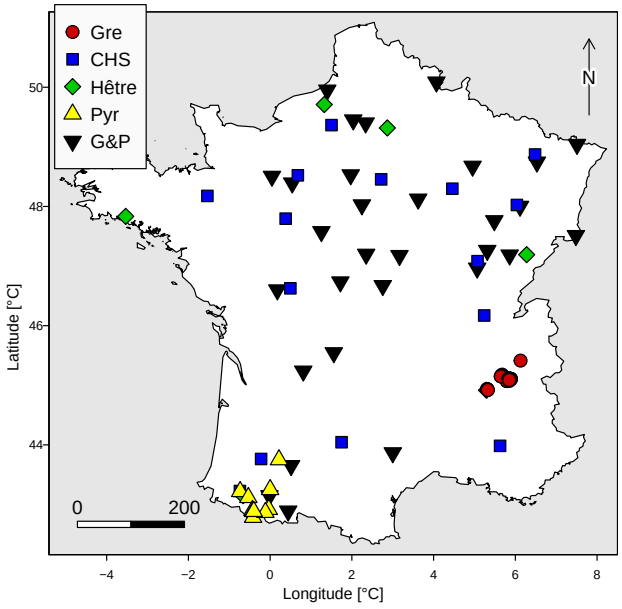

```
progressif(3)
```




```
progressif(4)
```




```
progressif(5)
```



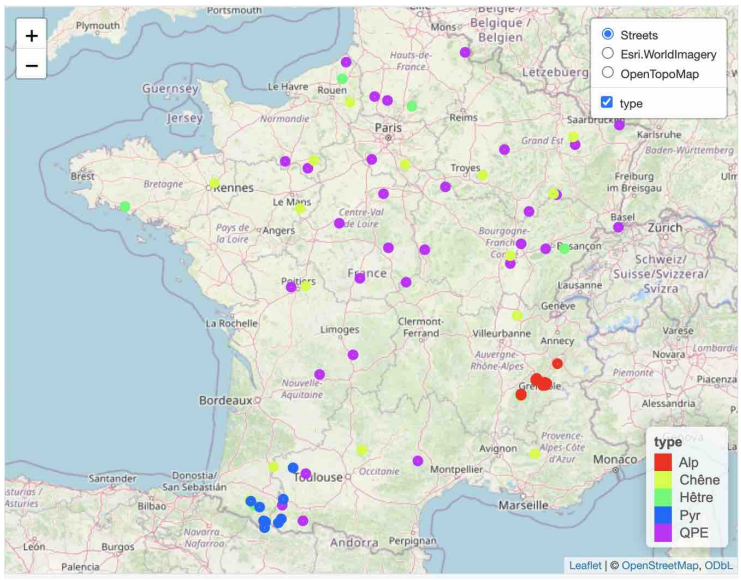


Figure 7 : Le stade initial de la carte interactive produite par la fonction `plet()` du paquet `terra` [5] et pris en charge par le paquet `leaflet` [3].

5 Annexes

5.1 Vivent les carte interactives !

Une carte interactive n'offre que peu d'intérêt dans le cadre d'un diaporama statique. Néanmoins, elle peut vous permettre de vérifier facilement que vos points sont correctement positionnés, avant le grand oral, tout en restant dans R. Ceci est possible grâce à la fonction `plet()` du paquet `terra` [5] qui va générer un objet pris en charge par le paquet `leaflet` [3]. Comme cela ne coûte *vraiment* pas beaucoup plus cher à ce stade, nous allons illustrer ce point. En entrant le *laconique* code ci-après dans la console R vous allez obtenir la carte interactive de la figure 7 page 16.

```
mamap <- plet(pts, "type", cex = 3)
library(leaflet)
mamap
```

À partir de là, vous êtes libres de musarder et gambader interactivement pour vérifier qu'il n'y a pas de *gag* dans la localisation de vos points. Par exemple dans la figure 8 page 17 nous avons vérifié qu'un point d'intérêt particulier était correctement porté sur la carte.

5.2 Sniff, je n'ai pas de degrés décimaux

Vous avez bien des coordonnées GPS mais celles-ci sont exprimées en degrés sexagésimaux au lieu de degrés décimaux, c'est à dire dans un format qui va typiquement ressembler à : `1°32'01" W`.

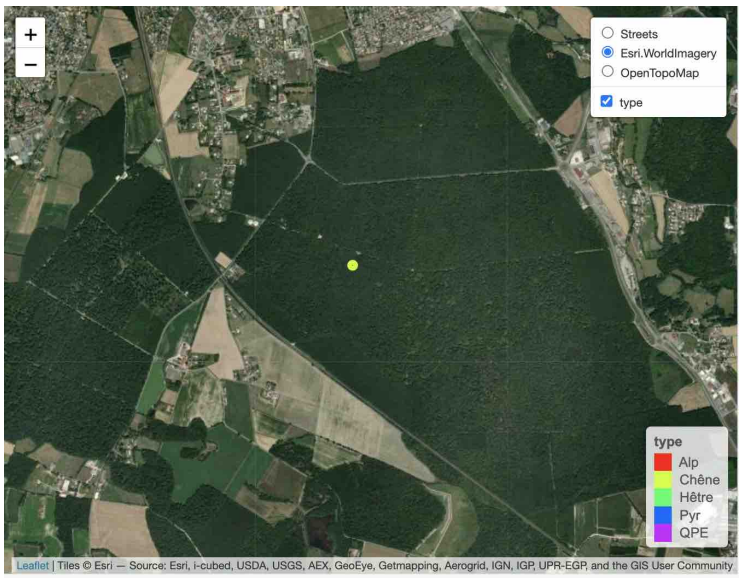


Figure 8 : Vérification de l'absence d'erreur cléricale pour un point d'intérêt particulier. En gambadant sur la carte de la figure 7 page 16 nous nous assurons que le point GPS pour le site `CHS01` dans la forêt domaniale de Seillon au sud de Bourg-en-Bresse dans le département de l'Ain est correctement positionné.

5.2.1 Solution manuelle

Si peu de points sont concernés, le plus simple est peut-être de faire la conversion directement « à la main » dans le tableur (voir l'exemple dans la figure 9 page 20). Par exemple dans le cas de `1°32'01" W` la conversion en degrés décimaux se calcule comme :

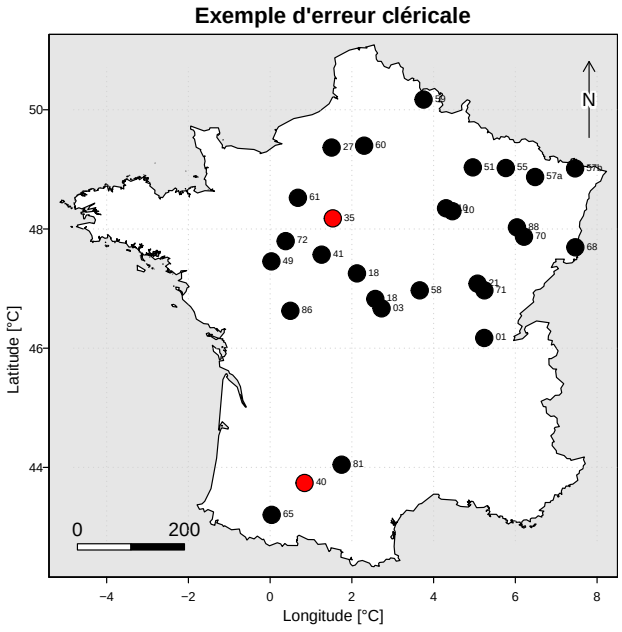
$$-1 \times (1 + \frac{32}{60} + \frac{1}{3600})$$

Soit dans R :

```
-1*(1 + 32/60 + 1/3600)
[1] -1.533611
```

Le problème avec cette approche est qu'il faut être extrêmement attentif pour ne pas commettre d'erreur cléricale. Par exemple, dans la première version de MASTREE+ [4], pour les séries temporelles de référence `RENECOFOR_2020`, les sites de chênaies `CHS 35` (Bretagne) et `CHP 40` (Landes) étaient du mauvais coté du méridien de Greenwich :

```
load(url("http://pbil.univ-lyon1.fr/R/donnees/mastree.Rda"))
#load("../../web/donnees/mastree.Rda")
ren <- subset(mastree, Reference == "RENECOFOR_2020" &
                  Species_code %in% c("QUEROB", "QUEPET"))
geoloc <- ren[!duplicated(ren$Site), ]
ptsREN <- with(geoloc, cbind(Longitude, Latitude), crs = "EPSG:4326")
macarte(fdc, main = "Exemple d'erreur cléricale")
add_grid()
mycol <- ifelse(geoloc$Site %in% c("CHS 35", "CHP 40"), "red", "black")
points(ptsREN, pch = 21, bg = mycol, cex = 2)
text(ptsREN, substr(geoloc$Site, 5, 7), pos = 4, cex = 0.5)
```



5.2.2 Solution automatisée

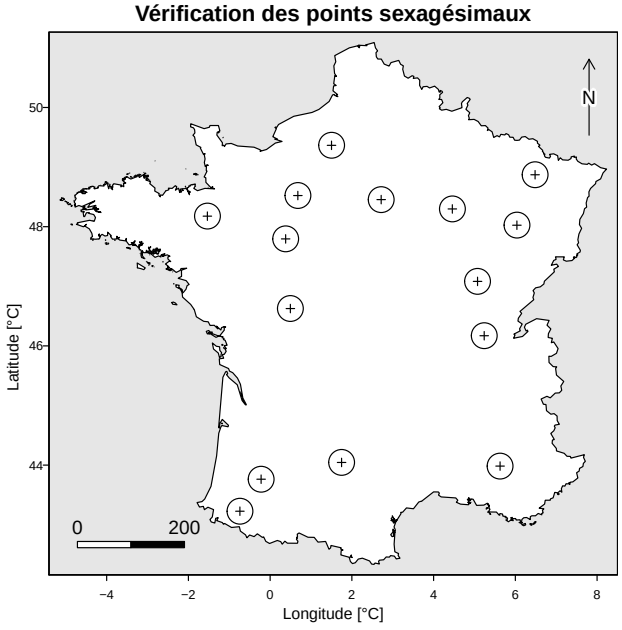
Nous pouvons utiliser la fonction `char2dms()` du paquet `sp` [7, 2] qui va transformer une chaîne de caractères en un objet de la classe `DMS` que l'on peut ensuite convertir en degrés décimaux avec la fonction `as.numeric()`.

```
DMS <- read_ods("PointsGPS/DMS.ods", as_tibble = FALSE)
library(sp)
DMS$xLon <- as.numeric(char2dms(DMS$Longitude, chd = "°", chm = "'", chs = "\""))
DMS$yLat <- as.numeric(char2dms(DMS$Latitude,  chd = "°", chm = "'", chs = "\""))
head(DMS)
```

```
    Code     Latitude  Longitude       xLon     yLat
1 CHS 01 46°10'17" N 5°14'22" E  5.239444 46.17139
2 CHS 10 48°17'54" N 4°27'36" E  4.460000 48.29833
3 CHS 21 47°04'58" N 5°04'30" E  5.075000 47.08278
4 CHS 27 49°21'58" N 1°30'15" E  1.504167 49.36611
5 CHS 35 48°10'41" N 1°32'01" W -1.533611 48.17806
6 CHS 57 48°52'18" N 6°29'02" E  6.483889 48.87167
```

Les arguments de la fonction permettent d'indiquer quels sont les caractères utilisés pour représenter les degrés, minutes et secondes. Notons que pour le site `CHS 35` qui est à l'ouest du méridien de Greenwich on récupère bien une longitude décimale négative. On fait une petite représentation graphique pour vérifier que nos points sont positionnés correctement :

```
ptsDMS <- with(DMS, vect(cbind(xLon, yLat), crs = "EPSG:4326"))
macarte(fdc, main = "Vérification des points sexagésimaux")
points(pts[pts$type == "Chêne", ], pch = 3)
points(ptsDMS, pch = 1, cex = 3)
```



5.3 Pire, je n'ai pas de degrés du tout !

Dans ce cas de figure, nous n'avons pas des coordonnées GPS mais les coordonnées cartésiennes dans un plan défini par un système de coordonnées

	A	B	C	D
1	Code	Latitude	Longitude	Long.dec
2	CHS 01	46°10'17" N	5°14'22" E	
3	CHS 03	46°40'05" N	2°43'37" E	
4	CHS 10	48°17'54" N	4°27'36" E	
5	CHS 18	47°15'17" N	2°07'29" E	
6	CHS 21	47°04'58" N	5°04'30" E	
7	CHS 27	49°21'58" N	1°30'15" E	
8	CHS 35	48°10'41" N	1°32'01" W	= -1*(1 + 32/60 + 1/3600)
9	CHS 41	47°34'09" N	1°15'36" E	
10	CHS 51	49°02'00" N	4°57'38" E	

Figure 9 : Exemple d'un tableur contenant des degrés sexagésimaux. On peut les convertir « à la main » en degrés décimaux en entrant une formule, par exemple ici pour le site de code `CHS 35`. Ne pas oublier que les longitudes à l'ouest du méridien de Greenwich doivent être négatives, de même pour les latitudes au sud de l'équateur.

géographique particulier. C'est d'une complexité effroyable, songez simplement aux effets de la dérive des plaques continentales ! Fort heureusement, il nous suffit de connaître le code `EPSG` utilisé (exemples courants section 5.4 page 21). En France métropolitaine un incontournable[7] est la projection dite de Lambert 93 (`EPSG:9794`). Le fichier `L93.ods` contient un exemple de ce type de coordonnées.

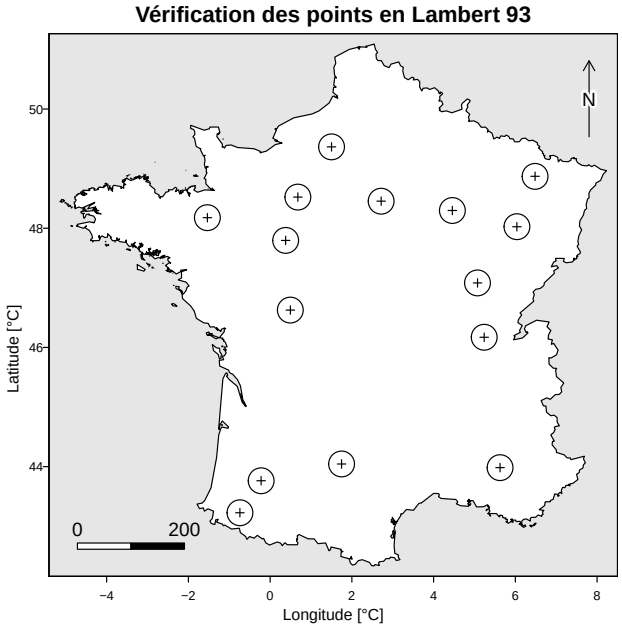
```
L93 <- read_ods("PointsGPS/L93.ods", as_tibble = FALSE)
head(L93)
```

```
    Code LambertX_93_m LambertY_93_m
1 CHS 01      872644.1       6565885
2 CHS 10      808199.1       6800660
3 CHS 21      857320.3       6666736
4 CHS 27      591233.1       6919419
5 CHS 35      363054.0       6795979
6 CHS 57      955392.5       6868954
```

On peut voir au premier coup d'œil que ce ne sont pas des coordonnées GPS puisque non comprises dans l'intervalle -180° à +180° pour les longitudes et dans l'intervalle -90° à +90°C pour les latitudes. Il s'agit en fait de distances en mètres sur le plan de projection, d'où les valeurs élevées. Pour les convertir en coordonnées GPS il suffit d'utiliser la fonction `project()` :

```
ptsL93 <- with(L93, vect(cbind(LambertX_93_m, LambertY_93_m), crs = "EPSG:9794"))
ptsGPS <- project(ptsL93, "EPSG:4326")
macarte(fdc, main = "Vérification des points en Lambert 93")
points(pts[pts$type == "Chêne", ], pch = 3)
points(ptsGPS, pch = 1, cex = 3)
```

[7] C'est parce que la loi dispose qu'elle s'impose aux services publics via le dernier décret en vigueur. Pour une mise à jour voir sur Légifrance à `https://www.legifrance.gouv.fr/loda/id/LEGITEXT000005630333` la version consolidée du décret n° 2000-1276 du 26 décembre 2000.



5.4 Quelques EPSG utiles

La projection de la surface d'un patatoïde sur un plan induisant nécessairement des distorsions, il y a énormément de systèmes de projections qui ont été utilisés en fonction des objectifs poursuivis. Il peuvent être définis dans `terra` avec une chaîne de caractères répondant à une syntaxe précise[8]. Heureusement pour nous, il existe une base de données dite `EPSG` (acronyme de *European Petroleum Survey Group*) qui donne une clef d'identification aux projections les plus courantes (il y en a quand même plusieurs milliers de définies). Il suffit d'utiliser par exemple la notation `"EPSG:4326"` pour faire référence au système de projection utilisé par les GPS. Voici une liste de codes `EPSG` que nous trouvons, localement parlant, utiles.

`4326` : WGS 84 - WGS84 - World Geodetic System 1984. C'est celui utilisé par les GPS. Il est incontournable et on en aura besoin tôt où tard. Par exemple ici pour récupérer le fond de carte.

`3035` : ETRS89-extended / LAEA Europe. C'est celui utilisé par l'agence environnementale européenne [8] pour la diffusion des données satellitaires sur le taux de couvert arborée.

`27572` : NTF (Paris) / Lambert II Carto (Centre) dit Lambert II étendue. C'est celle qui est utilisée par SAFRAN [11]. Mais attention dans les fichiers SAFRAN ils sont donnés en hectomètres.

`2154` : RGF93 v1 / Lambert 93 - France. Ancienne projection Lambert 93 utilisée au XX^{e} siècle.

[8] Type : `"+proj=lcc +lat_0=38.9072 +lon_0=-77.0369 +lat_1=33 +lat_2=45 +ellps=GRS80"`

9794 : RGF93 v2b / Lambert 93 - France. C'est la projection officielle actuelle pour la France métropolitaine. C'est celle utilisée par exemple pour l'inventaire forestier national [1].

Références

[1] Anonymous. Données brutes de l'inventaire forestier mises en ligne sur DataIFN. Technical report, Service de l'information statistique forestière et environnementale and Institut National de l'Information Géographique et Forestière, 2022. 8 pp. Version 2.0.

[2] R.S. Bivand, E. Pebesma, and V. Gomez-Rubio. *Applied spatial data analysis with R, Second edition*. Springer, NY, 2013.

[3] J. Cheng, B. Schloerke, B. Karambelkar, and Y. Xie. *leaflet : Create Interactive Web Maps with the JavaScript 'Leaflet' Library*, 2024. R package version 2.2.2.

[4] A. Hacket-Pain, J.J. Foest, I.S. Pearse, J.M. LaMontagne, W.D. Koenig, G. Vacchiano, M. Bogdziewicz, T. Caignard, P. Celebias, J. van Dormolen, M. Fernández-Martínez, J.V. Moris, C. Palaghianu, M. Pesendorfer, A. Satake, E. Schermer, A.J. Tanentzap, P.A. Thomas, D. Vecchio, A.P. Wion, T. Wohlgemuth, T. Xue, K. Abernethy, M.-C. Aravena A., M.D. Barrera, J.H. Barton, S. Boutin, E.R. Bush, S.D. Calderón, F.S. Carevic, C.V. de Castilho, J.M. Cellini, C.A. Chapman, H. Chapman, F. Chianucci, P. da Costa, L. Croisé, A. Cutini, B. Dantzer, R.J. DeRose, J.-T. Dikangadissi, E. Dimoto, F.L. da Fonseca, L. Gallo, G. Gratzer, D.F. Greene, M.A. Hadad, A.H. Herrera, K.J. Jeffery, J.F. Johnstone, U. Kalbitzer, W. Kantorowicz, C.A. Klimas, J.G.A. Lageard, J. Lane, K. Lapin, M. Ledwoń, A.C. Leeper, M.V. Lencinas, A.C. Lira-Guedes, M.C. Lordon, P. Marchelli, S. Marino, H. Schmidt Van Marle, A.G. McAdam, L.R. . Momont, M. Nicolas, L.H. de Oliveira Wadt, P. Panahi, G. Martínez Pastur, T. Patterson, P. Luis Peri, Ł. Piechnik, M. Pourhashemi, C. Espinoza Quezada, F.A. Roig, K. Peña Rojas, Y. Micaela Rosas, S. Schueler, B. Seget, R. Soler, M.A. Steele, M. Toro-Manríquez, C.E.G. Tutin, T. Ukizintambara, L. White, B. Yadok, J.L. Willis, A. Zolles, M. Żywiec, and D. Ascoli. MASTREE+ : time-series of plant reproductive effort from six continents. *Global Change Biology*, 00 :1–17, 2022.

[5] R.J. Hijmans. *terra : Spatial Data Analysis*, 2024. R package version 1.7-71.

[6] R.J. Hijmans, M. Barbosa, A. Ghosh, and A. Mandel. *geodata : Download Geographic Data*, 2023. R package version 0.5-9.

[7] E.J. Pebesma and R.S. Bivand. Classes and methods for spatial data in R. *R News*, 5(2) :9–13, November 2005.

[8] H. Peifer. About the EEA reference grid. Technical report, European Environment Agency, 2011. 2 pp.

[9] G.-J. Schutten, C.-h. Chan, P. Brohan, D. Steuer, and T.J. Leeper. *readODS : Read and Write ODS Files*, 2024. R package version 2.3.0.

[10] Sun Microsystems. XDR : external data representation standard. RFC 1014. Technical report, Network Working Group, 1987.

[11] J.-P. Vidal, E. Martin, L. Franchistéguy, M. Baillon, and J.-M. Soubeyroux. A 50-year high-resolution atmospheric reanalysis over France with the safran system. *International Journal of Climatology*, 30(11) :1627–1644, 2010.