Comment lire des données NetCDF avec R

Partie 2 : avec le package stars (et ncmeta)

Cyril Bernard, CEFE

15/10/2019

Le package `stars`

Le package `stars` permet de lire et gérer des données issues de NetCDF mais aussi de format raster courants dans les SIG tels que GeoTIFF, JP2.

La philosophie de `stars` est d'offrir une structure adaptée aux cubes de données spatiales : notamment les séries temporelles, avec une ou plusieurs variables.

L'autre avantage de `stars` est la possibilité d'utiliser des fonctions issues de `dplyr` pour extraire ou agréger des données.

Notez que l'installation (séparée) du package `ncmeta` est requise pour lire le NetCDF. L'installation de `dplyr` est évidemment recommandée.

Pour un tour complet, voir les 5 parties de la vignette : https://r-spatial.github.io/stars/articles/stars1.html

```
# chargement des packages
library(stars)
library(dplyr)
library(ncmeta)
```

Données à télécharger

Dans ce document, les 2 mêmes jeux de données que dans la partie 1 sont utilisés :

- CHIRPS : une grille mondiale à la résolution 0.05° et 0.25° avec une estimation quotidienne des precipitations (https://www.chc.ucsb.edu/data/chirps) de 1981 à aujourd'hui
- CRU TS Version 4.03 : une grille mondiale à la résolution 0.5° avec 9 variables climatiques (http://www.cru.uea.ac.uk/data/) de 1901 à 2016

Pour tester les codes de ce document, vous devez avoir téléchargé les données déjà utilisées dans la partie 1 :

- ftp://ftp.chg.ucsb.edu/pub/org/chg/products/CHIRPS-2.0/global_daily/netcdf/p25/chirps-v2.0.2018.days_p25.nc
- https://crudata.uea.ac.uk/cru/data/hrg/cru_ts_4.03/cruts.1905011326.v4.03/tmp/cru_ts4.03.1981.1990.tmp.dat.nc.gz

Exemple 1 : CHIRPS

Comment ouvrir un fichier NetCDF

2 fonctions sont disponibles pour lire un fichier dans `stars`.

- `read_ncdf` est bien adapté aux fichiers NetCDF. Requiert l'installation de `ncmeta`.
- `read_stars` repose sur GDAL, tout comme le package `rgdal`. En théorie, fonctionne sur les fichiers NetCDF. En pratique, fonctionne plutôt sur les formats raster autre que NetCDF.

```
f_ncdf <- "data/chirps-v2.0.2018.days_p25.nc"
chirps_2018 <- read_ncdf(f_ncdf)
```

```
## no 'var' specified, using precip
```

```
## other available variables:
##  latitude, longitude, time
```

```
## No projection information found in nc file.
##  Coordinate variable units found to be degrees,
##  assuming WGS84 Lat/Lon.
```

```
chirps_2018
```

```
## stars object with 3 dimensions and 1 attribute
## attribute(s), summary of first 1e+05 cells:
##  precip [mm/d]
##  Min.   : 0.00
##  1st Qu.: 0.00
##  Median : 0.00
##  Mean   : 1.12
##  3rd Qu.: 0.27
##  Max.   :50.55
##  NA's   :94224
## dimension(s):
##           from   to                  offset  delta
## longitude    1 1440                    -180   0.25
## latitude     1  400                     -50   0.25
## time         1  365 2017-12-31 12:00:00 UTC 1 days
##                                refsys point values
## longitude +proj=longlat +datum=WGS8...    NA   NULL [x]
## latitude  +proj=longlat +datum=WGS8...    NA   NULL [y]
## time                           POSIXct    NA   NULL
```

Comment est structuré un objet `stars`

Un objet `stars` comporte un ou plusieurs **attributs** ou variables. Ici, l'objet `chirps_2018` comporte 1 attribut : `precip`.

Les données de l'objet `stars` sont ordonnées selon N **dimensions**. Ici, l'objet `chirps_2018` comporte 3 dimensions : `longitude`, `latitude`, `time`.

Les valeurs associées à *from* - *to*, *offset*, *delta*, pour chaque dimension, nous renseignent sur :

- la largeur et hauteur en pixels (1440 x 400) ainsi que le nb de jours en profondeur (365)
- la valeur initiale pour les 3 dimensions (-179.875 en longitude, -49.875 en latitude, le 01/01/2018 pour *time*)
- la résolution pour les 3 dimensions (0.25 degrés en longitude, 0.25 degrés en latitude, 1 jour pour *time*)

```
# comment extraire les noms de variables
names(chirps_2018)
```

```
## [1] "precip"
```

```
# comment extraire les dimensions
st_dimensions(chirps_2018)
```

```
##           from   to                  offset  delta
## longitude    1 1440                    -180   0.25
## latitude     1  400                     -50   0.25
## time         1  365 2017-12-31 12:00:00 UTC 1 days
##                                refsys point values
## longitude +proj=longlat +datum=WGS8...    NA   NULL [x]
## latitude  +proj=longlat +datum=WGS8...    NA   NULL [y]
```

```
## time                           POSIXct    NA   NULL
```

```
# valeur associée à la dimension 3 (time)
t_dates <- st_get_dimension_values(chirps_2018, which="time")
head(t_dates)
```

```
## [1] "2017-12-31 12:00:00 UTC" "2018-01-01 12:00:00 UTC"
## [3] "2018-01-02 12:00:00 UTC" "2018-01-03 12:00:00 UTC"
## [5] "2018-01-04 12:00:00 UTC" "2018-01-05 12:00:00 UTC"
```

```
# remarque :
# pourquoi les dates commencent-elles au 31 déc de l'année précédente ?
# mystère ...
t_dates <- as.Date(t_dates) + 1
```

Extraire un sous-ensemble

Comment faire pour extraire les précipitations sur une surface correspondant à l'Espagne pour une date spécifiée (prenons le 1 avril 2018) ?

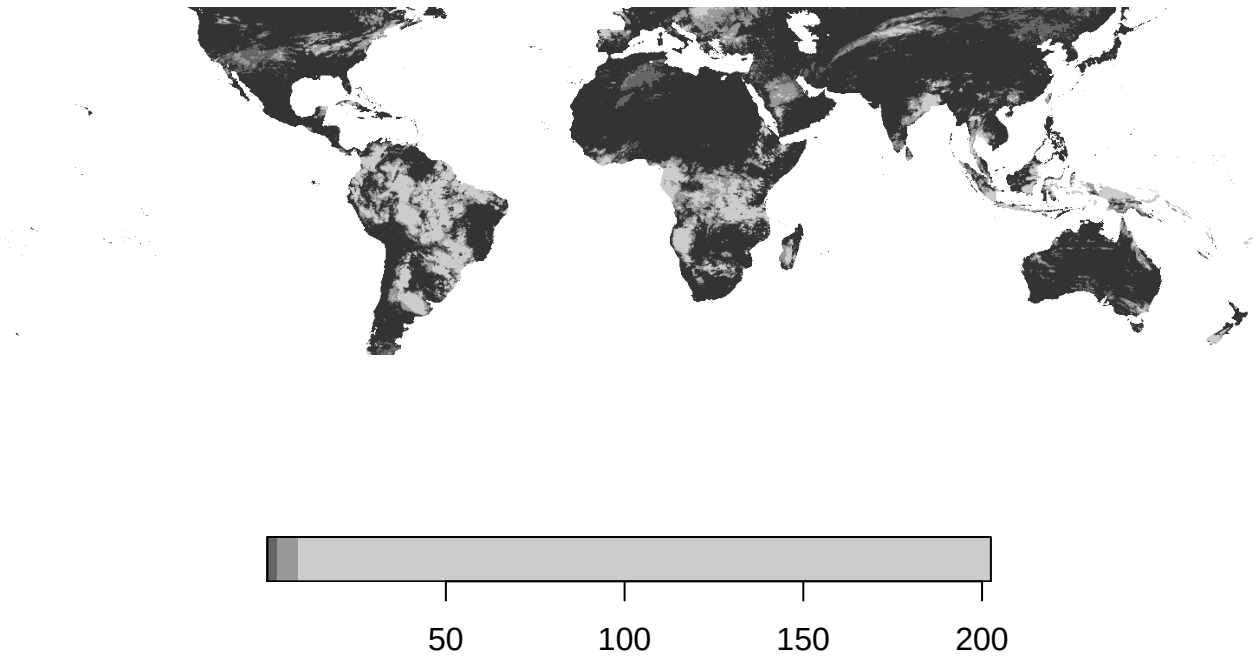
Il existe plusieurs techniques d'indexation, mais le plus simple est d'utiliser les verbes dplyr `select` pour extraire une variable, `slice` ou `filter` pour extraire une date, et la fonction `st_crop` pour extraire une étendue.

Pour plus d'infos sur les fonctions dplyr dans stars, consultez : https://r-spatial.github.io/stars/articles/stars3.html

```
# exemple avec slice : extraire les precip du 01/04/2018
chirps_20180401 <-  chirps_2018 %>%
  slice(time, index=which(t_dates=="2018-04-01"))
chirps_20180401
```

```
## stars object with 2 dimensions and 1 attribute
## attribute(s):
##  precip [mm/d]
##  Min.   :  0.0
##  1st Qu.:  0.0
##  Median :  0.0
##  Mean   :  2.5
##  3rd Qu.:  1.3
##  Max.   :202.5
##  NA's   :418679
## dimension(s):
##           from   to offset delta                       refsys point values
## longitude    1 1440   -180  0.25 +proj=longlat +datum=WGS8...    NA   NULL
## latitude     1  400    -50  0.25 +proj=longlat +datum=WGS8...    NA   NULL
##
## longitude [x]
## latitude  [y]
```

```
plot(chirps_20180401)
```




```
# exemple avec filter
chirps_20180401f <-  chirps_2018 %>%
  filter(time > as.POSIXct("2018-03-31",tz="UTC"),
         time < as.POSIXct("2018-04-01",tz="UTC"))
# Remarque : avec filter, il subsiste dans chirps_20180401f
# une 3eme dimension 'time', bien qu'il n'y ait qu'1 date.
chirps_20180401f %>% st_dimensions()
```

```
##           from   to offset delta                       refsys point
## longitude    1 1440   -180  0.25 +proj=longlat +datum=WGS8...    NA
## latitude     1  400    -50  0.25 +proj=longlat +datum=WGS8...    NA
## time         1    1     NA    NA                      POSIXct    NA
##                            values
## longitude                    NULL [x]
## latitude                     NULL [y]
## time       2018-03-31 12:00:00 UTC
```

```
# Pour ne conserver que les 2 dimensions longitude et latitude,
# utiliser adrop() comme ceci :
library(abind)
chirps_20180401f %>% adrop() %>% st_dimensions()
```

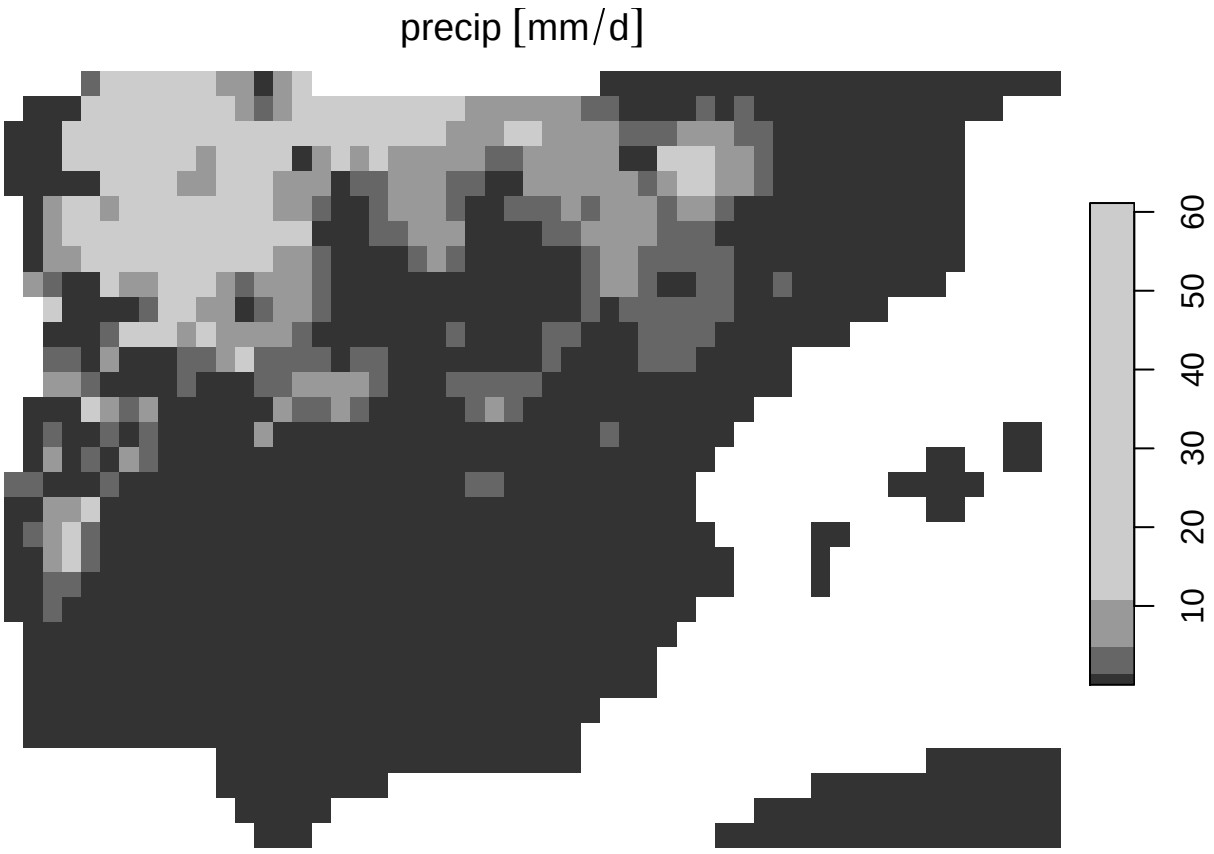
```
##           from   to offset delta                       refsys point values
## longitude    1 1440   -180  0.25 +proj=longlat +datum=WGS8...    NA   NULL
## latitude     1  400    -50  0.25 +proj=longlat +datum=WGS8...    NA   NULL
##
## longitude [x]
```

```
## latitude  [y]
```

```
# utilisation de st_crop pour extraire l'Espagne
sp_ext <- c(xmin=-9.3, xmax=4.4, ymin=35.9, ymax=43.8)
sp_bbox <- st_bbox(sp_ext, crs=st_crs("+proj=longlat +datum=WGS84 +no_defs"))
sp_20180401 <- chirps_20180401 %>% st_crop(sp_bbox)
```

```
## although coordinates are longitude/latitude, st_intersects assumes that they are planar
```

```
plot(sp_20180401)
```



Technique d'indexations avec stars

Il est possible d'utiliser des indices pour extraire des données, mais c'est moins intuitif que la solution vu précédemment.

Pour des données à 3 dimensions, comme dans notre NetCDF CHIRPS, la syntaxe est la suivante :

```
data_stars[v, x, y, t]
```

où v=index ou nom de la variable, x=index longitude, y=index latitude, t=index temps.

Par exemple :

```
lon <- st_get_dimension_values(chirps_2018, which="longitude")
lat <- st_get_dimension_values(chirps_2018, which="latitude")
iv <- 1 # variable precip (la seule) = 1
it <- which(t_dates=="2018-04-01")
ix <- which(lon >= -9.3 & lon <= 4.4)
iy <- which(lat >= 35.9 & lat <= 43.8)
```

```
# extraire la variable 1 pour la zone espagne, le 01/04/2018
chirps_extr <- chirps_2018[iv, ix, iy, it]
# extraire la variable 1 pour la zone espagne, avec ts les jours du 01/01 au 31/12
chirps_extr <- chirps_2018[iv, ix, iy, ]
# extraire tte les variables, sur la globalité du monde, pour le 01/04/2018
chirps_extr <- chirps_2018[, , , it]
```

Sauvegarder en .tif un objet stars

```
write_stars(sp_20180401, "data/esp_20180401_stars.tif", type="Float32")
```

Ouvrir un fichier sans charger les données en mémoire

Charger toutes les données d'un fichier peut occuper beaucoup de mémoire. Les données prennent souvent plus de place en mémoire que la taille initiale du fichier, car les nombres entiers sont converties en flottant par R.

La fonction `read_stars` offre un mécanisme pour ouvrir des données sans les charger en mémoire avec l'option `proxy=TRUE`.

La fonction `read_ncdf` ne propose pas cette option, par contre il est possible de préciser à l'ouverture du fichier une variable avec `var` et l'index de la partie à extraire avec `ncsub` (voir l'aide de cette fonction).

Exemple 2 : CRU TS

Lecture du fichier

Ce jeu de données contient 2 variables : tmp (température en °C) et stn (station counts)

```
f_ncdf2 <- "data/cru_ts4.03.1981.1990.tmp.dat.nc"
cruts_81_90 <- read_ncdf(f_ncdf2)
```

```
## no 'var' specified, using tmp, stn
```

```
## other available variables:
##  lon, lat, time
```

```
## No projection information found in nc file.
##  Coordinate variable units found to be degrees,
##  assuming WGS84 Lat/Lon.
```

```
cruts_81_90
```

```
## stars object with 3 dimensions and 2 attributes
## attribute(s), summary of first 1e+05 cells:
##    tmp [°*°C]         stn
##  Min.   : 1.00   Min.   :1.00
##  1st Qu.:20.90   1st Qu.:8.00
##  Median :24.70   Median :8.00
##  Mean   :23.61   Mean   :7.95
##  3rd Qu.:27.78   3rd Qu.:8.00
##  Max.   :33.80   Max.   :8.00
##  NA's   :94158   NA's   :94158
## dimension(s):
##      from  to offset delta                       refsys point
## lon     1 720   -180   0.5 +proj=longlat +datum=WGS8...    NA
## lat     1 360    -90   0.5 +proj=longlat +datum=WGS8...    NA
```

```
## time    1 120     NA    NA                       POSIXct    NA
##                         values
## lon                       NULL [x]
## lat                       NULL [y]
## time 1981-01-16,...,1990-12-16
```

Nous avons 3 dimensions : *lon*, *lat*, *time*. La résolution spatiale (*delta*) est de 0.5 degrés. Pour l'axe *time*, nous avons une date par mois.

```
t <- st_get_dimension_values(cruts_81_90, which="time")
head(t)
```

```
## [1] "1981-01-16 UTC" "1981-02-15 UTC" "1981-03-16 UTC" "1981-04-16 UTC"
## [5] "1981-05-16 UTC" "1981-06-16 UTC"
```

```
# obtenir mois sous la forme YYYY-MM
t_months <- format(as.Date(t), "%Y-%m")
```

Extraction de données

Dans cet exemple, nous cherchons à extraire les données température (variable *tmp*), pour la zone Amérique du Sud, et pour les mois de décembre uniquement.

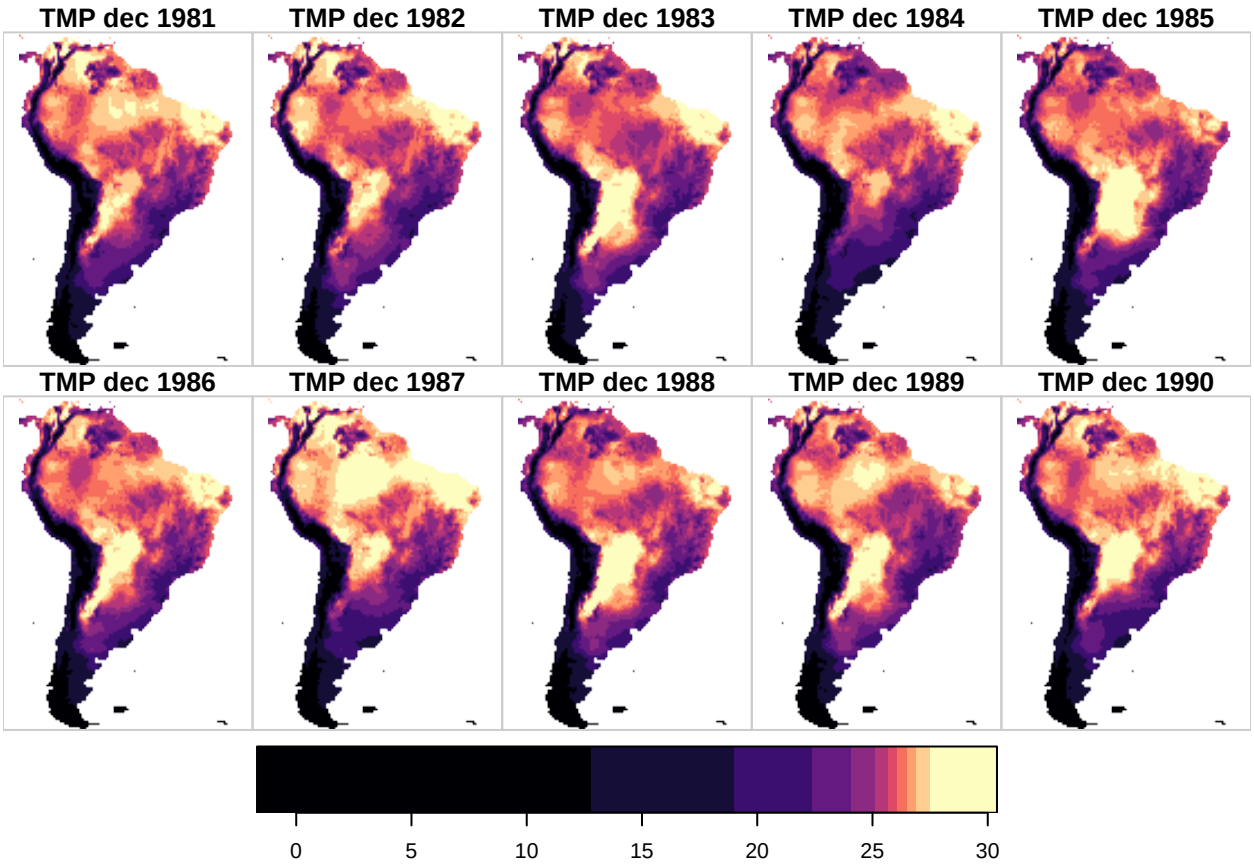
```
# définir étendue Amérique sud sous forme de bbox
sam_ext <- c(xmin=-82, xmax=-34, ymin=-56, ymax=13)
sam_bbox <- st_bbox(sam_ext, crs=st_crs("+proj=longlat +datum=WGS84 +no_defs"))
# définir indices des mois de décembre
y_m <- paste0(seq(1981, 1990), "-12") # de 1981-12 à 1990-12
i_time <- match(y_m, t_months)
# select (variable) + crop (étendue) +  slice (temps)
cruts_dec81_dec90_sam <- cruts_81_90 %>% select(tmp) %>%
  st_crop(sam_bbox) %>% slice(time, index=i_time)
```

```
## although coordinates are longitude/latitude, st_intersects assumes that they are planar
```

```
library(viridis)
```

```
## Loading required package: viridisLite
```

```
plot(cruts_dec81_dec90_sam, main=paste0("TMP dec ", seq(1981, 1990)), col=magma(11))
```



Conclusion

Une fois que l'on maîtrise la structure des objets `stars`, la syntaxe des commandes est concise et relativement simple. Cependant, la fonction `read_stars` ne semble pas fonctionner sur les fichiers NetCDF et c'est dommage. La gestion des dates au format POSIXct est un peu confuse (voir exemple CHIRPS)