
R_GIS 03



Le package qgisprocess

Octobre 2023





TABLE DES MATIERES

1. PREAMBULE	1
AUTEUR.....	1
LICENCE DE CE DOCUMENT	1
2. INTRODUCTION	2
3. FONCTIONNALITES LIEES AUX TRAITEMENTS DE DONNEES RASTER	3
3.1 INSTALLATION DU PACKAGE QGISPROCESS.....	3
3.2 DEFINITION DES CHEMINS VERS LES REPERTOIRE.....	3
3.3 ALGORITHMES DISPONIBLES DANS QGISPROCESS	4
3.4 EXEMPLE 1 – CREER DES BUFFERS MULTIPLES (ANNEAUX).....	6
3.5 EXEMPLE 2 – CREER UN POLYGONE MINIMUM CONCAVE	9
3.6 EXEMPLE 3 – CALCULER LA PENTE DU TERRAIN	10
3.7 TAMISAGE D’UNE COUCHE RASTER (GDAL SIEVE)	11
3.8 VIEWSHED (EXTENSION WHITEBOXTOOLS)	12
3.9 FONCTIONS HYDROLOGIQUES DE L’EXTENSION WHITEBOXTOOLS	15



1. Préambule

- Le présent document a été développé par l’Axe de Gestion des Ressources forestières de Gembloux Agro-Bio Tech – Université de Liège.
- Ce document a été écrit et vérifié par l’auteur. Cependant, il est possible que des erreurs subsistent et les éventuelles remarques et corrections sont toujours les bienvenues.
- La responsabilité de l’ULiège-GxABT et des auteurs ne peut, en aucune manière, être engagée en cas de litige ou dommage lié à l’utilisation de ce document.

Auteur


- Philippe Lejeune (p.lejeune@uliege.be)
- Justine Broers (****)

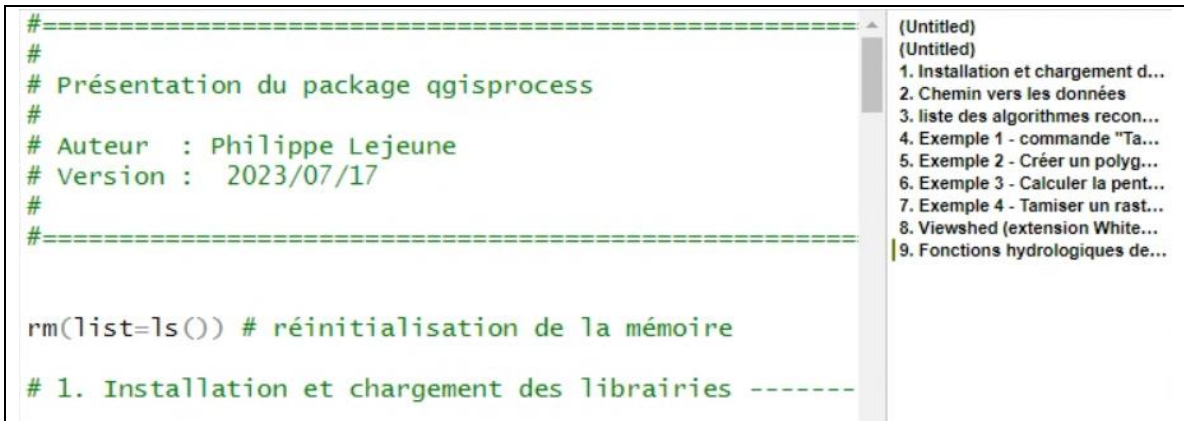
Licence de ce document

- La permission de copier et distribuer ce document à des fins pédagogiques est accordée sous réserve d’utilisation non commerciale et du maintien de la mention des sources.



2. Introduction

- L'objectif de ce tutoriel est de présenter les nombreuses possibilités offertes par le package `qgisprocess` qui permet d'exécuter les algorithmes présents dans l'environnement QGIS au sein d'un script R.
- Même si la plupart des géotraitements de base sont réalisables à l'aide des fonctions de packages tels que `sf`, `terra` ou encore `dplyr`, on peut trouver certains outils dans l'environnement QGIS qui n'ont pas d'équivalents dans ces librairies. Il est dès lors intéressant de pouvoir les exploiter en mode script dans R : c'est là que réside l'intérêt du package `qgisprocess`. Cet intérêt est d'autant plus marqué que `qgisprocess` donne également accès aux outils présents dans les extensions installées dans QGIS ou dans les logiciels auxquels QGIS a accès en tant que fournisseurs de service comme Grass ou SAGA.
- Le package `qgisprocess` est compatible avec des versions de QGIS $\geq 3.14.16$.
- Ce package est accessible depuis le lien <https://github.com/paleolimbot/qgisprocess>
- L'ensemble des exemples présentés dans ce tutoriel sont rassemblés au sein d'un script **R_GIS_03.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.

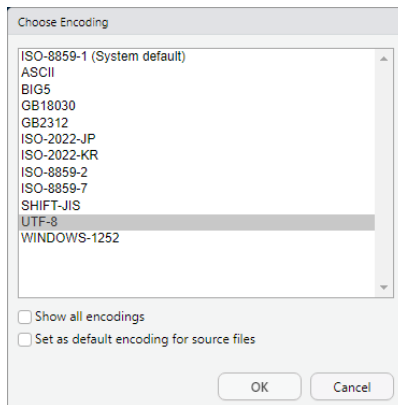


```
#=====
#
# Présentation du package qgisprocess
#
# Auteur : Philippe Lejeune
# Version : 2023/07/17
#
#=====

rm(list=ls()) # réinitialisation de la mémoire

# 1. Installation et chargement des librairies -----
```

- Les manipulations présentées dans ce tutoriel ont été testées dans l'environnement de R Studio, avec la version 4.0.3 de R.
- **Remarque** : le script **R_GIS_03.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 ».



3. Fonctionnalités liées aux traitements de données raster

3.1 Installation du package qgisprocess

- L'installation de la librairie qgisprocess s'effectue depuis son dépôt github (<https://github.com/paleolimbot/qgisprocess>).
- La fonction `qgis_configure()` permet de s'assurer que qgisprocess a bien détecté une version de QGIS avec laquelle elle peut fonctionner correctement. Dans le cas présent, il s'agit de la version 3.22.8 Biatowieza.

```
# 1. Installation et chargement des librairies -----
library(terra)
library(dplyr)
library(sf)
library(remotes)

remotes::install_github("paleolimbot/qgisprocess")
library(qgisprocess)

qgis_configure()

> qgis_configure()
getOption('qgisprocess.path') was not found.
Sys.getenv('R_QGISPROCESS_PATH') was not found.
Trying 'qgis_process' on PATH...
'qgis_process' is not available on PATH.
Found 1 QGIS installation containing 'qgis_process':
  C:/Program Files/QGIS 3.22.8/bin/qgis_process-qgis-ltr.bat
Trying command 'C:/Program Files/QGIS 3.22.8/bin/qgis_process-qgis-ltr.bat'
Success!
QGIS version: 3.22.8-Bia?owie?a
```

3.2 Définition des chemins vers les répertoires

- Les données utilisées pour cet exercice sont rangées dans le répertoire `/data_in`. Créer une variable qui contient l'adresse de ce répertoire, ainsi qu'une autre variable qui contient celle du



répertoire destiné à recevoir les résultats. Ces variables seront utilisées par la suite pour définir les noms des fichiers d'entrée et de sortie.

```
# 2. Chemin vers les données -----
path0="C:/tmp/R_GIS_03"
path_in=paste0(path0,"/data_in")
path_out=paste0(path0,"/output")
```

3.3 Algorithmes disponibles dans qgisprocess

- La fonction **qgis_algorithms()** dresse la liste des algorithmes présents dans l'environnement QGIS de votre ordinateur.

```
# 3. liste des algorithmes reconnus par qgisprocess -----
df=qgis_algorithms()
names(df)
> names(df)
[1] "provider"
[2] "provider_title"
[3] "algorithm"
[4] "algorithm_id"
[5] "algorithm_title"
[6] "provider_id"
[7] "can_cancel"
[8] "deprecated"
[9] "group"
[10] "has_known_issues"
[11] "help_url"
[12] "name"
[13] "requires_matching_crs"
[14] "short_description"
[15] "tags"
[16] "provider_can_be_activated"
[17] "default_raster_file_extension"
[18] "default_vector_file_extension"
[19] "provider_is_active"
[20] "provider_long_name"
[21] "provider_name"
[22] "supported_output_raster_extensions"
[23] "supported_output_table_extensions"
[24] "supported_output_vector_extensions"
[25] "supports_non_file_based_output"
[26] "provider_version"
[27] "provider_warning"
```

- Parmi les informations générées par cette fonction, on retrouve les noms des fournisseurs d'algorithmes. En fonction des extensions installées sur l'ordinateur utilisé, la liste proposée peut être différente de celle affichée dans la figure qui suit.

```
> table(df$provider)
```

3d Algorithms	gda1	grass7	native	qgis	saga
1	14	56	304	233	51

- Les algorithmes du fournisseur « qgis » correspondent à ceux qui sont utilisés dans les fonctionnalités de base de QGIS.



```

# liste des algo QGIS
df_qgis=df[df$provider=="qgis",]
df_qgis$algorithm

> df_qgis$algorithm
[1] "qgis:advancedpythonfieldcalculator"
[2] "qgis:barplot"
[3] "qgis:basicstatisticsforfields"
[4] "qgis:boxplot"
[5] "qgis:checkvalidity"
[6] "qgis:climbalongline"
[7] "qgis:concavehull"

```

- Les algorithmes du fournisseur « native » correspondent quant à eux au contenu de l'extension « Processing ».

```

# liste des algo natifs (extension processing)
df_native=df[df$provider=="native",]
df_native$algorithm

> df_native$algorithm
[1] "native:addautoincrementalfield"
[2] "native:addfieldtoattributetable"
[3] "native:adduniquevalueindexfield"
[4] "native:addxyfields"
[5] "native:affinetransform"
[6] "native:aggregate"
[7] "native:angletonearest"
[8] "native:antimeridiansplit"
[9] "native:arrayoffsetlines"
[10] "native:arraytranslatedfeatures"
[11] "native:aspect"

```

- Le fournisseur « gdal » propose pour sa part, l'ensemble des fonctionnalités de la librairie GDAL.


```

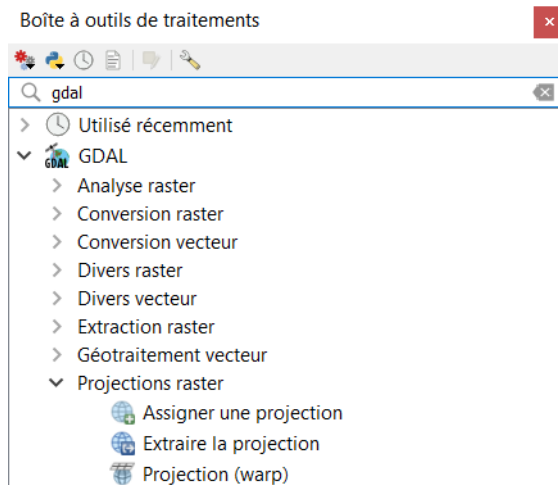
# liste des algo liés à GDAL
df_gdal=df[df$provider=="gdal",]
df_gdal$algorithm

> df_gdal=df[df$provider=="gdal",]
> df_gdal$algorithm
[1] "gdal:aspect"
[2] "gdal:assignprojection"
[3] "gdal:bufferectors"
[4] "gdal:buildvirtualraster"
[5] "gdal:buildvirtualvector"
[6] "gdal:cliprasterbyextent"
[7] "gdal:cliprasterbymasklayer"
[8] "gdal:clipvectorbyextent"
[9] "gdal:clipvectorbypolygon"
[10] "gdal:colorrelief"
[11] "gdal:contour"
[12] "gdal:contour_polygon"
[13] "gdal:convertformat"
[14] "gdal:dissolve"
[15] "gdal:executesql"
[16] "gdal:extractprojection"

```

- La correspondance avec les outils présents dans la boîte à outils de traitements de QGIS peut être établie assez facilement. Ainsi l’algorithme «**gdal :assignprojection**» correspond à l’outil

 Assigner une projection



- Etant donné le nombre très important d’algorithmes disponibles, il peut être utile de rechercher parmi ceux-ci la présence de mots clés, comme dans l’exemple qui suit où les algorithmes dont le nom contient le terme «**buffer**» sont extraits de la liste complète.

```
# recherche d'algorithme sur base d'un mot clé
df$algorithm[grepl(pattern = "buffer", df$algorithm)]
> df$algorithm[grepl(pattern = "buffer", df$algorithm)]
[1] "gdal:buffervectors"          "gdal:onesidebuffer"
[3] "grass7:r.buffer"            "grass7:r.buffer.lowmem"
[5] "grass7:v.buffer"            "native:buffer"
[7] "native:bufferbym"           "native:multiringconstantbuffer"
[9] "native:singlesidebuffer"    "native:taperedbuffer"
[11] "native:wedgebuffers"        "qgis:variabledistancebuffer"
[13] "saga:featuresbuffer"        "saga:rasterbuffer"
[15] "saga:rasterproximitybuffer" "saga:thresholdbuffer"
```

3.4 Exemple 1 – Créer des buffers multiples (anneaux)

- Dans ce premier exemple, nous utilisons l’algorithme «**native:multiringconstantbuffer**» qui permet de générer des buffers multiples à distance constante.
- La fonction **qgis_descriptions()** fourni une description de l’algorithme.

```
# Exemple 1 - commande "Tampons multi-anneaux" -----
algo="native:multiringconstantbuffer"
qgis_description(algo)
qgis_show_help(algo)
```




```
> qgis_description(algo)

native:multiringconstantbuffer
"This algorithm computes multi-ring ('donuts') buffer
for all the features in an input layer, using a fixe
d or dynamic distance and rings number."
```

- La description fournie est analogue à celle qui apparaît dans la boîte de dialogue de la commande dans l'interface de QGIS.



- La fonction **qgis_show_help()** décrit les différents paramètres à définir pour faire fonctionner correctement l'algorithme.

```
> qgis_show_help(algo)
Multi-ring buffer (constant distance) (native:multiringconstantbuffer)

-----
Description
-----
This algorithm computes multi-ring ('donuts') buffer for all the features in
an input layer, using a fixed or dynamic distance and rings number.

-----
Arguments
-----

INPUT: Input layer
      Argument type: source
      Acceptable values:
        - Path to a vector layer
RINGS: Number of rings
      Argument type: number
      Acceptable values:
        - A numeric value
DISTANCE: Distance between rings
      Argument type: distance
      Acceptable values:
        - A numeric value
OUTPUT: Multi-ring buffer (constant distance)
      Argument type: sink
      Acceptable values:
        - Path for new vector layer

-----
Outputs
-----

OUTPUT: <outputVector>
        Multi-ring buffer (constant distance)
```



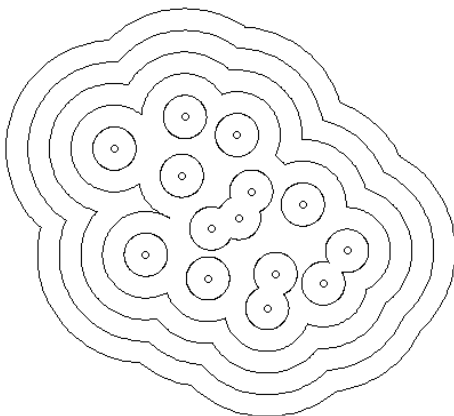
- L'exécution d'un algorithme est réalisée avec la fonction `qgis_run_algorithm()`. Dans l'exemple qui suit, 5 anneaux de 1000 m sont générés autour des points du shapefile `localites.shp`. Les arguments utilisés sont INPUT, RINGS, DISTANCE et OUTPUT. Il est important de respecter la casse des noms d'arguments. Dans le cas présent ils sont écrits en majuscule, mais ce n'est pas toujours le cas.

```
f_in=paste0(path_in,"/localites.shp")
f_out=paste0(path_out,"/buffers_5x1km.shp")
result = qgis_run_algorithm(
  algorithm=algo,
  INPUT = f_in,
  RINGS = 5,
  DISTANCE = 1000,
  OUTPUT = f_out,
  .quiet = TRUE)
```

- L'option « `.quiet = TRUE` » est utilisée pour ne pas afficher les messages envoyés par le système vers la console R Studio.

```
buff=st_read(f_out)
loc=st_read(f_in)
loc
plot(buff$geometry)
plot(loc$geometry,add=T)

> loc
Simple feature collection with 1 feature and 2 fields
Geometry type: MULTIPOINT
Dimension: XY
Bounding box: xmin: 679120.8 ymin: 611222.5 xmax: 689863.5 ymax: 620044.5
Projected CRS: ETRS89 / Belgian Lambert 2008
  fid ID geometry
1 1 1 MULTIPOINT ((679120.8 61858...
```



- Remarque : le résultat se présente sous la forme de buffers « fusionnés » car la couche de départ (`loc`) est constituée d'un seul multipoint.



3.5 Exemple 2 – Créer un polygone minimum concave

- Ce second exemple montre comment créer un polygone minimum concave autour d'un groupe de points.
- Le nom de l'algorithme peut être retrouvé dans la liste à l'aide du mot clé « concave ».

```
# 5. Exemple 2 - Créer un polygone minimum concave -----
df$algorithm[grepl(pattern = "concave", df$algorithm)]

algo="qgis:concavehull"
qgis_show_help(algo)

> df$algorithm[grepl(pattern = "concave", df$algorithm)]
[1] "qgis:concavehull"          "qgis:knearestconcavehull"
```

- L'algorithme utilisé est « **qgis:knearestconcavehull** ».

```
algo="qgis:knearestconcavehull"
qgis_show_help(algo)

-----
Arguments
-----

INPUT: Input layer
      Argument type: source
      Acceptable values:
        - Path to a vector layer
KNEIGHBORS: Number of neighboring points to consider (a lower number
is more concave, a higher number is smoother)
      Argument type: number
      Acceptable values:
        - A numeric value
FIELD: Field (set if creating concave hulls by class)
      Argument type: field
      Acceptable values:
        - The name of an existing field
        - ; delimited list of existing field names
OUTPUT: Concave hull
      Argument type: sink
      Acceptable values:
        - Path for new vector layer
```

- L'affichage de l'aide permet d'identifier les arguments à utiliser pour que l'algorithme fonctionne correctement.

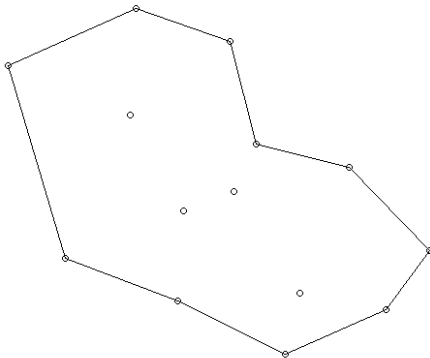
```
f_in=paste0(path_in,"/localites.shp")
f_out=paste0(path_out,"/concave_hull.gpkg")
result = qgis_run_algorithm(
  algorithm=algo,
  INPUT = f_in,
  OUTPUT = f_out,
  .quiet = T)
```



```
> f_in=paste0(path_in,"/localites.shp")
> f_out=paste0(path_out,"/concave_hull.gpkg")
> result = qgis_run_algorithm(
+   algorithm=algo,
+   INPUT = f_in,
+   OUTPUT = f_out,
+   .quiet = T)
Argument `KNEIGHBORS` is unspecified (using QGIS default value).
Argument `FIELD` is unspecified (using QGIS default value).
```

- Lorsque des arguments ne sont pas précisés dans la commande, ils reçoivent les valeurs par défaut prévues par QGIS.

```
# Affichage du résultat
hull=st_read(f_out)
plot(hull$geom)
loc=st_read(f_in)
plot(loc$geometry,add=T)
```



3.6 Exemple 3 – Calculer la pente du terrain

- Dans ce troisième exemple, une couche MNT est convertie en couche de pente, exprimée en pourcent, à l'aide de l'algorithme « **gdal:slope** ».

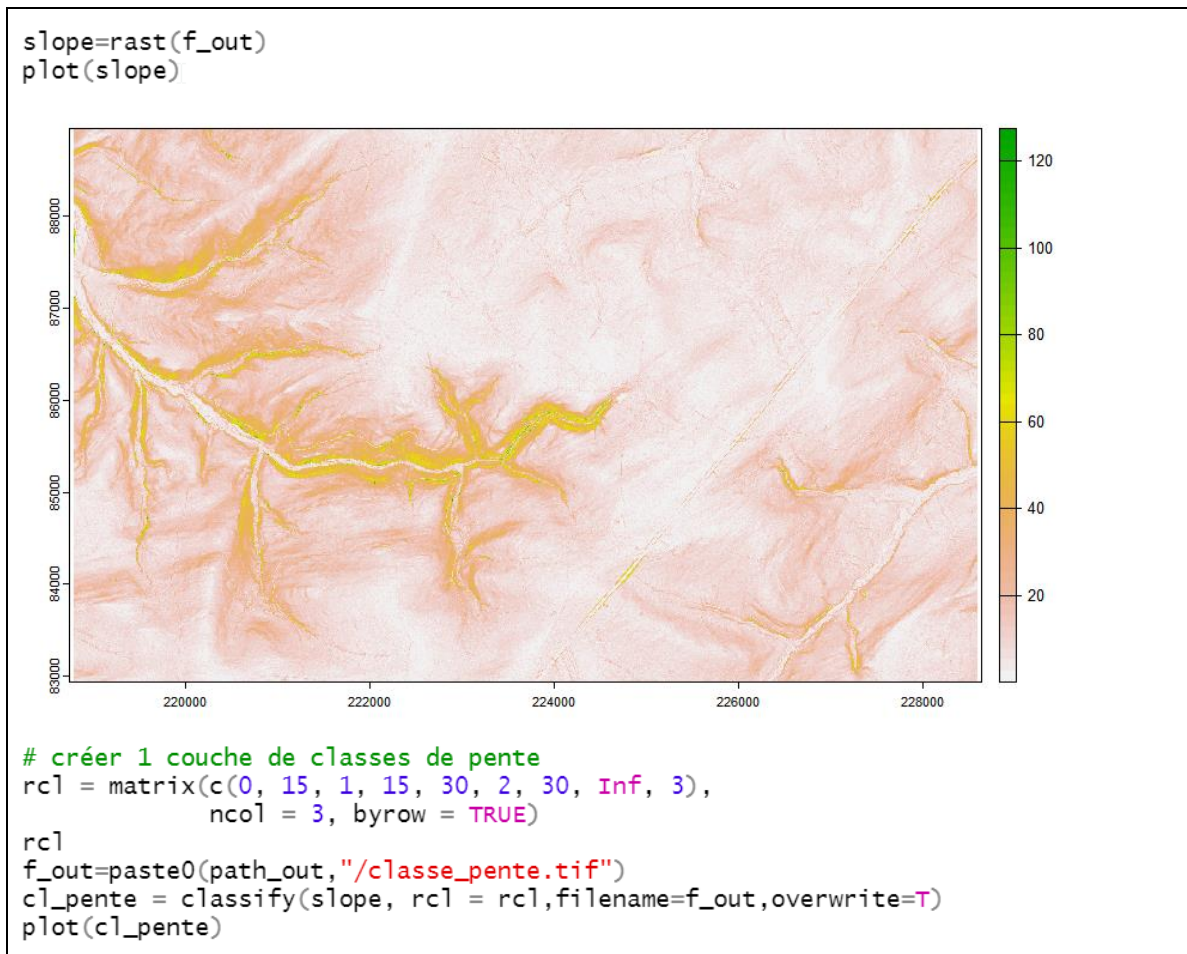
```
# 6. Exemple 3 - Calculer la pente du terrain (GDAL) ---

algo="gdal:slope"
qgis_show_help(algo)

f_in=paste0(path_in,"/mnt_2m.tif")
f_out=paste0(path_out,"/pente_pct.tif")
result = qgis_run_algorithm(
  algorithm=algo,
  INPUT = f_in,
  BAND = 1,
  AS_PERCENT = TRUE,
  OUTPUT = f_out,
  .quiet = TRUE)

slope=rast(f_out)
plot(slope)

loc=st_read(f_in)
plot(buff$geometry)
plot(loc$geometry,add=T)
```



3.7 Tamisage d'une couche raster (GDAL sieve)

- La couche de classe de pente qui a été générée au paragraphe précédent peut être « nettoyée » à l'aide d'un tamisage qui va supprimer les petits groupes de pixels (taille < 20).

```
# 7. Exemple 4 - Tamiser un raster (GDAL Sieve)

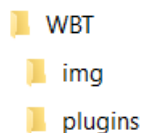
# appliquer 1 tamisage
algo="gdal:sieve"
qgis_show_help(algo)

f_in=paste0(path_out, "/classe_pente.tif")
f_out=paste0(path_out, "/classe_pente_tam20.tif")
result = qgis_run_algorithm(
  algorithm=algo,
  GRD_DEM = f_in,
  THRESHOLD = 20,
  OUTPUT = f_out,
  .quiet = TRUE)
```

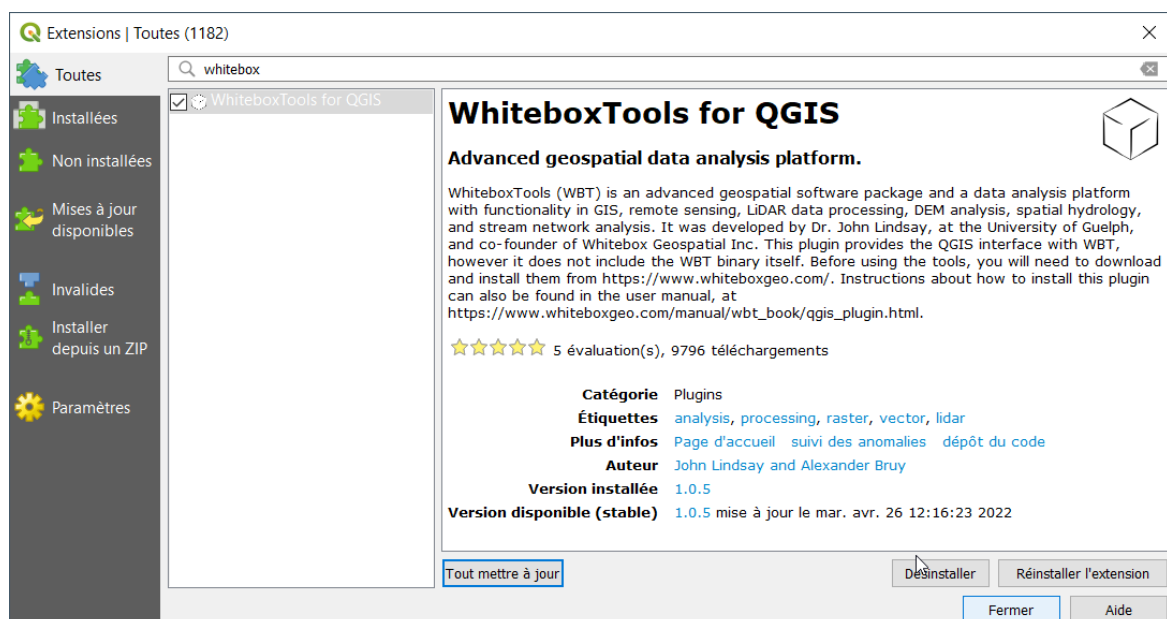



3.8 Viewshed (extension WhiteboxTools)

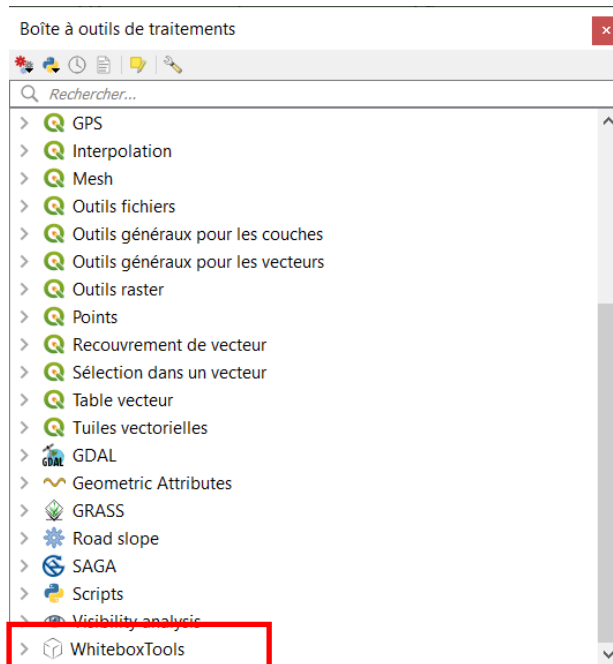
- L'extension WhiteBoxTools de QGIS qui est reliée à la boîte à outils WhiteboxTools. Celle-ci a été développée par la firme Whitebox Geospatial Inc (<https://www.whiteboxgeo.com/>) et est disponible en libre accès.
- Celle-ci rassemble un très grand nombre d'algorithmes de géotraitement.
- Pour rendre cette extension fonctionnelle, il convient tout d'abord de télécharger les fichiers constitutifs de la boîte à outils avec ce lien : <https://www.whiteboxgeo.com/download-whiteboxtools/>
- Une fois le fichier téléchargé, le fichier zip peut être décompacté sous la forme d'un répertoire WBT\.




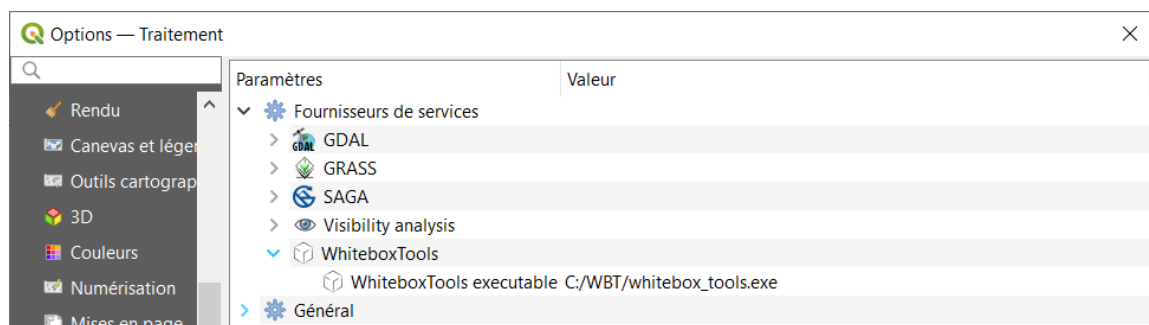
- L'extension QGIS, qui permet d'utiliser les fonctionnalités contenues dans ce répertoire, peut ensuite être installée. Pour cela, ouvrir une session QGIS et accéder au gestionnaire d'extensions avec la commande **[Extensions] → [Installer/Gérer les extensions]**.
- Taper le mot « whitebox » dans la boîte de recherche. L'extension WhiteboxTools devrait apparaître dans la liste des extensions disponibles. Sélectionner celle-ci et cliquer ensuite sur le bouton « Installer » pour installer l'extension.



- La boîte à outils WhiteboxTools est maintenant visible dans la boîte à outils de traitements qui est affichée avec le bouton .



- Pour rendre celle-ci opérationnelle, il faut définir ses paramètres en accédant aux propriétés de la boîte à outils de traitements avec le bouton .



- Ouvrir ensuite l'onglet « Fournisseur de services » qui reprend les différentes applications accessibles depuis l'environnement QGIS.
- Ouvrir l'onglet « WhiteboxTools ». Celui-ci contient 1 seul paramètre : il s'agit de l'emplacement de l'exécutable **whitebox_tools.exe**. Celui-ci doit normalement se trouver dans le répertoire \WBT.
- Après avoir rechargé la librairie qgisprocess, les algorithmes de WhiteboxTools (wbt) sont utilisables par cette dernière.

```
> table(df$provider)
```

	3d Algorithms	gda1	grass7	native	qgis	saga	wbt
	1	14	56	304	233	51	511
							529

- L'exemple qui suit utilise l'algorithme **viewshed** pour générer 1 couche raster représentant les zones visibles depuis 1 (des) point(s) de vue. Le relief est décrit par un MNT.

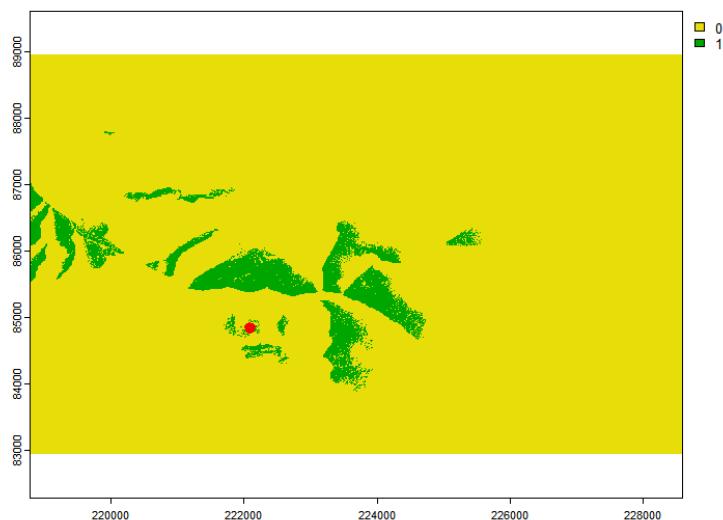
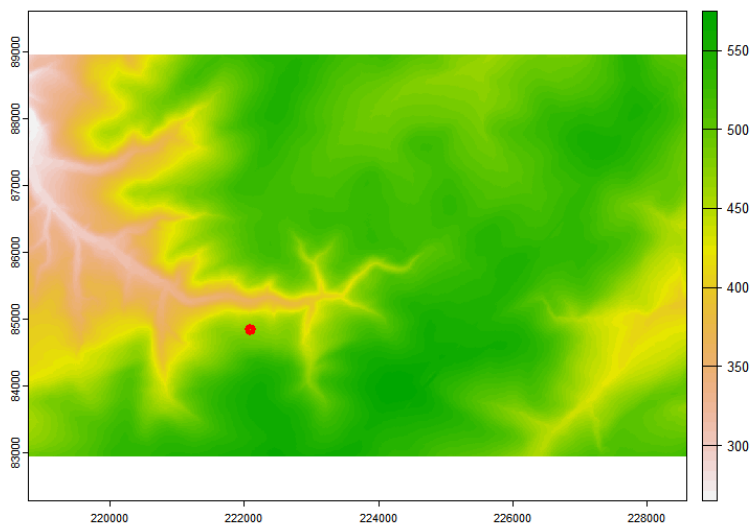
8. Viewshed (extension WhiteboxTools)

```
df_wbt=df[df$provider=="wbt",]
df_wbt$algorithm

algo="wbt:Viewshed"
qgis_show_help(algo)

f_pdv=paste0(path_in,"/point_de_vue.gpkg")
f_mnt=paste0(path_in,"/mnt_2m.tif")
f_out=paste0(path_out,"/viewshed.tif")

result <- qgis_run_algorithm(
  algo,
  dem = f_mnt,
  stations = f_pdv,
  height = 1.6,
  output = f_out,
  .quiet = TRUE)
```





3.9 Fonctions hydrologiques de l'extension WhiteboxTools

- WhiteboxTools comporte une série de fonctionnalités utilisables pour la création de couches hydrologiques dérivées de MNT.
- Dans l'exemple qui suit, nous utilisons 1 fichier **mnt_2m.tif** (MNT) et **exutoire.shp** (localisation d'un exutoire) pour délimiter le bassin versant situé en amont de l'exutoire.

```

# 9. Fonctions hydrologiques de l'extension whitetoolbox

# Input and output data
f_in=paste0(path_in,"/mnt_2m.tif") # MNT
f_exutoire=paste0(path_in,"/exutoire.shp") # exutoire du BV
f_fill=paste0(path_out,"/mnt_fill.tif") # MNT sans dépression
f_fl_dir=paste0(path_out,"/fl_dir.tif") # Direction d'écoulement
f_fl_acc=paste0(path_out,"/fl_acc.tif") # Accumulation d'écoulement
f_watershed=paste0(path_out,"/watershed.tif") # bassin versant
f_str=paste0(path_out,"/stream.tif") # axes d'écoulement
f_str_vect=paste0(path_out,"/stream.shp") # axes d'écoulement vectoriels
  
```

- La délimitation du bassin versant s'opère en 3 étapes :
 - production d'un MNT sans dépression
 - calcul des directions d'écoulement
 - délimitation du bassin versant situé en amont de l'exutoire.

```

# Délimitation d'un bassin versant

# step 1 : Fill depressions (remplissage des dépressions)

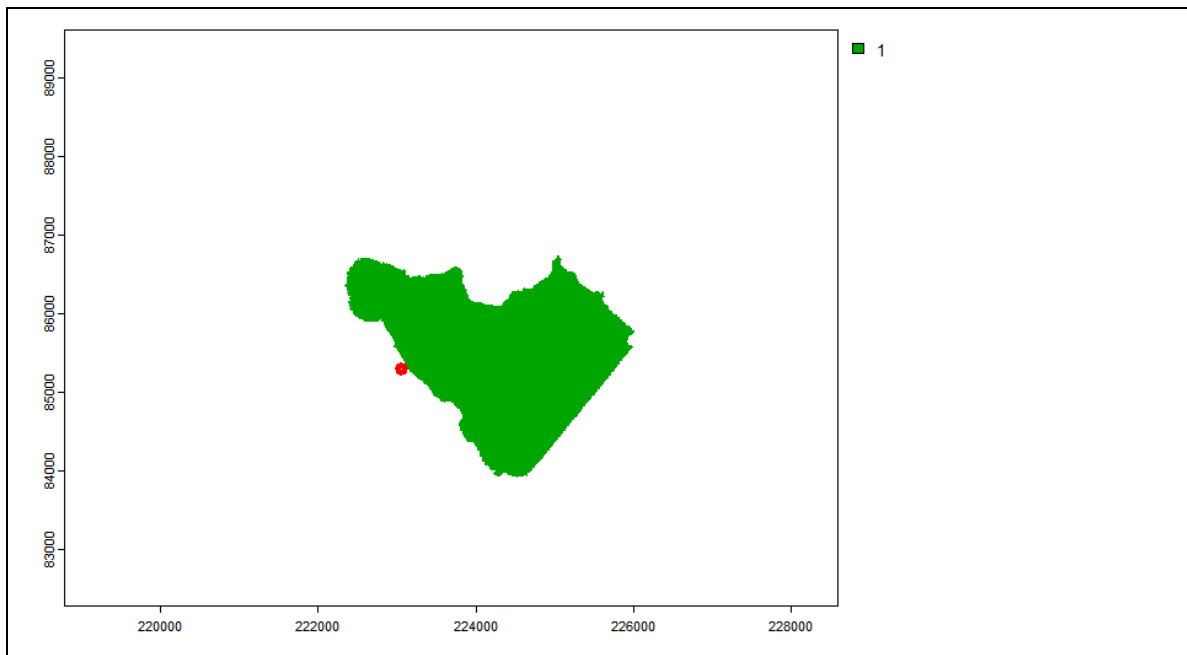
algo="wbt:FillDepressionsPlanchonAndDarboux"
result <- qgis_run_algorithm(
  algorithm="wbt:FillDepressionsPlanchonAndDarboux",
  dem = f_in,
  fix_flats= 1,
  output = f_fill,
  .quiet = TRUE)

# step 2 : D8 pointer (direction d'écoulement)
result <- qgis_run_algorithm(
  algorithm="wbt:D8Pointer",
  dem = f_fill,
  output = f_fl_dir,
  .quiet = TRUE)

# step 3 : Watershed (délimitation du bassin versant)

result <- qgis_run_algorithm(
  algorithm="wbt:Watershed",
  d8_ptr= f_fl_dir,
  pour_pts = f_exutoire,
  output = f_watershed,
  .quiet = TRUE)

r=rast(f_watershed)
plot(r)
exut=st_read(f_exutoire)
plot(exut,add=T,col="red",lwd=5)
  
```



- Avec les mêmes données de base, il est possible de tracer les axes d'écoulement dont la surface contributive (accumulation d'écoulement) est supérieure à une valeur seuil donnée (ici ssLa)
 - Calcul des accumulations d'écoulement
 - Seuillage des accumulation d'écoulement pour identifier les axes d'écoulement
 - Vectorisation des axes d'écoulement

```
# Extraction des axes d'écoulement

# step 1 : Accumulation d'écoulement
result <- qgis_run_algorithm(
  algorithm="wbt:D8FlowAccumulation",
  input = f_fill,
  output = f_fl_acc,
  .quiet = TRUE)

# step 2 : Extraction des axes d'écoulement
result <- qgis_run_algorithm(
  algorithm="wbt:ExtractStreams",
  flow_accum = f_fl_acc,
  threshold = 50000,
  output = f_str,
  .quiet = TRUE)

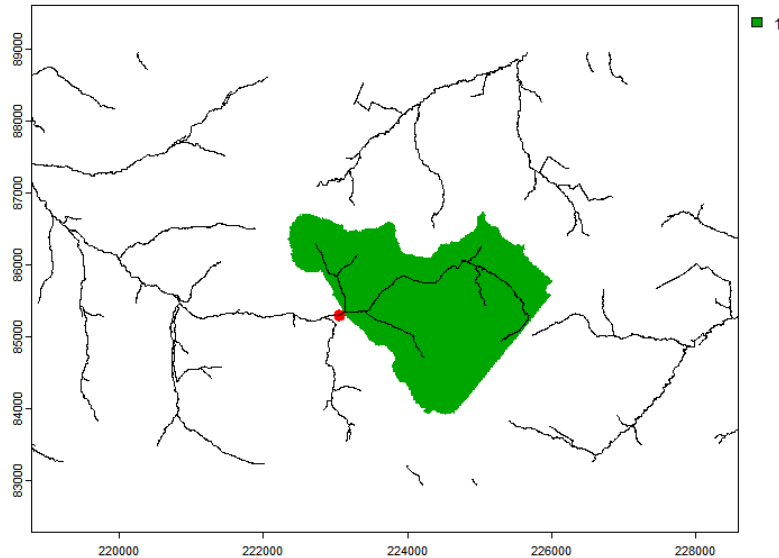
# step 3 : Vectorisation des axes d'écoulement
qgis_show_help("wbt:RasterStreamsToVector")
result <- qgis_run_algorithm(
  algorithm="wbt:RasterStreamsToVector",
  streams = f_str,
  d8_pntr = f_fl_dir,
  output = f_str_vect,
  .quiet = TRUE)
```

- **Remarque importante** : les algorithmes wbt ne supportent pas le format gpkg. Les couches vectorielles doivent donc être sauvegardées en format shapefile.



```
# affichage du résultat
str=st_read(f_str_vect)
plot(str$geometry,add=T)

> str=st_read(f_str_vect)
Reading layer 'stream' from data source
  `C:\PL\01_COURS\tthr_2022\R_GIS_03\output\stream.shp' using driver `ESRI Shapefile
Simple feature collection with 127 features and 2 fields
Geometry type: LINESTRING
Dimension:      XY
Bounding box:  xmin: 218780 ymin: 82941 xmax: 228598 ymax: 88949
CRS:            NA
```



- Remarque : le système de coordonnées de la couche vectorielle contenant les axes d'écoulement n'a pas été correctement sauvegardé. Il convient de le redéfinir avec la fonction `st_crs()`.