
R_GIS 02



Introduction aux géotraitements rasters avec R

Septembre 2026



TABLE DES MATIERES

1.	INTRODUCTION	1
2.	FONCTIONNALITES LIEES AUX TRAITEMENTS DE DONNEES RASTER	2
2.1	LECTURE , ECRITURE ET PROPRIETES D'OBJETS SPATRASTER	2
2.1.1	Lecture d'une couche raster mono-bande	2
2.1.2	Principales propriétés d'un objet SpatRaster	3
2.1.3	Sauvegarde d'un objet SpatRaster	5
2.1.4	Raster multi-bandes	6
2.2	MODIFIER LA GEOMETRIE D'UN RASTER.....	8
2.2.1	Rogner une couche raster (crop)	8
2.2.2	Rogner et masquer une couche raster (crop)	9
2.2.3	Reprojeter ou ré-échantillonner une couche raster	10
2.2.4	Agrégation - désagrégation	11
2.2.5	Créer un raster de toutes pièces (template)	12
2.3	ALGEBRE RASTER (FONCTIONS LOCALES)	14
2.3.1	Requête	14
2.3.2	Opérations mathématiques	16
2.3.3	Fonctions mathématiques.....	16
2.3.4	Création de classes.....	17
2.4	CALCUL DE STATISTIQUES	18
2.4.1	Statistiques globales (1 raster → 1 valeur)	18
2.4.2	Statistiques locales (x rasters → 1 raster).....	20
2.4.3	Corrélations globales	20
2.4.4	Calculs de surfaces par classes.....	20
2.5	EXTRACTION DE VALEURS AU SEIN D'UN RASTER	21
2.5.1	Extraction par points.....	21
2.5.2	Extraction par polygones (statistiques zonales)	22
2.6	CONVERSIONS SPATRASTER ↔ SPATVECTOR.....	23
2.6.1	SpatRaster → SpatVector	23
2.6.2	SpatVector → SpatRaster	23
2.7	CALCULS DE DISTANCES EUCLIDIENNES	26
2.8	PENTE, EXPOSITION, OMBRAGE	28
2.9	FONCTIONS FOCALLES (FILTRES)	30
2.9.1	Exemple 1 : filtre de lissage	30
2.9.2	Exemple 2 : filtre de lissage Gaussien.....	31
2.9.3	Exemple 3 : filtres morphologiques	31
2.10	RASTERS VIRTUELS (VRT)	34
3.	RESUME DES FONCTIONS R LIEES AUX GEOTRAITEMENTS	43

1. Introduction

- L'objectif de cet exercice est d'initier à l'utilisation des outils disponibles dans l'environnement R pour le traitement, la gestion et l'analyse de données spatiales de type raster.
- Les outils de traitement et d'analyse de données spatiales sont en pleine évolution dans l'environnement R. Anciennement, les opérations de géotraitement impliquaient de recourir à une multitude de packages : **sp** ou **sf** (gestion des objets vectoriels), **raster** (gestion des objets raster), **rgeos** (géotraitements vectoriels), **rgdal** (géotraitements raster)...
- Désormais, la plus grande partie des traitements peut être réalisée avec les 2 seuls packages **sf** (pour les géométries vectorielles) et **terra** pour les géométries raster et vectorielles. Le package **dplyr** est pour sa part utilisé pour la gestion des tables attributaires (sélection, jointures, agrégations...).
- L'ensemble des opérations présentées dans cet exercice sont rassemblées au sein d'un script **R_GIS_02.r** disponible dans le jeu de données qui accompagne le présent document. Les numéros des paragraphes de ces notes d'exercice peuvent être utilisés comme point de repère pour retrouver les lignes de code dans le script.
- La liste des paragraphes est accessible avec le bouton  accessible dans le bandeau de l'interface de R Studio.

The screenshot shows the R Studio interface. On the left, the script editor displays the following code:

```
#####
#
# Géotraitements raster avec R
#
# Auteur : Philippe Lejeune
# Version : 2025/09/09
#
#####
rm(list=ls()) # réinitialisation de la mémoire

# 1. Chargement des librairies -----

library(sf)
library(terra)
library(dplyr)
```

On the right, the table of contents lists the following sections:

- (Untitled)
- (Untitled)
- 1. Chargement des librairies
- 2. Lecture, écriture et propriétés d'objet SpatRaster
- 3. Modifier la géométrie des couches raster
- 3.1 Rogner un raster (crop)
- 3.2 Crop & Mask
- 3.3 Reprojecter et rééchantillonner un raster
- 3.4 Aggregate et Disaggregate
- 3.5 Créer un raster de toute pièce
- 4. Algèbre raster (fonctions locales)
- 4.1 Requêtes
- 4.2. Opérations mathématiques
- 4.3 Fonctions mathématiques
- ndvi_fun
- 4.4. Création de classes
- 5. Calcul de statistiques
- 5.1 Statistiques globales (sur les valeurs d'1 couc...
- 5.2 Statistiques locales (multi-bandes)

- Les manipulations présentées dans ce tutoriel ont été testées dans l'environnement de R Studio, avec la version 4.4.3 de R. Si la version de votre ordinateur sont trop différentes, il faut la mettre à jour.

```
# version de R
R.version$version.string

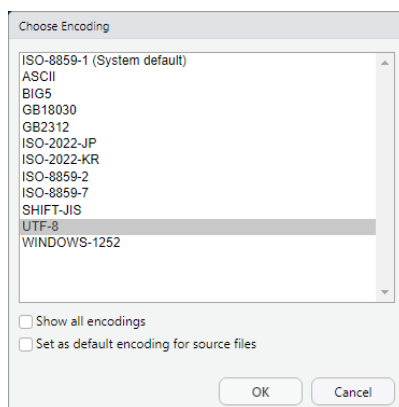
> R.version$version.string
[1] "R version 4.4.3 (2025-02-28 ucrt)"
```

- Il convient également de vérifier les versions des principaux packages utilisés et des les mettre à jour le cas échéant.

```
# version des packages utilisés dans ce tutoriel
packageVersion("sf")
packageVersion("terra")
packageVersion("dplyr")

> packageVersion("sf")
[1] '1.0.20'
> packageVersion("terra")
[1] '1.8.60'
> packageVersion("dplyr")
[1] '1.1.4'
```

- Le présent document décrit le déroulé de l'exercice, en le structurant en paragraphes. Dans chacun de ceux-ci sont présentés les différents concepts qui sont illustrés par des extraits du script de référence et des résultats obtenus par l'exécution de ces derniers.
- **Remarque :** le script **R_GIS_02.r** a été créé avec l'encodage « UTF-8 ». Si la version de R Studio dans laquelle le script est affiché utilise un autre encodage, certains caractères accentués ne s'afficheront pas correctement. Pour résoudre ce problème, il suffit de rouvrir le script avec la commande [File] → [Reopen with encoding ...] et de sélectionner l'encodage « UTF-8 ».



2. Fonctionnalités liées aux traitements de données raster

2.1 Lecture, écriture et propriétés d'objets SpatRaster

2.1.1 Lecture d'une couche raster mono-bande

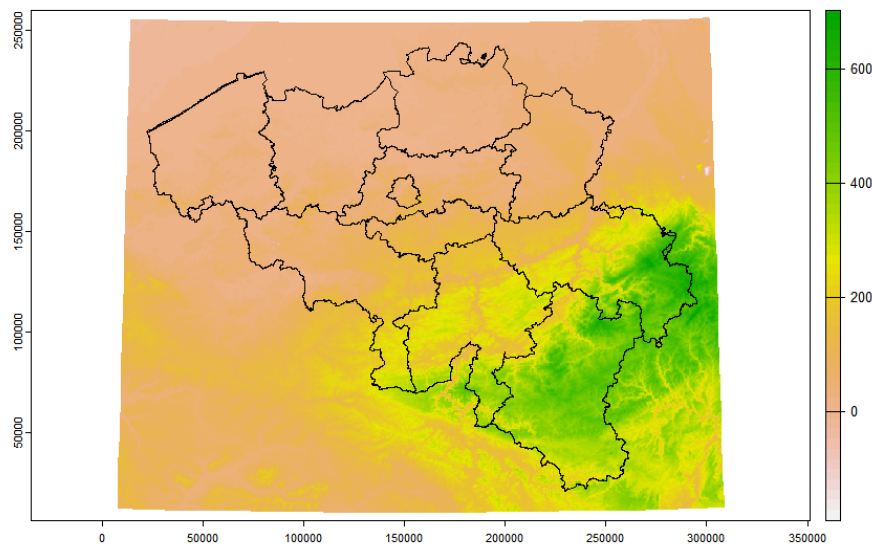
- Comme dans le tutoriel R_GIS_01, nous recommandons d'accéder aux sources de données en définissant de manière complète les noms des fichiers, tant à la lecture qu'à l'écriture. Cela implique de définir des variables qui contiennent l'adresse de répertoire où se trouvent les données d'entrée et celle relative au répertoire qui recevra les données de sortie. Cette approche permet notamment d'accéder facilement à des données situées dans différents répertoires.

```
# définir les chemins d'accès
path0="c:/tmp/R_GIS_02"
path_in=paste0(path0,"/data_in")
path_out=paste0(path0,"/output")
```

- La lecture d'une couche raster pour créer un objet **SpatRaster** utilise la fonction **rast()**.

```
# Lire une couche raster "monobande"
f_in=paste0(path_in,"/mnt_250m.tif")
mnt=rast(f_in)
plot(mnt)

# superposer 1 couche vectorielle
f_prov=paste0(path_in,"/provinces.gpkg")
prov=st_read(f_prov)
plot(prov$geom,add=T)
```



2.1.2 Principales propriétés d'un objet **SpatRaster**

- L'objet « **mnt** » dans lequel a été chargé le fichier **mnt_250m.tif** appartient à la classe **SpatRaster**. C'est avec cette classe que la librairie **terra** gère les données raster.
- Les principales propriétés d'un objet **SpatRaster** sont affichées en tapant le nom de cet objet dans la console : les dimensions (nombre de lignes, de colonnes et de couches), la résolution spatiale, l'emprise spatiale (xmin, xmax, ymin, ymax), le système de coordonnées, le nom du fichier source, le nom de la ou des couches, ainsi que les valeurs extrêmes.

```
> mnt
class           : SpatRaster
dimensions      : 1014, 1232, 1 (nrow, ncol, nlyr)
resolution      : 250, 250 (x, y)
extent          : 4000, 312000, 6500, 260000 (xmin, xmax, ymin, ymax)
coord. ref.     : Belge 1972 / Belgian Lambert 72 (EPSG:31370)
source          : mnt_250m.tif
name            : mnt_250m
min value       : -197.413
max value       : 702.726
```

- La fonction **summary()** permet également d'accéder aux statistiques de base du raster (valeurs min, max, quartiles, moyenne), ainsi qu'une indication du nombre de pixels contenant des nodata.
- Il faut cependant être conscient que ces statistiques sont calculées sur un échantillon des pixels constitutifs du raster.

```
> summary(mnt)
      mnt_250m
Min.   : -176.16
1st Qu.:  25.11
Median :  89.10
Mean    : 140.10
3rd Qu.: 215.42
Max.    : 702.73
NA's    : 8307
Warning message:
[summary] used a sample
```

- La fonction **st_crs()** renseigne sur le CRS de la couche raster.

```
> st_crs(mnt)
Coordinate Reference System:
  User input: Belge 1972 / Belgian Lambert 72
  wkt:
PROJCRS["Belge 1972 / Belgian Lambert 72",
  BASEGEOGCRS["Belge 1972",
    DATUM["Reseau National Belge 1972",
      ELLIPSOID["International 1924",6378388,297,
        LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
      ANGLEUNIT["degree",0.0174532925199433]],
    ID["EPSG",4313]],
  CONVERSION["Belgian Lambert 72",
```

- La fonction ***describe()*** offre une autre possibilité d'accéder aux métadonnées de la source de données. Cette fonction est l'équivalent de la fonction GDALInfo de la librairie GDAL.

```
> describe(f_in)
[1] "Driver: GTiff/GeoTIFF"
[2] "Files: C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/mnt_250m.tif"
[3] "Size is 1232, 1014"
[4] "Coordinate System is:"
[5] "PROJCRS[\"Belge 1972 / Belgian Lambert 72\",",
[6] "    BASEGEOGCRS[\"Belge 1972\",",
[7] "        DATUM[\"Rseau National Belge 1972\",",
[8] "            ELLIPSOID[\"International 1924\",6378388,297,",
[9] "                LENGTHUNIT[\"metre\",1]]],",
[10] "    PRIMEM[\"Greenwich\",0,",
[11] "        ANGLEUNIT[\"degree\",0.0174532925199433]],",
[12] "    ID[\"EPSG\",4313]],",
[13] "    CONVERSION[\"Belgian Lambert 72\",",
[14] "        METHOD[\"Lambert Conic Conformal (2SP)\",",
[15] "            ID[\"EPSG\",9802]],",
[16] "        PARAMETER[\"Latitude of false origin\",90,",
[17] "            ANGLEUNIT[\"degree\",0.0174532925199433]],",
[18] "            ID[\"EPSG\",8821]],",
[19] "        PARAMETER[\"Longitude of false origin\",4.36748666666667,"
```

- Il peut être intéressant de connaître précisément le type de données contenu dans la couche raster, c'est-à-dire la manière avec laquelle les valeurs des pixels sont codées. Il n'existe pas de fonction qui réponde à ce besoin dans la librairie terra. On doit donc se tourner vers la fonction ***raster::dataType()*** de la librairie raster.
- Pour que celle-ci fonctionne correctement, on doit convertir l'objet **SpatRaster** en objet **raster**.
- Le résultat de la fonction ***dataType()*** nous indique que les données contenues dans l'objet « **mnt** » sont codées en *float 32-bit* (FLT4S) (32 bit → 4 octets).

```
dataType(mnt)
> dataType(mnt)
[1] "FLT4S"
```

- Le tableau ci-dessous rappelle les gammes de valeurs associées aux différents types de données pouvant être stockés dans 1 raster. A titre d'exemple, le type de données **INT2S** correspond à un code en entier (INTeger) signé (S) de 16 bit (16-bit → 2 octets).

Datatype	definition	minimum possible value	maximum possible value
LOG1S	FALSE (0)		TRUE (1)
INT1S		-127	127
INT1U		0	255
INT2S		-32,767	32,767
INT2U		0	65,534
INT4S		-2,147,483,647	2,147,483,647
INT4U		0	4,294,967,296
FLT4S		-3.4e+38	3.4e+38
FLT8S		-1.7e+308	1.7e+308

2.1.3 Sauvegarde d'un objet **SpatRaster**

- La sauvegarde d'une couche raster dans 1 fichier tif s'opère avec la fonction ***writeRaster()***.

```
# Ecrire un couche raster dans un fichier .tif
f_out=paste0(path_out,"/mnt_250m.tif")
writeRaster(mnt,f_out,overwrite=TRUE)
```



- Remarque : l'option « **overwrite=TRUE** » permet d'écraser 1 fichier existant portant le même nom. Cette option est particulièrement utile lorsque le script est exécuté à plusieurs reprises.
- Il est possible de préciser différentes options lors de la sauvegarde d'une couche raster. Ainsi, dans l'exemple qui suit, on impose le type de données dans le fichier de sortie. Alors que l'objet « **mnt** » est codé en réel 16-bit, on sauvegarde celui-ci en entier 16-bit (INT2S).

```
# Ecrire un raster en forçant le type de données
f_out=paste0(path_out,"/mnt_250m_int2S.tif")
writeRaster(mnt,f_out,overwrite=TRUE, datatype="INT2S")
```

- Lorsque les objets rasters sont de grande taille, il peut être utile de les compresser lors de la création des fichiers. La librairie GDAL offre de telles possibilités qui peuvent être ajoutées comme option à la fonction **writeRaster()**, comme le montre l'exemple suivant.

```
# Ecrire un raster avec des options de compression du fichier
optINT2S = list(gdal=c("INTERLEAVE=BAND", "TILED=YES", "BIGTIFF=YES",
                      "COMPRESS=DEFLATE", "ZLEVEL=9", "NUM_THREADS=ALL_CPUS"),
               datatype='INT2S')
f_out=paste0(path_out,"/mnt_250m_int2S_compr.tif")
writeRaster(mnt,f_out,overwrite=TRUE, wopt=optINT2S)
```

2.1.4 Raster multi-bandes

- Le fichier **temp_mens.tif** contient 12 bandes (12 couches) correspondant aux températures mensuelles moyennes. Ces données proviennent de la base de données www.worldclim.org.

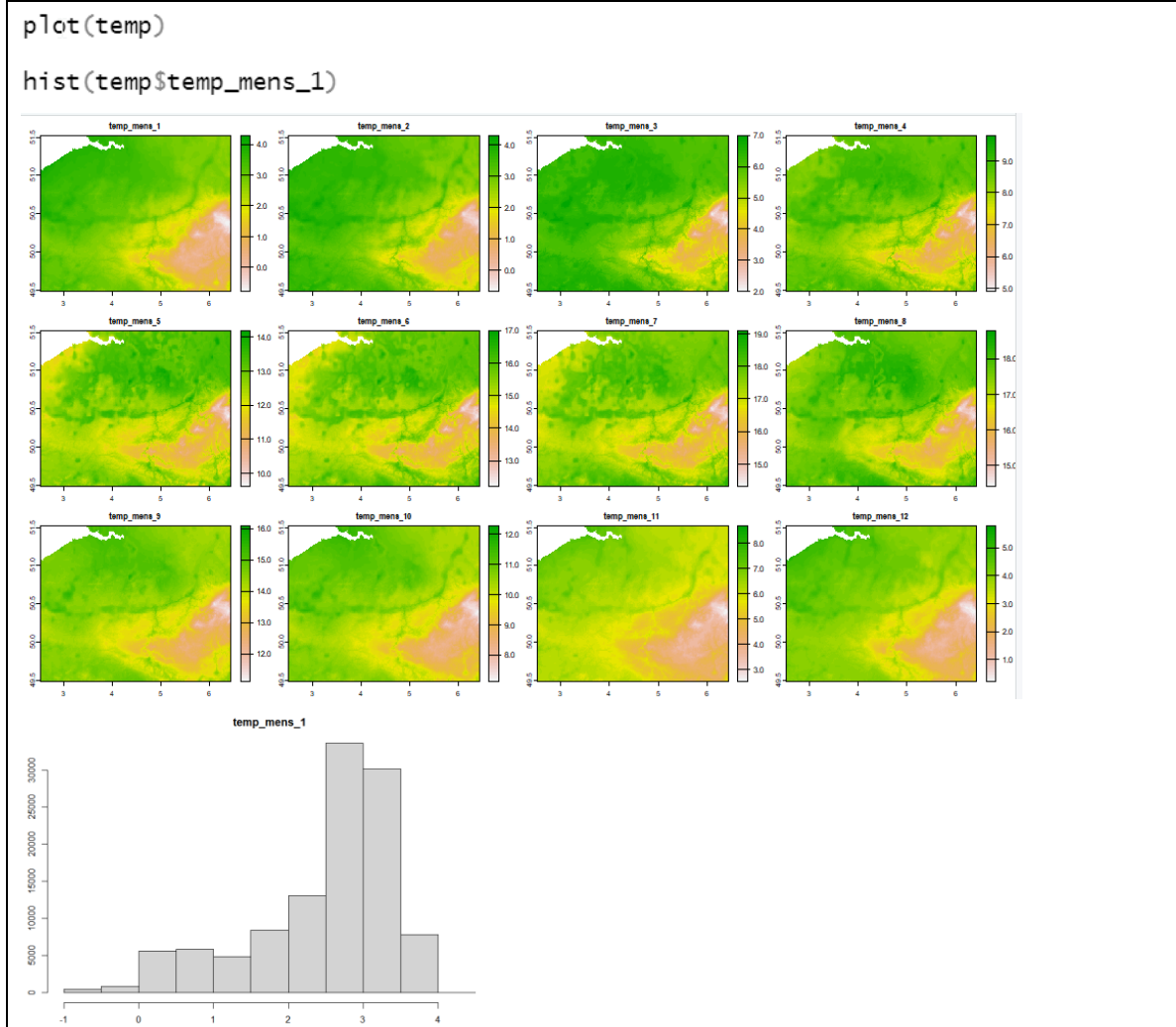
```
# Lire un raster comportant plusieurs bandes
f_in=paste0(path_in,"/temp_mens.tif")
temp=rast(f_in)

temp
> temp
class       : SpatRaster
dimensions  : 245, 470, 12  (nrow, ncol, nlyr)
resolution  : 0.008333333, 0.008333333  (x, y)
extent      : 2.516667, 6.433333, 49.48333, 51.525  (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat WGS 84 (EPSG:4326)
source      : temp_mens.tif
names       : temp_mens_1, temp_mens_2, temp_mens_3, temp_mens_4, temp_mens_5,
temp_mens_6, ...
min values   :      -0.8,      -0.7,      2.0,      4.9,      9.6,
12.2, ...
max values   :      4.3,      4.3,      7.0,      9.8,     14.2,
17.0, ...
```

- La fonction **names()** est utilisée pour afficher les noms des différentes bandes. A noter que l'utilisateur peut modifier ces noms s'il le souhaite.

```
> names(temp)
[1] "temp_mens_1" "temp_mens_2" "temp_mens_3" "temp_mens_4"
[5] "temp_mens_5" "temp_mens_6" "temp_mens_7" "temp_mens_8"
[9] "temp_mens_9" "temp_mens_10" "temp_mens_11" "temp_mens_12"
```

- La fonction **plot()** peut être utilisée pour afficher 1 ou plusieurs couches de l'objet « **temp** ».
- La fonction **hist()** quant à elle permet de visualiser la distribution de fréquence des valeurs des pixels.



- Il existe différents outils pour manipuler les couches d'un raster multi-bandes. Ainsi la fonction **subset()** est utilisée pour extraire certaines des bandes du raster.

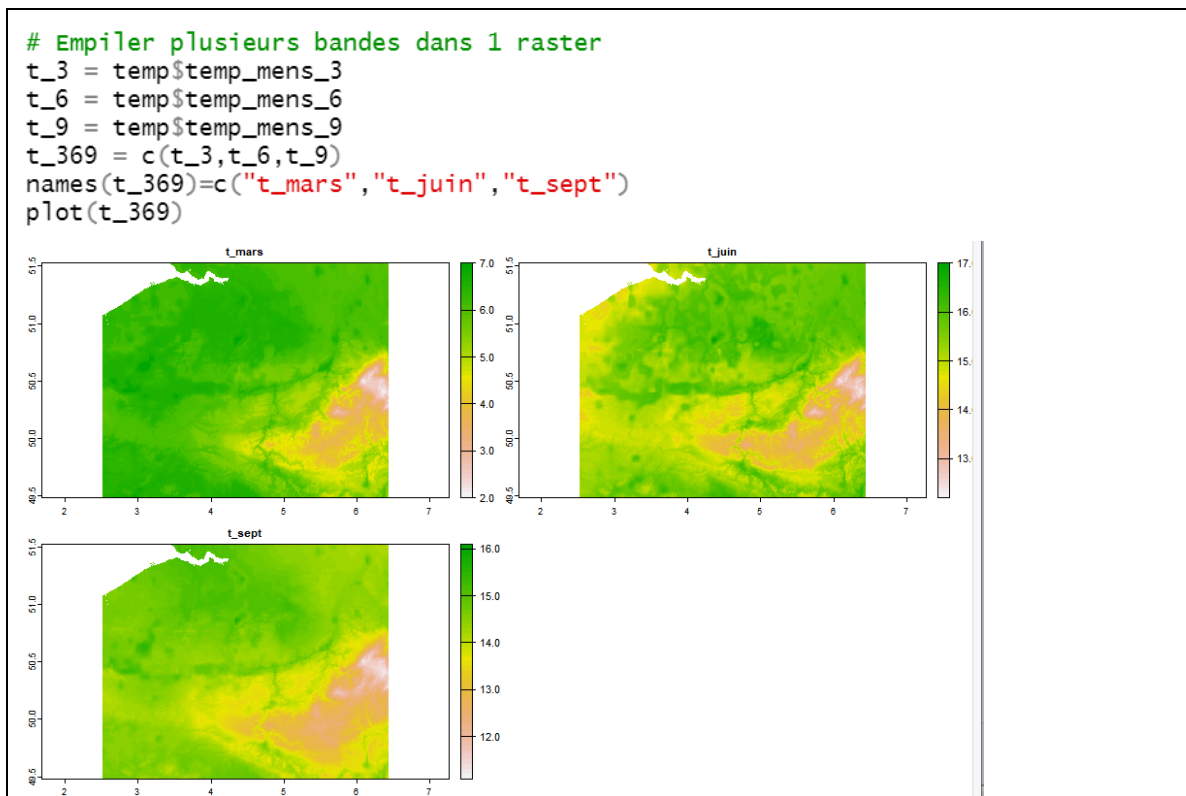
```
# Extraire certaines bandes d'un raster multi-bandes
temp_3456=subset(temp,c(3:6))
```



- Remarque importante :** les fonctions de la librairie **terra** qui génèrent des couches raster ont comme particularité qu'elles permettent de sauvegarder directement le résultat dans 1 fichier de sortie, évitant ainsi un stockage en mémoire vive. Cette possibilité est offerte par l'option « filename » qui indique le nom du fichier dans lequel est sauvegardé le résultat. Cette option peut être complétée des options utilisées dans la fonction **writeRaster()** : « overwrite » et « wopt ».

```
# Option "filename"
f_out=paste0(path_out,"/temp_3456.tif")
optFLT4S = list(gdal=c("INTERLEAVE=BAND", "TILED=YES", "BIGTIFF=YES",
                      "COMPRESS=DEFLATE", "ZLEVEL=9", "NUM_THREADS=ALL_CPUS"),
               datatype='FLT4S')
temp_3456=subset(temp,c(3:6),filename=f_out,overwrite=T,wopt=optFLT4S)
```

- Il est également possible de créer un raster multi-couches par empilement de différentes couches à l'aide de la fonction `c()`.



2.2 Modifier la géométrie d'un raster

2.2.1 Rogner une couche raster (crop)

- Très souvent, on est amené à modifier l'emprise spatiale d'un raster, pour que celui-ci colle avec l'emprise d'une autre couche, qu'elle soit vectorielle ou raster. Cette opération de rognage s'opère avec la fonction `crop()`.
- Dans l'exemple qui suit, la couche « **mnt** » est découpée aux limites de l'emprise spatiale de la province de Namur.
- La fonction `ext()` permet d'afficher les valeurs de la bounding box (emprise spatiale) de la nouvelle couche

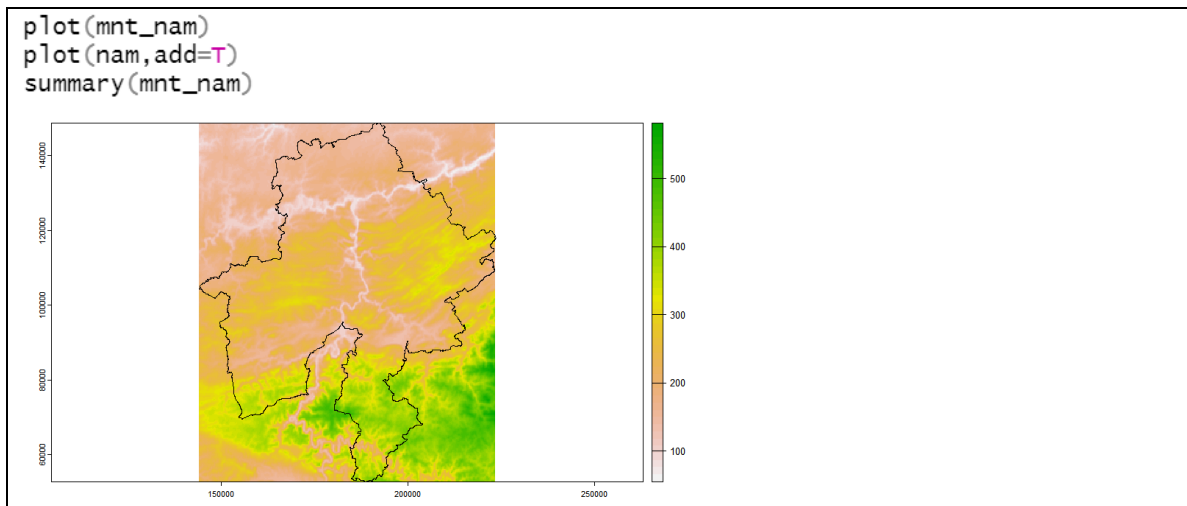
```
# step 1. déterminer l'emprise du nouveau raster
nam=prov[prov$NameFRE == "Province de Namur",]
nam=vect(nam) # convertir sf en SpatVector
ext(nam)

> ext(nam)
SpatExtent : 144085.864433491, 223574.0442208, 52802.5330247851,
148815.880906229 (xmin, xmax, ymin, ymax)

# step 2. rogner la couche mnt
mnt_nam = crop(mnt,nam)
ext(mnt_nam)

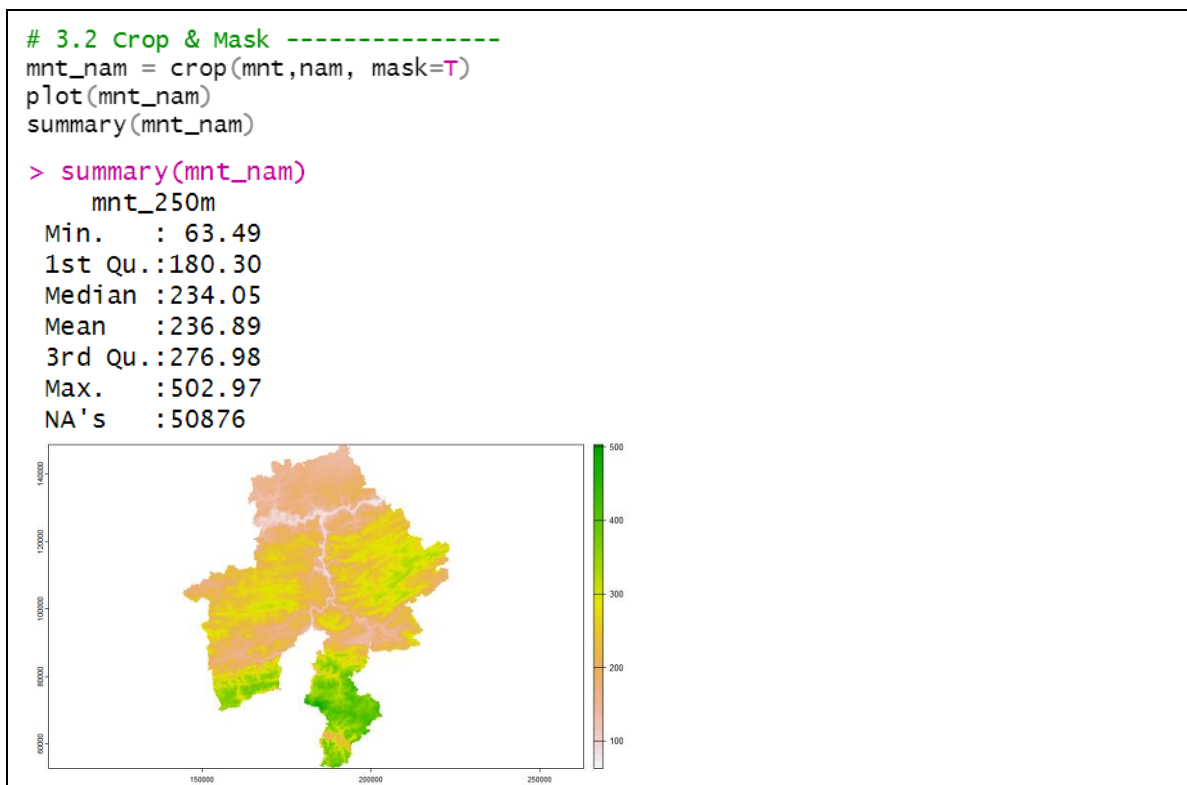
> ext(mnt_nam)
SpatExtent : 144000, 223500, 52750, 148750 (xmin, xmax, ymin, ymax)
```

- On notera que l'emprise du nouveau raster est légèrement différente de celle de la couche contenant le polygone de la province de Namur. Cela s'explique par le fait que le raster à une résolution spatiale de 250 m et que le découpage s'effectue de manière à englober l'emprise de la couche de référence, c'est-à-dire la couche « *nam* » dans le cas présent. Pour réaliser ce genre d'opération, il faut s'assurer au préalable que les 2 couches ont le même système de coordonnées.

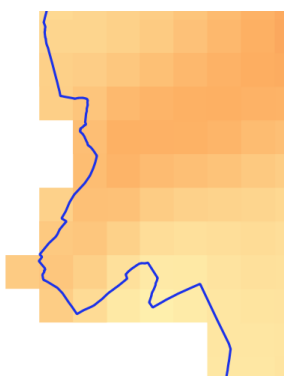


2.2.2 Rogner et masquer une couche raster (crop)

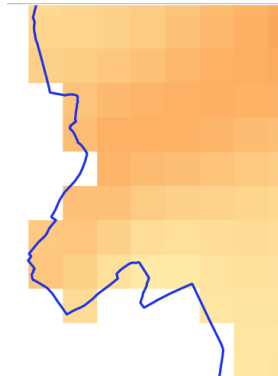
- L'option « `mask=TRUE` » complète l'opération de rognage avec l'application d'un masque en utilisant les limites de la couche de référence.



- L'option « touches » concerne les modalités d'application du masque. L'option par défaut (touches = T) intègre tous les pixels qui sont « touchés » par le masque. Pour l'option « touches = F », seuls les pixels dont le centre se trouve dans le masque sont pris en compte. On préfère généralement la seconde option qui est statistiquement plus représentative du polygone de masque.



Touches = T (default)



Touches = F

- En appliquant l'option « touches = F », le nombre de pixels situés hors du masque (valeurs NA) passe de 50876 à 52006.

```
mnt_nam2 = crop(mnt,nam, mask=T,touches=F)
summary(mnt_nam2)
> summary(mnt_nam2)
  mnt_250m
Min.   : 64.22
1st Qu.:180.48
Median :234.17
Mean   :236.74
3rd Qu.:276.64
Max.   :502.97
NA's   :52006
```

2.2.3 Reprojecter ou ré-échantillonner une couche raster

- La fonction **project()** est utilisée pour reprojeter une couche raster, c'est-à-dire changer le système de coordonnées (CRS) du raster.

```
# 3.3 Reprojecter un raster
st_crs(temp)==st_crs(mnt)
f_temp=paste0(path_out,"/temp_31370.tif")
temp_31370=project(temp, mnt, method="bilinear",filename=f_temp,overwrite=T)
```

- L'option « **method** » définit le type de ré-échantillonnage mis en œuvre lors de l'opération de re-projection.

```
st_bbox(temp)
st_bbox(temp_31370)
st_bbox(mnt)
```

```
> st_bbox(temp)
      xmin      ymin      xmax      ymax
2.516667 49.483333 6.433333 51.525000
> st_bbox(temp_31370)
      xmin      ymin      xmax      ymax
4000     6500    312000    260000
> st_bbox(mnt)
      xmin      ymin      xmax      ymax
4000     6500    312000    260000
```



- **Remarque importante** : la fonction **project()** peut également être utilisée pour ré-échantillonner un raster SANS changer son système de coordonnées. Elle peut ainsi être utilisée pour remplir le même rôle que la fonction **resample()**.

2.2.4 Agrégation - désagrégation

- Les fonctions d'agrégation ou de désagrégation effectuent un ré-échantillonnage sans changer l'emprise globale du raster.
- La fonction **aggregate()** va regrouper les pixels du raster de départ sur base d'un facteur d'agrégation. Dans les exemples qui suivent, le facteur d'agrégation est fixé à 2. Cela signifie que chaque pixel de la couche de sortie résulte de l'agrégation de 2x2 pixels de la couche d'entrée. Le paramètre « fun » désigne la manière avec laquelle les valeurs des cellules de la couche d'entrée sont agrégées (fonction d'agrégation). Alors que les fonctions de ré-échantillonnage ou de reprojection considèrent la valeur du plus proche voisin ou réalisent une moyenne pondérée des pixels les plus proches, la fonction **aggregate()** peut mettre en œuvre une très grande diversité de fonction d'agrégation comme le montrent les exemples qui suivent.
- Dans le premier exemple, la fonction d'agrégation est la moyenne.
- Remarque : on notera que dans le raster de sortie, la couche porte le même nom que la couche originale (mnt_250m). Pour que le nom de couche soit cohérent avec le contenu du raster, il peut s'avérer utile de renommer la couche.

```
# 3.4 Aggregate et Disaggregate -----
# Aggregate
mnt500_mean=aggregate(mnt, fact=2, fun=mean)
mnt500_mean
> mnt500_mean
class       : SpatRaster
dimensions  : 507, 616, 1  (nrow, ncol, nlyr)
resolution  : 500, 500  (x, y)
extent      : 4000, 312000, 6500, 260000  (xmin, xmax, ymin, ymax)
coord. ref. : Belge 1972 / Belgian Lambert 72 (EPSG:31370)
source      : memory
name        : mnt_250m
min value   : -183.379
max value   : 698.4906
names(mnt500_mean)="mnt500_mean"
```

- Le second exemple agrège les données avec la fonction d'agrégation « max ». Ces 2 premiers exemples utilisent des fonctions d'agrégation prédéfinies. Les fonctions prédéfinies disponibles sont : mean (moyenne), max (maximum), min (minimum), median (médiane), sum (somme), sd (écart-type) et modal (valeur la plus fréquente).

```
mnt500_max=aggregate(mnt,fact=2,fun=max)
mnt500_max
```

- L'exemple qui suit utilise une fonction personnalisée qui va agréger les pixels d'entrée en calculant le 90^{ème} percentile.

```
mnt500_q90 = aggregate(mnt,fact=2,
                        fun = function(x) quantile(x, probs = 0.9, na.rm = T))
```



- Remarque importante** : les fonctions utilisant les quantiles ou la médiane peuvent entraîner des temps de calculs très longs car elles impliquent de devoir trier les valeurs des pixels.
- La fonction **disagg()** opère en sens inverse de la fonction **aggregate()** : elle découpe les pixels de la couche d'entrée sur base de la valeur du paramètre « fact » : dans l'exemple qui suit, les pixels de 500 m de côté sont découpés en 4 (2x2) pixels, chacun de ceux-ci recevant la valeur du pixel original.

```
# Disaggregate
mnt125=disagg(mnt,fact=2,method="near")
mnt125
```

2.2.5 Créer un raster de toutes pièces (template)

- Très souvent, en début de projet, on doit construire une couche raster dont les propriétés géométriques vont servir de référence pour tous les rasters qui seront générés dans la suite du projet. Ces propriétés géométriques sont l'emprise spatiale (bbox), le système de coordonnées (CRS) et la résolution spatiale.
- Dans l'exemple qui suit, on génère un raster template dont l'emprise correspond à la province de Namur, le CRS est « epsg :3812 » (projection Lambert belge 2008) et la résolution spatiale est de 1 km.

```
# 3.5 Créer un raster de toute pièce -----
# paramètres à définir :
# - emprise (couvrir la province de Namur)
# - CRS : EPSG:3812 (Lambert belge 2008)
# - résolution spatiale : 1 km

# step 1 : générer l'objet SpatRaster
r0=rast() #
r0
# step 2 : définir son CRS
crs(r0)="epsg:3812"
```

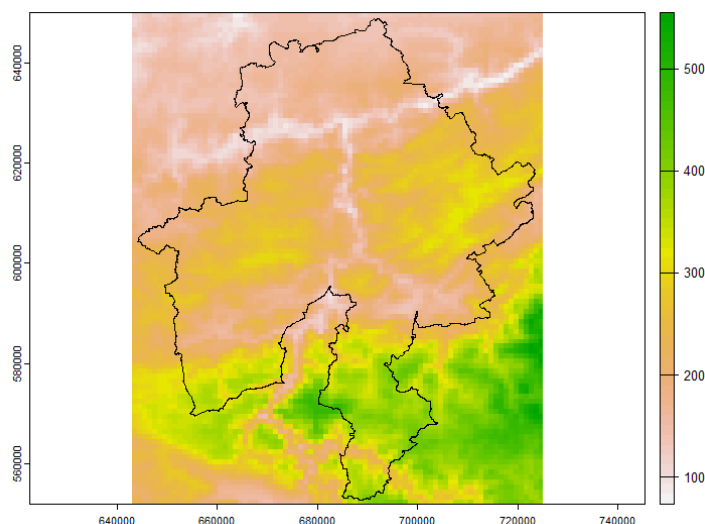
```
# step 3 : définir l'emprise
nam_3812=project(nam,"epsg:3812") # reprojette "nam" avec epsg:3812
ext=ext(nam_3812)
ext
# adapter l'emprise à la résolution
res0=1000 # résolution souhaitée
ext[1]=round(ext[1]/res0,0)*res0-res0
ext[2]=round(ext[2]/res0,0)*res0+res0
ext[3]=round(ext[3]/res0,0)*res0-res0
ext[4]=round(ext[4]/res0,0)*res0+res0
ext
# fixer l'emprise
ext(r0)=ext
r0
# step 4 : définir la résolution
res(r0)= res0
r0

> r0
class       : SpatRaster
dimensions  : 98, 82, 1 (nrow, ncol, nlyr)
resolution  : 1000, 1000 (x, y)
extent      : 643000, 725000, 552000, 650000 (xmin, xmax, ymin, ymax)
coord. ref. : ETRS89 / Belgian Lambert 2008 (EPSG:3812)
```

- Le template qui vient d'être créé est ensuite utilisé pour générer 1 couche mnt à 1 km de résolution.

```
# exemple d'utilisation du raster r0 pour
# créer un nouveau raster
mnt_1km = project(mnt, r0, method="bilinear")
names(mnt_1km)="mnt"

plot(mnt_1km)
plot(nam_3812,add=TRUE)
```



2.3 Algèbre raster (fonctions locales)

- Le terme « algèbre raster » fait référence aux fonctions « locales » qui effectuent des opérations logiques ou arithmétiques à l'échelle des pixels sans tenir compte de leur voisinage (fonctions focales) ou de l'ensemble du raster (fonctions globales).

```
# 4. Algèbre raster (fonctions locales) -----

# lecture des données
f_temp=paste0(path_out,"/temp_31370.tif")
temp=rast(f_temp) # températures mensuelles
f_mnt=paste0(path_in,"/mnt_250m.tif")
mnt=rast(f_mnt)

# Vérifier la cohérence géométrique des couches
ext(mnt)==ext(temp)
res(mnt)==res(temp)
```

- Lorsque des opérations d'algèbre raster mettent en œuvre plusieurs couches, il est important de s'assurer de la concordance géométrique de ces couches.

```
> mnt
class       : SpatRaster
dimensions  : 1014, 1232, 1  (nrow, ncol, nlyr)
resolution  : 250, 250  (x, y)
extent      : 4000, 312000, 6500, 260000  (xmin, xmax, ymin, ymax)
coord. ref. : Belge 1972 / Belgian Lambert 72 (EPSG:31370)
source      : mnt_250m.tif
name        : mnt_250m
min value   : -197.413
max value   : 702.726

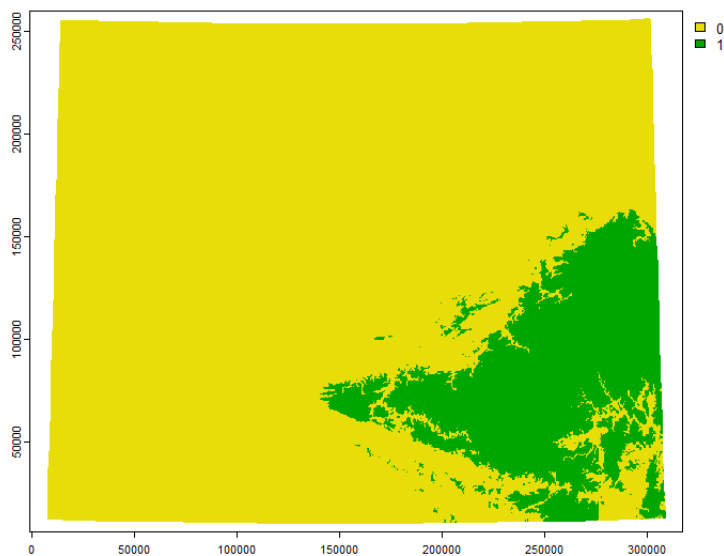
> temp
class       : SpatRaster
dimensions  : 1014, 1232, 12 (nrow, ncol, nlyr)
resolution  : 250, 250  (x, y)
extent      : 4000, 312000, 6500, 260000  (xmin, xmax, ymin, ymax)
coord. ref. : Belge 1972 / Belgian Lambert 72 (EPSG:31370)
source      : temp_31370.tif
names       : temp_mens_1, temp_mens_2, temp_mens_3, temp_mens_4, temp_mens_5,
```

2.3.1 Requête

- Les requêtes sont des opérations qui vont sélectionner les pixels répondant à un ou plusieurs critère(s) exprimé(s) par rapport à une ou plusieurs couche(s) raster.
- Le moyen le plus simple pour effectuer ces requêtes est de construire des expressions utilisant les opérateurs logiques classiques (>, <, ==, !=..) avec des objets SpatRaster et des valeurs ou des variables numériques. Les résultats de ces expressions sont des objets SpatRaster contenant des valeurs 1 ou 0 et éventuellement des NODATA.

3.1 Requêtes -----

```
res(mnt)
# Requête simple
r = mnt>300
summary(r)
plot(r)
> summary(r)
  mnt_250m
Min.   :0.000
1st Qu.:0.000
Median :0.000
Mean   :0.156
3rd Qu.:0.000
Max.   :1.000
NA's   :8307
Warning message:
[summary] used a sample
```



- La fonction *ifel()* offre plus de possibilités, comme le montrent les exemples qui suivent. Elle permet notamment de sauvegarder directement le résultat dans 1 fichier avec l'option « filename ».

```
# Fonction ifel()
r = ifel(mnt>300,1,0)
plot(r)
plot(prov$geom,add=TRUE)

f_out=paste0(path_out,"/mnt_gt300.tif")
r = ifel(mnt>300,1,0,filename=f_out,
        overwrite=T,wopt=opt1)

r = ifel(mnt>300,1,NA)
plot(r)

r = ifel(mnt>300,mnt,NA)
plot(r)

r= ifel(mnt>300 & temp$temp_mens_1<1,1,0,
        filename=f_out,overwrite=T,wopt=opt1)
plot(r)
```

2.3.2 Opérations mathématiques

- Comme pour les requêtes, la manière la plus simple d'appliquer des fonctions mathématiques à des couches rasters est d'utiliser les fonctions mathématiques de base (+,-,*,/...) avec des variables SpatRaster et des variables numériques.

```
# 3.2. Opérations mathématiques -----
r1 = log(mnt)
hist(r1)

f_temp=paste0(path_out,"/temp_31370.tif")
temp=rast(f_temp)
r2= log(mnt)*temp$temp_mens_6
hist(r2)
hist(mnt)
```

2.3.3 Fonctions mathématiques

- Si l'on souhaite sauvegarder directement le résultat d'une opération mathématique, il faut faire appel à 1 fonction (fun) en utilisant la fonction **lapp()**. Dans l'exemple qui suit, on utilise 1 image multispectrale Sentinel-2 contenant les bandes B2 (bleu), B3 (vert), B4 (rouge) et B8 (Proche infrarouge) pour calculer un indice de végétation NDVI. Pour stocker le résultat sous format entier 16-bit, on procède à une standardisation du NDVI. Le résultat s'exprime alors sur une échelle allant de 0 à 2000.

```
# 3.3 Fonctions mathématiques -----
f_in=paste0(path_in,"/s2_20160628_2348.tif")
s2=rast(f_in)
names(s2)
names(s2)=c("B2","B3","B4","B8")
# calcul du NDVI : (B8-B4)/(B8+B4)
ndvi_fun = function(nir, red){
  ((nir - red) / (nir + red)) * 1000 + 1000
}
f_out=paste0(path_out,"/ndvi_20160628.tif")
ndvi = lapp(s2[[c(4, 3)]], fun = ndvi_fun,
           filename=f_out,overwrite=T,wopt=optINT2S)

ndvi
hist(ndvi)

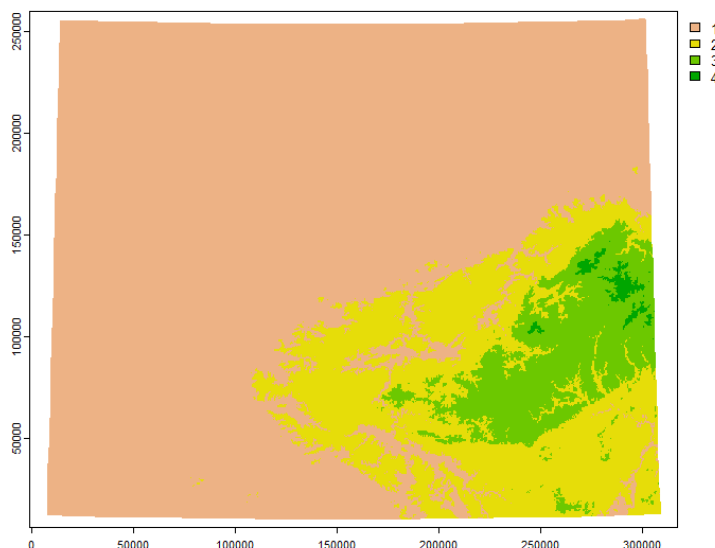
> ndvi
class       : SpatRaster
dimensions  : 473, 611, 1  (nrow, ncol, nlyr)
resolution  : 10, 10  (x, y)
extent      : 658060, 664170, 5518940, 5523670  (xmin, xmax, ymin, ymax)
coord. ref. : WGS 84 / UTM zone 31N (EPSG:32631)
source      : ndvi_20160628.tif
name        : lyr1
min value   : 849
max value   : 1948
```

2.3.4 Création de classes

- La création de classes au départ d'un raster contenant une variable continue est une opération très courante.
- Dans l'exemple qui suit, la couche « mnt » qui représente l'altitude, est convertie en un raster représentant des classes d'altitude numérotées de 1 à 4 : 1 : ≤ 200 , 2 : $200 < \leq 400$, 3 : $400 < \leq 600$, 4 : $600 <$. Par défaut les classes sont « fermées à droite » ($\min < \leq \max$). La définition de classes « fermées à gauche » nécessite de fixer l'option « right=FALSE ». Les classes ouvertes utilisent les valeurs -Inf et Inf comme bornes.
- Les paramètres pour la définition des classes sont organisés sous la forme d'une matrice.

```
# 3.4. Création de classes -----
summary(mnt)
rc1 = matrix(c(-Inf, 200, 1, 200, 400, 2, 400, 600, 3, 600, Inf, 4),
             ncol = 3, byrow = TRUE)
rc1
> rc1
  [,1] [,2] [,3]
[1,] -Inf 200  1
[2,] 200 400  2
[3,] 400 600  3
[4,] 600 Inf  4

cl_alti = classify(mnt, rc1 = rc1)
plot(cl_alti)
```



- Comme les classes sont représentées par des nombres entiers positifs < 255, le résultat peut être sauvegardé avec un codage « INT1U » (entier 8-bit).

```
optINT1U = list(gdal=c("INTERLEAVE=BAND", "TILED=YES", "BIGTIFF=YES",
                      "COMPRESS=DEFLATE", "ZLEVEL=9", "NUM_THREADS=ALL_CPUS"),
               datatype='INT1U')
f_out=paste0(path_out, "/cl_altitude.tif")
cl_alti = classify(mnt, rc1 = rc1, filename=f_out, overwrite=T, wopt=optINT1U)
```

2.4 Calcul de statistiques

- Le calcul de statistiques sur des couches raster est un volet important du traitement de ce type de données. Il convient de distinguer 2 manières d'envisager ces calculs :
 - Produire une statistique descriptive globale pour 1 raster, par exemple calculer pour un mnt l'altitude moyenne sur l'ensemble de la zone couverte par le MNT.
 - Produire 1 nouvelle couche raster dans laquelle 1 pixel contient le résultat d'une fonction statistique calculée sur les pixels correspondant dans 1 série de couches rasters : par exemple calculer sur les 12 couches raster correspondant aux températures moyennes mensuelles, la température moyenne annuelle.
- Lors de ces traitements, il convient d'être attentif à la présence de valeur nodata et à la manière de les gérer. Classiquement, on les retire du calcul avec l'option « na.rm=TRUE ».

2.4.1 Statistiques globales (1 raster → 1 valeur)

- La fonction **global()** est utilisée pour calculer 1 statistique globale à l'échelle d'un raster. L'option « na.rm = T » fait en sorte que les pixels sans données ne sont pas pris en compte lors de ces traitements.
- Sans cette option la présence de nodata entraîne un résultat incorrect.

```
# 4.1 Statistiques globales (sur les valeurs d'1 couche)
# -> renvoi 1 valeur numérique simple (dataframe !!)

# Moyenne
moy = global(mnt,"mean",na.rm=T)
moy
class(moy)

> moy = global(mnt,"mean",na.rm=T)
> moy
              mean
mnt_250m 140.0499
> class(moy)
[1] "data.frame"
```

- On remarque que le résultat est stocké dans 1 dataframe de taille 1x1. Il convient de convertir celui-ci en valeur numérique si on souhaite l'utiliser dans 1 calcul ultérieur.

```
# Convertir le résultat en variable numérique pour la suite
mnt_rel = mnt/moy # erreur
mnt_rel = mnt/as.numeric(moy)
hist(mnt_rel)
```

- Les exemples qui suivent illustrent le calcul d'autres variables statistiques globales : l'écart-type, le maximum, le minimum et différents quantiles.

```
# Ecart-type
(sd = as.numeric(global(mnt, fun = "sd", na.rm=T)))

# Max & min
(max = as.numeric(global(mnt, fun = "max", na.rm=T)))
(min = as.numeric(global(mnt, fun = "min", na.rm=T)))
(quant = as.numeric(global(mnt, quantile , na.rm=T))) # probs : 0.1,0.25,0.5,0.75,0.9

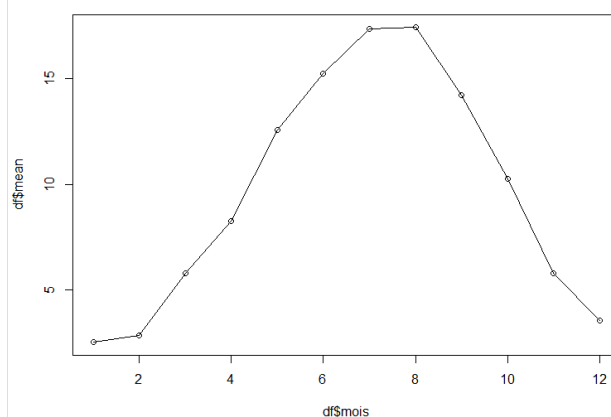
# Quantile (tdigest package)
library(tdigest)
(q50 = tquantile(tdigest(na.omit(mnt[])), probs = .5))
(quant = tquantile(tdigest(na.omit(mnt[])), probs = c(0,1,0.25, 0.5,0.75,0.9)))
```

- Lorsque l'on utilise la fonction **global()** sur un raster multi-couches, le dataframe de sortie contient 1 colonne avec 1 ligne par couche.

```
# Statistiques globale pour un raster multi-couches
df=global(temp,"mean",na.rm=T)
df$mois=c(1:12)
df
plot(df$mois,df$mean)
lines(df$mois,df$mean)

> df
```

	mean	mois
temp_mens_1	2.532420	1
temp_mens_2	2.863876	2
temp_mens_3	5.793298	3
temp_mens_4	8.263486	4
temp_mens_5	12.574658	5
temp_mens_6	15.252987	6
temp_mens_7	17.377351	7
temp_mens_8	17.420915	8
temp_mens_9	14.230771	9
temp_mens_10	10.267193	10
temp_mens_11	5.787497	11
temp_mens_12	3.582865	12



2.4.2 Statistiques locales (x rasters → 1 raster)

- Terra a implémenté une série de fonctions qui calculent des statistiques locales sur une piles de couches rasters. Ces fonctions sont : max, min, mean, median, prod, range, stdev, sum.

```
# 4.2 Statistiques locales (multi-bandes) -----
# -> renvoi 1 couche avec 1 valeur par pixel
t_moy=mean(temp)
t_stdev=stdev(temp)
t_med=median(temp,na.rm=T)
t_min=min(temp)
t_max=max(temp)

# quantile : génère 5 couches (q0,q25,q50,q100)
t_quant=quantile(temp,na.rm=T) # génère 5 couches : q0,q25,q50,q100
names(t_quant)
```

2.4.3 Corrélations globales

- La fonction **layerCor()** est utilisée pour calculer un coefficient de corrélation entre 2 couches raster, ou une matrice de corrélation entre une série de couches rasters.

```
# 4.3 Corrélation entre couches raster -----
layerCor(c(t_moy,mnt),fun="cor",na.rm=T)

> layerCor(c(t_moy,mnt),fun="cor",na.rm=T)
$correlation
      mean  mnt_250m
mean  1.0000000 -0.9636927
mnt_250m -0.9636927  1.0000000

$mean
      mean  mnt_250m
mean    NaN  9.658253
mnt_250m 140.0499    NaN

$n
      mean  mnt_250m
mean    NaN   972511
mnt_250m 1146086    NaN
```

2.4.4 Calculs de surfaces par classes

- La fonction **table()** peut être utilisée sur le vecteur des valeurs du raster **cl_alti** pour calculer un vecteur de fréquence des différentes classes présentes dans ce raster.
- Ces fréquences sont ensuite converties en surface en les multipliant par la surface des pixels fournies par la fonction **res()**.
- A noter que la colonne « Freq » qui contient les codes de classes est de type « factor ». Il peut être utile de la convertir en valeurs de type « numérique » ou « texte » pour un traitement ultérieur.

```

# 4.4 Surface par classe -----
tab1=as.data.frame(table(cl_alti[]))
tab1
pix_sqkm=prod(res(cl_alti))/1000000
tab1$sqkm=tab1$Freq*pix_sqkm
tab1
class(tab1$Var1)
> tab1
  Var1   Freq   sqkm
1    1 838134 52383.375
2    2 212482 13280.125
3    3  90504  5656.500
4    4   4966   310.375

tab1$class=as.integer(as.character(tab1$Var1))
tab1=dplyr::select(tab1,class,sqkm)
tab1
> tab1
  class   sqkm
1     1 52383.375
2     2 13280.125
3     3  5656.500
4     4   310.375

```

2.5 Extraction de valeurs au sein d'un raster

2.5.1 Extraction par points

- La fonction **extract()** est utilisée pour extraire des données d'un raster en considérant les objets d'une couche vectorielle contenue dans 1 objet de la classe SpatVector.
- Il faut veiller à la concordance des systèmes de coordonnées des 2 couches.
- Dans l'exemple qui suit, l'altitude est extraite de la couche « mnt » pour toutes les localités contenues dans le shapefile **localites.shp**.

```

# 5.1 Extraction -> points -----
f_in=paste0(path_in,"/localites.shp")
loc=vect(f_in) # utiliser 1 objet SpatVector
crs(loc)="epsg:31370"
df=terra::extract(mnt,loc)
loc$alt=df$mnt_250m
loc=loc[order(-loc$alt),]
head(loc)
> head(loc)
  ID  NOM one alt
1  85 Elsenborn 1 628.3987
2  83 Honsfeld 1 591.9815
3  92 Sourbrodt 1 586.6351
4  82 Bullange 1 584.5409
5  84 Butgenbach 1 583.3713
6 720 Chabrehez 1 579.0922

```

- A noter que par défaut, cette fonction extrait la valeur du pixel dans lequel se trouve le point. Il est possible de recourir à une interpolation bilinéaire en le spécifiant avec l'option « method ». Dans ce cas, la valeur attribuée au point est calculée par interpolation entre les valeurs des 4 pixels les plus proches du point.
- On constate que les résultats obtenus sont légèrement différents.

```
# Interpolation bilinéaire
df=terra::extract(mnt,loc,method="bilinear")
loc$alt=df$mnt_250m
loc=loc[order(-loc$alt),]
head(loc)

> head(loc)
```

	ID	NOM	one	alt
1	85	Elsenborn	1	626.3534
2	83	Honsfeld	1	591.9773
3	92	Sourbrodt	1	586.5268
4	84	Butgenbach	1	582.3289
5	82	Bullange	1	581.0288
6	720	Chabrehez	1	580.3869

2.5.2 Extraction par polygones (statistiques zonales)

- Lorsque la fonction d'extraction est utilisée avec 1 couche de polygones, on doit également préciser la fonction d'agrégation qui sera utilisée pour traiter les pixels contenus dans les polygones. Les fonctions par défaut sont les mêmes que celles présentées précédemment pour le calcul de statistiques globales : mean, max, min, sum...

```
# 5.2 Extraction -> polygones -----
f_in=paste0(path_in,"/communes.shp")
comm=vect(f_in)

df1=terra::extract(mnt,comm,fun="mean")
df2=terra::extract(mnt,comm,fun="max")
df3=terra::extract(mnt,comm,fun="min")
comm$alt_moy=df1$mnt_250m
comm$alt_max=df2$mnt_250m
comm$alt_min=df3$mnt_250m
comm$alt_range=comm$alt_max-comm$alt_min
comm=comm[order(-comm$alt_range),]
df=dplyr::select(as.data.frame(comm),ADMUNAFR,alt_moy:alt_range)
head(df)
```

- Cette extraction s'apparente aux opérations de statistique zonale.

2.6 Conversions SpatRaster ↔ SpatVector

- Les outils de conversion entre données rasters et vectorielles constituent également des maillons importants dans les chaînes de géotraitement.

2.6.1 SpatRaster → SpatVector

- Sauf cas particulier, la conversion d'une couche raster en polygones n'a vraiment de sens que lorsque ce raster contient des groupes de pixels de même valeur. C'est par exemple le cas avec la couche de classes d'altitude produite au § 4.4. Il faut veiller à ne pas oublier le paramètre « dissolve=TRUE » pour rassembler tous les pixels d'une même valeur en 1 seul polygone.

```
# 6.1 SpatRaster -> SpatVector -> sf -----
cl_alti=rast(paste0(path_out,"/cl_altitude.tif")) # classes d'altitude
pol = as.polygons(cl_alti,dissolve=T,na.rm=T)
class(pol)
pol = st_as_sf(pol) # convertir en objet sf
f_out=paste0(path_out,"/cl_altitude.gpkg")
st_write(pol,f_out,delete_layer = T)
> nrow(pol)
[1] 4
```

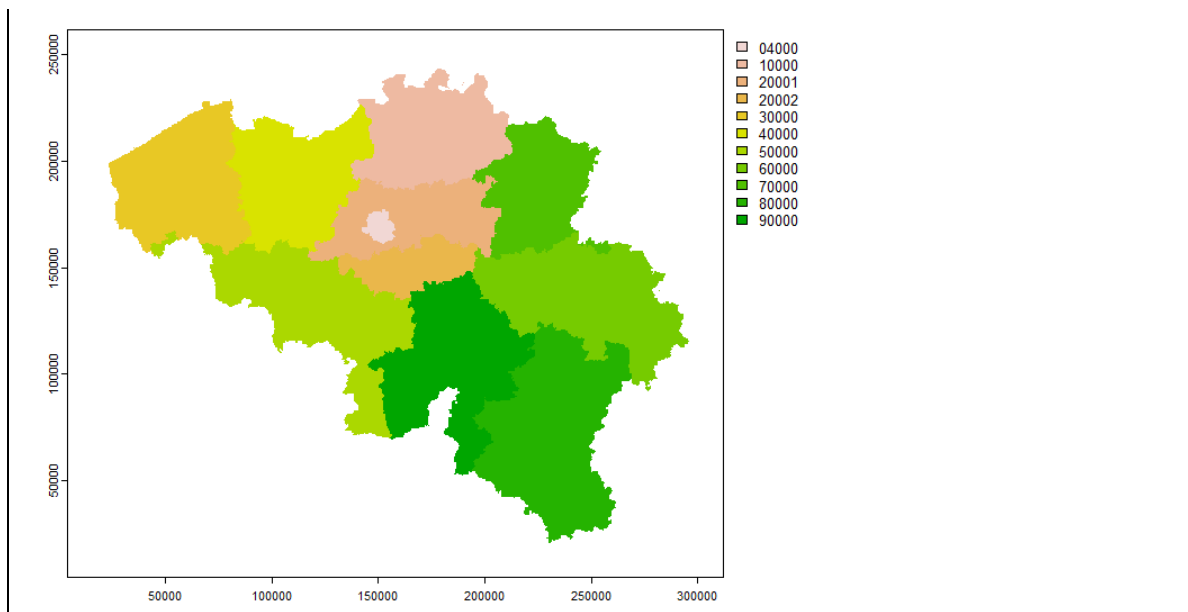
- Dans certains cas particuliers, on peut souhaiter convertir les pixels d'un raster en 1 couche de points. L'exemple qui suit génère 1 couche de points au départ du raster mnt_1km.

```
# raster -> couche de points
mnt_1km=rast(paste0(path_out,"/mnt_1km.tif")) # voir § 3.5
pnt=as.points(mnt_1km,na.rm=T)
pnt=st_as_sf(pnt)
nrow(pnt)
> nrow(pnt)
[1] 8036
```

2.6.2 SpatVector → SpatRaster

- La transformation inverse (vecteur → raster) est aussi très utilisée. Elle est mise en œuvre avec la fonction **rasterize()**. L'option « field » définit le champ du dataframe, ou la valeur numérique qui est utilisée pour coder les pixels.

```
# 6.2 SpatVector -> SpatRaster -----
# créer 1 raster avec le code de la province (ADPRKEY)
f_in=paste0(path_in,"/provinces.gpkg")
f_out=paste0(path_out,"/province.tif")
prov=vect(f_in)
class(prov$AdPrKey)
r=terra::rasterize(prov,mnt,field="AdPrKey",
                    filename=f_out,overwrite=T)
plot(r)
> class(prov$AdPrKey)
[1] "character"
```



- L'exemple présenté a la particularité que le champ utilisé est de type « character ». Cela se traduit par la création d'un raster « catégoriel » : les pixels contiennent des valeurs numériques qui sont reliées aux valeurs originales par une table de correspondance. Cette dernière est accessible avec la fonction **cats()**.

```
# liste des valeurs numériques contenues dans le raster
table(r[])

# table de concordance entre valeurs numériques et classes
cats(r)

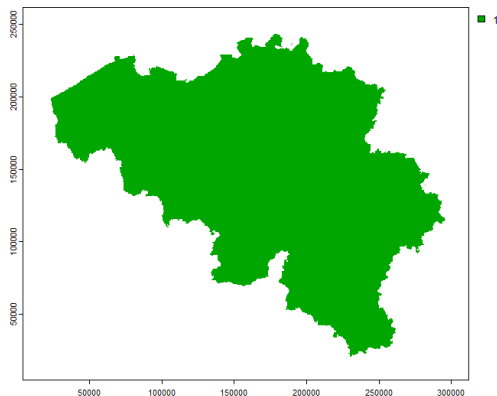
> table(r[])

 0    1    2    3    4    5    6    7    8    9   10   11
2597 46018 33895 17557 50662 48122 61025 61675 38851 71375 58795 470

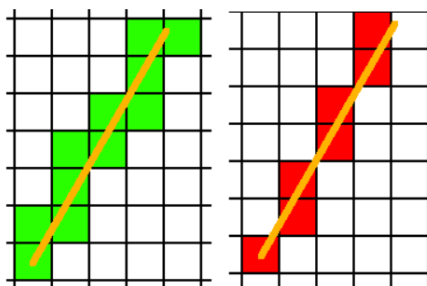
> cats(r)
[[1]]
  value AdPrKey
1      0  04000
2      1  10000
3      2  20001
4      3  20002
5      4  30000
6      5  40000
7      6  50000
8      7  60000
9      8  70000
10     9  80000
11    10  90000
12    11  ZZ000
```

- Le second exemple montre comment créer un masque raster correspondant à l'ensemble du territoire belge. Dans ce cas, l'option « field » est utilisée pour attribuer la valeur 1 à tous les pixels situés dans les polygones de la couche « prov ».

```
# créer 1 masque raster pour la Belgique
f_out=paste0(path_out,"/belgique.tif")
r=terra::rasterize(prov,mnt,field=1,
                  filename=f_out,overwrite=T)
summary(r)
plot(r)
```



- Le troisième exemple considère 1 couche de lignes. Dans cas, il peut être utile de recourir à l'option « touches=TRUE » pour prendre en compte un nombre plus important de pixels. La figure suivante illustre la différence entre un raster produit avec (pixels verts) ou sans (pixels rouges) l'option « touches=T ».



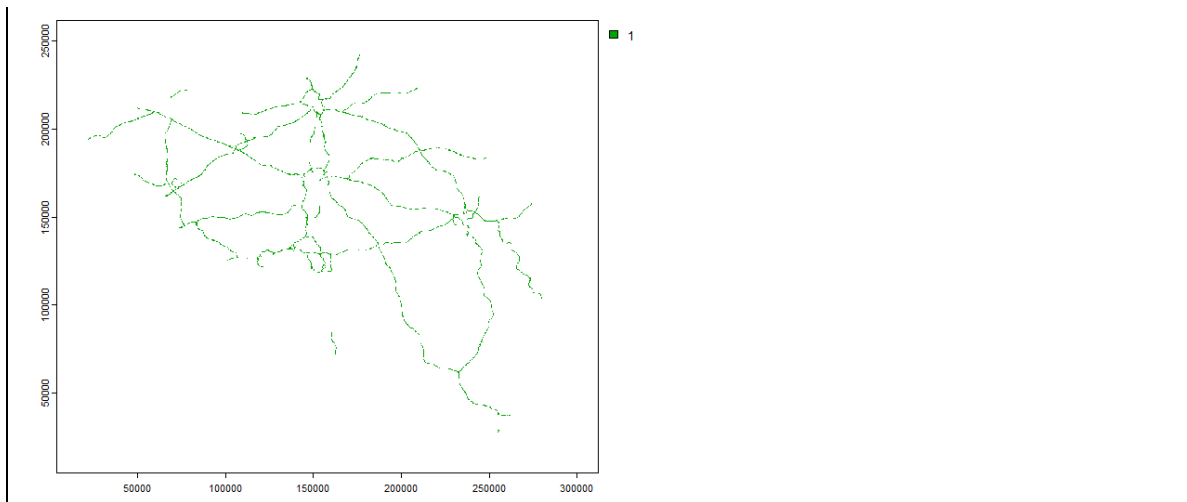
```
# convertir une couche de lignes -> raster
f_in=paste0(path_in,"/motorways_osm.gpkg")
roads=vect(f_in)
plot(roads)
f_out=paste0(path_in,"/motorways.tif")
r=terra::rasterize(roads,mnt,field=1,
                  filename=f_out,overwrite=T)
plot(r)
table(r[])

> table(r[])

 1
10286
r=terra::rasterize(roads,mnt,field=1,touches=T,
                  filename=f_out,overwrite=T)
table(r[])

> table(r[])

 1
9174
```

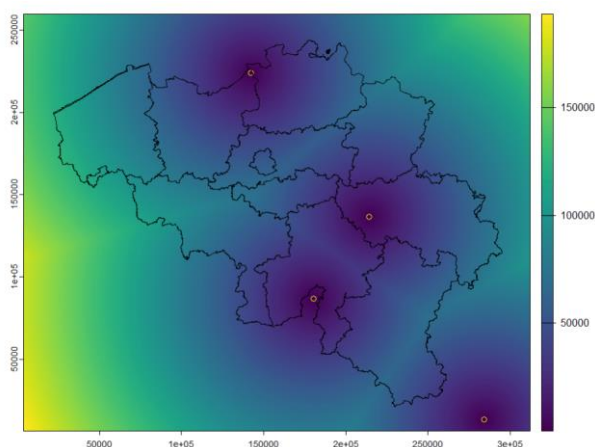


2.7 Calculs de distances euclidiennes

- La fonction **distance()** s'applique à une couche vectorielle en considérant 1 raster de référence. Dans l'exemple ci-dessous, on détermine la distance à la centrale nucléaire la plus proche en utilisant la couche du fichier **centrales_nucleaires.gpkg** et le template du raster mnt.

```
# distance aux centrales nucléaires
f_in=paste0(path_in,"/centrales_nucleaires.gpkg")
f_out=paste0(path_out,"/d2cnuc.tif")
cnuc=st_read(f_in)
cnuc=st_transform(cnuc,31370)

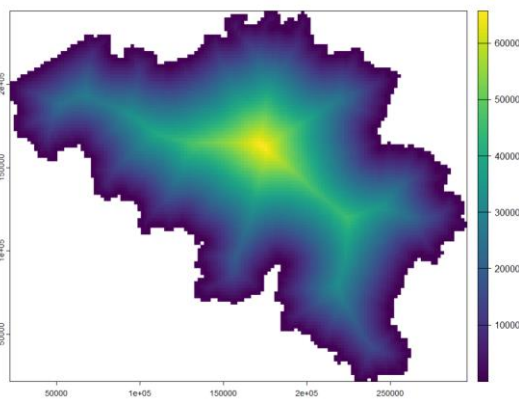
d2cnuc=distance(mnt,cnuc,filename=f_out,overwrite=T)
plot(d2cnuc)
plot(cnuc$geom,add=T,col="yellow")
f_prov=paste0(path_in,"/provinces.gpkg")
prov=st_read(f_prov)
plot(prov$geom,add=T)
```



- Le second exemple calcule la distance euclidienne par rapport à des lignes représentant la frontière belge. La résolution du raster « template » a été dégradée à 4 km pour réduire le temps de calcul.

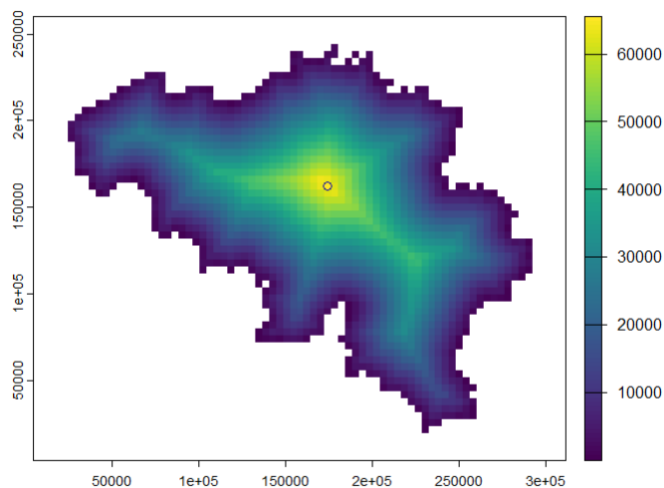
```
# Distance à la frontière belge

# step 1 : agréger provinces -> pays
f_prov=paste0(path_in,"/provinces.gpkg")
prov=st_read(f_prov)
pays=summarize(prov)
# step 2 : pays (polygone) -> frontière (ligne)
frontiere=st_cast(pays,"POLYGON")
frontiere=st_cast(frontiere,"LINESTRING") # polygone -> lignes
# step 3 : calcul de la distance à la frontière
grid_4km=aggregate(mnt,fact=16,fun=mean) # grid 4k résolution -> + rapide
d2bel=distance(grid_4km,pays) # distance au polygone (0 à l'intérieur)
d2front=distance(grid_4km,frontiere) # dist à la frontière
d2front=ifel(d2bel==0,d2front,NA) # dist à la frontière à l'intérieure du pays
plot(d2front)
```



- Cette carte de distance est ensuite utilisée pour localiser le point situé à l'intérieur du pays et qui est le plus éloigné de la frontière.

```
# Trouver l'endroit en Belgique qui est le + éloigné de la frontière
# on recherche le pixel qui a la valeur de distance maximum
center=d2front==as.numeric(global(d2front,fun="max",na.rm=T))
center=ifel(center==1,1,NA)
# on converti le pixel en 1 point
center=as.points(center) # spatvector
center=st_as_sf(center) # sf
plot(center,add=T)
```



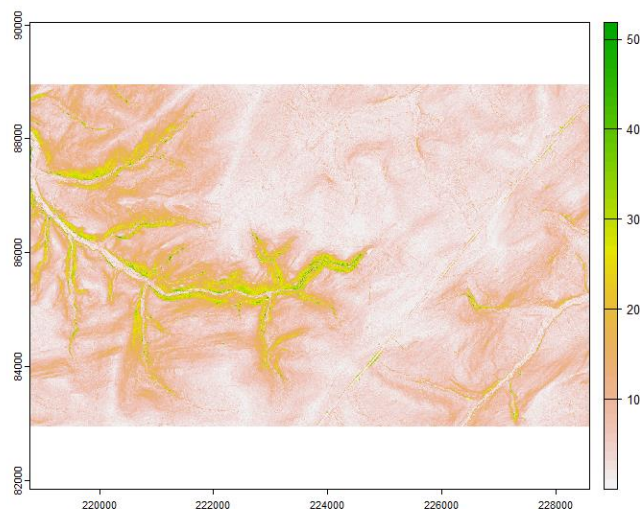
2.8 Pente, exposition, ombrage

- La fonction **terrain()** est utilisée pour générer des couches de pente ou d'exposition au départ d'un Modèle Numérique de Terrain. Elle comporte plusieurs options relatives à la variable à calculer (v= « slope » ou v= « aspect »), aux unités dans lesquelles le résultat est exprimé (unit = « degrees » , unit = « radians ») ou encore aux algorithmes utilisés pour le calcul de la pente ou de l'exposition en considérant 4 ou 8 voisins (neighbors=4 ou 8).

```
# 8. Pente, exposition, ombrage -----

# Lire le mnt (mnt à 2 m de résolution)
f_mnt=paste0(path_in,"/mnt_2m.tif")
mnt=rast(f_mnt)
plot(mnt)

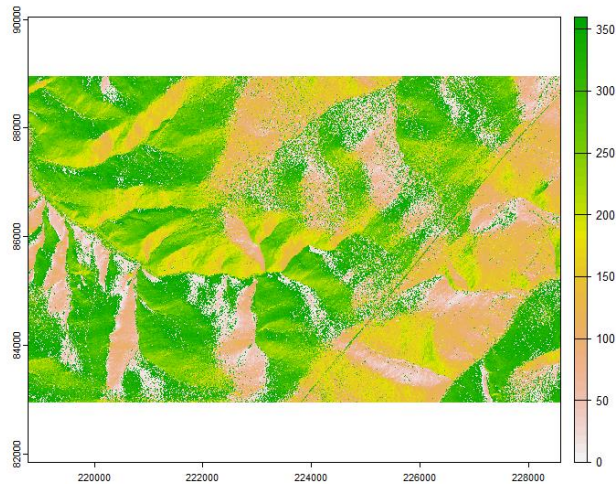
# Pente
f_out=paste0(path_out,"/pente_4d.tif")
pente_4d=terra::terrain(mnt,v="slope",unit="degrees",
                        filename=f_out,overwrite=T,neighbors=4)
plot(pente_4d)
f_out=paste0(path_out,"/pente_8d.tif")
pente_8d=terra::terrain(mnt,v="slope",unit="degrees",
                        filename=f_out,overwrite=T,neighbors=8)
plot(pente_8d)
```



```
diff=pente_8d-pente_4d
summary(diff)
> summary(diff)
      slope
  Min.   :-8.62129
 1st Qu.: -0.49340
  Median: -0.09914
   Mean  :-0.14116
 3rd Qu.:  0.24892
   Max.   :  7.10783
Warning message:
[summary] used a sample
```

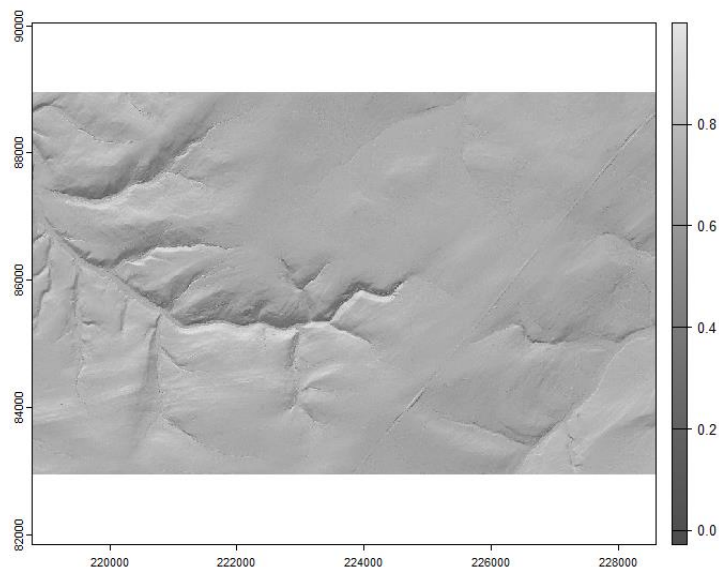
- On constate que les 2 algorithmes produisent des résultats très similaires.

```
# Exposition
f_out=paste0(path_out, "/aspect.tif")
expo=terra::terrain(mnt, v="aspect", unit="degrees",
                    filename=f_out, overwrite=T, neighbors=4)
plot(expo)
```



- La fonction **shade()** génère une couche d'ombrage (hillshade). Les couches entrantes (pente et exposition) doivent être exprimées en radians.

```
# Hillshade
f_out=paste0(path_out, "/hillshade.tif")
pente=terra::terrain(mnt, v="slope", unit="radians")
expo=terra::terrain(mnt, v="aspect", unit="radians")
ombre=shade(pente, expo, angle=45, direction=345,
            filename=f_out, overwrite=TRUE)
plot(ombre, col=gray.colors(255, gamma=0.1))
```



2.9 Fonctions focales (filtres)

- Les fonctions focales aussi appelées « filtres » représentent un volet très important des outils de traitement des données raster. Nous l'aborderons ici de manière très succincte, avec quelques exemples qui illustrent les principales fonctions à mettre en œuvre, soit au sein de la librairie **terra**, soit en utilisant des librairies plus spécifiques comme **mmand** utilisée pour l'application de filtres morphologiques.
- Une présentation détaillée des techniques de filtrage et concepts mathématiques qu'elles recouvrent sort du cadre de ce tutoriel.

2.9.1 Exemple 1 : filtre de lissage

- Lorsqu'une couche raster comporte des variations locales qui peuvent être assimilées à des bruits, le moyen le plus simple pour atténuer ces derniers est d'appliquer un filtre de lissage.
- L'application d'un filtre sur un raster s'effectue à l'aide de la fonction **focal()**. Les paramètres importants à définir sont d'une part la taille de la fenêtre (paramètres « w ») et d'autre part la fonction (« fun ») qui est appliquée sur les pixels contenus dans la fenêtre.
- Dans le premier exemple, le paramètre « w » est défini avec une valeur numérique impaire (3, ou 7) qui fixe la taille de la fenêtre (3 x 3 ou 7x7) dans laquelle va être appliquée une fonction (fun = « mean »). La couche de sortie contient donc en chaque pixel, la moyenne des pixels contenus dans 1 fenêtre de 3x3 ou 7x7 centrée sur chacun des pixels. L'option « na.rm=TRUE » fait en sorte que les données nodata ne sont pas prises en compte par la fonction.
- On peut observer l'effet du lissage sur les valeurs maximum de la couche originale et des couches lissées.

```
# Exemple 1 : filtre de lissage (lissage de la pente) ----
f_in=paste0(path_out,"/pente_8d.tif") # créé au § 10
pente = rast(f_in)
plot(pente)

pente_sm3=focal(pente, w=3 ,fun="mean" , na.rm=TRUE)
plot(pente_sm3)

pente_sm7=focal(pente, w=7 ,fun="mean" , na.rm=TRUE)

global(pente,fun="max",na.rm=T)
global(pente_sm3,fun="max",na.rm=T)
global(pente_sm7,fun="max",na.rm=T)

> global(pente,fun="max",na.rm=T)
      max
slope 59.8745
> global(pente_sm3,fun="max",na.rm=T)
      max
focal_mean 54.27885
> global(pente_sm7,fun="max",na.rm=T)
      max
focal_mean 45.60864
```

2.9.2 Exemple 2 : filtre de lissage Gaussien

- Il est possible de remplacer la valeur numérique définissant la fenêtre du filtre, par une matrice de « poids ». C'est par exemple le cas des filtres gaussiens qui sont aussi fréquemment utilisés pour lisser des rasters.
- Il convient dans ce cas de créer au préalable cette matrice de poids dont la somme est égale à 1. La matrice est construite avec la fonction **focalMat()**.
- Dès lors que l'on utilise une matrice de poids, la fonction « fun » qui est mise en œuvre dans le filtre n'est plus la moyenne, mais bien une sommation.

```
# Exemple 2 : filtre gaussien (lissage de la hauteur) ----
f_in=paste0(path_in,"/mnh_20cm.tif")
mnh=rast(f_in)
plot(mnh)
gf= focalMat(mnh, d=0.15, type=c('Gauss'), fillNA=FALSE) # Créer la matrice
gf # matrice
sum(gf)
mnh_sm=focal(mnh, w=gf ,fun="sum" , na.rm=TRUE)
plot(mnh_sm)
summary(mnh)
summary(mnh_sm)

> gf # matrice
      [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.0002310286 0.003324944 0.008087679 0.003324944 0.0002310286
[2,] 0.003324944 0.047852321 0.116397203 0.047852321 0.003324944
[3,] 0.008087679 0.116397203 0.283127519 0.116397203 0.008087679
[4,] 0.003324944 0.047852321 0.116397203 0.047852321 0.003324944
[5,] 0.0002310286 0.003324944 0.008087679 0.003324944 0.0002310286
> sum(gf)
[1] 1
```

- A la différence des filtres de lissage simple du premier exemple, le filtre gaussien attribue plus d'importance aux pixels situés au centre de la fenêtre.

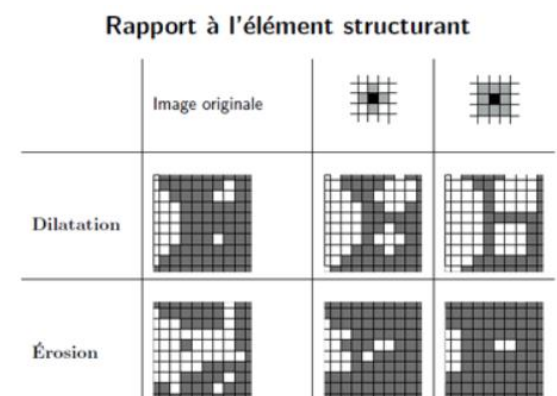
2.9.3 Exemple 3 : filtres morphologiques

- Les filtres morphologiques sont généralement utilisés sur des couches raster binaires issues d'un processus de classification ou de seuillage. Ces filtres vont agir sur les groupes de pixels de valeur 1 en ajoutant/retirant un certain nombre de couches de pixels à la périphérie des groupes. Ces opérations permettent ainsi de nettoyer les rasters pour améliorer la morphologie des groupes de pixels de valeur 1.



- Remarque importante** : les filtres morphologiques fonctionnent sur des couches raster binaire (0/1) et non sur des couches avec 1 seule valeur (NA/1).
- Les filtres morphologiques se déclinent en 4 opérations :
 - Dilatation : ajout de « couche(s) » de pixels de valeur « 1 » à la périphérie des groupes de pixels « 1 ».

- Erosion : suppression de « couches(s) » de pixels de valeur « 1 » à la périphérie des groupes de pixels « 1 ».
- Ouverture : application successive d'une érosion et d'une dilatation. Utilisée pour supprimer les petits groupes de valeur « 1 ».
- Fermeture : application successive d'une dilatation et d'une érosion. Utilisée pour supprimer les petits groupes de valeur « 0 ».
- Ces opérations morphologiques sont mises en œuvre dans les exemple qui suivent à l'aide de fonctions issues de la librairie **mmand** (Mathematical Morphology in Any Number of Dimensions). Les fonctions **dilate()**, **erode()**, **opening()** et **closing()** sont appliquées sur des objets de type matrice.
- Le kernel (fenêtre) qui est mis en œuvre est généré au préalable avec la fonction **shapeKernel()**. Les paramètres de celle-ci concernent la taille de la fenêtre (une fenêtre de 3x3 va agir sur 1 couche de pixel, une fenêtre de 5x5 sur 2 couches de pixel...) et la forme de l'élément structurant contenu dans la fenêtre. Celui-ci définit les pixels qui sont impactés par le filtre morphologique comme le montre la figure suivante (tirée de <http://vokvince.free.fr/IMAC/TS/morphologie.html>). Les formes d'éléments structurants proposées par la fonction **shapeKernel()** sont « box », « disc » et « diamond ».



- Après avoir appliqué le filtre, il convient de transformer la matrice « résultat » en sens inverse pour produire un objet SpatRaster. Cette opération est réalisée avec la fonction **setValues()**.
- Dans l'exemple qui suit, un raster constitué de valeurs 0/1 est traité avec les 4 opérateurs morphologiques

```
# Exemple 3 : filtres morphologiques
library(mmand)

f_in=paste0(path_in,"/test.tif")
test=rast(f_in)
plot(test)

m=as.matrix(test,wide=T)
k <- shapeKernel(c(3,3), type="disc")
k|
k <- shapeKernel(c(3,3), type="diamond")
k
```

```
> k <- shapeKernel(c(3,3), type="diamond")
> k
      [,1] [,2] [,3]
[1,]    0    1    0
[2,]    1    1    1
[3,]    0    1    0
attr(,"class")
[1] "kernelArray" "kernel"

# dilate
d=dilate(m,k)
test_d= setValues(test, d)

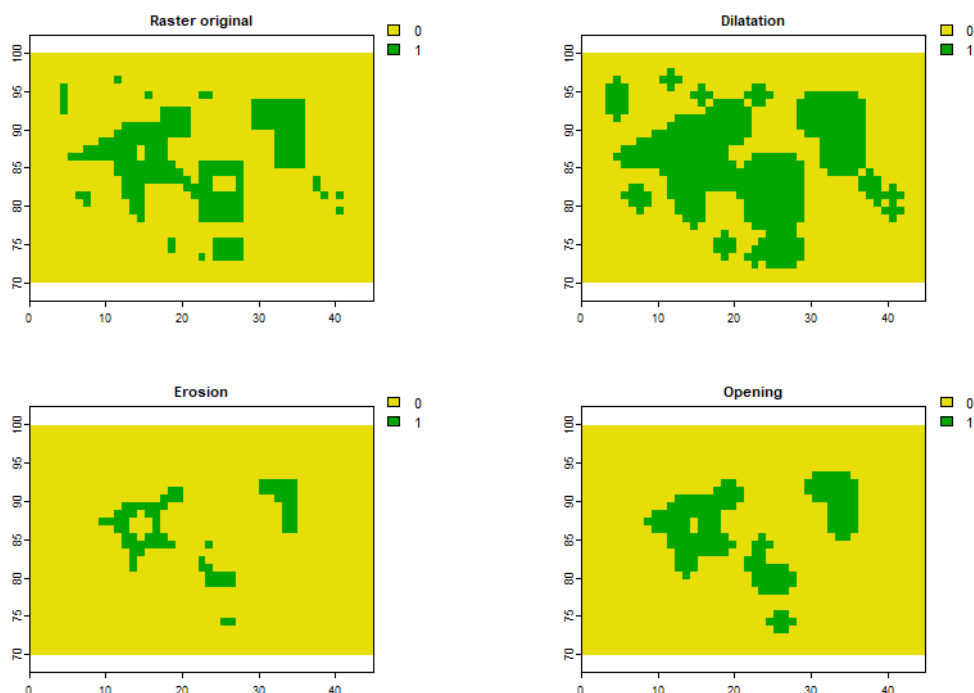
diff=test_d*10+test
plot(diff)

# Erosion
e=erode(m,k)
test_e= setValues(test, e)
diff=test_e*10+test
plot(diff)

# Opening : érosion puis dilatation
o=opening(m,k)
test_o= setValues(test, o)
plot(test_o)

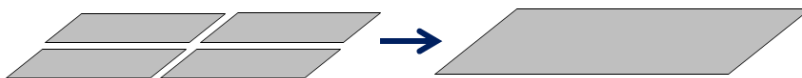
# Closing : dilatation puis érosion
c=closing(m,k)
test_c= setValues(test, c)
plot(test_c)

par(mfrow=c(2,2))
plot(test, main="Raster original")
plot(test_d, main="Dilatation")
plot(test_e, main="Erosion")
plot(test_o, main="Opening")
```

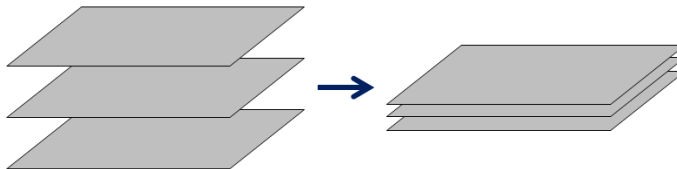


2.10 Rasters virtuels (vrt)

- Les rasters virtuels sont des outils de gestion de données rasters particulièrement efficaces.
- Ils sont implémentés dans la librairie GDAL et rendus accessibles dans l'environnement R au travers de la librairie **gdalUtils**.
- Le principe de ces rasters virtuels (vrt) est de générer des produits dérivés de couches raster (c'est-à-dire des nouvelles couches raster) sans devoir recréer physiquement de nouveaux fichiers raster, mais simplement en définissant sous la forme de fichiers texte très légers les caractéristiques des nouvelles couches et la manière avec laquelle elles peuvent être générées au départ des rasters originaux. Le principal avantage réside dans le gain d'espace disque (qui peut se chiffrer en centaines de Go) et de temps de calcul nécessaire à la création de nouveaux fichiers tif.
- Globalement, la création de rasters virtuels correspond à 2 types d'assemblage de couches raster :
 - le **mosaïquage** qui rassemble différentes couches jointives (tuiles) en 1 couche continue (mosaïque)



- l'**empilement** qui rassemble différentes couches couvrant la même emprise en 1 pile de couches



- Ces assemblages peuvent être complétés de différentes opérations comme le rognage (modifier l'emprise) ou le rééchantillonnage (modifier la résolution).
- Nous allons illustrer l'intérêt de ces rasters virtuels (vrt) au travers de quelques exemples simples.

Exemple 1 : Mosaïquage de rasters mono-couche

- Dans ce premier exemple, on considère un ensemble de rasters mono-couches organisés en tuiles contigües correspondant à un MNT. Ceux-ci sont assemblés en 1 mosaïque continue, comme schématisé dans la figure suivante.



- Ce mosaïquage est réalisé en 2 étapes. La première consiste à dresser la liste des fichiers tif correspondant aux tuiles à assembler, à l'aide de la fonction **list.files()** assortie de l'option « *full.names=TRUE* ». Dans notre exemple, cette liste comporte 20 fichiers.

```
# Exemple 1 : mosaïquage de rasters mono-bandes (exemple : MNT)

path_mnt=paste0(path_in,"/tuiles_mnt")

# step 1 : créer 1 liste des fichiers à assembler
list_file=list.files(path_mnt, pattern="*.tif$",full.names = TRUE)
length(list_file)
head(list_file)

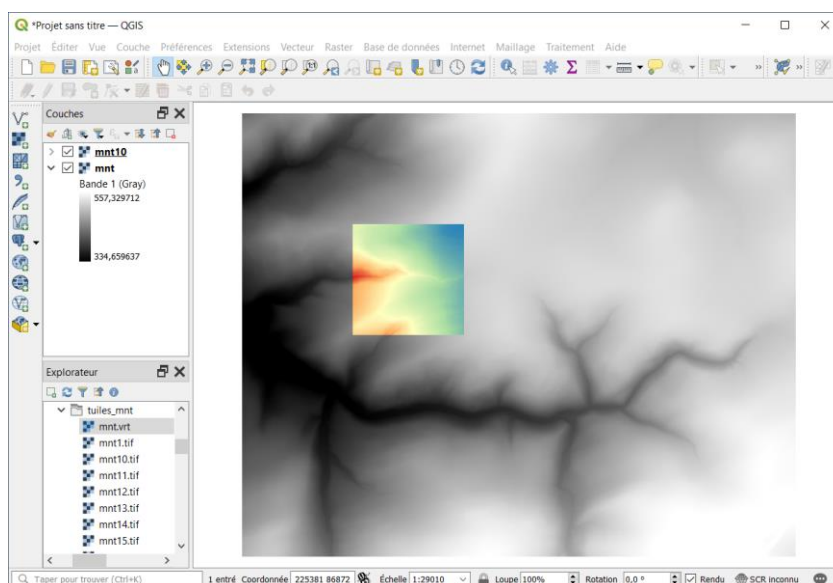
> length(list_file)
[1] 20
> head(list_file)
[1] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_mnt/mnt1.tif"
[2] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_mnt/mnt10.tif"
[3] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_mnt/mnt11.tif"
[4] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_mnt/mnt12.tif"
[5] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_mnt/mnt13.tif"
[6] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_mnt/mnt14.tif"
```

- Le fichier vrt est ensuite construit à l'aide de la fonction **sf::gdal_utils()**.

```
# step 2 : générer le raster virtuel
f_vrt=paste0(path_mnt,"/mnt.vrt")
sf::gdal_utils(util = "buildvrt", source = list_file, destination = f_vrt)

mnt=rast(f_vrt)
plot(mnt)
```

- Le résultat du mosaïquage se présente sous la forme d'un fichier vrt qui peut être utilisé comme source de données raster dans QGIS. La figure qui suit illustre l'affichage du fichier vrt dans QGIS. Le raster affiché en pseudo-couleurs correspond à 1 des tuiles.



- Le fichier vrt peut être ouvert avec n'importe quel éditeur de texte. Il est structuré avec un format xml. Il comporte différentes rubriques renseignant les propriétés du raster virtuel : sa taille, son système de coordonnées, les bandes (couches) qu'il contient, ainsi que les sources de données qui sont utilisées pour générer ces bandes.
- Le fichier vrt qui vient d'être produit correspond à 1 raster dont la taille est de 2500 x 2000 pixels. Il est basé sur le CRS « Lambert belge 1972 » et comporte 1 bande codée en Float32. Cette couche unique est constituée d'une série de fichiers sources (mnt1.tif, mnt10.tif, mnt11.tif...) dont la taille est de 500x500. La position relative de chaque tuile est indiquée avec la balise <DstRect xOff yOff>.

```

<VRTDataset rasterXSize="2500" rasterYSize="2000">
  <SRS>PROJCS["unnamed",GEOGCS["International 1909 (Hayford)",DATUM["unknown",SPHEROID["intl",6378388,297],TOWGS84[
  <GeoTransform> 2.2000100000000000e+005, 2.0000000000000000e+000, 0.0000000000000000e+000, 8.8000000000000000e+004
  <VRTRasterBand dataType="Float32" band="1">
    <NoDataValue>-3.4e+038</NoDataValue>
    <ColorInterp>Gray</ColorInterp>
    <ComplexSource>
      <SourceFilename relativeToVRT="1">mnt1.tif</SourceFilename>
      <SourceBand>1</SourceBand>
      <SourceProperties RasterXSize="500" RasterYSize="500" DataType="Float32" BlockXSize="500" BlockYSize="4" />
      <SrcRect xOff="0" yOff="0" xSize="500" ySize="500" />
      <DstRect xOff="500" yOff="0" xSize="500" ySize="500" />
      <NODATA>-3.4e+038</NODATA>
    </ComplexSource>
    <ComplexSource>
      <SourceFilename relativeToVRT="1">mnt10.tif</SourceFilename>
      <SourceBand>1</SourceBand>
      <SourceProperties RasterXSize="500" RasterYSize="500" DataType="Float32" BlockXSize="500" BlockYSize="4" />
      <SrcRect xOff="0" yOff="0" xSize="500" ySize="500" />
      <DstRect xOff="500" yOff="500" xSize="500" ySize="500" />
      <NODATA>-3.4e+038</NODATA>
    </ComplexSource>
    <ComplexSource>
      <SourceFilename relativeToVRT="1">mnt11.tif</SourceFilename>
      <SourceBand>1</SourceBand>
      <SourceProperties RasterXSize="500" RasterYSize="500" DataType="Float32" BlockXSize="500" BlockYSize="4" />
      <SrcRect xOff="0" yOff="0" xSize="500" ySize="500" />
      <DstRect xOff="500" yOff="500" xSize="500" ySize="500" />
      <NODATA>-3.4e+038</NODATA>
    </ComplexSource>
  </ComplexSource>
</VRTDataset>

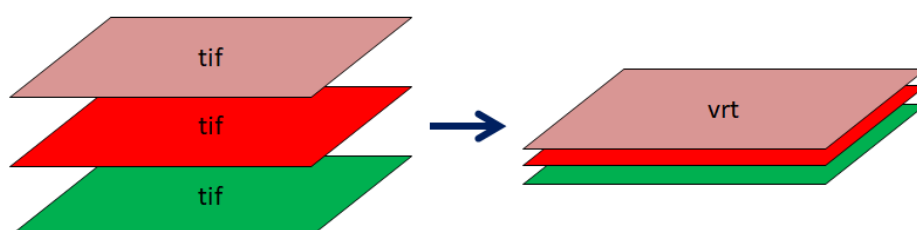
```



- **Remarque importante :** pour assurer la cohérence entre le fichier vrt et les données sources, il est conseillé de sauvegarder le fichier vrt dans le même répertoire que les données sources.

Exemple 2 : Empilement de rasters mono-couches dans 1 raster multi-couches

- Ce deuxième exemple illustre l'empilement de rasters mono-couches dans 1 vrt multi-couches. Cette opération est souvent utilisée pour empiler les bandes spectrales d'une image satellitaire.



- Dans l'exemple qui suit, les bandes B03, B04 et B08 d'une image Sentinel 2 sont assemblées pour former un raster multispectral utilisable pour l'affichage d'une composition colorée dans QGIS. Les fichiers utilisés dans cet exemple sont extraits de l'image S2A_MSIL2A_20190920T104021_N0213_R008_T31UFR_20190920T120330.SAFE téléchargée sur le site <https://scihub.copernicus.eu/>.
- La démarche suivie est très semblable à celle utilisée pour le mosaïquage : on génère d'abord une liste de fichiers qui sont ensuite assemblés avec la fonction `gdalbuildvrt()`. La différence réside dans l'utilisation de l'option « **-separate** » qui spécifie que les différents fichiers seront placés dans des couches séparées.

```
# Exemple 2 : empilement de rasters mono-bandes -> raster multi-bandes

path_s2=paste0(path_in,"/sentinel2")

# step 1 : créer 1 liste des fichiers à assembler
list_file=list.files(path_s2, pattern="*.tif$",full.names = TRUE)
list_file

> list_file
[1] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/sentinel2/s2_20190920_B03.tif"
[2] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/sentinel2/s2_20190920_B04.tif"
[3] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/sentinel2/s2_20190920_B08.tif"

# step 2 : générer le raster virtuel : !! SEPARATE = TRUE
f_vrt=paste0(path_s2,"/s2.vrt")
sf::gdal_utils(util = "buildvrt", source = list_file,
               destination = f_vrt, options = c("-separate"))

s2=rast(f_vrt)
plotRGB(s2,scale=6000,stretch="lin",r=3,g=2,b=1)
```



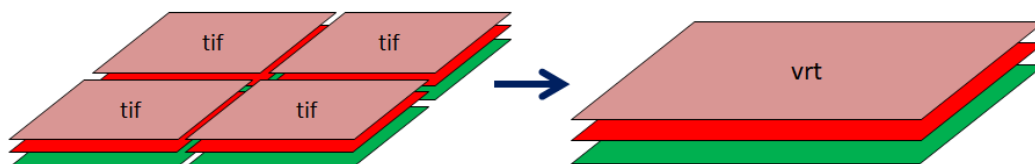
- Si l'on observe le contenu du fichier vrt, on retrouve les informations suivantes :
 - Taille du raster : 1030 colonnes x 650 lignes
 - Système de coordonnées : WGS84 - UTM 31N

- Codage des pixels : Entier 16-bit non signés (UInt16)
- Nombre de bandes : 3
- Bande 1 : fichier source : s2_20190920_B03.tif
- Bande 2 : fichier source : s2_20190920_B04.tif
- Bande 3 : fichier source : s2_20190920_B08.tif

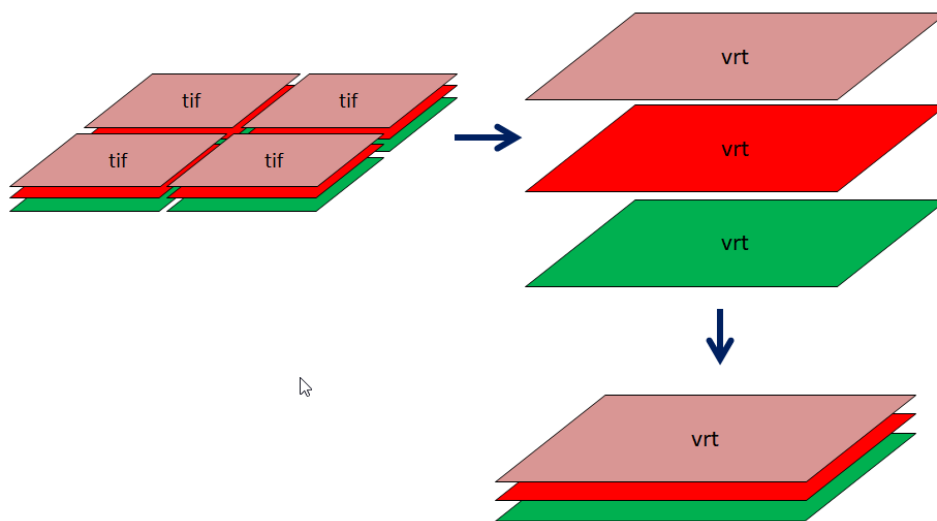
```
s2.vrt - Bloc-notes
Fichier Edition Format Affichage Aide
<VRTDataset rasterXSize="1030" rasterYSize="650">
  <SRS>PROJCS["WGS 84 / UTM zone 31N",GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,AUTH
  <GeoTransform> 6.665000000000000e+005, 1.000000000000000e+001, 0.000000000000000e+000, 5.553600000000000e+006
  <VRTRasterBand dataType="UInt16" band="1">
    <SimpleSource>
      <SourceFilename relativeToVRT="1">s2_20190920_B03.tif</SourceFilename>
      <SourceBand>1</SourceBand>
      <SourceProperties RasterXSize="1030" RasterYSize="650" DataType="UInt16" BlockXSize="1030" BlockYSize="4" />
      <SrcRect xOff="0" yOff="0" xSize="1030" ySize="650" />
      <DstRect xOff="0" yOff="0" xSize="1030" ySize="650" />
    </SimpleSource>
  </VRTRasterBand>
  <VRTRasterBand dataType="UInt16" band="2">
    <SimpleSource>
      <SourceFilename relativeToVRT="1">s2_20190920_B04.tif</SourceFilename>
      <SourceBand>1</SourceBand>
      <SourceProperties RasterXSize="1030" RasterYSize="650" DataType="UInt16" BlockXSize="1030" BlockYSize="4" />
      <SrcRect xOff="0" yOff="0" xSize="1030" ySize="650" />
      <DstRect xOff="0" yOff="0" xSize="1030" ySize="650" />
    </SimpleSource>
  </VRTRasterBand>
  <VRTRasterBand dataType="UInt16" band="3">
    <SimpleSource>
      <SourceFilename relativeToVRT="1">s2_20190920_B08.tif</SourceFilename>
      <SourceBand>1</SourceBand>
      <SourceProperties RasterXSize="1030" RasterYSize="650" DataType="UInt16" BlockXSize="1030" BlockYSize="4" />
      <SrcRect xOff="0" yOff="0" xSize="1030" ySize="650" />
      <DstRect xOff="0" yOff="0" xSize="1030" ySize="650" />
    </SimpleSource>
  </VRTRasterBand>
</VRTDataset>
```

Exemple 3 : Mosaïquage d'images multi-bandes

- Ce troisième exemple combine les opérations de mosaïquage et d'empilement.
- L'objectif est de créer une mosaïque au départ d'une série d'images multispectrales organisées en tuiles jointives.



- Le format vrt ne permet pas de mosaïquer des images multi-bandes.
- Il faut donc procéder en deux étapes : premièrement, on génère des mosaïques « mono-bandes » qui sont ensuite empilées sous la forme d'une mosaïque multi-bandes.



- Avant d'aborder la première étape, on construit la liste des fichiers d'entrées.

```
# Exemple 3 : tuiles d'images multi-bandes -> 1 mosaïque multi-bandes

# On procède en 2 étapes :
# step 1 : création d'1 mosaïque pour chaque bande (.vrt monobandes)
# step 2 : empilement des vrt monobandes -> 1 vrt multi-bandes

# créer la liste des images à traiter
path_ortho=paste0(path_in,"/tuiles_ortho")
list_file=list.files(path_ortho, pattern="*.tif$",full.names = TRUE)
list_file

> list_file
[1] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_ortho/tuiles_ortho.0.tif"
[2] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_ortho/tuiles_ortho.1.tif"
[3] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_ortho/tuiles_ortho.2.tif"
[4] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_ortho/tuiles_ortho.3.tif"
[5] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_ortho/tuiles_ortho.4.tif"
[6] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_ortho/tuiles_ortho.5.tif"
[7] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_ortho/tuiles_ortho.6.tif"
[8] "C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_ortho/tuiles_ortho.7.tif"
```

- La création des mosaïques mono-bandes nécessite d'utiliser l'option « b » qui permet d'extraire une bande spécifique d'un raster multi-bande. Cette opération est intégrée dans 1 boucle qui va traiter chacune des 3 bandes. L'indice i de la boucle est utilisé dans l'option « -b » de la fonction **gdal_utils()**.
- En vue de réaliser la seconde étape, une liste contenant les noms des 3 fichiers vrt produits lors de la première étape est créée (variable list_vrt).

```
# step 1 : mosaïquer les bandes IR,R,G des 8 images dans 3 vrt

list_vrt=NULL # liste qui va contenir les noms des 3 fichiers .vrt
for (i in 1:3){
  f_vrt=paste0(path_ortho,"/band",i,".vrt")
  sf::gdal_utils(util = "buildvrt", source = list_file, destination = f_vrt,
                 options = c("-b", i))
  list_vrt=rbind(list_vrt,f_vrt)
}
class(list_vrt)
list_vrt=list_vrt[,1] # crée une liste de fichiers
> list_vrt
                                file_vrt
"C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_ortho/band1.vrt"
                                file_vrt
"C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_ortho/band2.vrt"
                                file_vrt
"C:/PL/01_COURS/tthr_2022/R_GIS_02/data_in/tuiles_ortho/band3.vrt"
```

- La seconde étape utilise à nouveau la commande **gdal_utils()** pour cette fois empiler les 3 bandes dont les fichiers vrt sont repris dans la liste « list_vrt ». L'empilement est assuré par l'option « -separate ».

```
# step 2 : empiler les 3 vrt pour créer un vrt multispectral
f_vrt=paste0(path_ortho,"/ortho_IRRG.vrt")
sf::gdal_utils(util = "buildvrt", source = list_vrt, destination = f_vrt,
               options = c("-separate"))
```

- Le fichier vrt final est constitué de 3 bandes dont les sources de données respectives sont les fichiers band1.vrt, band2.vrt et band3.vrt produits à l'étape précédente.

```
ortho_IRRG.vrt - Bloc-notes
Fichier Edition Format Affichage Aide
<VRTDataset rasterXSize="6327" rasterYSize="3940">
  <SRST>LOCAL_CS["Belge 1972 / Belgian Lambert 72",GEOGCS["Belge 1972",DATUM["unknown",SPHEROID["unretrievable - usi
  <GeoTransform> 1.6510000000000000e+005, 3.000638912596807e+000, 0.0000000000000000e+000, 1.4557000000000000e+005
  <VRTRasterBand dataType="Byte" band="1">
    <NoDataValue>0</NoDataValue>
    <ComplexSource>
      <SourceFilename relativeToVRT="1">band1.vrt</SourceFilename>
      <SourceBand>1</SourceBand>
      <SourceProperties RasterXSize="6327" RasterYSize="3940" DataType="Byte" BlockXSize="128" BlockYSize="128" />
      <SrcRect xOff="0" yOff="0" xSize="6327" ySize="3940" />
      <DstRect xOff="0" yOff="0" xSize="6327" ySize="3940" />
      <NODATA>0</NODATA>
    </ComplexSource>
  </VRTRasterBand>
  <VRTRasterBand dataType="Byte" band="2">
    <NoDataValue>0</NoDataValue>
    <ComplexSource>
      <SourceFilename relativeToVRT="1">band2.vrt</SourceFilename>
      <SourceBand>1</SourceBand>
      <SourceProperties RasterXSize="6327" RasterYSize="3940" DataType="Byte" BlockXSize="128" BlockYSize="128" />
      <SrcRect xOff="0" yOff="0" xSize="6327" ySize="3940" />
      <DstRect xOff="0" yOff="0" xSize="6327" ySize="3940" />
      <NODATA>0</NODATA>
    </ComplexSource>
  </VRTRasterBand>
  <VRTRasterBand dataType="Byte" band="3">
    <NoDataValue>0</NoDataValue>
    <ComplexSource>
      <SourceFilename relativeToVRT="1">band3.vrt</SourceFilename>
      <SourceBand>1</SourceBand>
      <SourceProperties RasterXSize="6327" RasterYSize="3940" DataType="Byte" BlockXSize="128" BlockYSize="128" />
      <SrcRect xOff="0" yOff="0" xSize="6327" ySize="3940" />
      <DstRect xOff="0" yOff="0" xSize="6327" ySize="3940" />
      <NODATA>0</NODATA>
    </ComplexSource>
  </VRTRasterBand>
</VRTDataset>
```

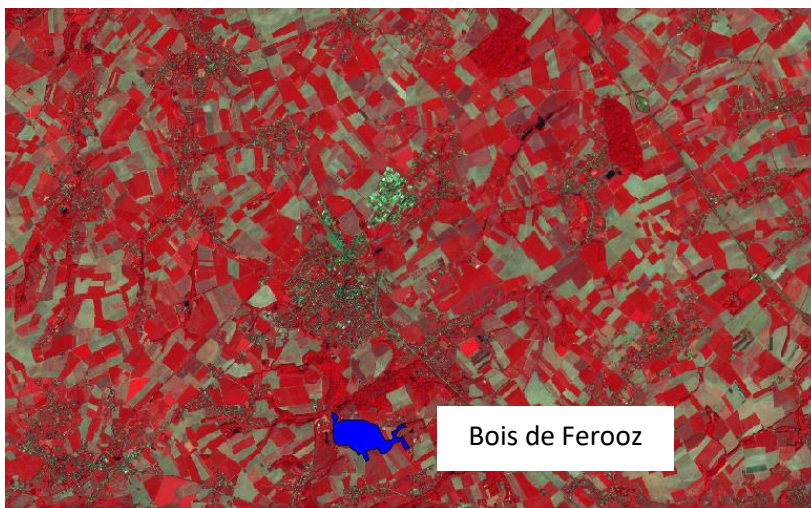
```
# Afficher le résultat
ortho=rast(file_vrt)
plotRGB(ortho)
```



Exemple 4 : Empilement avec changement de résolution et d'emprise

- Pour ce dernier exemple, nous allons combiner différentes sources de données et mettre à profit les options de modification d'emprise et de résolution.
- Les données d'entrée sont constituées de l'orthomosaïque qui vient d'être produite dans l'exemple précédent et dont la résolution spatiale est de 3 cm.
- On souhaite produire 1 nouveau raster virtuel contenant la même ortho-mosaïque, mais avec 1 résolution spatiale de 2 m et 1 emprise spatiale correspondant au Bois de Ferooz qui est situé sur le territoire de la commune de Gembloux.

```
# Utiliser l'emprise du bois de Ferooz (commune de Gembloux)
f_bois=paste0(path_in,"/bois_ferooz.gpkg")
bois=vect(f_bois)
plot(bois,add=T,col="blue")
```



- Une variable « bbox » contenant des coordonnées (xmin, ymin, xmax, ymax) qui englobent la forêt est définie. Elle sera utilisée ultérieurement dans la fonction **gdalbuildvrt()**.

```
> st_bbox(bois)
      xmin      ymin      xmax      ymax
172725.4 135031.3 174515.8 136181.7
> bbox="172500 134900 174600 136300"
```

- Le nouveau vrt est généré en repartant du vrt créé précédemment et en utilisant 2 paramètres supplémentaires : « -te » définit l'emprise spatiale et « -tr » fixe la résolution à 2 m.

```
# Créer un nouveau vrt basé sur le précédent avec emprise et résolution modifiées
f_out=paste0(path_ortho, "/ortho_Ferroz.vrt")

gdal_utils(util = "buildvrt",
           source = f_vrt,
           destination = f_out,
           options = c("-te", bbox, "-tr", 2, 2))

> ortho
class       : SpatRaster
dimensions  : 700, 1050, 3  (nrow, ncol, nlyr)
resolution  : 2, 2  (x, y)
extent      : 172500, 174600, 134900, 136300  (xmin, xmax, ymin, ymax)
coord. ref. :
source      : ortho_Ferroz.vrt
names       : ortho_Ferroz_1, ortho_Ferroz_2, ortho_Ferroz_3
```



3. Résumé des fonctions R liées aux géotraitements

Fonctions	Descriptif	Réf
<i>rast()</i> <i>raster::dataType()</i> <i>summary()</i> <i>st_crs()</i> <i>describe()</i> <i>WriteRaster()</i>	Lecture d'une couche raster dans 1 objet <i>SpatRaster</i> Indique le type de codage du raster Présentation des statistiques de base d'un objet <i>SpatRaster</i> Indique le système de coordonnées d'un objet <i>SpatRaster</i> Affiche les métadonnées d'un fichier raster (équivalent de GDALinfo) Sauvegarde un objet <i>SpatRaster</i> dans 1 fichier tif	2.1
<i>crop()</i> <i>project()</i> <i>aggregate()</i> <i>disagg()</i> <i>rast()</i>	<i>Modifier la géométrie d'un objet SpatRaster</i> Rognage (et masquage) d'un objet <i>SpatRaster</i> (modification de l'emprise spatiale) Reprojection ou ré-échantillonnage d'un objet <i>SpatRaster</i> Réduction de la résolution d'un raster par agrégation des pixels Augmentation de la résolution d'un raster par désagrégation des pixels Création d'un raster de toutes pièces	2.2
<i>>, <, !=, ==, &, </i> <i>+, -, *, /, ^...</i> <i>ifel()</i> <i>lapp()</i> <i>classify()</i>	<i>Algèbre raster</i> Création de requêtes sur des couches raster Opérations arithmétiques sur des couches raster Création de requêtes sur des couches raster Application de fonctions mathématiques sur des couches raster Re-Classification d'un raster	2.3
<i>global()</i> <i>mean(), stdev(), median(), min(), max()</i> <i>layerCor()</i>	<i>Calculs de statistiques</i> Calcul de statistiques descriptives globale pour 1 objet <i>SpatRaster</i> . Le résultat se présente sous la forme d'un dataframe Calcul de statistiques locales sur les pixels d'un ensemble de couches rasters. Le résultat se prend la forme d'une couche raster Calcul d'un coefficient de corrélation entre 2 rasters ou d'une matrice de corrélation pour 1 séries de couches rasters	2.4
<i>extract()</i>	<i>Extraction de valeurs d'un raster</i> Extraction de valeurs au niveau de points ou de polygones	2.5

<i>spatSample()</i>	Extraction d'échantillons au sein des pixels d'un raster raster	
<i>as.polygons()</i> <i>as.points()</i> <i>rasterize()</i> <i>cats()</i>	<i>Conversion SpatRaster ↔ SpatVector</i> Conversion d'une couche raster en couche de polygones Conversion d'une couche raster en couche de points Conversion d'une couche vectorielle en couche raster Affiche la table de correspondance d'un raster catégoriel	2.6
<i>distance()</i>	<i>Calculs de distances euclidiennes</i> Création d'une couche raster exprimant les distances euclidiennes par rapport à des objets contenus dans une couche vectorielle	2.7
<i>terrain()</i> <i>shade()</i>	<i>Variables topographiques</i> Calculs de couches rasters exprimant la pente, l'exposition ou encore différents indices topographiques au départ d'un Modèle Numérique de Terrain Calcul d'une couche d'ombrage (hillshade)	2.8
<i>focal()</i> <i>focalMat()</i> <i>dilate(), erode(),</i> <i>opening(), closing()</i>	<i>Fonctions focales (filtres)</i> Application d'un filtre à une couche raster Création d'un kernel pour l'application d'un filtre Filtre morphologiques (package mmand)	2.9
<i>gdal_utils()</i>	Création de rasters virtuels	2.10