Tutoriel Programmation SIG

Utilisation des outils Python libres

T. Guyet

Empty.

Version Version 0.10, Dernière modification : 10/2019

Table des matières

CHAPITRE

I Introduction

Ce cours présentation l'utilisation des libraries [1] de l'OGR : GDAL/GEOS en langage Python. Ces librairies de programmation sont utilisées dans un grand nombre d'outils libres de SIG (Mapnik, GeoServer, GRASS, QGis, gvSIG, GoogleEarth, VTP, ...). Ces librairies se décomposent en 4 principales librairies complémentaires :

- OGR désigne l'ensemble des fonctions de gestion des formats et des données sous le format vectoriel
- GEOS désigne l'ensemble des fonctions liées au transformations géométriques des données en format vectoriel
- PROJ4 désigne l'ensemble des fonctions liées aux transformations des systèmes de référencement géographiques
- GDAL désigne l'ensemble des fonctions consacré à la gestion et à la manipulation des données raster.

Comprendre le fonctionnement général de ces API est un moyen de cerner le potentiel et les limites des traitements SIG actuels.

1 Python et les géotraitements

L'ensemble de ces librairies est développé en C++, un langage plus générique que python et qui offre de très bonne performances. Comme nous allons travailler sous python, les librairies python ne sont que des *binding*, c'est-à-dire des fonctions qui font indirectement référence aux fonctions C++. Généralement, les fonctions ont les mêmes noms. Les informations que vous trouverez en C++ sont donc également utiles pour les versions python. Néanmoins, certaines fonctionnalités ne sont pas fonctionnelles en python bien que les fonctions existes (j'ai personnellement rencontré le problème !).

Python est un langage de programmation assez récent (1989). C'est un langage multi-paradigme et multi-plateformes. Il est multi-paradigme car il permet de réaliser des scripts, de faire de la programmation matricielle (avec numpy), de faire de la programmation procédurale et de la programmation objet (il reste quand même assez orienté objet). Il est également multi-plateformes puisque l'environnement Python est disponible aussi bien sous Windows, Linux et Mac. Les mêmes programmes fonctionnent dans les trois environnements.
L'une des visées du Python est de pouvoir écrire rapidement des fonctionnalités avancées. Pour cela, le langage offre une syntaxe concise et s'accompagne d'un très grand nombre de librairies complémentaires qui offre un accès à des fonctionnalités avancées telles que celles permettant d'accéder aux données géographiques.
L'exécution d'un programme Python passe par un **interpréteur** qui doit être installé sur l'ordinateur exécutant le script. Une des contraintes de Python est l'absence de compilation, c'est à dire d'une étape précédent l'exécution qui permet, entre autre, au programmeur de vérifier que le programme qu'il a été est "'correct"'. Dans le cas de Python, on ne saura que l'écriture n'est pas correcte que lors de l'exécution du script. Ceci rend les programmes Python assez difficile à finaliser proprement.
Le langage Python est en cours de mutation entre une version 2 et une version 3. La version 3 apportant des modifications non-compatibles avec la version 2 du langage. Il est généralement recommandé de développer pour la version 3 du langage afin d'avoir un développement fonctionnel à moyen terme. Néanmoins, dans ce tutoriel, nous utiliserons une version 2.7 pour des raisons de disponibilité sous Windows de l'ensemble des librairies dont nous nous servirons.

2 Librairies Python pour les géotraitements

Le langage Python offre trois possibilités pour la mise en place de fonctionnalités de géotraitements :

- l'utilisation de la suite de librairies de l'OGR/GDAL

1. on parle également d'API pour *Application Programming Interface*

— l'utilisation de la librairie accompagnant le logiciel ArcGIS : ArcPy
— l'utilisation des fonctionnalités intégrées à QGis

La première solution est la solution la plus générique : elle est indépendante de quelconque autre logiciel, elle peut être utilisé très largement (librairies gratuites et open-sources, installable sur Windows, Linux et Mac). C'est une solution qui est complète mais basique. En effet, l'objectif de ces librairies est de fournir les fonctionnalités de base de traitement de l'information géographique, mais elle ne comprend que de très rare fonctionnalités avancées. Pour développer des traitements avancés, il faut soit développer ces nouvelles fonctionnalités, soit faire appel à des librairies plus spécialisées (utilisant elles-mêmes les librairies de l'OGR comme support).
La seconde solution est une solution payante, elle limite donc l'usage des outils qui sont mis en place uniquement aux utilisateurs de ArcGIS. Cette librairie de fonctionnalité a été faite pour augmenter les capacités du logiciel ArcGIS par des procédures développées par l'utilisateur. L'intérêt principal de cette librairie est de donner accès aux traitements de haut-niveau disponibles dans ArcGIS. Mais c'est aussi une contrainte de cette librairie qui reste très cloisonnée à ces fonctionnalités. On tire le maximum de l'intérêt de ArcPy avec le *modeler*. Une procédure créée graphiquement avec le *Modeler* peut engendrer un code Python qu'un géomaticien aguerris à la programmation pourra adapté à son besoin spécifique.
La troisième solution est intermédiaire. Initialement prévu pour le développement de plugins dans QGis la librairie Python de QGis offre des fonctionnalités très similaires à celle de l'OGR (en faite, tout le logiciel QGis s'appuie sur cette librairie) et ajoute la possibilité d'accéder aux fonctionnalités de visualisation, d'interaction et de traitement disponibles dans QGis. Il est néanmoins à noter que QGis est un logiciel prévu pour faire de la visualisation et certaines fonctionnalités de modification des données de la librairie de GDAL (en version 1.9, je n'ai pas regardé depuis) semblent accessibles depuis QGis mais ne sont en faite pas implémentées. C'est le cas, en particulier, pour l'accès aux données raster. Il est parfois nécessaire de revenir à des objets de la librairie GDAL pour développer son plugin complet (et ce n'est pas très propre !).

Dans le cadre de ce tutoriel, nous avons choisis de présenter la librairie OGR/GDAL, d'une part, pour présenter les fondements partagés par ces librairies de géotraitement, ensuite, pour l'aspect générique des traitements qui peuvent être développés par cette librairie. Une bonne pratique de la programmation informatique (voir, par exemple, les modèles MVC) est de décorréler les fonctionnalités de traitement (la définition des fonctions de traitement) et les fonctionnalités d'interaction (comment l'utilisateur utilise les fonctions). Nous nous intéresserons ici à développer des fonctionnalités de traitement. Finalement, l'usage premier de ces outils par le géomaticien consiste souvent à effectuer des transformations simples des données et à déléguer le réalisation des fonctionnalités de traitement avancé à des informaticiens. Dans ce sens, on peut supposer que les traitements avancés seront des fonctions utilisables en Python (voir le chapitre **??**, sur l'automatisation des traitements).

3 Installation de l'environnement de travail

L'ensemble des outils qui ont été sélectionnés sont les logiciels **open sources** et **libres d'usage** et multi-plateformes. Vous pouvez donc les installer et les utiliser gratuitement dans tous vos développements.
Ce dont nous aurons besoin pour ce tutoriel :

— les librairies de l'OGR :
  — OGR : pour la gestion des formats de données vectoriels (dont SHP et bases de données géographiques),
  — GEOS : pour la manipulation des géométries,
  — PROJ4 : pour la gestion des systèmes de projection,
  — GDAL : pour la gestion des formats de données raster.
— les librairies permettant l'utilisation de R sous python,
— le système d'information géographique QGis.

3.1 Installation sous Linux

3.1.1 Installation de Python et GDAL

L'installation sous Linux est plutôt simple relativement à l'installation sous Windows. En effet, tous ces outils sont développés et utilisés sous Linux en premier lieu.
Les distributions de Linux les plus usuels (Ubuntu/Fedora) dispose de ces librairies dans leur offre logiciel, il suffit donc de les installer directement avec l'outil d'installation. Personnellement, j'utilise les outils d'installation en ligne de commande (`apt-get` pour ubuntu et `yum` pour fedora), mais vous pouvez les retrouver à l'aide des outils d'installation graphiques.

Pour Ubuntu, la liste des *packages* à installer sont les suivants (certains *packages* supplémentaires, comme les librairies GDAL seront automatiquement ajoutées à votre installation) :

— python3
— python3-numpy
— python3-scipy
— python3-gdal

3.1.2 Installation des outils utiles au tutoriel

Installation de R/python Toujours en utilisant les *packages* :

— r-base (logiciel R)
— python-rpy2 (prévu pour la version python 2.7, mais fonctionnel avec les versions python 3.X)

3.2 Installation sous Windows

Pour la version Windows, à la date de la rédaction de ce document, il est nécessaire d'utiliser une version de Python 2.7 (32 bits, même si votre ordinateur est 64 bits !) pour que l'ensemble de l'environnement soit cohérent. La limitation de version vient de QGis qui intègre une console Python uniquement en version 2.7.

! Attention ! - Incompatibilité des versions de Python sous Windows

Attention, il est important d'avoir des librairies qui correspondent à la même version de Python (même numéro de version et même type d'architecture 32 bits vs 64 bits) que celle que vous avez installé si vous voulez que votre installation fonctionne (ce dont je ne doute pas ! !).
La sous-version de python n'a pas d'importance. La version 2.7.0 sera compatible avec la version 2.7.7.
Notez qu'il est tout à fait possible d'installer des versions de Python concurrente ... mais après, on ne sait plus trop laquelle est utilisée ensuite !

3.2.1 Installation de Python

On commence donc par effectuer une installation de Python 2.7.

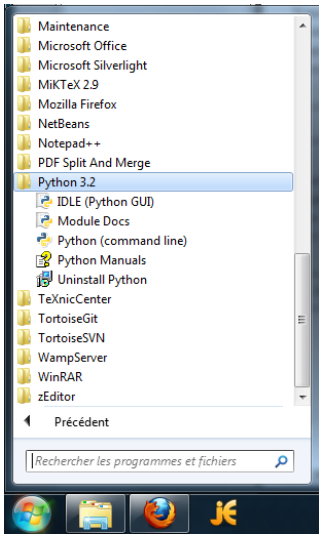
1. Vous devez vous rendre sur le site officiel de Python : `http://www.python.org/`.
2. Cliquez sur le lien `Download` dans le menu principal de la page.
3. Sélectionnez la version de Python que vous souhaitez utiliser.
4. On vous propose un (ou plusieurs) lien(s) vers une version Windows : sélectionnez la version 2.7 en version "x86" (32 bits), cette version permet la compatibilité avec GDAL.
5. Enregistrez puis exécutez le fichier d'installation et suivez les étapes. Ce n'est ni très long ni très difficile.
   ⇒ il est recommandé de laisser le répertoire d'installation par défaut `C:\Python27` (aide pour la suite de l'installation), sinon retenir votre chemin d'installation.
6. Une fois l'installation terminée, vous pouvez vous rendre dans le menu `Démarrer>Tous les programmes`. Python devrait apparaître dans cette liste (*cf.* figure ci-dessous).

3.2.2 Installation de GDAL

On passe maintenant à l'installation de la librairie GDAL proprement dite.

3.2.3 Installation des librairies Python utiles

On passe maintenant à des étapes d'installation de librairies Python. Cette étape n'est pas triviale sous Windows (mais elle n'est pas difficile non-plus !). En effet, si le langage Python est multi-plateforme, ce n'est pas le cas des langages de programmation tels que C/C++ sur lesquels s'appuient les librairies que nous souhaitons installer. Il faut alors disposer de version qui ont été préparés spécifiquement pour votre version de Windows. Cette étape est très largement facilité grâce à l'initiative de Christoph Gohlke de



l'Université de Californie. Ce chercheur met à disposition des versions dites "'compilées"' des librairies fréquemment utilisées en calcul scientifique qu'il suffit de choisir et d'installer.
Les trois librairies à installer (dans cet ordre) sont les suivantes :

- `Numpy` : librairie de calcul matriciel
- `scipy` : librairie de calcul scientifique
- `scikit-learn` (aussi appelé `sklearn` : librairie d'algorithmes d'apprentissage automatique
- `GDAL` : librairie incluant OGR/GDAL/GEOS et Proj4

Pour chacune de ces librairies, vous procéderez ainsi :

1. aller sur la page web `http://www.lfd.uci.edu/~gohlke/pythonlibs`
2. rechercher la librairie souhaitée dans la liste des librairies disponibles
3. télécharger la librairie en version `win-32` et pour `py2.7` (les téléchargements peuvent être longs)
4. exécuter le fichier téléchargé et suivre les instructions de l'interface :
   ⇒ lors de l'étape du choix de version de Python, l'installateur doit trouver automatiquement le répertoire (sauf si vous avez changé la destination manuellement lors de l'installation de Python)

3.3 Environnement de programmation adapté

Nous avons vu en introduction que Python était un langage interprété pour lequel il est difficile de vérifier l'exactitude des programmes que nous écrivons. Pour faciliter notre travail, l'utilisation d'un environnement de programmation adapté est une aide importante. Il est très utile d'investir un peu de temps à apprendre l'utilisation d'un logiciel qui aidera à la programmation.
Pour Python, dans le cadre d'un développement sous Windows, je recommande l'utilisation du logiciel libre `PyScripter` dont l'interface est illustrée dans la figure I.1.

3.3.1 Installation de PyScripter

Pour l'installation de `PyScripter` :

1. aller sur le site `https://code.google.com/p/pyscripter/`
2. dans l'onglet `download`
3. télécharger la version avec un setup (préférer la version x86 pour architecture 32 bits par cohérence avec les librairies Python, bien que je ne sois pas sûr que cela soit lié !).
4. lancer le setup et suivre les instructions

3.3.2 Fonctionnalités élémentaires de PyScripter

L'interface de PyScripter se compose de la sorte :

- le bandeau supérieur de l'interface contient les menus et les icônes d'accès aux fonctionnalités les plus utiles du logiciel. Comme dans tous les logiciels, vous pouvez organiser ces icônes en fonction de vos besoins les plus courants.

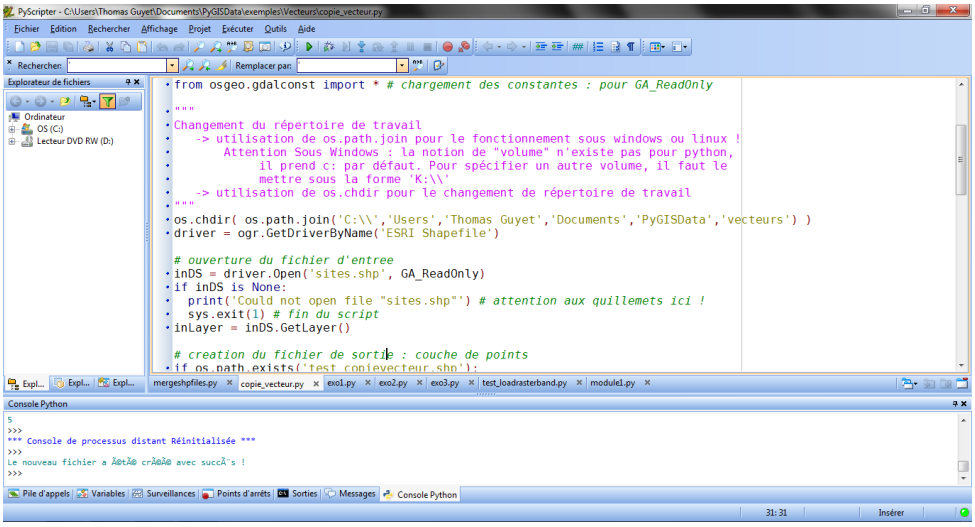


FIGURE I.1 – Illustration de l'interface de `PyScripter`

— la partie de gauche de l'interface permet d'avoir une vue organisée sur votre script. Trois onglets vous sont proposés par défaut (fichier, projet ou code). La vue la plus utile pour vous sera la vue 'code' qui représente l'ensemble des fonctions définies dans votre script. Pour un script long, cela permet d'accéder rapidement à la partie du code qui vous intéresse !
  — l'éditeur effectue de la coloration syntaxique : c'est à dire que des mises en forme permettent de mieux structurer visuellement le code (commentaires en rose, chaînes de caractère en jaune, etc.)
  — les points bleus dans la colonne de gauche indique qu'il s'agit de ligne d'instruction, en cliquant sur ce point, il est possible d'ajouter un point d'arrêt (*cf.* un peu plus loin) représenté par un gros point rouge.
— au centre de l'interface se trouve les scripts en cours d'édition. Dans les options de l'éditeur, il est possible d'ajouter le numéro de ligne ou de changer la police de caractères, etc.
— la liste des fichiers ouverts se trouve sous la zone d'affichage du code
— la partie inférieure de l'interface contient une console python dans laquelle il est possible de tester des commandes et d'exécuter vos scripts

La fonctionnalités importantes de PyScripter sont les suivantes :

— **auto-complétion** : lorsque vous tapez le début d'un mot, le programme vous propose automatiquement l'ensemble des variables ou mots clés qu'il connaît. Vous n'êtes pas obligé d'en tenir compte ... si vous continuer à taper votre texte, les propositions s'ajusteront ou disparaitront (moins vous touchez à la souris et plus vous serez efficaces). L'intérêt de cette fonctionnalité est double :
  1. c'est un aide-mémoire pour le nom exacte d'une fonction,
  2. c'est un outil pour explorer les fonctionnalités d'un type de données ou d'une librairie.
— **aide à l'exécution du code** : l'interface propose un façon rapide d'exécuter votre programme : l'icône en forme de flèche verte, ou la combinaison de touche `Ctrl+F9` ou `F9` (quasiment-équivalent si il n'y a pas de point d'arrêt dans votre programme).
— **aide au débuggage** : l'interface permet de faire de l'exécution pas-à-pas, c'est-à-dire d'arrêter le programme pendant son exécution et observer l'état des variables. Cette fonctionnalité est très utile pour trouver une erreur dans un programme Python. Le principe est d'introduire des points d'arrêt dans le programme. Lors de l'exécution en mode débuggage, le programme s'interrompra automatiquement dès qu'il atteindra le point d'arrêt. Il donne ensuite la main au programmeur pour :
  — consulter la valeur des variable en utilisant des commandes d'affichage directement dans la console de PyScripter
  — de faire exécuter la suite du programme instruction par instruction (pas-à-pas).
  — continuer normalement l'exécution (tant qu'il ne retombe pas sur un point d'arrêt)
— la gestion de gros projet de développement
— contrairement à d'autres logiciels, PyScripter ne met pas à disposition d'outil d'indentation automatique. Ceci est particulièrement regrettable lorsqu'on développe des scripts Python pour lesquels l'indentation a beaucoup d'importance.

4 Aides et documents de référence

La librairie GDAL est une librairie écrite en C/C++. Il est plus facile de trouver de la documentation sur la librairie dans ce langage qu'en python. En particulier la documentation de référence en python n'est qu'une liste de fonction sans description ... elle réfère systématiquement à la documentation C++.

— Documentation python sur l'API : `http://www.gdal.org/python/`
— Tutoriel officiel : `http://www.gdal.org/gdal_tutorial.html`

L'Internet contient également beaucoup de ressources utiles : sur les blogs (spécialisés ou non), dans les fils de discussions, etc. n'hésitez pas à utiliser et adapter ces ressources toutes faites.

5 Importation des librairies GDAL/GEOS

Du fait de l'inclusion des libraries GDAL/GEOS dans une même librarie OGR, il existe actuellement différents chemins d'accès à ces libraires. Les lignes de programme ci-dessous donne une manière assez générique d'importer les librairies. Ce programme teste deux possibilités de nommage des librairies.
Les lignes ci-dessous seront à utiliser systématiquement au début de vos codes python.

```
#Chargement de la librairie GDAL/GEO (doit etre installe pour que ca fonctionne)
try:
  import osgeo.gdal as gdal
  import osgeo.ogr as ogr
  import osgeo.osr as osr
  from osgeo.gdalconst import *
except ImportError:
  try:
    import gdal, ogr, osr
    from gdalconst import *
  except ImportError:
    import sys
    sys.stderr.write("GDAL needs to be installed. Exiting...\n")
    sys.exit(-1)
```

Dans la suite du document, on présente tout d'abord l'utilisation des méthodes liées à l'usage de données vectorielles, puis on présente l'utilisation des traitements sur des données raster.

CHAPITRE

II Quelques rappels de python

Ce chapitre reprends les éléments essentiels pour la compréhension de la suite, mais ne se veut aucunement exhaustif sur la présentation du langage Python.
Pour découvrir l'ensemble des fonctionnalités, je vous invite à consulter le tutoriel suivant : `http://fr.openclassrooms.com/informatique/cours/apprenez-a-programmer-en-python`.

Python est un langage :
- simple en première approche : langage interprété, sans déclaration et plutôt intuitif.
  ⇒ il est idéal pour automatiser rapidement des petits traitements
- modèle de programmation complet : orienté-objet, fonctionnel et procédural, il répond aussi bien au développement de gros logiciels.
- modulaire : il existe de nombreux modules disponibles (et grauit) qui répondent à vos besoins de fonctionnalité.
- bien outillés
  - le langage et les modules sont généralement bien documentés (avec un documentation uniforme)
  - il existe de nombreux outils pour vous aider à programmer
- multi-plateforme : vos programmes fonctionnent aussi bien sous Windows, Mac ou Linux !

Pour toutes ces raisons, c'est un langage très utilisé actuellement. On le retrouve notamment en support de développement de modules complémentaires à de nombreux logiciels.

1 Programmation Python

1.1 Exemple de programme

Un programme python est un texte (un script) écris dans un fichier. Généralement, les programmes python sont enregistrés avec l'extension de fichier `.py`.
Un script Python peut être édité avec un logiciel comme NotePad ou le bloc note, mais ce n'est pas l'idéal. Il existe de nombreux outils de développement intégrés (IDE pour Integrated Development Environment) pour vous aider dans le développement. Il permettent en particulier de faire :
- la coloration syntaxique : les différents éléments du code sont représentés par des typographies/-couleurs différentes
- la completion : lorsque vous taper le début d'une fonction, il vous propose les fins possibles. Ceci peut servir d'aide mémoire.
- la vérification des erreurs syntaxiques les plus évidentes
- l'exécution de votre programmme directement dans l'IDE (sans changer de fenêtre) pour vérifier fréquemment votre progression.

```
# -*- coding: utf-8 -*-

#Chargement des modules utiles au programme
import os , sys , Image

#Definition d'une fonction a 2 parametres
def rotate_all (path, angle) :
  for root, dirs, files in os .walk(path):
    for i in files :
      im=Image.open(os.path.join(root, i )) # ouvre l'image
      print i, im.format, im. size, im.mode # affiche qq infos sur l'image
      im.show() # affichage de l'image avant rotation
      out = im.rotate(angle) # rotation
      out.show() # affichage de l'image apres rotation
      out.save(os .path. join (root , i )) # ecrase l'image
```

```
#Point d'entree du programme
if __name__ == '__main__':
  rotate_all ("/home/login/images", 90.0)
```

La première ligne du programme indique l'encodage de vos fichiers textes. Dans la mesure où on travaille sous Linux, les fichiers sont encodés en UTF-8. Sous Windows, les encodages sont des ISO. Si l'encodage n'est pas correctement indiqué, vous aurez des messages d'erreur lors de l'exécution de votre programme. Il indique alors des problèmes sur les accents.
Dans ce programme, les autres parties du code qui se trouve après un symbole '#' sur une ligne sont des commentaires. Ils ne seront pas pris en compte lors de l'exécution, mais ils sont important pour que le programmeur qui découvre le code puisse aisément en comprendre son fonctionnement.
L'utilisation de retraits en début de ligne (on parle d'indentation) a beaucoup d'importance dans les programmes Python. Il est TRÈS important de les respecter et d'avoir une pratique uniforme : soit se sont toujours des tabulations, soit se sont toujours des espaces (et toujours le même nombre d'espace). Ces retraits indiquent la structure en bloc d'un programme.

1.2 Exécution d'un script python

Il existe plusieurs façons de lancer des scripts Python :
- en tapant le code directement dans l'interpréteur ("IDLE" sous Windows) : on parle de lignes de commande
- en exécutant un script à partir de l'interpréteur : cela consiste à appeler les commandes contenues dans un fichier pour qu'il les exécute une à une.
- sous la forme d'une ligne de commande (`python votre_script.py`)
- en utilisant un interpréteur "caché" dans un outil de développement Python : c'est le cas par exemple pour Spyder2 qui vous permet d'exécuter votre script en utilisant la touche `F5`.

Quelque soit la façon d'exécuter votre script, les résultats seront similaires.

2 Les variables python

2.1 Variables

Le python est un langage typé, c'est à dire que toutes les variables sont associés à un type. Le type d'une variable peut être connu grâce à la fonction `type()`. Il est très important de savoir quel est le type d'une variable puisque c'est de ce type que dépendent les opérations qui peuvent être faites "avec" ou "sur" elle.
Une variable est déclarée lors de sa première utilisation :
- Il n'y a pas d'instruction spécifique à la déclaration d'une variable
- Une variable est dite "mutable" : le type de la variable dépend de la valeur qu'elle contient

L'affectation d'une variable, c'est à dire l'opération qui désigne la valeur que contient la variable, est réalisée avec '='.

Exemple 1

```
>>> x=23
>>> type(x)
<type 'int'>
>>> y=x*23
>>> print(y)
529
>>> print(u)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'u' is not defined
>>> x='coucou'
>>> type(x)
<type 'str'>
>>> y=x*3
>>> y
'coucoucoucoucoucou'
>>> y=x/23
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

2.2 Affichages et saisies

La fonction print permet d'afficher le contenu d'une variable dans le terminal d'exécution du programme Il est utile de savoir également faire des affichages complexes pour rendre les message d'informations plus clairs. Pour cela, il faut utiliser des traitements sur les chaînes de caractères. En premier lieu, la concaténation de texte peut être réalisé au moyen de l'opérateur +. Cet opérateur ne fonctionne qu'entre chaînes de caractères. Lorsque vous avez des variables non-textuelles, la fonction str() permet de construire un texte à partir d'une variable.

```
x=34
print 'la valeur de x est ', x
print('la valeur de x est '+str(x)+'\n')
```

La récupération d'une valeur auprès de l'utilisateur par une saisie clavier peu se faire grâce aux fonctions input ou raw_input. La première ne permet de récupérer qu'un entier. La seconde récupère un texte que vous pouvez ensuite transformer dans le type souhaité. À chaque saisie, n'oubliez pas d'ajouter un message d'information sur la saisie à effectuer.

```
x = input('saisir un entier\n')
print('vous avez saisi la valeur ' + str(x) +'\n')
x = int( raw_input('saisir un entier\n') )
f = float( raw_input('saisir un nombre\n') )
val=f/x
print('resultat de la division : ' + str(val) +'\n')
```

3 Structures de contrôle usuelles

3.1 Structure alternative

```
# syntaxe generale ( if / else )
if x>0 : # noter les deux points
  print 'positif' # l'indentation indique l' instruction depend de la condition
elif x<0 :
  print 'negatif'
else :
  print 'nul'
```

Indiquer si un point (avec des coordonnées données) est à l'intérieur d'un rectangle (orienté Nord-Sud), de coordonnées :

- Xmin = 235067, Xmax = 237980
- Ymin = 2108934, Ymax = 2110234

Solution 1 :

```
# coordonnee du rectangle
Xmin,Xmax=235067,237980
Ymin,Ymax=2108934,2110234
# coordonnee du point
Xpt,Ypt=235085,2109654

resultat ="Le point est en dehors du site " # on initialise en disant que le point est en dehors
if Xpt >= Xmin and Xpt <= Xmax:
  if Ypt >= Ymin and Ypt <= Ymax:
    # si toutes les conditions sont remplies , on change le resultat
    resultat ="Le point est dans le site "
print ( resultat )
```

Solution 2 (on ne remet pas les 5 premières lignes) :

```
isin = true
if Xpt < Xmin or Xpt > Xmax:
  isin = false

if Ypt < Ymin or Ypt > Ymax:
  isin = false

if isin :
  print ("Le point est dans le site")
```

```
else :
  print ("Le point est en dehors du site ")
```

Solution 3 :

```
isin = true
if Xpt < Xmin or Xpt > Xmax or Ypt < Ymin or Ypt > Ymax:
  isin = false

if isin :
  print ("Le point est dans le site")
else :
  print ("Le point est en dehors du site ")
```

Solution 4 :

```
isin = false
if Xpt >= Xmin and Xpt <= Xmax and Ypt >= Ymin and Ypt <= Ymax:
  isin = true

if isin :
  print ("Le point est dans le site")
else :
  print ("Le point est en dehors du site ")
```

Ces 4 solutions sont équivalentes ! Elles donnent les mêmes résultats et les temps de calculs seront en moyenne similaires. Personnellement, je trouve les deux dernières plus éléguantes.
Vous vous assurerez d'être bien d'accord avec l'utilisation des opérateurs logiques `or` ou `and`. La vraie difficulté des structures conditionnelles est de construire la condition.

3.2 Structure répétitive

Il y a classiquement deux façons de faire répéter des opérations :
- les boucles `for` qui font répéter un bloc d'instructions pour tous les éléments d'une séquence de valeurs
- les boucles `while` qui font répéter un bloc d'instruction tant qu'une condition est vérifiée.

```
# Exemple de boucles while
i,j=1,5
while j>=i:
  print ( j )
  j=j-1 # attention a ce que les valeurs de la condition changent (sinon boucle infinie )

# Exemple de boucle
liste =['a', 'b', 'c']
for lettre in liste :
  print lettre
```

La fonction `range()` permet d'utiliser facilement la boucle `for` pour des valeurs entières.

```
for i in range(1,10) :
  print ( i ) # ecrit 1,2,...,9 ( ligne par ligne )
```

! Attention ! - Utilisation de `range()`

Je recommande d'utiliser toujours la fonction `range()` avec 2 paramètres (début et fin de la séquence) :
- `range(10)` construit une séquence de 0 à 9
- `range(0,9)` construit une séquence de 0 à 9

4 Les structures de données usuelles

Le python dispose de structures de données riches pour manipuler des ensembles de données. Les plus usuelles sont d'une part les listes qui permettent de manipuler des collections d'objet, et les dictionnaires qui sont des structures de données dites "associatives".

4.1 Les listes

Une liste contient une collection ordonnée d'éléments. L'ordre des éléments est imposé par construction de la liste. L'avantage de la liste est la grand facilité de la manipulation de ses éléments. Il est possible facilement d'ajouter des éléments (en début ou en fin), de supprimer des éléments (à n'importe quelle position), de parcourir la liste et de rechercher des éléments.

```
semaine = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi']
```

Quelques exemple de manipulation à connaître :

- **print semaine** affichera toute la liste
- **print semaine**[2] affichera le 3eme élément de la liste : on accède à un élément par la notation crochetée. Les indices commencent à 0.
- **len**(**semaine**) donne le nombre d'éléments de la liste
- **semaine.append**('dimanche') ajoute un élément à la liste
- **semaine.remove**('mardi') supprime un élément s'il existe
- **semaine.clear**() supprime tous les éléments de la liste

! Attention ! - Liste d'éléments différents

Les éléments d'une liste peuvent être de différents types :

```
liste2 = ['lundi', 'mardi', 85, 'vendredi', 32,17]
```

Remarque 2 - Liste de listes

Il est possible d'avoir des listes de liste. Les différentes listes de la liste principal n'auront pas nécessairement la même taille, et ne contiendra pas nécessairement des éléments de même type ... (*cf.* remarque précédente)

```
l0 = []          # creation d'une liste vide
l1=['a', 'b', 'c']
l2=[45, 67]
l3=range(1, 34)      # range engendre une liste !
L=[l0, l1, l2, l3]   # construction d'une liste de listes
print(L)
L[0].append(34)  # ajoute un element a la premiere liste
```

Le programme affiche ainsi :

```
[[], ['a', 'b', 'c'], [45, 67], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
```

Exemple 2 - Parcours des éléments d'une liste

On prend l'exemple de la somme des éléments d'une liste **liste** *contenant des nombres. La première solution consiste à utiliser la notation crochetée pour accéder aux éléments 1 à 1 par leur indice :*

```
liste =[30,0,-15,8.0,73]
somme=0
i=0
n=len(liste)
while i<n :
  somme=somme+liste[i]
  i=i+1
print "La somme est" + str(somme)
```

*Une autre solution consiste à utiliser la notion d'**itérateur**. Un itérateur est une variable qui va prendre successivement les valeurs des éléments de la liste.*

```
liste =[30,0,-15,8.0,73]
somme=0
for val in liste :
  somme=somme+val #on ajoute directement val, comme element de la liste
  i=i+1
print "La somme est" + str(somme)
```

4.2 Les dictionnaires

Les dictionnaires sont des structures de données associatives, c'est-à-dire qu'elles associent une valeur à une autre. Il faut donc distinguer :

— l'ensemble des clés : elles sont uniques,
— l'ensemble des valeurs contenues dans le dictionnaires.

Le principe du dictionnaire est d'associer une valeur à une clé : la clé sert à accéder aux éléments du dictionnaire et elles sont donc uniques. Vous pouvez voir le dictionnaire comme une extension de l'indexe des listes. Pour une liste, l'indexe est nécessairement un entier positif. Dans le cas d'un dictionnaire l'indice peut être un entier, mais tout autre chose.
Alors que les listes sont associés aux notations crochetées, les dictionnaires sont associés aux accolades.

Exemple 3 - Création d'un dictionnaire

Dans cet exemple, on créé un dictionnaire qui contient les notes d'étudiants.

```
mon_dico = {} #creation d'un disctionnaire vide
mon_dico[ "Thomas" ] = 9
mon_dico[ "Sylvaine" ] = 15
mon_dico[ "Alban" ] = 16
mon_dico[ "Ambre" ] = 12
mon_dico[ "Martin" ] = 12
mon_dico[ "Thomas" ] = 5
print ( mon_dico )
```

Cet exemple affichera :

```
{'Martin': 12, 'Thomas': 5, 'Sylvaine': 15, 'Ambre': 12, 'Alban': 16}
```

Notez qu'il n'y a, au final, qu'une seule valeur pour 'Thomas', mais que la valeur 12 peut être utilisée plusieurs fois.

Quelques exemples de manipulation utile à connaître :

— `mon_dico["bar"]` donne la valeur associé à l'entrée "foo" si elle existe (sinon erreur !)
— `len(mon_dico)` donne le nombre d'entrée du dictionnaire
— `mon_dico.keys()` donne la liste des entrées du dictionnaire, il n'y a pas de doublons dans cette liste.
— `mon_dico.values()` donne la liste des valeurs du dictionnaire, il peut y avoir des doublons.
— `mon_dico.has_key("Julie")` indique si le dictionnaire à l'entrée Julie
— `del mon_dico["Thomas"]` supprimera l'entrée pour Thomas
— `mon_dico.clear()` supprime tous les éléments du dictionnaire

Exemple 4 - Parcours des éléments d'un dictionnaire

Pour parcourir un dictionnaire, il existe deux syntaxes usuelles. La première consiste à parcourir l'ensemble des clés du dictionnaire, et de récupérer les valeurs par l'intermédiaire de la clé. Comme pour les listes, on utilise ici des itérateurs.

```
for k in mon_dico.keys(): #k est l'une des clees
  print ( k + " a pour note " + str( mon_dico[k] ) )
```

Pour éviter d'utiliser la notation crochetée pour accéder aux valeurs, il est possible d'utiliser la syntaxe suivante :

```
for k,v in mon_dico.items():
  print ( k + " a pour note " + str(v) )
```

5 Les fonctions

Les fonctions sont des outils primordiaux pour structurer votre code. L'objectif des fonctions est de décomposer votre programme en petites fonctionnalités qui peuvent être réutilisées et sont plus simples à implémenter séparément.
Les fonctions et les classes d'objets sont différentes structures de sous-programmes qui ont été imaginées par les concepteurs des langages de haut niveau afin de décomposer les programmes Python.
Nous allons décrire ici l'utilisation de fonctions sous Python, utile pour la réalisation de scripts même simples.

5.1 Définir une fonction

La syntaxe Python pour la définition d'une fonction est la suivante :

```
def nomDeLaFonction(liste de parametres):
    ...
    bloc d'instructions
    ...
```

Comme les structures de contrôle **if** et **while**, l'instruction **def** est une instruction composée. La ligne contenant **cette instruction se termine obligatoirement par un double point**, lequel introduit un bloc d'instructions que vous ne devez pas oublier d'**indenter** (comme pour l'intérieur d'une boucle ou d'une condition).
La liste de paramètres spécifie quelles informations il faudra fournir en guise d'arguments lorsque l'on voudra utiliser cette fonction (les parenthèses peuvent parfaitement rester vides si la fonction ne nécessite pas d'arguments).
Une fonction s'utilise pratiquement comme une instruction quelconque. Dans le corps d'un programme, un appel de fonction est constitué du nom de la fonction suivi de parenthèses.
Si c'est nécessaire, on place dans ces parenthèses le ou les arguments que l'on souhaite transmettre à la fonction. Il faudra en principe fournir un argument pour chacun des paramètres spécifiés dans la définition de la fonction, encore qu'il soit possible de définir pour ces paramètres des valeurs par défaut (voir plus loin).

5.1.1 Procédure : une fonction sans paramètres

Commençons par un exemple de procédure :

```
def table7():
    n = 1
    while n <11 :
        print n * 7
        n = n +1
```

En entrant ces quelques lignes, nous avons défini une fonction très simple qui calcule et affiche les 10 premiers termes de la table de multiplication par 7. Notez bien les parenthèses, le double point, et l'indentation du bloc d'instructions qui suit la ligne d'en-tête (c'est ce bloc d'instructions qui constitue le corps de la fonction proprement dite).
Si vous exécutez votre code à ce stade, il ne se passera rien. Vous avez uniquement demandé au programme de créer la fonction. Pour utiliser la fonction que nous venons de définir, il faut faire un appel de fonction ainsi :

```
table7()
```

L'instruction **table7** peut être maintenant considéré comme une nouvelle instruction.
Nous pouvons maintenant réutiliser cette fonction à plusieurs reprises, autant de fois que nous le souhaitons. Nous pouvons également l'incorporer dans la définition d'une autre fonction, comme dans l'exemple ci-dessous :

```
def table7triple ():
    print 'La table par 7 en triple exemplaire :'
    table7()
    table7()
    table7()

#appel de la fonction
table7triple ()
```

Une fonction est donc en quelque sorte une nouvelle instruction personnalisée, que vous ajoutez vous-même librement à votre langage de programmation.

5.1.2 Fonction avec paramètres

Dans nos derniers exemples, nous avons défini et utilisé une fonction qui affiche les termes de la table par 7. Supposons à présent que nous voulions faire de même avec la table par 9. Nous pouvons bien entendu réécrire entièrement une nouvelle fonction pour cela.
Lorsque nous appellerons cette fonction, nous devrons bien évidemment pouvoir lui indiquer quelle table nous souhaitons afficher. Cette information que nous voulons transmettre à la fonction au moment même où nous l'appelons s'appelle un argument.

Dans la définition d'une telle fonction, il faut prévoir une variable particulière pour recevoir l'argument transmis. Cette variable particulière s'appelle un paramètre. On lui choisit un nom en respectant les mêmes règles de syntaxe que d'habitude (pas de lettres accentuées, etc.), et on place ce nom entre les parenthèses qui accompagnent la définition de la fonction.

```
#definition de la fonction
def table(base):
    n = 1
    while n <11 :
        print n * base,
        n = n +1

# utilisation de la fonction
table(13)
table(9)
```

La fonction **table()** telle que définie ci-dessus utilise le paramètre base pour calculer les dix premiers termes de la table de multiplication correspondante. Pour tester cette nouvelle fonction, il nous suffit de l'appeler avec un argument.
Dans ces exemples, la valeur que nous indiquons entre parenthèses lors de l'appel de la fonction (et qui est donc un argument) est automatiquement affectée au paramètre dans la fonction.

De la même manière, il est possible d'avoir des fonctions avec plusieurs arguments

```
#definition de la fonction
def tableMulti(base, debut, fin):
    print 'Fragment de la table de multiplication par', base,':'
    n = debut
    while n <= fin :
        print n,'x',base, '=', n * base
        n = n +1

# utilisation de la fonction
tableMulti(12, 5, 9)
```

5.1.3 Utilisation d'une variable comme argument

Dans les exemples qui précèdent, l'argument que nous avons utilisé en appelant les fonctions était à chaque fois une constante (la valeur 13, puis la valeur 9). Cela n'est nullement obligatoire. Comme dans l'exemple ci-dessous, l'argument dans l'appel d'une fonction peut être une variable ou même le résultat d'une autre opération.

```
t= 11
d= 5
f= 2
while t<21:
    tableMulti(t, d, f*5)
    t = t+1
    d = d+3
    f += 1 # autre syntaxe de l'incrementation
```

Remarque 3 - Nom des arguments

Le nom d'une variable que nous passons comme argument n'a rien à voir avec le nom du paramètre correspondant dans la fonction.
Ces noms peuvent être identiques si vous le voulez, mais vous devez bien comprendre qu'ils ne désignent pas la même chose (en dépit du fait qu'ils puissent contenir une valeur identique).

5.1.4 Ce que calcule la fonction

Dans les deux cas précédents, les fonctions réalisaient des affichages. Dans de nombreux cas, une fonction à pour objectif de calculer quelque chose. Dans ce cas, on dit que la fonction doit "retourner" un résultat. En poursuivant notre exemple, nous allons modifier la fonction pour que celle ci construise une liste qui contiendra les chaînes de caractères de la table de multiplication.

```
def tableMulti(base, debut, fin):
    l=list()
    n = debut
    while n <= fin :
        t = str(n) + 'x' + str(base) + '=' + str( n * base )
        l.append( t )
        n = n+1
    return l

#appel de la fonction et recuperation du resultat dans 'r'
r = tableMulti(7, 2, 5)
```

Si vous exécutez cet exemple, il ne se passe "rien" ... enfin, rien ne s'affiche, mais la fonction a bien été utilisée et le résultat qu'elle a calculée (variable `l` dans la fonction) a été récupéré lors de l'appel de fonction dans la variable `r`. Cette variable peut être utilisée dans la suite du programme (`r` est ici une liste). Pour afficher la table de multiplication, il suffit alors de faire :

```
for t in r:
    print t
```

5.2 Variables locales, variables globales

Lorsque nous définissons des variables à l'intérieur du corps d'une fonction, ces variables ne sont accessibles qu'à l'intérieur de la fonction. On dit que ces variables sont des **variables locales** à la fonction.
Les variables définies à l'extérieur d'une fonction sont des **variables globales**. Leur contenu est "visible" de l'intérieur d'une fonction, mais la fonction ne peut pas le modifier. Exemple :

Exemple 5 - Visibilité des variables

Considérons le programme ci-dessous.
A l'intérieur de cette fonction, une variable `p` *est définie, avec 20 comme valeur initiale.*
Au niveau principal, les deux variables globales `p` *et* `q` *sont initialisés par 15 et 38. Ainsi le même nom de variable* `p` *a été utilisé ici à deux reprises, pour définir deux variables différentes : l'une est globale et l'autre est locale.*
On constate en effet que lorsque la fonction `mask()` *est lancée, la variable globale* `q` *y est accessible, puisqu'elle est imprimée correctement. Pour* `p`*, par contre, c'est la valeur attribuée localement qui est affichée.*
On pourrait croire d'abord que la fonction `mask()` *a simplement modifié le contenu de la variable globale* `p` *(puisqu'elle est accessible). Les lignes suivantes démontrent qu'il n'en est rien : en dehors de la fonction* `mask()`*, la variable globale* `p` *conserve sa valeur initiale.*

```
def mask():
    p = 20
    print p, q

p, q = 15, 38
mask()
print p, q
```

Ce programme affiche :

```
20 38
15 38
```

Il est très utile d'avoir des variables qui soient locales, c'est-à-dire en quelque sorte confinées à l'intérieur d'une fonction. Cela signifie en particulier que vous pourrez définir toutes vos fonctions sans vous préoccuper le moins du monde des noms de variables qui y sont utilisées dans les autres fonctions : ces variables ne pourront en effet jamais interférer avec celles que vous aurez vous-même définies par ailleurs.
En contrepartie, il est nécessaire de bien savoir jouer avec les paramètres et les retours de fonctions, seule manière propose de transmettre des valeurs entre l'intérieur de la fonction et l'extérieur.

5.3 Exemple

Exemple 6 - Savoir si un point se situe dans un cercle de centre et rayon fixés

Dans un programme, on a besoin de savoir si un point se trouve dans un cercle donné (cercle défini par sa position et son rayon). Il peut être intéressant de séparer cette fonctionnalité du reste du code pour le rendre lisible et peut être réutilisée notre fonction plus tard. Je veux donc pouvoir faire écrire facilement quelque chose comme :

```
if isincircle (x,y, x0,y0,r):
  print "le point est dans le cercle !"
```

Il faut donc construire une fonction **isincircle** *qui aura comme paramètre les coordonnées du point, les coordonnées du centre du cercle et son rayon.*
Si dessous, j'ai même décomposé la fonction en construisant une fonction dédiée au calcule de la distance euclidienne entre deux points.

```
def distance (x1,y1,x2,y2):
  return (((x1-x2)**2+(y1-y2)**2)**0.5)

def isincircle (x1,y1,x0,y0,r):
  if ( distance(x1,y1,x0,y0)<=r ):
    return True
  else :
    return False

# utilisation de la fonction
if isincircle (1,2,4,6,5) :
  print "le point est dans le cercle !"
else :
  print "le point n'est pas dans le cercle !"
```

CHAPITRE

III Données géographiques vectorielles

Les données géographiques représentées sous la forme vectorielle permettent de représenter des points, des lignes ou des polygones géométriques ayant des caractéristiques propres (leurs attributs).

1 Lire et écrire des informations depuis des fichiers Shapefile

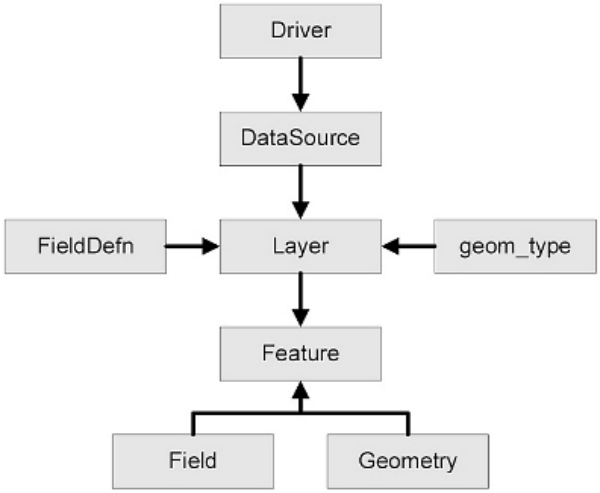


FIGURE III.1 – Organisation logique des objets consituant une source de données vectorielles.

La Figure III.1 illustre l'organisation logique des données vectorielles pour la librairie GDAL. Il est intéressant de noter que cette organisation est structurante pour l'ensemble de traitement qui devront être réalisés.
Dans le cas des fichiers Shapefile, cette structure est également fortement lié à l'organisation des fichiers : un fichier Shapefile ne comprend finalement que la partie droite du schéma : les formes ! Mais les autres informations concernant les attributs (*fields*) ne sont pas dans ce fichier mais dans un fichier `.dbf` qui correspond à une représentation de la table attributaire.

1.1 Les éléments de base

Lorsqu'on utilise une librairie, la première difficulté consiste à comprendre la logique de l'organisation des traitements. Dans le cas qui nous intéresse, je l'aborde par les objets.
Dans la librairie GDAL, il est important de connaître les différents On s'intéresse ici à des données issues d'un format vectoriel, typiquement un fichier Shapefile.

1.1.1 Ouverture et fermeture d'un fichier Shapefile en lecture

La lecture d'un fichier nécessite de disposer d'un outil capable d'interpréter correctement un format de fichier spécifique, c'est le rôle du **driver** (*cf.* exemple ci-dessous). Dans le cas de l'ouverture d'un Shapefile, il faut utiliser un driver initialisé avec la chaîne de caractère `'ESRI Shapefile'` (exactement).
L'exemple ci-dessous illustre l'ouverture d'un fichier Shapefile :

```
1 import os #chargement de la librairie os
2 import sys #chargement de la librairie systeme
3 import ogr #chargement de la librairie OGR
4 from osgeo.gdalconst import * # chargement des constantes : pour GA_ReadOnly
5
```

```
6  #on va ouvrir un fichier shapefile
7  driver = ogr.GetDriverByName('ESRI Shapefile')
8
9  #se placer dans votre repertoire de travail
10 os.chdir('/home/guyet/data/ProgSIG')
11
12 #ouverture du fichier
13 datasource = driver.Open('sites.shp', GA_ReadOnly)
14 if datasource is None:
15   # cas d'erreur ( inexistence du fichier )
16   print ('Could not open file')
17   sys.exit(1) #sortir du programme
```

La fonction `driver.Open(...)` permet de récupérer un objet qui représente une source de données. Dans le cadre général, ce pourrait être une base de données, un flux internet (WFS), une base de données géographique ou un fichier. Dans le cas de l'exemple proposé, nous chargeons un fichier.

- Le premier paramètre de la fonction donne le nom du fichier ! Ici, nous avons commencé par imposer de changer le répertoire de travail grâce à la ligne 10 (fonction **chdir**), il suffit alors de donner le nom du fichier sans son chemin d'accès. Une alternative aurait consister à donner son chemin "absolu", *c.-à-d.* `"/home/guyet/data/ProgSIG/sites.shp"`
- Le second paramètre permet d'indiquer le mode de lecture du fichier **GA_ReadOnly** pour ouvrir le fichier en lecture seule (pas de modification possible) et **GA_Update**, pour ouvrir le fichier avec possibilité de modification. Dans le cas où vous avez importé les constances de GDAL, vous pouvez utiliser **GA_ReadOnly** et **GA_Update**.

La fermeture de la source de données peut se faire à l'aide de l'appel suivant :

```
datasource.Destroy()
```

Remarque 4 - Alternative à l'utilisation des constantes GDAL

Dans la plupart des cas, les programmes n'utilisent pas les constantes prédéfinies de GDAL. Celles-ci peuvent être substituées directement par leurs valeurs. **GA_ReadOnly** correspond en fait à la valeur 0 et **GA_Update** à la valeur 1.
Vous pourrez vous en rendre compte en tapant les lignes suivantes directement en ligne de commande d'une console Python :

```
from osgeo.gdalconst import *
print (GA_Update)
print (GA_ReadOnly)
```

Remarque 5 - Utilisation préférable de OpenEx

Dans les versions plus récente de OGR/GDAL, il semble préférable d'utiliser la fonction **OpenEx** de GDAL pour ouvrir une source de données vectorielles.
Dans ce que j'ai pu montré ci-dessus, l'ouverture d'un fichier se fesait de cette manière

```
driver = ogr.GetDriverByName("GeoTiff")
datasource = driver.Open(source_path, 0)
```

avec la nouvelle méthode, il est possible de faire la même chose de la sorte :

```
src = gdal.OpenEx(source_path, 0)
```

Il n'y a donc plus besoin d'avoir à définir un driver dans cette solution. Simplement à donner le nom du fichier et son mode d'ouverture. Le type de fichier est identifié automatiquement par la structure du fichier à ouvrir.
Néanmoins, le concept de *driver* reste utile lors de la création de nouveaux fichiers (cf. Section 1.3).

1.1.2 La couche vectorielle

L'objet le plus générique pour de l'information géographique, est la notion de **couche** (*layer*). Lorsqu'on ouvre un fichier *shapefile*, on récupère une couche de données vectorielles.
L'accès à une couche vectorielle depuis un fichier Shapefile se fait au travers de la fonction **GetLayer()** de la source de données.

```
layer = datasource.GetLayer()
```

L'objet **layer** représente la couche de données vectorielles. Il est possible de récupérer des informations sur cette couche :

— récupération du nombre de formes (nommés *features* en anglais)

```
numFeatures = layer.GetFeatureCount()
print('Feature count: ' + str(numFeatures))
print('Feature count:', str(numFeatures))
```

— l'étendue de la couche

```
extent = layer.GetExtent()
print('Extent:', extent)
print('UL:', extent[0], extent[3]) # upper-left corner
print('LR:', extent[1], extent[2]) # lower-right corner
```

— le nom de la couche

```
print( layer.GetName() )
```

— le type de géométrie (points, lignes, polygones, etc.). On reviendra plus tard sur les différents types de géométrie. Les types de géométrie sont représentés par des entiers [1]

```
print( layer.GetGeomType() )
```

Il est surtout intéressant de savoir parcourir chacune des formes (*feature*) de la couche. Pour cela, on dispose d'au moins trois manières. Dans le cas simple où vous souhaitez traiter individuellement chacune des features qui compose votre couche, le schéma le plus simple est le suivant :

```
for f in layer:
  # do something with f
```

Cette écriture très ■ pythonic ■ peut se lire ainsi : ■ pour chaque feature **f** dans la couche **layer** faire *quelque chose avec f* ■. La partie à l'intérieur du code décrira le traitement individuel à appliquer à **f** (*p.ex.* la transformation d'un attribut, la modification de sa géométrie, etc).

Ce schéma est le plus simple qui existe, mais il n'est malheureusement pas approprié à toutes les situations. Dans certaines situations particulières, il peut être utile de mieux maîtriser l'ordre de traitement de la boucle. Les deux solutions suivantes permettent de mieux contrôler le déroulement de la boucle :

— accès à un élément par sa position dans la liste des éléments. Dans l'exemple ci-dessous, on peut récupérer le premier élément de la couche, puis le dixième !

```
feature = layer.GetFeature(0)
feature = layer.GetFeature(10)
```

On peut alors parcourir chacune des features d'une couche en utilisant un index **i** qui va varier de 0 au nombre de feature total :

```
for i in range(1, layer.GetFeatureCount()):
  feature = layer.GetFeature( i )
  # do something here with feature
```

— utilisation d'un ■ iterateur interne ■ à la couche :

```
feature = layer.GetNextFeature()
while feature:
  # do something here
  feature = layer.GetNextFeature()
layer.ResetReading() #rembobine pour recommencer un parcourt !
```

La fonction **GetNextFeature**() passe à la ■ feature suivante ■. En fait, il faut imaginer qu'il y a un pointeur interne à la couche qui retient à quel feature la lecture en est. Lorsqu'on utilise cette fonction, le pointeur est décalé (tant que c'est possible) et elle donne le nouvelle feature.

1. Des valeurs constantes permettent de rendre ces entiers plus intelligibles : **ogr.wkbPoint**, **ogr.wkbPolygon**, ... La signification de ces entiers peut être trouvée dans la norme OGC 06-103r4 "OpenGIS Implementation Standard for Geographic information – Simple feature access – Part 1 : Common architecture", v1.2.1 (voir p. 66)

Remarque 6 - Penser à rembobiner

Cette méthode est à utiliser avec la plus grande attention ! En particulier, il faut bien penser à ■ rembobiner ■ la lecture lorsqu'on doit parcourir plusieurs fois une couche. Sinon, à la fin de la première lecture, l'itérateur reste à la fin ... Dans le code proposé, on rembobine après avoir lu, mais on pourrait très bien commencer par rembobiner !

1.2 Les *feature*

Dans un format vectorielle, une forme est associée à :

- une géométrie : elle donne les caractéristiques géométrique d'une forme, son type (point, ligne, polygone, multi-polygone) et sa description (positions spatiale du point, etc.)
- une liste de valeurs définies pour des attributs. Dans le langage des SIG, un attribut est appelé un *Field*. Les attributs sont définies à l'échelle de la couche et chaque forme a ses propres valeurs pour les attributs de la couche.

Pour accéder aux éléments de la couche, il est possible d'utiliser les fonctions suivantes :

- **GetField**('...'), en précisant le nom de l'attribut dont on souhaite la valeur
- **GetFieldAsInteger**('...') ou **GetFieldAsString**('...') sont des spécialisations de la fonction précédente pour être sûr de récupérer une information avec le type souhaité (si les données le permettent !)

Pour l'utilisation de ces fonctions, il est nécessaire de connaître les noms des attributs pour y accéder. Dans le cadre de ce tutoriel, on supposera que ce sera toujours le cas. Néanmoins, il est bien évidemment possible de lister l'ensemble des attributs d'une couche (avec leur noms et leurs caractéristiques) pour avoir des programmes génériques. Il faut conserver en tête que plus le programme est générique, plus il sera complexe. Ce n'est pas notre objectif que de faire des choses artificiellement complexes, cherchons déjà à traiter nos données.

Pour accéder à la géométrie d'un élément de la couche, il existe la fonction **GetGeometryRef()**. Cette fonction construit une représentation de la géométrie. En fonction de son type, la géométrie offre différentes possibilités.
Dans le cas d'un point, on peut récupérer ses positions spatiales (x, y) :

```
geometry = feature.GetGeometryRef()
x = geometry.GetX()
y = geometry.GetY()
```

Remarque 7 - Signification des valeurs X et Y

Nous reviendrons plus tard sur le gestion des systèmes de coordonnées. D'ores et déjà, il est intéressant de noter que du point de vue informatique, les valeurs enregistrées comme coordonnées géographique d'un point n'est soumis à aucune contrainte ! Il s'agit simplement de nombre ... et leur signification n'est possible que parce que la couche est associée à un SRS (Spatial Reference System).

Lorsque les traitements sur une forme sont terminés, il est utile de détruire explicitement cet objet par l'instruction **Destroy()**. Ceci permet de conserver de la mémoire pour la suite du traitement.

```
feature.Destroy()
```

1.2.1 Exemple 1 : afficher les arrêts de bus de Rennes

L'exemple ci-dessous illustre le parcours d'un fichier shapefile `tco-bus-topologie-pointsarret-td.shp` contenant l'ensemble des arrêts de bus de Rennes [2]. L'exemple affiche uniquement les arrêts de la ville de Rennes et les compte.

Le programme peut être téléchargé ici.

```
import ogr, os, sys
from osgeo.gdalconst import * # chargement des constantes : pour GA_ReadOnly

os.chdir('./LiveData/data/vecteurs/bus_star/')
driver = ogr.GetDriverByName('ESRI Shapefile')
```

2. Fichier disponible à l'url suivante : `https://data.rennesmetropole.fr/`.

```
# ouverture du fichier d'entree
arretsBusDS = driver.Open('tco-bus-topologie-pointsarret-td.shp', GA_ReadOnly)
if arretsBusDS is None:
  print ('Could not open file')
  sys.exit(1)
arretsBusLayer = arretsBusDS.GetLayer()

#Pour toutes les formes de la couche d'entree, faire
cnt = 0
for arretBusFeature in arretsBusLayer:
    if arretBusFeature.GetField('nomcommune')=="Rennes":
        geom = arretBusFeature.GetGeometryRef()
        print ("Arret '" + arretBusFeature.GetField('nom') +"', localisation: ("+ \
          str(geom.GetX()) +","+ str(geom.GetY()) +").")
        cnt += 1

print ("Il y a " + str(cnt) + " arrets de bus dans Rennes.")

# Fermeture des sources de donnees
arretsBusDS.Destroy()
```

Vous pourrez tester et modifier le programme pour n'afficher, par exemple, que les arrêts de la ville de Betton, ou entre d'afficher le numéro de l'arrêt (regarder la couche en utilisant QGis pour connaître le nom des attributs ou de leurs valeurs).

1.2.2 Lister les attributs d'une couche

Le code ci-dessous permet de lister les attributs d'une couche. La fonction **GetLayerDefn()** permet d'obtenir une "liste" des attributs de la couche. Il ne s'agit pas d'une liste Python, mais d'un objet spécifique. Pour accéder à un élément de cette liste, il est nécessaire d'utiliser une fonction **GetFieldDefn(i)** où **i** est l'index de l'attribut. Finalement, les quatre propriétés des attributs sont accessibles via des fonctions dédiées (voir l'exemple ci-dessous).

```
driver = ogr.GetDriverByName('ESRI Shapefile')

dataSource = driver.Open("tco-bus-topologie-pointsarret-td.shp", )
layer = dataSource.GetLayer()

layerDefinition = layer.GetLayerDefn()
print "Name - Type Width Precision"
for i in range(layerDefinition.GetFieldCount()):
    fieldName = layerDefinition.GetFieldDefn(i).GetName()
    fieldTypeCode = layerDefinition.GetFieldDefn(i).GetType()
    fieldType = layerDefinition.GetFieldDefn(i).GetFieldTypeName(fieldTypeCode)
    fieldWidth = layerDefinition.GetFieldDefn(i).GetWidth()
    GetPrecision = layerDefinition.GetFieldDefn(i).GetPrecision()

    print fieldName + " - " + fieldType+ " " + str(fieldWidth) + " " + str(GetPrecision)
```

1.3 Écriture d'un fichier shapefile

La création d'un fichier *shapefile* reprend la même logique que la lecture d'un fichier. Pour la création d'une couche éditable, il faudra définir un *driver* et une source de données, puis chaque feature qui doit composer la couche sera ajoutée une à une.

1.3.1 Création d'une couche éditable

La création d'un nouveau fichier suit le même principe que l'ouverture d'un fichier : il est nécessaire d'avoir un driver qui saura comment enregistrer le fichier :

```
fout = '/home/guyet/data/ProgSIG/output.shp'
dsout = driver.CreateDataSource(fout)
```

dsout est alors une source de données correspondant au fichier **fout** dans lequel on va pouvoir écrire des données. Ici, on utilise un chemin absolu, mais si le répertoire de travail a été défini au préalable, il est possible de se limiter à doner le nom du fichier.

Pour la création d'une couche, il est nécessaire d'indiquer quel va être le type de forme que la couche contiendra (points, lignes, etc.). La fonction d'ajout d'une couche à une source de données est la suivante :

```
layerout = dsout.CreateLayer('Nouvelle couche', geom_type=ogr.wkbPoint)
```

Notez que ceci ne peut être fait que sur une source de données qui a été créée pour enregistrer des données.

Une fois que la couche a été définie, il est possible (et souvent nécessaire) de lui ajouter les attributs des formes de la couche. Je rappelle ici qu'il s'agit de définir uniquement les entêtes de la table des attributs. On donnera une valeur à ces attributs pour chaque forme !
Dans l'exemple ci-dessous, on définit un nouvel attribut nommé `id` et on indique qu'il s'agira de nombre entier. On ajoute ensuite cet attribut à la couche **layerout**. Puis, on ajoute un second attribut `descr` qui est une chaîne de caractères de longueur 40 au plus.

```
fieldDefn = ogr.FieldDefn('id', ogr.OFTinteger)
layerout.CreateField(fieldDefn)
fieldDefn = ogr.FieldDefn('descr', ogr.OFTString)
fieldDefn.SetWidth(40)
layerout.CreateField(fieldDefn)
```

Dans ce code, la fonction **FieldDefn** permet de créer un nouvel attribut à partir de deux informations : le nom de l'attribut et son type (entier, texte, ...). Ensuite, la fonction **CreateField** d'une couche de données ajoute cet attribut à la couche. Dans certains cas, comme pour l'attribut `descr`, l'attribut peut être configurer plus finement. Ici, on donne la taille maximum de l'attribut.

1.3.2 Création de features

La création de nouvelles formes/features ne peut se faire qu'une fois qu'une couche a été définie ainsi que ses attributs. En particulier, il est nécessaire de connaître la taille de chaque attribut pour savoir comment les enregistrer. Avant tout chose, on a besoin de récupérer la description de tous les attributs :

```
featureDefn = layerout.GetLayerDefn()
```

La création d'une nouvelle forme se fait ensuite en quatre étapes :

1. On créé effectivement la nouvelle forme avec les bons attributs pour la couche

   ```
   feature = ogr.Feature(featureDefn)
   ```

2. On donne une géométrie à la forme

   ```
   point = ogr.Geometry(ogr.wkbPoint)
   point.AddPoint(10,20)
   feature.SetGeometry(point)
   ```

3. Ajouter les valeurs pour chaque attribut

   ```
   feature.SetField('id', 23)
   ```

4. Ajouter la forme à la couche

   ```
   layerout.CreateFeature(feature)
   ```

1.3.3 Exemple 2 : recopie d'un fichier shapefile

L'exemple ci-dessous illustre la recopie d'un fichier shapefile, *feature* par *feature*.

```
import ogr, os, sys
from osgeo.gdalconst import *

os.chdir('/home/guyet/data')
driver = ogr.GetDriverByName('ESRI Shapefile')

# ouverture du fichier d'entree
inDS = driver.Open('sites.shp', GA_ReadOnly)
if inDS is None:
  print('Could not open file')
  sys.exit(1)
inLayer = inDS.GetLayer()

# creation du fichier de sortie : couche de points
if os.path.exists('test.shp'):
```

```
  driver.DeleteDataSource('test.shp')
outDS = driver.CreateDataSource('test.shp')
if outDS is None:
  print('Could not create file')
  sys.exit(1)

outLayer = outDS.CreateLayer('test', geom_type=ogr.wkbPoint)

# definition des attributs de la couche de sortie par recopie
fieldDefn = inLayer.GetFeature(0).GetFieldDefnRef('id')
outLayer.CreateField(fieldDefn)
fieldDefn = inLayer.GetFeature(0).GetFieldDefnRef('cover')
outLayer.CreateField(fieldDefn)

# featureDefn decrits les attributs de la couche outLayer
featureDefn = outLayer.GetLayerDefn()

#Pour toutes les formes de la couche d'entree, faire
for inFeature in inLayer:
  # Creation d'une forme par recopie
  outFeature = ogr.Feature(featureDefn)
  outFeature.SetGeometry(inFeature.GetGeometryRef())
  outFeature.SetField('id', inFeature.GetField('id'))
  #NB: seul l'attribut 'id' est recopie ici !

  # Ajouter de la forme a la couche de sortie
  outLayer.CreateFeature(outFeature)

  # destruction des formes
  inFeature.Destroy()
  outFeature.Destroy()

# Fermeture des sources de donnees
inDS.Destroy()
outDS.Destroy()
```

Le programme peut être téléchargé ici.

On peut noter qu'à la fin de cet exemple, deux instructions viennent conclure l'exécution du programme appelant les fonctions de destruction des sources de données. Ces instructions ont pour effet de forcer à ■ fermer ■ le fichier. Cette étape est souvent indispensable après l'écriture d'information. C'est parfois la seule trace d'une modification.

1.4 Exercices

Exercice 1 (Lecture d'un fichier Shapefile)

Question a) Commencez par explorer les données avec QGis

Question b) Créer un script Python pour déterminer le nombre de features de la couche *Pour cette première question, il s'agit de mettre en place correctement votre script : chargement des librairies, définition du répertoire de travail et ouverture d'une couche (inutile de parcourir les features)*

Question c) Compléter ensuite votre script pour afficher le type de forme que comporte la couche (vous comparer la valeur obtenu aux constantes **ogr.wkbPoint**, **ogr.wkbLineString** et **ogr.wkbPolygon**), l'étendue de la couche ainsi que la liste de ses attributs.

Question d) Compléter votre script Python pour afficher en ligne (**print**) les `ID`, `COVER` et les positions `x y` de chaque point du fichier `sites.shp`.

Exercice 2 (Stations de vélos) *À l'aide d'un programme Python et de la couche des stations de vélo de Rennes (`supports-velos.shp`), répondre aux questions suivantes :*

Question a) Combien y a-t-il de stations de vélo à rennes ?

Question b) Combien y a-t-il d'emplacements de vélo à rennes ?

Question c) Combien y a-t-il de stations de vélo qui ont strictement plus de 10 supports ?

Question d) Lister toutes les rues qui ont un support de vélo (doublons autorisés) dans la ville de Rennes ?

Question e) Faire un programme qui demande l'identifiant d'une station et affiche ses coordonnées spatiales et le nombre de supports de vélo. Vous pourrez utiliserez l'instruction `input()`.

Question f) Pour une station donnée (par exemple, à la place de la gare, support vélo avec l'attribut *objectid* égal à 258) :
- *quelle est la station la plus proche (à vol d'oiseaux) ?*
- *combien y a-t-il de stations à moins de 500m ?*

Pour répondre à ces questions, vous pourrez utiliser les algorithmes étudiés en cours d'algorithmique.

Question g) Déterminer toutes les distances (à vol d'oiseau) entre pairs de station de vélos ? Quelle est la moyenne des distances ?

Exercice 3 (Décomposition d'un fichier Shapefile)

Question a) Écrire un script Python créant un fichier `output.shp` qui ne contiendra que les points correspondant à la valeur `trees` de l'attribut `COVER` du fichier `sites.shp`.
Le fichier ne conservera que l'attribut `ID` du fichier d'origine, l'autre attribut étant inutile.

Question b) Transformer votre script en **une fonction** qui prendra en paramètre le nom du fichier d'entrée, le nom du fichier de sortie et la valeur de l'attribut dont on conserve les points.

Question c) Écrire un script Python utilisant votre fonction pour créer un fichier pour chacune des modalités de l'attribut `COVER` *Vous pourrez récupérer la liste des modalités de l'attribut en allant visualiser les données sous QGis.*

Exercice 4 (Fusion de deux fichiers shapefile)

Question a) Écrire un script qui construit un fichier shapefile unique à partir de deux fichiers shapefile de points.
Vous supposerez que les deux fichiers utilisent le même système de projection.

Question b) Tester votre script en reconstruisant un fichier `site_rebuilt.shp` à partir des décompositions obtenues lors de l'exercice précédent.

Question c) Modifier votre script pour ajouter un nouvel attribut dans le fichier fusion qui contiendra l'information sur l'origine de chaque forme. Vous utiliserez un attribut au format texte (**ogr.OFTString**) de longueur 10.

1.5 Utilisation des filtres

Revenons maintenant quelques instants sur l'exemple de l'affichage des bus de Rennes ou de la sélection des sites correspondant à un type d'objet. Cette opération qui consiste à parcourir une couche selon une sélection d'objets est assez usuel, et plutôt que de le faire par de la programmation, il est possible d'utiliser des **filtres**.
Les filtres vont sélectionner un sous-ensemble de features de votre couche selon un critère que vous aller définir (par exemple, les features d'arrêt correspondant à Rennes), une fois ce filtrage appliqué, le parcours de la couche se fera en tenant compte de votre sélection.
L'exemple ci-dessous reprend le parcours des arrêts de bus de *Rennes* uniquement. Il est très important de bien comprendre l'argument de la fonction `SetAttributeFilter` qui contient une chaîne de caractères qui exprime la restriction à opérer. Dans ce cas, on a `"nomcommune=='Rennes'"` : Il s'agit bien d'une chaîne de caractères définie entre guillemets, mais comme la restriction se fait sur un attribut texte, la valeur `'Rennes'` doit également être mise entre guillemets pour définir que c'est ce texte auquel doit correspondre l'attribut `nomcommune`. Il faut ici forcément alterner les deux types de guillemets : simples et doubles pour que Python comprenne cette expression.

```
1 import ogr, os, sys
2 from osgeo.gdalconst import * # chargement des constantes : pour GA_ReadOnly
3
4 driver = ogr.GetDriverByName('ESRI Shapefile')
5
6 # ouverture du fichier d'entree
7 arretsBusDS = driver.Open('tco-bus-topologie-pointsarret-td.shp', GA_ReadOnly)
8 if arretsBusDS is None:
9   print ('Could not open file')
```

```
  sys.exit(1)
arretsBusLayer = arretsBusDS.GetLayer()

arretsBusLayer.SetAttributeFilter("nomcommune=='Rennes'")

for arretBusFeature in layer:
        geom = arretBusFeature.GetGeometryRef()
        print("Arret '" + arretBusFeature.GetField('nom') +"', localisation: ("+ \
         str(geom.GetX()) +","+ str(geom.GetY()) +").")

# Fermeture des sources de donnees
arretsBusDS.Destroy()
```

Un autre type de filtre similaire permet de faire une sélection des features à parcourir par une **restriction spatiale**. Un objet représentant une couche offre deux fonctionnalités principales :

- la fonction **SetSpatialFilter (geom)** qui prend en paramètre une géométrie et qui applique un filtre en utilisant une géométrie (de type polygone ou multi-polygone a priori)
- la fonction **SetSpatialFilterRect (xm, ym, xM, yM)** qui permet une sélection à partir des coordonnées d'un rectangle. Cette fonction se charge simplement de créer la forme rectangulaire pour vous ...

L'exemple ci-dessous illustre l'ajout d'un filtre spatial défini par un polygone créé "à la main" (ici, construit par une séquence WKT, c'est-à-dire une succession de points).

```
driver = ogr.GetDriverByName("ESRI Shapefile")
dataSource = driver.Open("tco-bus-topologie-pointsarret-td.shp", GA_ReadOnly)
layer = dataSource.GetLayer()

wkt = "POLYGON ((-103.81402655265633 50.253951270672125,-102.94583419409656
     51.535568561879401,-100.34125711841725 51.328856095555651,-100.34125711841725
     51.328856095555651,-93.437060743203844 50.460663736995883,-93.767800689321859
     46.450441890315041,-94.635993047881612 41.613370178339181,-100.75468205106476
     41.365315218750681,-106.12920617548238 42.564247523428456,-105.96383620242338
     47.277291755610058,-103.81402655265633 50.253951270672125))"

layer.SetSpatialFilter( ogr.CreateGeometryFromWkt(wkt) )

for feature in layer:
    print feature.GetField('nom')
```

Dans l'exemple ci-dessous, j'utilise les fonctionnalités de filtrage pour classer les arrêts de bus de la STAR par commune en croisant les informations relatives aux limites des communes de la métropole (couche de polygones) avec les données de la couche des arrêts de bus.

```
import ogr, os, sys
from osgeo.gdalconst import * # chargement des constantes : pour GA_ReadOnly

os.chdir('../../LiveData/data/vecteurs/bus_star/')
driver = ogr.GetDriverByName('ESRI Shapefile')

# ouverture du fichier d'entree
arretsBusDS = driver.Open('tco-bus-topologie-pointsarret-td.shp', GA_ReadOnly)
if arretsBusDS is None:
  print('Could not open file')
  sys.exit(1)
arretsBusLayer = arretsBusDS.GetLayer()


# ouverture du fichier d'entree
communesDS = driver.Open('limites-communales-referentielles-de-rennes-metropole-polygones.shp',
      GA_ReadOnly)
if communesDS is None:
  print('Could not open file')
  sys.exit(1)
communesLayer = communesDS.GetLayer()

# Pour toutes les communes, faire:
for Comm in communesLayer:
    print("====" + Comm.GetField("nom") + " ====")

    # Selection spatiale des arrets
    arretsBusLayer.SetSpatialFilter( Comm.GetGeometryRef() )
    # Pour chaque arret de la selection , faire :
```

```
    for arretBusFeature in arretsBusLayer:
        geom = arretBusFeature.GetGeometryRef()
        print ("Arret '" + arretBusFeature.GetField('nom') + "'")
    arretsBusLayer.ResetReading()

# Fermeture des sources de donnees
arretsBusDS.Destroy()
communesDS.Destroy()
```

Le programme peut être téléchargé ici.

Exercice 5 (Filtre interactif)

Question a) Écrire un script qui utilise les filtres pour afficher à l'écran la liste de tous les arrêts de bus d'une ville de la métropole de Rennes qui sera donné par l'utilisateur.
Pour demander à l'utilisateur la ville qui l'intéresse, vous pourrez utiliser la fonction `ville = input("donner une ville")`. *La difficulté réside ici dans la construction du filtre.*

1.6 Autres librairies similaires

Il existe d'autres modules qui permettent de manipuler et créer des couches vectorielles ou objets géographiques. On retrouve des fonctionnalités semblables, mais souvent pas aussi large que celle de GDAL. La plupart de ces librairies sont en fait elle-même basées sur GDAL. Leur intérêt est d'offrir souvent des syntaxes plus simples (et donc plus facile à écrire).
On peut donner comme exemples les libraires `fiona`, `shapefile`, Shapely, GeoDjango ou GeoPandas.

1.6.1 Utilisation de la librairie `fiona`

Ci-dessous, je propose un très court exemple de la librairie `fiona` qui intéressera certainement beaucoup d'entre vous.

```
import fiona

couche = fiona.open('tco-bus-topologie-pointsarret-td.shp')
# La description de la couche: les attributs et les types de geometrie
print (couche.schema)

#{'geometry': 'Point', 'properties': OrderedDict([(u'id', 'int:10'), (u'dip', 'int:2'), (u'dip_dir', 'int:3'), (u'dir',
        'int:9'), (u'type', 'str:10'), (u'x', 'int:10'), (u'y', 'int:10')])}

# premier feature
feature = couche.next()
print (feature) #c'est un dictionnaire

print ("coordonnées:" + str( feature['geometry']['coordinate'] )
print ("attributs:" + str( feature['properties'] )
print ("valeur attribut:" + str( feature['properties']['nom'] )

# parcours de tous les elements de la couche
for elem in couche:
  print ("Arrêt bus:" + str( elem['properties']['nom'] )

# récupération d'une liste de tous les features
elements = [elem for elem in couche]
```

1.6.2 Utilisation de la librairie `shapefile`

Lorsque vous ne traitez que des fichiers shapefile, une alternative possible à l'utilisation de la librairie GDAL/OGR est l'utilisation de la librairie Python nommée `pyshp` (nom du *package*) ou `shapefile` (nom de l'*import*).
Cette librairie a été conçue pour mettre à disposition, peut être plus facilement, des fonctionnalités usuelles de manipulation des shapefile.

```
import shapefile

# ouverture du fichier d'entree
r = shapefile.Reader("tco-bus-topologie-pointsarret-td.shp")
```

```
#### acces aux features ####
shapes = r.shapes()
len(shapes)

# type de premiere geometrie
print ( shapes[1].shapeType )

#### attributs de la couche ####
#lister les attributs de la couche
print ( r. fields )

# acces aux valeurs des attributs du 2e element (meme ordre que pour shapes)
print ( r.record(2) )

### Cas d'une couche de polygones ####

r = shapefile.Reader("limites-communales-referentielles-de-rennes-metropole-polygones.shp")

pg_shapes = r.shapes()
print ( shapes[1].shapeType )

# description du polygone
print ( len(shapes[1].points) ) # nb de points qui le compose
```

Le programme peut être téléchargé ici.

La librairie `shapefile` peut être utilisée également pour créer (resp. modifier) des shapefiles grâce à l'utilisation d'objet **Writer** (resp. **Editor**). Leur utilisation est plus technique et le lecteur intéressé pourra consulter l'aide en ligne[3] sur cette librairie.
On peut néanmoins noté que les notions de GDAL se retrouve de manière très similaire dans les librairies alternatives.

2 Manipulation des formes géométriques

2.1 Les formes de base

Toutes les formes sont des objets de la classe **geometry** représentent des formes géométriques de bases. Cette classe peut être spécialisée comme illustré par la Figure III.2.

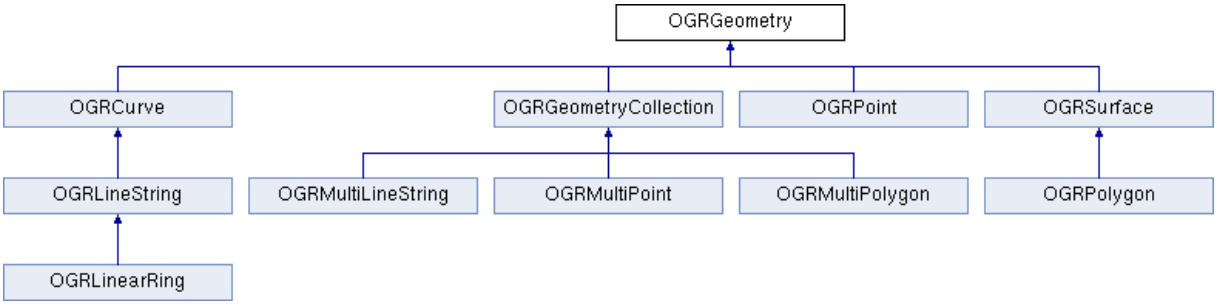


FIGURE III.2 – Classes de geometry

Les différents types spécifiques de géométries sont les suivantes :
- point (`wkbPoint`) : géométriquement défini par des coordonnées (x, y),
- une polyligne (`wkbLineString`) : une succession de points liés par des lignes (droites),
- un polyligne fermée (`wkbLinearRing`) : une successions de points qui définissent un anneau (le premier point est utilisé pour "fermer" la courbe,
- un polygone (`wkbPolygon`) : défini par une polyligne fermée dite "extérieure" et un ensemble de polylignes fermées dites "intérieures" (qui représente des extrusions de la forme définie par la polyligne "extérieure".

Finalement, ces formes de bases peuvent être regroupées sous la forme de MultiLineString, MultiPoint et MultiPolygone.

! Attention ! - Shapefile

Un fichier shapefile qui contient des polygones peut contenir des polygones ou des multi-polygones ...

3. `https://pypi.python.org/pypi/pyshp`

Remarque 8 - Pas de courbes !

Contrairement à d'autres librairies de dessin vectoriel (dans les usages différents de ceux de l'information géographique), les couches SIG vecteurs ne permettent pas de définir des formes curvilignes. Si vous souhaitez représenter un cercle ou bien une courbe, il faudra nécessairement l'approximer par des droites.

2.2 Création d'une nouvelle géométrie

Lors de la création d'une nouvelle géométrie, il faut :

1. créer un objet représentant la géométrie (utiliser la fonction **ogr.Geometry**)
2. spécifier les systèmes de coordonnées de la géométrie,
3. définir les propriétés géométriques de la géométrie (dépend de la nature de la géométrie)

Dans les exemples ci-dessous, on illustre la création de formes géométriques. On reviendra plus tard sur la définition du système de coordonnées.

Création d'un point

```
point = ogr.Geometry(ogr.wkbPoint)
point.AddPoint(10,20)
```

Création d'une ligne

```
line = ogr.Geometry(ogr.wkbLineString)
line.AddPoint(10,10)
line.AddPoint(20,20)
line.SetPoint(0,30,30)
```

Dans la dernière ligne, la fonction **SetPoint** transforme les coordonnées du premier point (identifié par l'indice 0)
Pour accéder aux informations d'une polyligne :

- **line.GetPointCount()** permet d'avoir le nombre de points de la ligne
- **line.getX(34)** et **line.getY(34)** permettent d'accéder aux coordonnées du 34eme points de la ligne (si il existe !)

Création d'un polygone
L'opération de création d'un polygone est plus complexe, elle nécessite de créer un anneau extérieur puis de faire les anneaux intérieurs.

```
#creation d'un anneau exterieur
ring = ogr.Geometry(ogr.wkbLinearRing)
ring.AddPoint(0,0)
ring.AddPoint(100,0)
ring.AddPoint(100,100)
ring.AddPoint(0,100)
ring.CloseRings()

#creation d'un anneau interieur
inring = ogr.Geometry(ogr.wkbLinearRing)
inring.AddPoint(25,25)
inring.AddPoint(75,25)
inring.AddPoint(75,75)
inring.AddPoint(25,75)
inring.CloseRings()

#creation du polygone
polygon = ogr.Geometry(ogr.wkbPolygon)
polygon.AddGeometry(ring)
polygon.AddGeometry(inring)
```

2.3 Destruction des géométries

À chaque fois que vous créer une géométrie, celle-ci est retenue en mémoire de l'ordinateur. Il est préférable d'indiquer explicitement à l'ordinateur qu'une géométrie ne sera plus utile dans la suite du programme pour qu'il libère de la mémoire [4]. Ceci est possible grâce à l'instruction **Destroy()**, par exemple :

```
polygon.Destroy()
```

2.4 Exemple de création d'un fichier Shapefile à partir de rien ...

L'exemple ci-dessous permet de créer des points dans un Shapefile à partir de saisies clavier de l'utilisateur. L'intérêt du programme est de donner la possibilité, en l'adaptant de générer automatiquement des formes complexes pour, par exemple, mener des analyses régionalisées.

Le programme peut être téléchargé ici.

```
# recuperation d'un driver
driver = ogr.GetDriverByName('ESRI Shapefile')

# creation d'un nouveau fichier
fin='test_generation.shp'
if os.path.exists(fin):
  driver.DeleteDataSource(fin)
ds = driver.CreateDataSource(fin)
if ds is None:
  print('Could not create file ', fin)
  sys.exit(1)

layer = ds.CreateLayer('test', geom_type=ogr.wkbPoint)

# ajoute d'un attribut aux points
fieldDefn = ogr.FieldDefn('id', ogr.OFTInteger)
layer.CreateField(fieldDefn)

while 1:
  val1=raw_input("Donnez une valeur de X: ")
  val1= int(val1)
  val2=raw_input("Donnez une valeur de Y (-1 pour quitter): ")
  val2= int(val2)
  if val2==-1:
    break

  # creation d'un nouveau point
  point = ogr.Geometry(ogr.wkbPoint)
  point.AddPoint(val1, val2)

  # creation de la feature avec ses attributs associe a la forme
  featureDefn = layer.GetLayerDefn()
  feature = ogr.Feature(featureDefn)
  feature.SetGeometry(point)
  feature.SetField('id', id)
  id = id+1

  #ajout a la couche
  layer.CreateFeature(feature)

  point.Destroy()
  feature.Destroy()

#fermeture du fichier
ds.Destroy()
```

2.5 Exercices

Exercice 6 (Marche aléatoire)

4. Pour l'exécution d'un programme la mémoire importante est la mémoire RAM. Celle-ci est en quantité limitée. Les programmes qui ne libère pas la mémoire qu'ils utilisent peuvent non-seulement s'arrêter, mais par la même provoquer des erreurs sur tout le fonctionnement de l'ordinateur. On parle alors de fuite mémoire.

Question a) Écrire un script Python créant un fichier `output.shp` qui contiendra une polyligne qui passe par tous les points du fichier `sites.shp`. On ne s'intéressera pas à l'ordre dans lequel sont pris en compte les points.

L'exemple ci-dessous illustre comment construire une liste de points (listes de tuples) ordonnées par les X :

```
shapeData = osgeo.ogr.Open('sites.shp')
layer = shapeData.GetLayer()
points = []
for index in xrange(layer.GetFeatureCount()):
  feature = layer.GetFeature(index)
  geometry = feature.GetGeometryRef()
  points.append( geometry )
```

L'exemple suivant illustre l'utilisation des nombres aléatoires :

```
import random
x=random.randint(1,10) #genere un nombre entier aleatoire entre 1 et 10 compris,
y=random.uniform(1,10) # genere un nombre reel aleatoire, selon une lois uniforme, entre 1 et 10.
```

Question b) Écrire un script Python créant un fichier `output.shp` qui génère une marche aléatoire de longueur l à partir des points de `sites.shp`.
Une marche aléatoire est une polyligne qui passe d'un point à un autre de manière aléatoire. Pour réaliser ce script, vous commencerez par construire un vecteur de points à partir des points localisant les sites donnés dans le fichier `sites.shp`. Ensuite, vous générerez une polyligne de l points en tirant aléatoirement des points dans ce vecteur. Cette polyligne sera ensuite enregistrée dans le Shapefile.

Exercice 7 (Décomposition de lignes) *Lors de la manipulation de couche de lignes comme des routes ou des cours d'eau, il n'est pas rare d'avoir l'intégralité d'un cours d'eau (et ses affluents) ou du réseau routier dans un seul objet ligne. La manipulation de cet objet peut se montrer difficile. On rappelle que dans les données vectorielles, une ligne est décrite par une succession de segments (ligne droite entre deux points).*
Dans cet exercice, l'objectif est de décomposer une couche de longues lignes en une couche de lignes équivalentes, mais pour laquelle tous les objets sont des segments (une ligne droite entre deux points).

Question a) Décomposition d'une ligne *Écrire une fonction* **decompose(line)** *qui prend en paramètre une géométrie de lignes et qui retourne une liste de géométries de ligne correspondant à tous les segments de la ligne en entrée.*

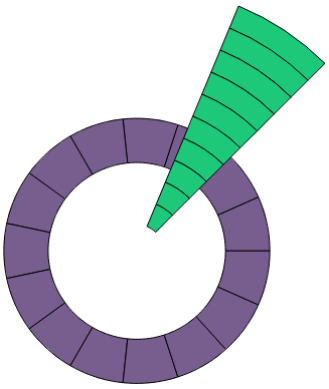
Question b) Décomposition d'une couche de lignes *Écrire une fonction* **decompose_file(fname)** *qui prend en paramètre un nom de fichier shapefile (lignes) et qui créé un nouveau fichier shapefile de lignes contenant le résultat de la décomposition des lignes du fichier par la fonction précédente. Tous les attributs des lignes seront recopiés.*

Exercice 8 (Création de transects aléatoires) *L'objectif de ce projet est de concevoir, développer et mettre à disposition une méthode de traitement d'images (indice de végétation SAVI) pour mettre en évidence le gradient de végétation ouest-est à partir d'imagerie multi-spectrale (dont rouge/proche infrarouge).*

Question a) Données de transects
Un transect désigne une ligne droite dans l'espace servant de guide pour localiser des relevés de valeurs physiques. Ces relevés peuvent alors être traités comme des successions de valeurs pour mettre en évidence un comportement spécifique de la valeur physique selon l'axe d'étude. Dans le cas des images de télédétection, la valeur physique à prélevée se trouve être la valeur des pixels dans l'image. Pour construire un transect, il faut donc prélever des valeurs sur l'image régulièrement selon une droite.
L'objectif est donc de construire une fonction qui vous permettra de mettre en évidence un gradient dans une image satellite. La fonction devra permettre de facilement choisir les points de départ et d'arrivé du transect, ainsi que la distance entre chaque point. La fonction construira un vecteur contenant la suite des valeurs prélevées sur le transect.

Exercice 9 (Création de "secteurs" (⋆)) *Dans cet exercice, on vous propose de créer des formes géométriques qui peuvent être utiles pour l'analyse de la propagation d'un phénomènes autour d'un point en s'intéressant à deux paramètres :*

- *l'angle de diffusion (en mauve sur la Figure ci-dessous)*
- *la distance de diffusion (en vert sur la Figure ci-dessous)*



Pour chacun de ces paramètres, on cherche à générer des collections de géométries, c'est-à-dire un ensemble de géométries, qui vont permettre d'analyser un phénomène en s'intéressant à son évolution entre les différentes régions que délimitent les éléments de la collection.

Commençons avec un peu de géométrie.

3 Opérations géométriques

3.1 Présentation des opérations géométriques

La librairie OGR/GDAL offre des possibilités élémentaires de manipulation des objets géométriques. Si on considère deux géométries (par défaut, des polygones) **g** et **h**, la librairie propose les opérations suivantes :

- **geom** = **g**.**Intersection**(**h**) : construit la géométrie **geom** par l'intersection des deux géométries **g** et **h**,
- **geom** = **g**.**Union**(**h**) : construit la géométrie **geom** par l'union des deux géométries **g** et **h**,
- **geom** = **g**.**Difference**(**h**) : construit la géométrie **geom** par le découpage de la géométrie **g** de sa partie commune avec **h**,
- **geom** = **g**.**Buffer**(34) : construit la géométrie **geom** en ajoutant un buffer d'une distance de 34 à la géométrie **g**,
- **geom** = **g**.**GetEnvelope**() : construit la géométrie **geom** définissant le rectangle englobant de **g**,
- **geom** = **g**.**ConvexHull**() : construit l'enveloppe convexe de **g**.

S'ajoutent à ces fonctions, un ensemble de fonctions qui peuvent servir à identifier des relations particulières entre deux géométries. Considérons de nouveau deux géométries **g** et **h**.

- **g**.**Contains**(**h**) : teste si la forme **h** est incluse dans la forme **g**, cette relation peu, en particulier être utilisée pour savoir si un point se trouve à l'intérêt d'une forme.
- **g**.**Touches**(**h**) : teste si la forme **g** touche la forme **h**, c'est-à-dire s'ils ont un bord commun,
- **g**.**Overlaps**(**h**) : teste si les formes **h** et **g** se superpose (partiellement). Il faut préférer l'usage de cette fonction à l'usage de la fonction **Intersects** qui ne regarde que les enveloppes.
- **g**. **Disjoint** (**h**) : teste si les formes **h** et **g** sont disjointes (c'est l'inverse de *overlaps*)
- **g**.**Distance**(**h**) : calcule la distance entre les formes **h** et **g**. La distance entre deux polygones est la distance la plus courte entre deux points des polygones respectifs.

Exercice 10 *L'objectif de cet exercice est de transformer un couche de polygônes en utilisant la fonction* **Buffer***() sur toute les géométries d'une couche. Vous pourrez utiliser la couche RPG disponible dans le répertoire* `data/vecteurs/RPG/`.

Question a) Commencer par faire un programme qui recopie intégralement un fichier shapefile dans un autre fichier shapefile (vous vous inspirerez de l'exercice sur la recopie de la couche de points, sans recopier les attributs).

> *Question b)* Modifier votre code pour que, à la place de la géométrie originale – appelons cette variable **geom** – vous construisiez une nouvelle géométrie à l'aide de la fonction **geom.Buffer(50)**. Cette nouvelle géométrie sera celle effectivement utilisée pour la second couche. La valeur `50` est indicative (vous pourrez la modifier pour voir les effets, y compris avec des valeurs négatives).

3.2 Exemple avancé : fusion des sous-bassins versants des Grands Lacs

On dispose d'une couche vecteur de polygones délimitant les sous-bassins versants dans la région des Grands Lacs (`glwsheds.shp`) On cherche à construire un nouveau fichier `greatlakesBV.shp` regroupant chaque bassin versant dans un minimum de géométries. Par rapport aux données disponibles, il faut fusionner les formes géométriques qui se touchent et qui appartiennent aux même bassin versant.
On notera que chaque sous-bassin versant est caractérisé par des attributs dont l'attribut `LAKEBASIN` qui indique à quel bassin versant appartient un sous-bassin.
La tâche semble assez simple (et doit se faire assez simplement (?) avec GRASS ou autres outils de manipulation de données SIG), mais il y a une difficulté algorithmique pour faire la fusion de plusieurs formes géométriques ...

Bien ! Discutons maintenant stratégie. Pour s'attaque à ce problème, nous allons commencer par récupérer toutes les géométries en les classant par BV (on utilisera pour cela l'attribut `LAKEBASIN`). Nous utiliserons pour cela une structure de données de type dictionnaire qui va associer un BV à une liste de forme géométriques.
Une fois que nous avons les listes de formes géométriques pour chaque BV, il faut fusionner deux à deux les formes qui se touchent. On construit une nouvelle liste de formes géométriques fusionnées **outputgeoms**, et ajoute successivement chaque élément `e` de la liste de géométrie initiale.
L'algorithme élémentaire pour cet ajout est le suivant :

```
FOR h in outputgeoms
   SI h touche e ALORS
      fusionner e dans h
      break
```

Néanmoins, en fonction de l'ordre de traitement et de l'organisation spatiale des formes, cette méthode ne conduit pas à assurer la bonne fusion de toutes le formes. La solution ci-dessous est meilleure : on introduit une variable `modifie` qui indique si la liste a subit des modifications par l'ajout de `e` (ie la fusion de `e` avec un élément de la liste)). Si tel est le cas, on parcourt de nouveau cette liste pour savoir si l'élément fusionné n'a pas lui même de nouvelles connexion avec d'autres éléments.

```
modifie=true
TANT QUE modifie
   modifie = false
      FOR h in outputgeoms
         SI h touche e ALORS
            fusionner e dans h
            e <- h
            modifie = true
```

Les nouvelles listes de formes géométriques seront elles aussi organisées dans un dictionnaire, et on se servira de ce dictionnaire de formes géométrique pour générer le fichier de sortie.

Ouverture des fichiers On commence notre script de manière "habituelle" en ouvrant les fichiers qui nous seront utiles :

— le fichier contenant la description des sous-bassins versants,
— le fichier de sortie près à l'écriture.

```
import ogr, os, sys

os.chdir('/home/tguyet/Enseignements/2012-2013/TASE-GAPE/ProgSIG/Tutoriel/data/greatlakes/')

filelakes = "glwsheds.shp"
fileout = "output.shp"
driver = ogr.GetDriverByName('ESRI Shapefile')
```

```
# Ouverture de la couche des sous-BV des lacs
sBV = driver.Open(filelakes, 0)
if sBV is None:
    print 'Could not open file ' + filelakes
    sys.exit(1)
sBVLayer = sBV.GetLayer()

# creation de la couche de sortie
if os.path.exists( fileout ):
    driver.DeleteDataSource(fileout)
outDS = driver.CreateDataSource(fileout)
if outDS is None:
    print 'Could not create file'
    sys.exit(1)
outLayer = outDS.CreateLayer("BV", srs=sBVLayer.GetSpatialRef(), geom_type=ogr.wkbPolygon)


# definition des attributs de la couche de sortie par recopie
fieldDefn = sBVLayer.GetFeature(0).GetFieldDefnRef('LAKEBASIN')
outLayer.CreateField( fieldDefn )

# featureDefn decrits les attributs de la couche outLayer
featureDefn = outLayer.GetLayerDefn()
```

Construction du dictionnaire des sous-bassins versants. Pour chaque forme du fichier d'entrée, on regarde la valeur de son attribut et on ajoute sa géométrie à la liste adéquate

```
sousBVdict={}

sfeat = sBVLayer.GetNextFeature()
while sfeat:
    #on recupere la geometrie d'un site
    geom = sfeat.GetGeometryRef().Clone() #attention ici a bien cloner la geometrie !

  #on recupere l'attribut indiquant le BV d'appartenance
    lakeBV = sfeat.GetField('LAKEBASIN')

    # si c'est le premier element pour un BV, on cree une liste vide
    if not sousBVdict.has_key( lakeBV ) :
        sousBVdict[lakeBV]=list()

    #on ajoute la geometrie dans la liste des ssBV correspondant
    sousBVdict[lakeBV].append( geom )

    sfeat.Destroy() #on peut detruire car on a clone la geometrie !
    sfeat = sBVLayer.GetNextFeature()
```

Fusion des géométries qui se touchent. On traite séparément chacun des bassins versants

```
BVdict={}
for lake in sousBVdict: #pour chaque bassin versant
    outputgeoms = list() #liste des geometries fusionnees

    for geom in sousBVdict[lake]: # pour chaque geometrie
        modified=True
        while modified:
            modified=False
            for h in outputgeoms:
                if geom.Touches(h): # test sur la propriete geometrique
                    geom=geom.Union(h) #fusion des formes
                    outputgeoms.remove(h)
                    modified=True
        outputgeoms.append(geom)

    #on associe la nouvelle liste au bassin-versant 'lake'
    BVdict[lake]=outputgeoms
```

Enregistrement du fichier de sortie. On enregistre chacune des géométries on mentionne dans ses attributs sont BV d'appartenance.

```
for lake in BVdict:
```

```
    for geom in BVdict[lake]:
        outFeature = ogr.Feature(featureDefn)
        outFeature.SetGeometry(geom)
        outFeature.SetField('LAKEBASIN', lake)
        outLayer.CreateFeature(outFeature)
        geom.Destroy() #si on n'en a plus besoin par la suite !
        outFeature.Destroy()
outDS.Destroy()
```

3.3 Exercices

Exercice 11 (Reconstructions des Grands Lacs) *On dispose d'une couche vecteur de polygone délimitant les régions administratives autour des Grands Lacs (`glpolit_gen.shp`). On souhaite construire des nouvelles couches à partir de ces données. On utilise pour cela des opérations géométriques entre les géométries.*

Question a) Construire les limites du lac supérieur *On souhaite maintenant extraire le limites du lac supérieur.*

1. *Commencer par identifier à la main (dans QGis) les limites approximatives de la boîte englobante du lac supérieur, et notez ses coordonnées,*
2. *Construire un programme dont le principe sera :*
   - *construire un polygone correspondant à la boîte englobante identifiée manuellement,*
   - *extruder le polygone avec chacun des polygones de la couche des limites administratives.*

Question b) Identification des berges du Lac Supérieur *On cherche à obtenir uniquement les berges du lac (de distance 0.03 unités). À la suite du programme précédent, utiliser la fonction* **Buffer** *uniquement à partir de la géométrie du Lac.*

Question c) Faire de même pour obtenir la surface du Lac à plus de 0.03 unités de la terre.

Exercice 12 (Voisinage d'une parcelle) *Dans le cadre des actions sanitaires de la DRAAF, il peut être utile d'identifier facilement (et rapidement) un ensemble de parcelles sur lesquels appliquer des traitements curatifs ou préventifs à la suite de l'identification d'un problème sanitaire sur une parcelle agricole.*
Le jeu de donnée `intern_pern_polygon.shp` est un extrait du RPG (Registre Parcellaire Graphique) reprenant les parcelles déclarées par les agriculteurs.
On souhaite mettre en place un programme pour identifier facilement un ensemble de parcelles à traiter. On s'intéressera particulièrement aux numéros d'identifiant de parcelles (attribut `IDPARCEL`) pour identifier une parcelle (NB : plusieurs polygones peuvent avoir le même identifiant de parcelle, c'est normal). Pour le début de cet exercice, vous vous intéresserez au voisinage de la parcelle `046001946_002`.

Question a) Identification des parcelles dans le voisinage de la parcelle `046001946_002` *Dans cette question, il s'agit de s'intéresser aux parcelles qui dont le centroïde est à une distance inférieure à* 300 *unités du centroïde de la parcelle `046001946_002`. Vous* ***afficherez à l'écran le numéro des parcelles*** *qui sont dans le voisinage de la parcelle cible.*
Vous commencerez par réfléchir à la stratégie générale pour arriver à cette solution, et passerez ensuite à l'implémentation.

Aide : *Quelques fonctions qui seront utiles :*

- **ResetReading**() *: fonction pour recommencer à zéro le parcours des features d'une couche*
- *pour récupérer la position le centroide d'un polygone, vous utiliserez la fonction* **Centroid**() *sur une géométrie. Cette fonction retourne un* **OGRPoint** *sur lequel vous pourrez récupérer les coordonnées grâce aux fonctions* **GetX**() *et* **GetY**()
- *le calcul de la distance entre deux points (x, y) et (lx, ly) peut se faire au moyen du code ci-dessous :*

```
import math
d = math.sqrt( (x-lx)**2 + (y-ly)**2 )
```

Question b) Identification des parcelles dans une zone tampon de la parcelle `046001946_002` *La stratégie est ici globalement la même que précédemment. On cherche sélectionner les parcelles qui se trouvent dans une région tampon autour de la parcelle cible. On utilise pour cela un buffer de taille* 300 *autour du polygone. Toutes les parcelles qui intersectent sont alors considérées comme appartenant à la zone tampon.*
Vous ***afficherez à l'écran le numéro des parcelles*** *qui sont dans le voisinage de la parcelle cible.*

Question c) Demander à l'utilisateur le numéro de la parcelle *Modifier l'un des programmes précédent pour que l'utilisateur saisisse le numéro de la parcelle cible à rechercher. L'exemple ci-dessous permet de récupérer une valeur auprès de l'utilisateur (saisie clavier). Vous ferez les modifications nécessaire pour qu'il n'y ait pas d'erreur lorsque la saisie de l'utilisateur ne correspond à aucune parcelle existante (vous pourrez même le signaler à l'utilisateur par un message à l'écran).*

```
val = raw_input("saisir le numero de la parcelle cible")
```

Question d) Gestion de deux zones *On souhaite maintenant considérer deux seuils : les parcelles qui se trouvent à moins de* 300 *unités (en distance ou en tampon) doivent subir un traitement curatif, les parcelles entre* 300 *et* 600 *unités doivent subir un traitement préventif.*
Un numéro de parcelle ne peut être que dans une seule liste. Si une parcelle est dans les deux listes, elle doit disparaître de liste des "préventifs" (elle subit le traitement plus plus contraignant)

Aide : *plutôt que d'afficher les résultats, vous utiliserez des listes pour retenir les résultats.*
Pour faire des différences de listes, vous pourrez utiliser l'exemple ci-dessous (notez que vous perdrez également tous les doublons) :

```
>>> l1=[45, 56, 67]
>>> l2=[12, 56]
>>> list(set(l1) - set(L2))
[67, 45]
```

.

Exercice 13 (Récupération de l'information par localisation) *L'objectif de cet exercice est de faire la fusion d'information entre couche en utilisant la localisation des polygones. Cet exercice provient d'un problème pratique de récupération d'informations issue d'une couche d'ilôts de culture*[5] *pour l'ajouter à une couche de parcelles.*
Normalement, il existe un outil dans QGis capable de faire ce genre de transformation ... sauf qu'il ne marchait pas (sur les nouvelles versions, j'espère qu'il a été modifié) et dans tous les cas, je ne savais pas exactement ce qu'il faisait ! Donc, la seule solution était de refaire un petit programme Python pour faire la même tâche.
Le principe de la méthode est le suivant : on dispose de deux couches, les ilôts et les parcelles. Pour chaque parcelle, on va son calculer isobarycentre P. Ensuite, pour chacun des ilôts I, on vérifie si P se trouve dans le polygone i. Si c'est bien le cas, on ajoute à la parcelle l'information utile, et on passe à la parcelle suivante.

Question a) Écrire en pseudo-code l'algorithme de fusion des informations par localisation. (solution données plus bas)

Dans les données proposées se trouvent dans les fichiers `Parcels.shp` *et* `Ilots.shp`.
On récupèrera l'information `PAE_ID_EXP`*, c'est-à-dire l'identifiant de l'agriculteur, ainsi que l'attribut* `COMMUNE_IL`*. Ces deux attributs sont des nombres entiers. Il n'est pas impossible qu'une parcelle ne corresponde à aucun ilôt. Dans ce cas, vous n'ajouterez aucune information à la parcelle pour ces attributs.*

Question b) Créer une fonction pour la fusion des informations par localisation en supposant que les deux fichiers ont les mêmes systèmes de coordonnées (en pratique, les fichiers proposés sont en Lambert-II)

Question c) (⋆⋆) Modifier votre programme pour faire une(des) fonction(s) de votre programme. Vous pourrez prendre en paramètre de votre fonction les noms des fichiers ainsi que deux listes d'attributs (1 pour chaque couche).

Question d) (⋆)Même question que la question 13 en utilisant le fichier `Ilots_RGB93.shp` qui reprend les mêmes ilôts mais en coordonnées Lambert 93.

! Attention ! - Temps de calcul

L'algorithme proposé ici est un algorithme basique, mais il est très peu efficace ! La complexité de l'algorithme sera de l'ordre de $I \times P$, c.-à-d. le nombre d'ilôts fois le nombre de parcelles. Pour de grandes couches de données, le temps de calcul qui en découle peut être très important. Il faudrait alors améliorer l'algorithme pour réaliser la même tâche, mais plus efficacement.

Remarque 9 - Aide : algorithmes

Le principe de l'algorithme est celui d'une recopie de la couches de parcelles, à la seule différence que lorsqu'on recopie d'une forme, il faut ajouter des attributs provenant des attributs d'un ilot couche

```
- ouvrir la couche de parcelles LP
- ouvrir la couche d'ilots LI
- creer une nouvelle couche de parcelles avec les attributs de LP et les attributs de la co

Pour toute parcelle P faire
    creer P' par recopie de P
    creer un point C, barycentre de P

    #recherche d'un ilot
    Pour tout ilot I faire
        Si C est dans le polygone de I
            ajouter les attributs de I à P'
            arreter la recherche des ilots

- fermer tous les fichiers
```

4 Utilisation des projections

Par défaut, les géométries n'ont pas de système de projection de défini. Lors de l'utilisation des données, il y a donc une ambiguïté sur la signification des coordonnées.

4.1 Création d'un système de coordonnées

La première façon de récupérer un système de coordonnées est de la construire à partir de l'un des systèmes de description d'une projection.
L'exemple suivant illustre la construction d'un système de coordonnées en utilisant un code EPSG. La seconde ligne créé un objet représentant un système de coordonnées et la troisième ligne définie le système de coordonnées à partir de l'identifiant EPSG. Notez que l'utilisation des transformations nécessite l'utilisation de la librairie **osr** (complément de **ogr**).

```
1 import osr
2 coordSys = osr.SpatialReference()
3 coordSys.ImportFromEPSG(32612)
```

Il existe de nombreuses fonctions d'importation d'un système de coordonnées :

— **ImportFromWkt**(<**wkt**>) : <**wkt**> est une chaîne de caractères qui contient la description du système de projection (*p.ex.* `PROJCS["UTM Zone 12, Northern Hemisphere", GEOGCS["WGS_1984", DATUM["WGS_1 SPHEROID["WGS 84",6378137,298.2572235630016], TOWGS84[0,0,0,0,0,0,0]], PRIMEM["Greenwich UNIT["degree",0.0174532925199433], AUTHORITY["EPSG","4326"]], PROJECTION["Transverse_Mer PARAMETER["latitude_of_origin",0], PARAMETER["central_meridian",-111], PARAMETER["scale_ PARAMETER["false_easting",500000], PARAMETER["false_northing",0], UNIT["Meter",1], AUTHORITY["EPSG","32612"]] `),

- **ImportFromEPSG**(<**epsg**>) : <**epsg**> est un nombre, identifiant dans la base des EPSG
- **ImportFromProj4**(<**proj4**>) : <**proj4**> est une chaîne de caractères qui contient la description dans le formalisme "proj4" (*p.ex.* "`+proj=utm +zone=12 +ellps=WGS84 +datum=WGS84 +units=m +no_defs`").
- **ImportFromESRI**(<**proj_lines**>) : voir la doc
- **ImportFromPCI**(<**proj**>, <**units**>, <**parms**>) : voir la doc
- **ImportFromUSGS**(<**proj_code**>, <**zone**>) : voir la doc
- **ImportFromXML**(<**xml**>) : voir la doc

4.2 Récupération et attribution d'un système de coordonnées

4.2.1 Avec des couches

Lors de l'utilisation de données, il est possible de récupérer le système dans lequel sont exprimées les données en utilisant la fonction **GetSpatialRef**().
Dans le code suivant, on affiche le code WKT du système de coordonnées d'une couche désignée par la variable **layer** :

```
coordSys = layer.GetSpatialRef()
if coordSys is None:
  print 'pas de systeme de coordonnees pour cette couche'
else:
  print 'le systeme de coordonnees est ' + str( coordSys.ExportToWkt() )
```

La fonction **ExportToWkt**() permet de récupérer une chaîne de caractères décrivant le système de projection.

Remarque 10

L'utilisation de la fonction **GetSpatialRef**() fonctionne pour les couche vecteur mais également pour les couches raster.

L'attribution d'un système de coordonnées à une couche se fait lors de sa création par la fonction **CreateLayer** . Le second argument de cette fonction permet au programmeur de définir le système de projection décrit dans le formalisme PROJ4.
L'exemple ci-dessous illustre l'utilisation d'un système de coordonnées lors de la création d'une couche.

```
import osr
coordSys = osr.SpatialReference()
coordSys.ImportFromEPSG(32612)
layerout = dsout.CreateLayer('Nouvelle couche', coordSys.ExportToProj4(), geom_type=ogr.wkbPoint )
```

! Attention !

Attention, l'attribution d'un système de coordonnées à une couche ne transforme pas les données. Si vous insérer des géométries exprimées dans un autre système de coordonnées, dans la couche, aucune modification ne sera faite sur les données : le résultat sera totalement faux !

4.2.2 Avec des géométries

Cette fonction peut également être appelée sur une géométrie.

```
geom.AssignSpatialReference(out_srs)
```

4.3 Transformation de système de coordonnées

L'intérêt de pouvoir récupérer des systèmes de coordonnées est de pouvoir passer d'un système à l'autre pour travailler sur des données exprimées dans des systèmes différents.
Alors l'objet décrivant le système de coordonnées n'est pas en mesure de faire des transformations, il est nécessaire de faire appel à un objet annexe qui spécifiera la transformation d'un système donné vers une autre.
Pour créer une transformation, il faut :

1. Créer ou récupérer le système de coordonnées des données (SRS source)
2. Créer ou récupérer le système de coordonnées dans lequel on veut exprimer ces même données (SRS destination)
3. Créer la transformation du SRS Source vers le SRS destination

```
#Creation d'un SRS source : 12N WGS84
sourceSR = osr.SpatialReference()
sourceSR.ImportFromEPSG(32612)
#Creation d'un SRS destination : WGS84
targetSR = osr.SpatialReference()
targetSR.ImportFromEPSG(4326)
coordTrans = osr.CoordinateTransformation(sourceSR, targetSR)
```

La transformation peut s'utiliser de deux manières :

1. pour transformer directement les géométries (**geom.Transform(coordTrans)**) : dans ce cas, on utilise l'objet de transformation dans une fonction de la géométrie. Cette fonction modifie directement et efficacement la géométrie **geom**.
2. pour projeter un point (**coordTrans.TransformPoint(point)**) : dans le cas, il s'agit d'une fonction de l'objet **coordTrans**.

4.4 Exemple : recopie d'un fichier dans un système de coordonnées Lambert II

Le fichier `sites.shp` est défini avec un système de coordonnées (peu importe lequel !). Mais, je souhaite recopier le fichier dans un système de coordonnées Lambert II (EPSG27572).
Le programme ci-dessous permet cette transformation :

Le programme peut être téléchargé ici.

```
os.chdir('/home/tguyet/Enseignements/2012-2013/TASE-GAPE/ProgSIG/Tutoriel/data/data_transfogeo/')

filein ='sites.shp'
fileout ='sitesLII.shp'

driver = ogr.GetDriverByName('ESRI Shapefile')

# ouverture du fichier d'entree
inDS = driver.Open(filein, GA_ReadOnly)
if inDS is None:
  print 'Could not open file'
  sys.exit(1)
inLayer = inDS.GetLayer()

#recuperation de l'EPSG
coordSys = inLayer.GetSpatialRef()
if coordSys is None:
  print 'pas de systeme de coordonnees pour cette couche'
  sys.exit(1)
else:
  print 'le systeme de coordonnees est ' + str( coordSys.ExportToWkt() )

# creation du fichier de sortie : couche de points
if os.path.exists( fileout ):
  driver.DeleteDataSource(fileout)
outDS = driver.CreateDataSource(fileout)
if outDS is None:
  print 'Could not create file'
  sys.exit(1)

#definition du systeme de coordonnees Lambert II
coordSysLII = osr.SpatialReference()
coordSysLII.ImportFromEPSG(27572)

#creation de la couche avec son systeme de coordonnees
outLayer = outDS.CreateLayer('couche reprojectee', srs=coordSysLII, geom_type=ogr.wkbPoint)

#creation de la transformation geometrique
coordTrans = osr.CoordinateTransformation(coordSys, coordSysLII)

# definition des attributs de la couche de sortie par recopie
fieldDefn = inLayer.GetFeature(0).GetFieldDefnRef('ID')
outLayer.CreateField( fieldDefn )
```

```
fieldDefn = inLayer.GetFeature(0).GetFieldDefnRef('COVER')
outLayer.CreateField( fieldDefn )

# featureDefn decrits les attributs de la couche outLayer
featureDefn = outLayer.GetLayerDefn()

#Pour toutes les formes de la couche d'entree, faire
inFeature = inLayer.GetNextFeature()
while inFeature:
  # Creation d'une forme avec recopie des attributs
  outFeature = ogr.Feature(featureDefn)
  outFeature.SetField('id', inFeature.GetField('id'))
  outFeature.SetField('COVER', inFeature.GetField('COVER'))

  # Transformation de la geometry et attribution en sortie
  geom = inFeature.GetGeometryRef()
  geom.Transform( coordTrans )
  outFeature.SetGeometry( geom )

  # Ajouter de la forme a la couche de sortie
  outLayer.CreateFeature(outFeature)
  outFeature.Destroy()

  # destruction des formes
  inFeature.Destroy()

  # on regarde combien on a traite de forme avant de continuer
  inFeature = inLayer.GetNextFeature()

# Fermeture des sources de donnees
inDS.Destroy()
outDS.Destroy()
```

4.5 Exercice

Exercice 14 *Écrire une programme qui affiche à l'écran (ou dans un fichier) la liste des points de* `sites.shp` *avec leurs attributs qui se situent dans l'étendue (exprimé dans le système de coordonnées Lambert 93, dont l'EPSG est 2154)*

- *Upper-Left : [-5853294,11884675]*
- *Lower-Right : [-5831808,11873003]*

Question a) Votre programme commencera par transformer les coordonnées des points des limites en des points exprimés dans le référentiel du fichier

Question b) Une autre façon (plus générique) de faire sera de faire construire au programme une forme géométrique rectangulaire à partir des coordonnées brutes et de transformer cette forme dans le référentiel des données. La forme géométrique pourra ensuite être utilisée avec la fonction **contains** pour savoir si une autre forme est incluse ou non dans cette géométrie.

Exercice 15 *Le fichier* `zones.shp` *contient une information complémentaire de* zone *aux données* `sites.shp`*. On veut ajouter un attribut* `zone` *aux points de* `sites.shp` *correspondant au numéro de zone dans lequel se trouve un point.*

Question a) Écrire un programme qui affiche (**print**) pour chaque point la `zone` dans laquelle il se situe. Le fichier `zones.shp` n'est pas dans le même référentiel que le fichier `sites.shp`.

Aide :

- *vous utiliserez la fonction* **contains** *pour savoir si un point est dans une géométrie*
- *le principe de l'algorithme est donné ci-dessous*

```
Pour chaque point P de sites.shp faire :
  Pour chaque zone Z de zones.shp faire :
    Si Z contient le point P alors:
      afficher "le point P est dans la zone Z"
```

Question b) Écrire un programme qui créé un fichier `sites_zones.shp` qui contiendra tous les points de `sites.shp` avec les attributs `id`, `COVER` et `Zone`. Si un point n'appartient à aucune zone, la valeur de l'attribut `Zone` restera `NULL`.

5 Exercices bilans

5.1 À l'attaque des carottes

Le contexte de ce sujet est le développement d'un outil d'aide à la décision pour des cultivateurs de carottes envahis par des lapins. Ce problème est illustré par la carte ci-dessous (figure III.4). Les données, totalement factices, contiennent deux fichiers *shapefile* :

- `champs.shp` : un fichier de polygones qui décrit les champs disponibles à la culture de carottes pour les agriculteurs. Chaque champ (un polygone) est associé à une évaluation du rendement en carottes (*i.e.* le nombre de tonnes de carottes qui sont produites par hectare, représenté en dégradé de couleurs dans l'illustration).
- `terriers.shp` : un fichier de points localisant les terriers de lapins. Le chiffre indiqué à côté des terriers correspond à la quantité de lapins qu'ils contiennent. Cette couche a été obtenue par télédétection (oui, j'ai dis que c'était factice !).

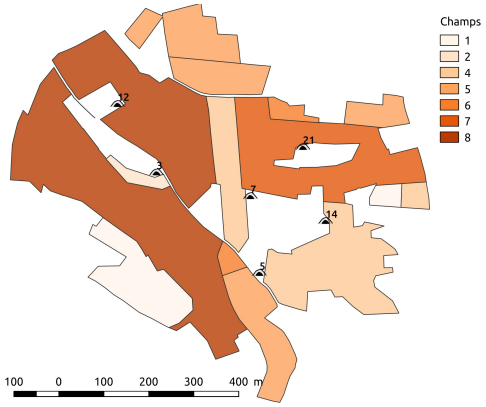


FIGURE III.3 – Carte représentant les données du sujet (vue extraite avec QGis)

En fin de sujet, on vous propose un code source python utilisant la librairie OGR/GDAL. L'objectif de ces exercices est de comprendre le programme et de l'adapter pour en compléter la fonctionnalité.

5.1.1 Compréhension du code

Exercice 16 (Estimation des zones d'attaques de lapin) *Dans cet exercice, on cherche à identifier les zones dans lesquelles les lapins vont attaquer les carottes. Le premier programme à télécharger effectue un calcul qui estime ces zones.*

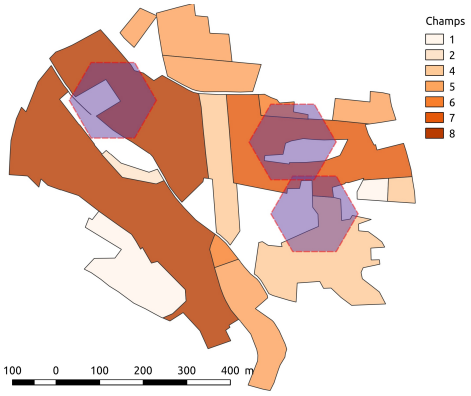
Question a) Décrire la signification des variables `nb`, `x` et `y` aux lignes 25-27.

Question b) Expliquez en détails ce que font les lignes 31 à 39.

Question c) Compléter l'illustration en dessinant le résultat attendu par l'exécution du programme. Utiliser le dessin en fin de sujet
NB : on attend un schéma qui illustre des mesures réalistes ! Vous pourrez compléter le dessin par des informations textuelles sur les distances utilisées. ***Vous pourrez vous limiter à dessiner 3 éléments au choix de la couche.***

Question d) Décrire ce que fait ce programme, de manière précise et concise ?

Exercice 17 (Consommation de carottes) *On s'intéresse maintenant au second programme à télécharger. Ce programme prend en entrée une couche de polygones et un polygone correspondant à des zones d'attaque de lapins sur des champs de carottes (un lapin qui attaque un champ, mange toutes les carottes sur lesquelles il tombe).*
On illustre ce problème par la carte ci-dessous. NB : Les zones proposées dans l'illustration ne correspondent pas nécessairement à celles de l'exercice précédent.



Question a) Expliquer ce que fait la ligne 11 du programme : décrire ce que représentent les variables et indiquer ce que calcule la fonction **Intersection**.

Question b) La ligne 7 est-elle réellement utile ? pourquoi ?

Question c) Sachant que **geom** représente une zone d'attaque de lapins et que **layer** représente les parcelles de carottes, en déduire ce que calcule la fonction **ma_fonction** (soyez précis et conçis).

Question d) Décrire ce que calcul le programme principal (lignes 15 à 30)

Vous pourrez télécharger le programme ayant servi à faire les illustrations avec les hexagones ici.

5.1.2 Généralisation du code

Exercice 18 (Mixer les deux programmes) *Le premier exercice s'est intéressé à estimer des régions d'attaque de lapin à partir de la localisation des terriers et le second exercice s'est intéressé à calculer les dégâts provoqués dans des régions quelconques d'attaque.*

Question a) (⋆) En utilisant les programmes précédents, créer un programme unique qui réunit les deux fonctionnalités pour *calculer les dégâts* (en tonnes de carottes) sur les champs de carottes à partir de la localisation des terriers.
Pour répondre à la question, vous pourrez soit recopier tout le code, soit "réutiliser" des parties du code en faisant des "copier-collers" mélangés avec des lignes de codes manuscrites supplémentaires.

Question b) (⋆) Créer ensuite une fonction **process(burrowsfile , plotsfile , radius , growth)** où **radius** et **growth** sont des valeurs pour le modèle d'estimation des zones d'attaque.

5.2 Ça déraille

Le contexte de ce sujet est la mise en place d'un outil pour l'évaluation des risques liés au transport de l'ammoniac par voie ferrée. Ce problème est illustré par la carte ci-dessous (figure III.4). Les données, totalement factices, contiennent deux fichiers *shapefile* :

- `bati.shp` : un fichier de points contenant les localisations du bâti. Il existe trois types de bâti : maisons, usines et écoles.
- `troncons.shp` : un fichier de ligne décrivant les rails de chemin de fer. La couche contient plusieurs lignes. Chaque ligne est associé un un niveau de risque d'acccident. Par exemple, les secteurs tournant sont plus risqués (niveau 3) que les secteurs rectilignes (niveau 1). Ces niveaux sont illustrés par des couleurs sur la carte.

L'objectif de ces exercices est de comprendre le programme et de l'adapter pour en compléter la fonctionnalité.

5.2.1 Compréhension du code

On vous propose de télécharger un code source python utilisant la librairie GDAL.

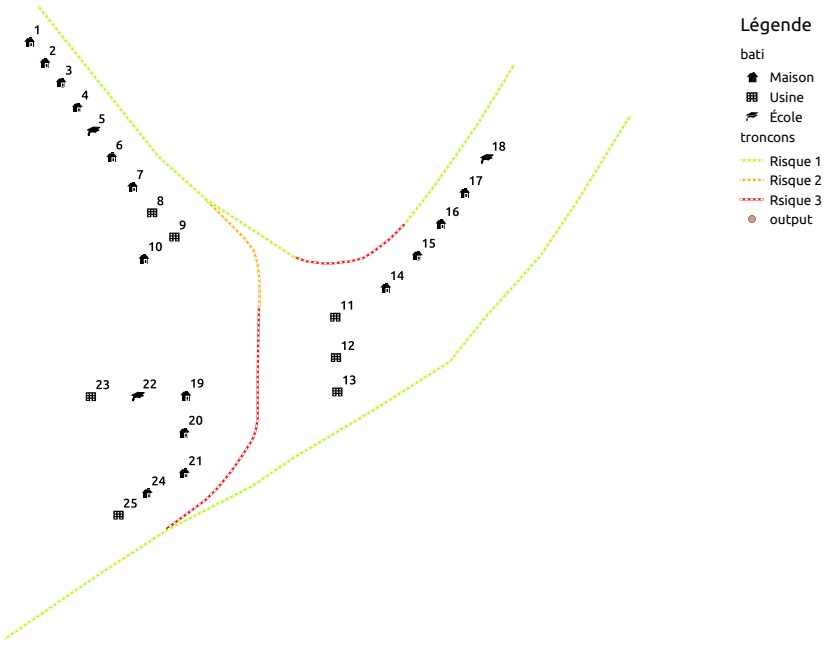


FIGURE III.4 – Carte représentant les données du sujet (vue extraite avec QGis)

Exercice 19 (Compréhension de la fonction `mafonction`)

Question a) Expliquez en détails ce que fait **line**.**GetGeometryRef**().**Distance**(**point**) à la ligne 8.

Question b) Quel schéma classique reconnaît-on dans les lignes 7 à 12 de la fonction ?

Question c) Sachant que **point** est une géométrie de type "point", et que **layer** est une couche de lignes, décrire ce que fait la fonction **mafonction**. Vous indiquerez en particulier la signification de la valeur calculée (et retournée) par la fonction.

Question d) Quelle est la signification de la ligne 4 ? (⋆) Pourquoi est-elle nécessaire dans cette fonction ?

Exercice 20 (Compréhension du programme principal) *On s'intéresse maintenant au programme principal, dont le début se trouve à la ligne 15. Ce programme fait appel à la fonction précédemment étudiée.*

Question a) Expliquer ce que contient la variable **outputLayer** dans le programme ?

Question b) Que permettent de faire les lignes 46 à 51 ?

Question c) En utilisant les fichiers du jeu de données d'exemple, indiquer la valeur qui sera donnée à l'attribut `risque` pour les bâtis suivants : 11, 12, 13, 19, 20 et 22

Question d) (⋆) Expliquer ce que fait le programme.

Question e) La ligne 57 est-elle réellement utile ?

5.2.2 Généralisation du code

Exercice 21 (Traitement de couches multiples) *Le traitement ci-dessus doit être appliqué par la SNCF. Elle dispose d'un fichier contenant les informations pour toutes les rails, mais elle récupère des informations sur la localisation du bâti des différentes communes, et dispose donc de multiples fichiers de points ! On suppose que tous les fichiers de points fournis ont la même structure que le fichier utilisé précédemment (en particulier, le fichier a les mêmes attributs `id` (nombre entier) et `type` (texte de 15 caractères).*

On dispose maintenant d'une collection de fichiers de points contenus dans une liste **inputfiles** *(toujours un unique fichier* `troncons.shp`*), par exemple :*

```
inputfiles = ['bati_com1.shp', 'bati_com2.shp','bati_com3.shp', 'bati_com4.shp']
```

L'identifiant de chaque bâti étant normalisé, il n'y a pas de redondance d'identifiant entre les différents fichiers de bâti.

Question a) En réutilisant des parties du code précédent, proposer un programme qui construit un fichier **unique**, `output.shp`, qui contient le résultat du traitement précédent pour tous les fichiers de **inputfiles** ?

CHAPITRE

IV Utilisation des données Raster

On aborde maintenant l'accès aux couches raster (format matriciel) à l'aide de la librairie OGR/GDAL. L'organisation générale de la librairie est très semblable à celle des données représentées sous un format vectoriel. On y retrouve l'organisation structurelle logique des données en couches imbriquées :

- le *driver* : pour la définition du format de données
- le *datasource* : pour désigner la source physique des données. Dans le cas des images raster, il s'agira quasi-uniquement de fichiers
- les *couches* : qui, pour une image raster, sont nommées des bandes et qui peuvent être multiples (là où il était plus rare d'avoir couches multiples pour les fichiers vectoriels)

Cette structure permettra de comprendre la logique des programme qui vous donneront accès aux données des fichiers raster. Ces données vont alors être récupérées sous la forme de matrices qui pourront être manipulées par votre programme.

1 Lire un fichier Raster

On commence par la lecture d'un fichier raster pour introduire le premier programme pour accéder aux données, ainsi que les premières bases pour la manipulation de ces données (traitement par pixels).

1.1 Chargement des drivers spécifiques (optionnel)

Pour des formats de données spécifiques, il peut être utile de charger des *drivers* qui permettront à la librarie GDAL de comprendre le format de votre fichier [1]. Dans les dernières versions de GDAL, il apparaît que ce ne soit pas indispensable pour ouvrir les formats très standards (surtout *open source* comme le geotiff). Cette étape reste néanmoins utile pour l'enregistrement de nouvelles couches.
Pour charger tous les drivers en une fois (pour lire un raster seulement, pas pour écrire), utiliser :

```
gdal.AllRegister()
```

Pour charger un driver spécifique, créez votre objet *driver* [2] :

```
driver = gdal.GetDriverByName('SRTMHGT')
```

puis enregistrez le :

```
driver.register()
```

1.2 Chargement des données d'un fichier Raster

L'opération de chargement des données d'un fichier raster consiste à transformer le contenu d'un fichier sous la forme d'une matrice de valeurs qui pourra être exploitée dans un programme (pour analyser son contenu, le transformer, etc.).

1.2.1 Ouverture du fichier

Vous pouvez maintenant lire un fichier raster sous forme de jeu de données (dataset) :

```
file = "N43E004.hgt"
ds = gdal.Open(file, GA_ReadOnly)
if ds is None:
    print("impossible d'ouvrir " + file)
    sys.exit(1)
```

1. La liste de format supporté par GDAL se trouve ici : `www.gdal.org/formats_list.html`.
2. Dans cet exemple, on propose de charger un fichier au format HGT, qui correspond à un format pour la représentation d'un modèle d'élévation de terrain sous une format matricielle : chaque pixel de l'image contient l'élévation à sa localisation.

La méthode **Open**() prend deux paramètres :
- le chemin du fichier
- la méthode de lecture. Vous avez deux constantes possibles pour la méthode de lecture : **GA_ReadOnly** = 0, **GA_Update** = 1

Si python vous renvoie un message d'erreur sur la constante **GA_ReadOnly**, vous pouvez la remplacer par sa valeur (0 donc) ou bien importer les constantes de GDAL.
Voici quelques méthodes définies pour récupérer de l'information sur vos données :
- **ds.RasterXSize** et **ds.RasterYSize** permettent de connaître les dimensions de l'image d'origine (en nombre de pixels)
- **ds.RasterCount** indique le nombre de bande dans l'image
- **ds.GetProjection**() indique le système de projection de la couche
- **ds.GetGeoTransform**() indique les caractéristiques géométriques de la couche (notamment la position des points des angles et les tailles des pixels)

NB : **ds.RasterXSize** et **ds.RasterCount** sont bien des valeurs tandis que **ds.GetProjection**() et **ds.GetGeoTransform**() sont des fonctions.

1.2.2 Accès à une bande de données

L'accès aux informations d'un fichier raster se fait bande par bande. Le principe de lecture du fichier est de sélectionner une bande, puis de sélectionner une région rectangulaire dont on va récupérer les données. Pour travailler sur les pixels, nous devons obtenir la bande (la première bande est numérotée 1) :

```
band = ds.GetRasterBand(1)
```

Puis récupérons les données sous la forme d'un tableau à deux dimensions. L'accès aux données d'une bande se fait en donnant les positions (en nombre de pixels) de la zone rectangulaire à récupérer sous la forme d'un tableau.

```
data = band.ReadAsArray(xOffset, yOffset, 100, 12)
```

- 100,12 est la taille de la cellule que nous voulons récupérer.
- $xOffset$ et $yOffset$ sont obtenus en calculant le nombre de pixels entre le bord en haut à gauche et le point pour chaque axes (ordonnés et abscisses). Nous connaissons les coordonnées du point haut gauche, la taille d'une cellule en pixel.

Pour récupérer toute une bande, il suffit d'utiliser l'instruction suivante

```
data = band.ReadAsArray(0, 0, ds.RasterXSize, ds.RasterYSize)
```

ou plus encore

```
data = band.ReadAsArray()
```

Les données (c.-à-d. la variable **data**) est un tableau en 2 dimensions de la taille qui a été définir plus haut $(100, 12)$. Pour récupérer une valeur du tableau :

```
value = data[23][3]
```

Le tableau de valeur est un tableau de colonne, les deux valeurs sont bien des colonnes et des lignes et non des coordonnées. De plus la première ligne et la première colonne commencent à 0 !

! Attention !

Ne lisez pas un pixel à chaque fois mais récupérer les tous en une fois, puis traiter les. Ne lisez qu'un pixel à la fois si vous êtes sur d'en avoir besoin que d'un ou deux ! Malheureusement pour de gros jeux de données, cela peut poser problème : la fonction **ReadAsArray** va chercher les informations sur le disque dur. Chaque appel de cette fonction est très lent (beaucoup plus que l'accès à un élément d'une matrice). La solution est d'utiliser la taille des blocs ou de lire une ligne et de faire le traitement voulut, puis la ligne suivante.

Remarque 11

Chargement des séries temporelles d'images satellite Il n'est pas possible de faire une lecture "transversale" d'une pile de couches simplement. Pour les séries temporelles d'images satellite (par exemple 1 couche par date sur 23 dates), il serait intéressant de récupérer facilement l'ensemble des 23 données relatives à un pixel. Mais cela n'est structurellement pas pensé ainsi dans la librairie GDAL.
Cette situation rend les programmes de traitement d'images satellite un peu techniques. Il faut d'abord charger l'intégralité des données, couche par couche, puis les organisées sous une forme appropriée pour leur traitement. Ceci pose souvent de gros problème d'utilisation de la mémoire RAM de l'ordinateur (mémoire de travail). Des stratégies doivent alors être mises en place pour permettre les traitements sur de grandes images.

1.3 Parcours des pixels de la matrice

On dispose maintenant des matrices de données que l'on va pouvoir analyser. De la même manière que je vous proposais des "idioms" pour le parcours d'une couche de *features* pour les données vectorielles, il existe aussi des "idioms" pour parcourrir les matrices de pixels. L'objectif ici n'est pas de faire du traitement d'images (qui s'intéressent à des algorithmes complexes de traitements des matrices). On se contentera de mener des analyses pixel par pixel (les plus simples).

L'exemple ci-dessous illustre l'ouverture (en reprenant les étapes précédentes) et le parcours "type" de la première bande d'un fichier raster. Le parcours pixel à pixel se fait à l'aide d'une **double-boucle** : une boucle pour les lignes, indice `j`, imbriquée dans une boucle pour les colonnes, indice `i`.

```
dataset = gdal.Open(file, GA_ReadOnly)
band = dataset.GetRasterBand(1)
data = band.ReadAsArray()
for i in range(dataset.RasterXSize):
  for j in range(dataset.RasterYSize):
    # faire ici un traitement du pixel data[j][i] !!!
```

! Attention ! - Indicage colonne/ligne

Faites très attention à ce que le premier indice corresponde aux colonnes et que le second corresponde aux lignes. On verra en particulier lors de la création d'une couche raster, que l'ordre n'est pas toujours celui-ci !!

1.4 Coordonnées des pixels

L'intérêt de traiter des images de télédétection réside en partie sur la possibilité de géolocaliser l'information [3]. Il est donc intéressant de savoir comment faire la correspondance entre les coordonnées dans l'images (position d'un pixel dans la la matrice) et les coordonnées "physiques" exprimées dans le système de projection de la couche (*cf.* fonction **GetProjection**()).
Ceci est possible grâce à la transformation géométrique qui accompagne les données. Il s'agit d'un tableau de 6 valeurs dont 4 qui nous intéressent plus particulièrement, comme l'illustre le code ci-dessous :

```
geotransform = ds.GetGeoTransform()
originX = geotransform[0]
originY = geotransform[3]
pixelWidth = geotransform[1]
pixelHeight = geotransform[5]
```

L'utilisation du code ci-dessus permet de récupérer des informations essentielles sur la spatialisation physique de l'image à l'aide de variables qui seront plus facilement manipulables par la suite. Il est important de noter que le `geotransform` a toujours la même structure (*p.ex.* la valeur de l'origine en X sera toujours identifié par la première valeur du vecteur **geotransform**).

! Attention ! - Pas de `geotransform` sans projection !

Les valeurs contenues dans le vecteur **geotransform** sont des valeurs numériques ! Leur signification est à interpréter en fonction du système de projection de la couche.

3. C'est une différence majeure par rapport au traitement des images visuelles, issues de caméra par exemple.

Remarque 12 - Uniquement des transformations de SO ?

1: Illustrer les notions de projection et les conséquences/limites d'un geotransform simple ()

1.4.1 Passer des coordonnées aux pixels : cas d'extraction de l'image entière

Dans le cas où on a récupéré toute la matrice, les informations données dans le **geotransfom** sont directement exploitables (*cf.* Figure IV.1).

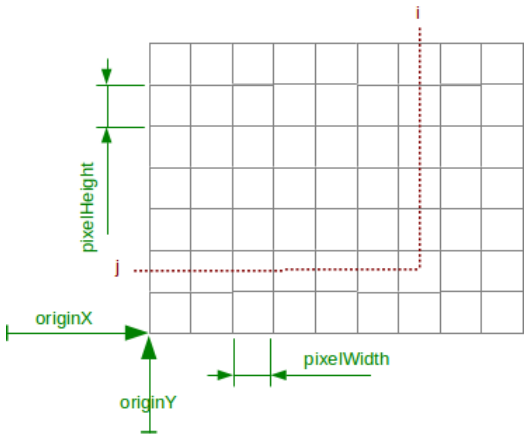


FIGURE IV.1 – Illustration de la correspondance entre coordonnées physiques (en vert) et index des pixels (en rouge).

Si nous cherchons la position dans la matrice de la coordonnées physiques $(43.2, 4.2)$, on peut utiliser les opérations suivantes :

```
i = int((4.2 − originX) / pixelWidth)
j = int((43.2 − originY) / pixelHeight)
```

Ces valeurs doivent être dans les limites acceptables de la matrice, sinon, il n'existera pas de pixel correspondant à votre position, et vous risquez de provoquer une erreur ! Si vous avez un offset négatif, il est fort probable que vous ayez choisit un point en dehors de la zone de couverture du raster.

À l'inverse, si on cherche les coordonnées physique d'un pixel (i, j), on peut utiliser les opérations suivantes :

```
x = originX + i∗ pixelWidth
y = originY + j∗ pixelHeight
```

Remarque 13

geotransform[2] et **geotransform**[4] indique des rotations éventuelles de l'image. Ces valeurs sont très souvent nulle et non-utiles ... S'il arrivait qu'elle ne soient pas nulle, c'est que vous n'avez pas de chance ... et aller voir une documentation plus complète !

Remarque 14 - Les demi-pixels comptent aussi

Les correspondances proposées dans ces formules localisent des pixels par leurs angles. Si vous travailler avec des images de résolution faible (*p.ex.* images MODIS à 1km), la taille du pixel peut avoir son importance et dans ce cas, il faudra prendre en compte des coordonnées au milieu du pixel. Par exemple :

```
x = originX + i∗ pixelWidth + pixelWidth/2
y = originY + j∗ pixelHeight + pixelHeight/2
```

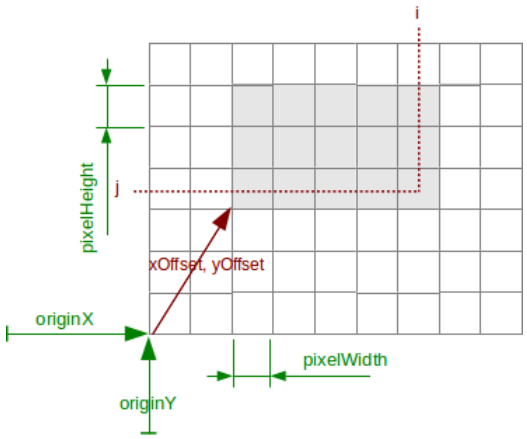


FIGURE IV.2 – Illustration de la correspondance entre coordonnées physiques (en vert) et index des pixels (en rouge) dans le cas d'une extraction de la zone grisée.

1.4.2 Passer des coordonnées aux pixels : cas de l'extraction d'une sous image

Il est important de noter que le **geotransform** est donné au niveau du **datasource**, il ne tient pas compte de la matrice qui a été effectivement extraite au moyen de la fonction **ReadAsArray**. Cette situation est illustré par la Figure IV.2.
Dans le cas où vous n'extrayez pas l'intégralité d'une couche, il faut donc en tenir compte lors de la mise en correspondance entre les coordonnées pixels et les coordonnées physiques. L'information du décalage est données en nombre de pixel par les **xOffset**, **yOffset**
On a alors :

```
x = originX + (i + xOffset) * pixelWidth
y = originY + (j + yOffset) * pixelHeight
```

et

```
i = int((4.2 - originX) / pixelWidth) - xOffset
j = int((43.2 - originY) / pixelHeight) - yOffset
```

1.4.3 Exemples

L'exemple ci-dessous charge une image raster dans un format "Géographique" classique (GeoTiff), récupère une matrice correspondant à la seconde bande de cette image et calcule le minimum et le maximum de la bande.

Le programme peut être téléchargé ici.

```
#affichage de quelques proprietes de l'image
print ('Nombre de couches : ' + str(dataset.RasterCount) )
print ('Taille de l\'image : ' + str(dataset.RasterXSize) + 'x' + str(dataset.RasterYSize) )
print ('Projection : ' + str(dataset.GetProjection()) )

if dataset.RasterCount < 2:
  print ("Warning : invalid raster file, not enought layers" )
  sys.exit(1)

#recuperation d'une bande de l'image sous la forme d'une matrice
band = dataset.GetRasterBand(2)
array=band.ReadAsArray()

bmin = array[1][1]
bmax = array[1][1]
for i in range(dataset.RasterXSize):
  for j in range(dataset.RasterYSize):
      if array[j][i]>bmax:
        bmax=array[j][i]
      if array[j][i]<bmin:
        bmin=array[j][i]

print ('Valeurs extremes de la couche : ' + str(bmin) +'-'+str(bmax))
```

Le second exemple présente plusieurs techniques qui peuvent être utiles :

— L'utilisation d'une liste de bandes pour facilement faire des traitements à partir de plusieurs bandes (impossible pour de grandes images !!)
— L'extraction d'une sous-zone de l'image à partir de coordonnées

Le programme peut être téléchargé ici.

```
# open the image
ds = gdal.Open('United_Kingdom_Ireland.2012247.terra.721.1km.tif', GA_ReadOnly)
if ds is None:
  print('Could not open image')
  sys.exit(1)

# liste de points d'interet (en WGS-84, systeme de coordonnees de la couche)
xValues = [-1.199, -0.049]
yValues = [50.567, 52.069]

geotransform = ds.GetGeoTransform()
xOrigin = geotransform[0]
yOrigin = geotransform[3]
pixelWidth = geotransform[1]
pixelHeight = geotransform[5]

imin = int((xValues[0] - xOrigin) / pixelWidth)
imax = int((xValues[1] - xOrigin) / pixelWidth)
jmin = int((yValues[1] - yOrigin) / pixelHeight)
jmax = int((yValues[0] - yOrigin) / pixelHeight)

# creation d'une liste de bandes
bandList = []

# lecture des bandes uniquement pour la region d'interet
for i in range(ds.RasterCount):
  band = ds.GetRasterBand(i+1)
  data = band.ReadAsArray(imin, jmin, imax-imin+1, jmax-jmin+1)
  bandList.append(data)

#Parcours des donnees
for i in range(0, imax-imin+1):
  for j in range(0, jmax-jmin+1):
    #calcul du NDVI
    ndvi = (float(bandList[1][j][i]) - float(bandList[2][j][i])) / (float(bandList[1][j][i]) + float(bandList[2][j][i])
        )
    s = str(i) + ' ' + str(j) + ' : ' + str(ndvi)
    print(s)
```

2 Création d'une couche raster

2.1 Principe

Nous allons maintenant créer des images rasters. Le principe général de la création d'un raster est le suivant :

1. Commencer par créer un objet correspondant à un fichier raster en précisant les informations nécessaires : sa taille, son nombre de bande, sa géométrie, etc.
2. Pour chaque couche, créer à côté une matrice **numpy** à deux dimensions sur laquelle vous pourrez travailler facilement
3. "Copier" les données de la matrice **numpy** dans la couche de votre choix

Rentrons maintenant un peu plus dans le détail.

De même que pour les images vecteurs, il faut tout d'abord créer un *driver* spécifique à un format de fichier. Dans mon cas, je n'utilise que du GeoTiff, pour les autres noms de driver, voir les documentations de GDAL.

```
driver = gdal.GetDriverByName( "GTiff" )
```

La création d'une nouvelle couche peu se faire grâce à la fonction **Create** du driver. Cette fonction prend 5 arguments :

— le nom du fichier,
— le nombre de lignes et de colonnes de l'image (toutes les bandes ont la même taille),
— le nombre de bandes,

— le codage des éléments de la bande. Les valeurs pour ce paramètre sont, par exemple, **GDT_Byte**, **GDT_Int16**, **GDT_Int32**, **GDT_Float32**, **GDT_Float64**.

L'exemple ci-dessous va donc créer un fichier d'image contenant 5 bandes de taille $(XSize, YSize)$ dont les pixels auront des valeurs entières (codées sur 32 bits)

```
dst_ds=driver.Create("output.tif", XSize, YSize, 5, gdal.GDT_Int32)
```

Il est ensuite possible d'attribuer une projection et une transformation à la couche en utilisant les fonctions ci-dessous. Ces attributs sont souvent récupérés par recopie de ceux d'une couche d'origine.

— **dst_ds.SetProjection** (...) pour définir une projection (voir l'utilisation des projections dans le chapitre précédent)

— **dst_ds.SetGeoTransform**(...) pour définir une transformation géométrique. La fonction attend en paramètre une matrice de taille 6 (*cf.* Section 1.4).

La création d'une matrice **numpy** doit se faire en concordance avec les attributs d'une couche. Il faut :

— que la taille de la matrice corresponde à la taille de la couche (en nombre de pixels),

— que le type de la matrice **numpy** correspond au type des éléments d'une bande de la couche. On retrouve les mêmes types que pour le cinquième paramètres de la création d'une couche pour les matrices **numpy** : **bool**, **int16**, **int32**, **float32**, **float64**.

L'exemple ci-dessous créé une matrice de 0 avec les propriétés nécessaires à la bande du raster : taille et type (**float64**).

```
dst_ds=driver.Create("output.tif", XSize, YSize, 1, gdal.GDT_Float64)
size = (YSize, XSize)
array= np.zeros( size , dtype='float64')
dst_ds.GetRasterBand(1).WriteArray( array )
```

Les valeurs d'une matrice sont attribuées à une bande à l'aide de la fonction **WriteArray** d'une bande :

```
dst_ds.GetRasterBand(1).WriteArray( array )
```

! Attention ! - Attention aux tailles des matrices numpy

Il est très important de noter que dans l'exemple ci-dessus, la taille de la matrice **numpy** doit être données en spécifiant d'abord la taille de Y puis la taille en X. On a bien :

```
dst_ds=driver.Create("output.tif", XSize, YSize, 1, gdal.GDT_Float64)
size = ( YSize, XSize)
array= np.zeros( size , dtype='???')
```

Remarque 15 - Des erreurs difficiles à trouver !

L'utilisation des matrices numpy et l'utilisation de la librairie GDAL donne accès à des informations de très bas niveau sur l'organisation des fichiers. Pour être efficace, il y a très peu de vérification effectuées sur la validiter des opérations de copies de données des matrices. Donc, si votre matrice n'est pas en correspondance avec la déclaration faite lors de la création du fichier (erreur de taille de la matrice ou erreur sur le format des données), alors le programme n'engendrera aucune erreur lors de l'exécution ! Mais ... l'image résultante sera totalement étrange.

2.2 Quelques exemples

On propose ci-dessous 2 exemples de manipulations de fichiers raster : le calcul d'une image NDVI et un recodage des pixels.

2.2.1 Exemple : calcul d'une image NDVI

Dans cet exemple, on illustre la création d'une image NDVI à partir d'une image MODIS (bandes 7, 2 et 1).

Le programme peut être téléchargé ici.

```
import matplotlib

os.chdir("/home/tguyet/Enseignements/2016-2017/M2/ProgSIG/Tutoriel/LiveData/data/raster")
dataset = gdal.Open('United_Kingdom_Ireland.2012247.terra.721.1km.tif', GA_ReadOnly)
if dataset is None:
  print('Could not open image')
  sys.exit(1)

if dataset.RasterCount != 3:
  print("Warning : invalid raster file : must have 3 bands\n")
  sys.exit(1)

#recuperation des donnees sous la forme de matrices
band = dataset.GetRasterBand(2)
array_nir =band.ReadAsArray()
band = dataset.GetRasterBand(3)
array_red=band.ReadAsArray()

# Construction d'une matrice vide pour l'image NDVI
array= np.zeros( (dataset.RasterYSize,dataset.RasterXSize) , dtype='float64')

# Remplissage de la matrice
for j in range(dataset.RasterXSize):
  for i in range(dataset.RasterYSize):
    if float( array_nir[i][j]) + float(array_red[i][j]) != 0:
      array[i][j]= (float( array_nir[i][j]) - float(array_red[i][j])) / (float( array_nir[i][j]) + float(array_red[i][j])
          )

# Enregistrement de l'image transformee
driver = gdal.GetDriverByName( "GTiff" )
dst_ds = driver.Create( "NDVI.tif", dataset.RasterXSize, dataset.RasterYSize, 1, gdal.GDT_Float64 )

if dst_ds is None:
  print("Error : impossible to create the file !\n")
  sys.exit(1)

if not dataset.GetGCPProjection() is None:
  dst_ds.SetProjection( dataset.GetGCPProjection() ) # meme projections
  geotransform = dataset.GetGeoTransform()  # meme transformations geometriques

if not geotransform is None:
  dst_ds.SetGeoTransform( geotransform )

dst_ds.GetRasterBand(1).WriteArray( array )

import matplotlib.pylab as plt
plt.imshow(array)

#fermeture des fichiers
dst_ds=None
dataset=None
```

2.2.2 Exemple : conversion d'un codage des valeurs gdal.GDT_Int8 vers gdal.GDT_Byte

Dans cet exemple, on dispose d'une image codant des informations de présence ou d'absence (1 ou 0) de nuages (je crois !) sur 8 jours. Le codage utiliser pour cette image est assez spécifique puisqu'il s'agit d'une unique bande codée par un entier sur 8 bits. Chaque bit de l'entier correspond à la présente pour un jour donné. Par exemple, si un pixel vaut 68, soit 01000010 en binaire, on indique qu'il y avait des nuages en second jour et l'avant dernier. Ce codage est astucieux, économique en place, mais peu pratique à utiliser !
L'idée du programme est de créer une nouveau fichier contenant 8 couches codées sous la forme de valeurs binaires (gdal.GDT_Byte).

Le programme peut être téléchargé ici.

```
#ouverture du fichier RASTER
dataset = gdal.Open("LST_days_night_clear_04juillet_11juillet_2010.tif", GA_ReadOnly )

#affichage de quelques proprietes de l'image pour info
print('Nombre de couches : ' , dataset.RasterCount )
print('Taille de l\'image : ', dataset.RasterXSize,'x', dataset.RasterYSize )
print('Projection : ', dataset.GetProjection() )
```

```
#recuperation des donnees sous la forme d'une matrice (ici, il n'y a qu'une bande, donc on recupere bien une matrice
      en deux dimensions)
array=dataset.ReadAsArray()

#Creation d'une nouvelle image avec 8 couches d'entiers codes sur 8bits (gdal.GDT_Byte)
driver = gdal.GetDriverByName( "GTiff" )
dst_ds = driver.Create( "output.tiff", dataset.RasterXSize, dataset.RasterYSize, 8, gdal.GDT_Byte )

dst_ds.SetProjection( dataset.GetGCPProjection() ) # meme projections
geotransform = dataset.GetGeoTransform()  # meme transformations geometriques
if not geotransform is None:
  dst_ds.SetGeoTransform( geotransform )

#on decompose l'image selon l'apparition d'une puissance de 2 dans le codage
for p in range(0,8):
  print("decomposition de la bande " + str(p))
  rasterlayer = numpy.zeros( (dataset.RasterYSize, dataset.RasterXSize), dtype=numpy.uint8 )
  val=2**p #ici, val correspond a la valeur de la puissance de 2 de la bande a extraire

  #on parcourt l'image, pixel par pixel
  for i in range(dataset.RasterXSize):
    for j in range(dataset.RasterYSize):
      #La ligne suivante est centrale et astucieuse : & est un operateur BitWised
      # il retourne le resultat d'une operation bit a bit
      # si array[j][i] vaut 17 (soit, en binaire, 10001000) et que val vaut 16 (soit 00001000)
      # l'operateur retourne la valeur 00001000 : un ET logique bit a bit.
      # Ce test est donc non-nul si array[j][i] "contient" le p-eme bit a 1.
      if array[j][i] & val:
        #ici, on attribut la valeur 255 a la p-eme couche de sortie
        rasterlayer[j][i] = 255;
      else:
        #ici, on attribut la valeur 0 a la p-eme couche de sortie
        rasterlayer[j][i] = 0;

  #on ecrit maintenant dans la bande de l'image de sortie
  dst_ds.GetRasterBand(p+1).WriteArray( rasterlayer )

#fermeture des fichiers
dst_ds=None
dataset=None
```

2.3 Création de données en mémoire

Dans certains cas, il n'est pas nécessaire de créer des couches sous la forme de fichiers, il est possible de créer des couches uniquement en mémoire. Cela permet de continuer à manipuler des couches avec les mêmes fonctionnalités que pour une source extérieure. Pour cela, il suffit de créer une couche à partir d'un driver `'MEM'` (en mémoire).

```
driver = gdal.GetDriverByName('MEM') # In memory dataset
target_ds = driver.Create('', cols, rows, 1, gdal.GDT_UInt16)
target_ds.SetGeoTransform(geo_transform)
target_ds.SetProjection( projection )
```

Cette fonctionnalité est utilisé par exemple lorsqu'on souhaite rasterizer une couche vectorielle pour réaliser des calculs dessus. Dans ce cas, on a besoin pendant le programme d'obtenir la matrice issue de la rasterization, mais on n'a pas besoin de concrétiser cette matrice physiquement dans un fichier.

2: TO BE FILLED

3 Manipulation d'une matrice numpy

Dans les exemples de la section 2.2, nous avons illustré des calculs réalisé en algorithmique "classique" sur les matrices construites directement à partir des données lues des fichiers (bande par bande).
Par réalisation algorithmique "classique", j'entends l'utilisation d'une double boucles **for** imbriquées. Cette manière de programmer est peu efficace en Python. La librairie **numpy** permet de réaliser des opérations similaires au moyen d'**opérations matricielles**. À la place de programmer des traitements valeur par valeur, **numpy** va traiter des vecteurs ou des matrices de valeurs. L'intérêt de procéder ainsi est de gagner en performance, puisqu'une opération matricielle sera beaucoup plus efficace qu'une opération réalisée par des boucles.

Prenons l'exemple ci-dessous qui affiche le temps nécessaire pour multiplier par deux toutes les composantes d'un vecteur de taille 10000. La boucle est 100 fois plus lente que l'opération matricielle.

```
vec=np.ones( 10000 )

shape = np.shape(vec)

start = time.time()
for i in range( shape[0] ):
    vec[i] *= 2
duration = time.time()-start
print (duration)

start = time.time()
vec *= 2
duration = time.time()-start
print (duration)
```

3.1 La forme des matrices et opérations élémentaires

La notion de forme d'une matrice désigne le nombre de dimension d'une matrice et le nombre de composantes par dimension. Dans votre cas, vous êtes amener à rencontrer des tableaux de dimension 1, 2 ou 3 :

- Dimension 1 : il s'agit d'un vecteur (**np.zeros**(300))
- Dimension 2 : il s'agit d'une matrice (**np.zeros**((300,10)))
- Dimension 3 : il s'agit d'un "cube" (**np.zeros**((3,10,10)))

La forme d'une matrice (obtenue grâce à la fonction **np.shape**) est un tuple qui contient le nombre de compostante selon chaque dimension. Dans le cas du vecteur ci-dessus, il y a 300 composantes. Pour la matrice, la forme (300,10) indique une matrice de taille 300 lignes par 10 colonnes. [4] Pour le cube, on peut le voir comme 3 matrices de taille 10 fois 10.
Dans la suite, j'utilise le terme *matrice* pour désigner des *tableaux* de n'importe quelle dimension.

Connaître la taille des matrices et savoir éventuellement les réorganiser est très important puisque ces tailles et l'organisation des matrices conditionne les traitements matriciels qui peuvent être réalisés.
En particulier, toutes les opérations arithmétiques (composante par composante) entre les matrices nécessitent d'avoir des matrices qui ont exactement les mêmes formes.

```
a = np.floor(10*np.random.random((3,4)))
print (a)
print (a.shape) # (3, 4)

# exemple de manipulation des formes d'un tableau
flata = a.ravel()  # transformation sous forme de vecteur
print ( flata )

ra = a.reshape(6,2) #transformation de la forme
print (ra)

rae = a.reshape(5,2) #provoque une erreur: pas le bon nombre de cases !

ta = a.T  # transposée de la matrice
print (ta)

a. resize (6,2)  #comme reshape, avec modification de a
print (a)
```

! Attention ! - Ordre du reshape

Les données d'une matrice sont traiter dans un ordre spécifique qui va contraindre le **reshape** sur les réorganisations des données selon les dimensions.

Une autre opération (magique) sera certainement très utile lorsqu'on traite des images de plusieurs bandes est la fonction **dstack** qui permet de transformer le point de vue sur une image. On peut en effet avoir deux visions d'une image

4. La notion de ligne et colonne est conventionnelle

— une pile de k couches (matrices de dimension 2), où k est le nombre de bandes de votre image,
— une matrice de pixels (eux-même de dimension k)

Il est très aisé de passer de la première représentation (celle des fichiers ouverts par GDAL) à la seconde grâce à la fonction **dstack**.

```
img = np.floor(10*np.random.random((3,10,10)))
dimg=np.dstack(img)
print (dimg.shape) #(10,10,3)
```

Sous matrices : il est possible de sélectionner uniquement certaines bandes de l'image. Dans l'exemple ci-dessous, on illustre la sélection de 2 composantes sur 3 sur la première dimension, en conservant l'intégralité des compostantes sur autres dimensions.

```
img = np.floor(10*np.random.random((3,10,10)))
b13 = img[(0,2) ,:,:]
print (b13.shape) #(2, 10, 10)
```

! Attention ! - Indices

Contrairement aux structures indexées de Python (comme les listes), les indices des tableaux **numpy** commencent à 0. C'est de nouveau un source fréquente de bug.

3: TO BE FILLED

3.2 Traitement matricielle bande par bande

Le programme peut être téléchargé ici.

Reprenons l'exemple du calcul de l'indice NDVI

```
dataset = gdal.Open('United_Kingdom_Ireland.2012247.terra.721.1km.tif', GA_ReadOnly)
if dataset is None:
  print ('Could not open image')
  sys.exit(1)

#recuperation des donnees sous la forme de matrices
band = dataset.GetRasterBand(2)
array_nir =band.ReadAsArray()
band = dataset.GetRasterBand(3)
array_red=band.ReadAsArray()

# Construction d'une matrice vide pour l'image NDVI
denom=array_nir + array_red
denom[denom==0]=0.001 #traitement des zeros
array= (array_nir-array_red)/denom

# Enregistrement de l'image transformee
driver = gdal.GetDriverByName( "GTiff" )
dst_ds = driver.Create( "NDVI.tif", dataset.RasterXSize, dataset.RasterYSize, 1, gdal.GDT_Float64 )

if dst_ds is None:
  print("Error : impossible to create the file !\n")
  sys.exit(1)

if not dataset.GetGCPProjection() is None:
  dst_ds.SetProjection( dataset.GetGCPProjection() ) # meme projections
  geotransform = dataset.GetGeoTransform()  # meme transformations geometriques

if not geotransform is None:
  dst_ds.SetGeoTransform( geotransform )

dst_ds.GetRasterBand(1).WriteArray( array )

#fermeture des fichiers
dst_ds=None
dataset=None
```

3.3 Traitement matricielle pixel par pixel

Imaginons (là, j'ai pas d'exemple sous la main) qu'on souhaite appliquer un modèle linéaire pour faire une détection à partir des bandes 20m d'une image sentinel-2 (B5, B6, B7, B11 et B12)... prenons le cas d'une équation sous la forme

$$q = 3.45 \times B5 + 1.512 \times B6 + 4.196 \times B7 + 13.25 \times B11 - 5.245 \times B12$$

On souhaite alors construire une image pour avoir cet indice. C'est ce que réalise le programme ci-dessous. Le programme peut être téléchargé ici.

```
ds = gdal.Open('extract_sentinel.tif', GA_ReadOnly)
if ds is None:
  print ('Could not open image')
  sys.exit(1)

bandList=[]
for i in range(ds.RasterCount):
    band = ds.GetRasterBand(i+1)
    data = band.ReadAsArray()
    bandList.append(data)

img=np.array(bandList)
img=img[(4,5,6,7,8) ,:,:]
img=np.dstack(img)

q = img.dot( [3.45, 1.512, 4.196, 13.25, -5.245] )
print (q.shape) #dimension 2, taille de l'image

# Enregistrement de l'image transformee
driver = gdal.GetDriverByName( "GTiff" )
dst_ds = driver.Create( "NDVI.tif", ds.RasterXSize, ds.RasterYSize, 1, gdal.GDT_Float64 )

if dst_ds is None:
  print ("Error : impossible to create the file !\n")
  sys.exit(1)

if not ds.GetGCPProjection() is None:
  dst_ds.SetProjection( ds.GetGCPProjection() ) # meme projections
  geotransform = ds.GetGeoTransform()  # meme transformations geometriques

if not geotransform is None:
  dst_ds.SetGeoTransform( geotransform )

dst_ds.GetRasterBand(1).WriteArray( q )

#fermeture des fichiers
dst_ds=None
ds=None
```

4 Exercices

Exercice 22 (Calcul d'un indice SAVI) *Le SAVI (Soil Adjusted Vegetation Index) est un indice de végétation amélioré*

$$SAVI = (1 + L) \times \frac{PIR - R}{PIR + R + L}$$

avec $L = 0.5$, PIR la bande de proche infra-rouge (bande 2 de l'image), R la bande de rouge (bande 3 de l'image).
Écrire un programme qui calcule une image SAVI à partir d'une image MODIS `United_Kingdom_Ireland.2012247.terra.721.1km.tif` (ou autre) d'une part en utilisant des boucles puis en utilisant des opérations matricielles.

Exercice 23 (Compréhension générale de Python/GDAL) *On fournit le code suivant :*

```
1 import gdal, ogr
2 from gdalconst import *
3 import numpy
4 import os
5
```

```
6  dataset = gdal.Open("image_raster.tif", GA_ReadOnly )
7
8  data=dataset.ReadAsArray()
9  #on force la matrice a ne contenir que des valeurs entieres :
10 data = numpy.array(data, dtype = int)
11
12 driver = ogr.GetDriverByName('ESRI Shapefile')
13
14 if os.path.exists('test.shp'):
15   driver.DeleteDataSource('test.shp')
16 outDS = driver.CreateDataSource('test.shp')
17
18 outLayer = outDS.CreateLayer('test', geom_type=ogr.wkbPoint)
19
20 geotransform = dataset.GetGeoTransform()
21 featureDefn = outLayer.GetLayerDefn()
22
23 for i in range(dataset.RasterXSize):
24   for j in range(dataset.RasterYSize):
25     if (data[j][i]==1) | (data[j][i]==2):
26       Xgeo = geotransform[0] + i*geotransform[1] + j*geotransform[2]
27       Ygeo = geotransform[3] + i*geotransform[4] + j*geotransform[5]
28       feature = ogr.Feature(featureDefn)
29       point = ogr.Geometry(ogr.wkbPoint)
30       point.AddPoint(Xgeo,Ygeo)
31       feature.SetGeometry(point)
32       outLayer.CreateFeature(feature)
33       feature.Destroy()
34
35 dataset=None
36 outDS=None
```

Compréhension générale du code *On commence l'exercice par quelques questions précises sur le fonctionnement du code. Ces questions sont également là pour vous aider à arriver à la compréhension globale du code.*

Question a) Que permet de faire la ligne 15

Question b) Que se passe-t-il à la ligne 6 si le fichier `image_raster.tif` n'existe pas ? Que faut-il ajouter pour traiter ce cas dans le programme ?

Question c) (⋆) Que se passe-t-il si on oublie la ligne 33

On passe maintenant à la compréhension de sous parties du code.

Question d) Quel type de schéma classique reconnait-on dans les lignes 23-24 ?

Question e) Que permettent de faire les lignes 26 et 27 ?

Question f) Indiquer maintenant ce que font les lignes 28 à 32 ?

Question g) On donne l'image ci-dessous pour `image_raster.tif`. Chaque pixel contient une valeur entière tel qu'indiqué dans la matrice. Quel sera le résultat du tratement ? (faire un dessin et expliquer)

3	4	2	1
1	3	2	1
0	2	5	7
0	1	3	5

Question h) (⋆) Expliquer ce que fait le programme.

Modification du code

Pour cette partie, on attend ***un seul code*** *qui doit contenir toutes les modifications pour chacune des questions ci-dessous. Prenez garde à bien lire toutes les fonctions avant de vous lancer dans l'exercice. Pour répondre à la question, vous pouvez soit recopier tout le code, soit "réutiliser" des parties du code en faisant des "copier-coller".*

Toutes les explications accompagnant votre proposition sont les bienvenues et permettront de vous évaluer avec plus d'indulgence en cas d'erreur ou de coquille.

Question i) Modifier le code pour que l'utilisateur puisse choisir les valeurs de l'image à prendre en compte à la ligne 25. Vous utiliserez la fonction **input** pour demander la saisie d'un entier à l'utilisateur.

Question j) Créer une fonction **process(filein , fileout)** pour encapsuler le traitement proposé dans le programme ci-dessus. La fonction traitera le fichier **filein** en créant un fichier **fileout** selon le même principe que le code étudié. Vous pourrez modifier le profil de fonction pour ajouter de nouveau paramètres

La correction peut être téléchargée ici.

Exercice 24 (Construction d'un tableau "sklearn-ready") *Dans cet exercice, l'objectif est de préparer un jeu de données pour faire de l'apprentissage automatique. On va utiliser pour cela la librairie* **panda** *qui propose une structure de données de* **DataFrame**, *semblable à celle disponible en* **R**.
Vous utiliserez pour cela les images `zone1.tif` *et* `zone1_cl.tif` *disponibles dans le répertoire* `./batch/DonneesPayTal` *du répertoire des données du cours. La première image est (de mémoire) une image SPOT 5 dans la région Angevine et la seconde image est une classification obtenue par analyse automatique et corrigée par photo-interprétation. L'image SPOT comportent 5 bandes (valeurs entières). L'image de classe comporte une unique bande (11 classes principales codées par des valeurs entières). Les deux images sont dans le même système de coordonnées et sont alignées géométriquement (c.-à-d. correspondance des pixels).*
On cherche ici à préparer un jeu de données pour faire de l'apprentissage automatique. Pour particulièrement, on souhaite construire un classifieur capable d'étiquetter automatiquement les pixels d'une image SPOT5 (par exemple, pour classer les autres images similaire du répertoire) ! Pour cela, il faut contruire un tableau de données comportant 6 colonnes : une colonne pour chaque bande de l'image + 1 colonne pour la classe.

Question a) L'objectif de cet exercice est de construire un **DataFrame panda** correspondant à ces images. *Pour chaque pixel, vous irez chercher les données utiles dans les deux images.*
NB : pour éviter tes temps de calcul trop long, il pourrait être judicieux de ne prendre qu'un pixel tous les 100 pixels (cas simple de l'exercice sur les échantillonage) !

Pour utiliser un **DataFrame panda**, *vous aurez besoin de quelques instructions élémentaires :*

```
#import de la librairie Panda
import panda as pd

# creation d'un dataframe avec les noms de colonnes
df = pd.DataFrame(columns=['col1', 'col2', 'col3'])

# exemple d'ajout de valeurs ( aleatoires dans le tableau de donnees)
for i in range(5):
  df.loc[i] = [randint(-1,1) for n in range(3)]

# autre exemple d'ajout d'elements avec la fonction append
df2 = pd.DataFrame(columns=['col1', 'col2', 'col3'])
for i in range(5):
  df2.loc[i] = [randint(-1,1) for n in range(3)]

df = df.append( df2 )

# creation d'un Data.Frame a partir d'une liste de vecteurs
data=[]
for i in range(5):
  data.append( [randint(-1,1) for n in range(3)] )

# creation d'un dataframe avec les noms de colonnes
df = pd.DataFrame( data, columns=['col1', 'col2', 'col3'] )
```

Exercice 25 (Modification de l'échantillonnage d'une image) *On dispose d'une image MODIS de taille $H \times L$. Pour se simplifier le travail, on utilise une image avec une seule bande spectrale (utiliser par exemple l'image,* `urbain.tif`*). On cherche à modifier la résolution de cette image. Cette opération nécessite de créer une nouvelle image raster en faisant attention à deux aspects de l'images :*

- *la matrice de pixel dont la taille est modifiée,*
- *la transformation géométrique qui doit être adapté pour conserver un référencement spatial correct.*

Question a) Recopie d'une image *Commencer par créer un script qui construit une recopie d'une image pixel par pixel. L'image devra avoir les mêmes caractéristiques géographiques (même projection et même transformation).*

Question b) Cas simple : réduction de la résolution par 2. *Dans le cas de la réduction de la résolution par 2, les opérations sont assez simples. On doit ainsi construire une nouvelle matrice qui comportera 4 fois moins de pixels (c.-à-d. une matrice de taille $H/2 \times L/2$). La transformation géographique doit,*

quant à elle, mentionner des tailles de pixels qui sont 2 fois grands (valeurs aux positions 1 et 5 dans le vecteur de transformation).
Modifier le script de recopie pour préparer une nouvelle image avec ces caractéristiques.
Une fois qu'on a établi les caractéristiques générales de l'image, il faut construire la matrice proprement dite. Le principe est de parcourir l'image transformée pixel par pixel, et pour chaque pixel (x, y), d'aller chercher les informations dans l'image originale pour construire la valeur du pixel (x, y).
Considérons pour cela un pixel (x, y) de l'image transformée, la valeur de ce pixel va être calculée comme la moyenne des 4 pixels couverts par le pixels (x, y).
Quels seront les coordonnées dans l'image original de ces 4 pixels (en fonction de x et y) ?
Finaliser le script pour réduire l'image.

Question c) Cas général ($\star$) *On note k un entier > 1. Modifier votre script pour réduire l'image d'un facteur k.*

Exercice 26 (Histogrammes($\star$))

Question a) Écrire une script qui calcule l'histogramme des valeurs d'une bande du fichier `extract.tif` [5], dans le système de projection Lambert-93 .
L'exemple ci-dessous illustre la construction d'un histogramme à partir d'un vecteur. L'histogramme est un vecteur qui contient le nombre d'occurrence d'une valeur du vecteur. Adapter ce code pour traiter la matrice récupérée depuis un fichier raster.

```
import numpy as np
vec=np.random.randint(10, size=1000) #generation d'un vecteur de taille 1000 contenant des valeurs aleatoires
      entre 0 et 10

histo=np.zeros( np.max(vec)+1 ) # "max(vec)+1" donne le nombre de valeurs distincte dans le vecteur

for i in range( len(vec) ):
  histo[ vec[i] ] = histo[ vec[i] ] +1

print ('classe majoritaire : '+str(np.argmax(histo)))
```

NB : il existe une fonction `np.histogram` *mais il est intéressant pour nous de refaire partiellement cette fonction que nous pourrons adapter plus facilement dans la question suivante !*

Question b) Histogramme d'une région particulière
On définit maintenant le polygone ci-dessous (coordonnées exprimées en Lambert-93) :

```
ring = ogr.Geometry(ogr.wkbLinearRing)
ring.AddPoint(216656.0,6771560.8)
ring.AddPoint(216545.9,6771809.5)
ring.AddPoint(216907.7,6771904.4)
ring.AddPoint(217043.1,6771845.5)
ring.AddPoint(216985.7,6771741.4)
ring.AddPoint(216756.2,6771709.2)
ring.AddPoint(216769.2,6771602.1)
ring.CloseRings()

limits = ogr.Geometry(ogr.wkbPolygon)
limits.AddGeometry(ring)
```

On souhaite construire l'histogramme des pixels qui se trouvent à l'intérieur de cette géométrie.
Pour des raisons d'efficacité, on propose de procéder de la sorte :

1. *Commencer par récupérer l'enveloppe de la géométrie (utiliser la fonction* `limits.GetEnvelope()`*). L'enveloppe est un vecteur de quatre valeurs :* `[xmin, xmax, ymin, ymax]`*.*
2. *Extraire une matrice pour la bande de données uniquement pour la zone du raster limitée à l'enveloppe (spécifier correctement la zone à extraire dans la fonction* `ReadAsArray`*)*
3. *Construire ensuite l'histogramme à partir de cette matrice, mais vous ne prendrez en compte que les pixels qui sont à l'intérieure de la géométrie* `limits` *(utiliser la fonction* `limits.Contains()`*. Pour utiliser cette fonction vous serez obligé de créer une géométrie de type point à partir des coordonnées d'un pixel.*
4. *afficher l'histogramme et l'identifiant de la classe majoritaire (*`am=np.argmax(histo)`*)*

Question c) Modifier votre code de sorte à créer une fonction avec le profil ci-dessous où **theband** est la bande à traiter, **thegeom** est le polygone et **am** est la classe majoritaire retournée par la fonction.

```
def geom_getclass(thegeom, theband, geotransform):
  ...
  return am
```

Question d) Projection sur le cadastre
On dispose maintenant d'une fonction capable de données la classe majoritaire présente dans une bande pour n'importe quel polygone. Nous allons utiliser cette fonction pour traiter tous les polygones d'un fichier shapefile `CADASTRE.shp`*. L'objectif est de créer une fichier* `projection.shp` *qui contiendra les formes du cadastre pour lesquelles un attribut* `class` *retiendra la classe majoritaire du fichier* `extract.tif`
Pour cette question, il n'est pas nécessaire de se poser la question des projections. Toutes les coordonnées sont exprimées en Lambert-93.
Le principe de l'algorithme est le suivant :

```
Recuperer la bande b du fichier 'extract.tif'
Creer une nouvelle couche vectorielle 'projection.shp'

Pour chaque polygone P du cadastre faire:
  am = geom_getclass(P, b) #votre fonction !
  copier la geometrie de P
  attribuer la valeur am à la forme
  ajouter la forme a projection.shp
```

Exercice 27 (Masquage des nuages)

Question a) Écrire un script qui charge toute l'image raster et qui compte le nombre de pixels "blanc" *Un pixel est considéré comme blanc lorsque la valeur de sa première et sa troisième bande est supérieure à 180.*

Question b) Écrire un script qui créé une image raster au format GeoTiff avec les mêmes caractéristiques géométriques que l'image d'origine mais contenant des **GDT_Byte**. Un pixel vaudra 0 si le pixel de l'image d'origine était "blanc" et 0 sinon.

Exercice 28 (Raster to point : utilitaire pour la création de Heatmap) *Le problème pratique provient d'un cas où on dispose d'une image raster représentant la présence (ou l'absence d'une observation géolocalisée) et on souhaite construire une carte de chaleur (heatmap) qui va donner en tout point une densité de présence du phénomène. Il se trouve qu'un plugin QGis propose la construction d'une heatmap, je ne vais donc pas refaire cette fonctionnalité complexe. En revanche, ce plugin nécessite d'utiliser en entrée une image vectorielle contenant des points localisant le phénomène.*
L'objet de l'exercice est donc d'écrire un programme python qui transforme une image raster (utilisez l'image `urbain.tif`*) en une image vecteur (couche de points). Un point sera créé pour chaque pixel qui n'est pas nul (les différentes valeurs correspondent à des décennies d'urbanisation).*

5 Statistiques zonales

L'objet de cette section est de réaliser des statistiques zonales simple ... par exemple la moyenne des valeurs d'une bande raster pour une zone délimitée par un polygone. Bien évidemment, on peut avoir besoin de réaliser cette même opération pour une collection de polygones, voire de calculer d'autres statistiques que la moyenne.
Dans cette section, l'objectif est d'étudier en détail le problème du calcul des statistiques zonales. Dans un premier temps,

5.1 Version algorithmique

Dans cette partie, l'idée est de vous faire programmer l'idée naïve de l'algorithme des statistiques zonales consistant

5.2 Version numpy

5.3 Utilisation du module rasterstat

rasterstats est un module Python permettant de calculer des statistiques de zones de données raster à partir de géométries vectorielles.
L'utilisation basique de ce module utilise directement des fichiers

```
from rasterstats import zonal_stats
zonal_stats("polygons.shp", "elevation.tif", stats="count min mean max median")
```

CHAPITRE V

Classer des images avec `sklearn`

La classification d'images est une tâche usuelle en traitement d'image satellite. Que se soit en version non-supervisée pour simplifier le contenu des images ou bien en superviser pour aller vers l'automatisation de la classification des pixels des images dans des classes d'occupation du sol, de type de surface, etc. la mise en place de procédure d'apprentissage automatique (machine learning) est de plus en plus courante. Ces fonctionnalités de classification autoamtique sont disponibles dans les outils de traitement d'images tels que ENVI ou ArcGis. L'objectif est ici de vous rendre plus autonomes vis-à-vis de ces solutions logicielles propriétaires : à la fois pour être en mesure de développer des analyses sans avoir à payer la licence onéreuse de ces outils mais également en s'affranchissant des limitations imposées par des choix propres à ces outils.
En utilisant la programmation à l'aide d'outils libres, vous vous donnez accès à une panoplie de méthode d'apprentissage (et de validation) qui vous permettront de décider en connaissance des meilleurs méthodes d'apprentissage à utiliser pour vos données.

Ce chapitre introduit l'utilisation de la librairie `sklearn` ou `scikit-learn`[1]. C'est une librairie dédiée aux méthodes d'apprentissage automatique (supervisées et non-supervisées) qui connait un grand succès dans le monde Python. Elle met à disposition un grand nombre d'algorithmes classiques et également une documentation très riche (à la fois sur la théorie de l'apprentissage automatiquement et sur l'utilisation pratique de **sklearn**).
Quelques intérêts à l'utilisation de **sklearn** pour découvrir l'apprentissage automatique :

- Elle dispose d'une excellente documentation fournissant de nombreux exemples
- Elle dispose d'une API uniforme entre tous les algorithmes, ce qui fait qu'il est facile de basculer de l'un à l'autre
- Elle est très bien intégrée avec les Librairies Pandas et Seaborn
- C'est un projet open source avec une grande communauté et plus de 800 contributeurs référencés sur GitHub !
- Son code est rapide

1 Introduction à la librairie sklearn et consorts

Scikits-learn est une librairie d'apprentissage automatique couvrant une très grande gamme des tâches usuelles de cette discipline : apprentissage supervisé, non supervisé, par renforcement, par transfert. On y retrouve les algorithmes classiques tels que :

- Non-supervisé
  - K-Means
  - DBScan
- Supervisé
  - Linear Regression (régression linéaire),
  - Decision Tree (arbre de décision),
  - SVM (machines à vecteur de support),
  - Naive Bayes (classification naïve bayésienne),
  - KNN (Plus proches voisins),
- Réduction de dimensionnalité (*p.ex.* PCA)

La librairie **sklearn** est très bien pensée pour faciliter le travail de la mise en place d'une chaîne de traitement incluant le chargement de données, sa sélection et l'affichage de résultat. Pour cela il se connecte facilement à d'autres librairies python également classiques dans le domaine des *data science* avec Python :

- la librairie **numpy** pour la gestion efficace des tableaux de données
- la librairie **pandas** qui offre la possibilité de manipuler des jeux de données sous forme tabulaire

1. Plus d'info et aide sur cette librairie : `http://scikit-learn.org`.

(**DataFrame**) assez aisée et intuitive.
— la librairie **matplotlib** pour l'affichage de graphiques
— la librairie **seaborn** utilisée également pour l'affichage de graphique orienté vers l'analyse de données (fonctionnalités similaires à celles de `R`)

Dans la section suivante, on présente la "philosophie" de la librairie **sklearn**

1.1 Philosophie et concepts de la librairie

La philosophie générale de **sklearn** celui de l'apprentissage automatique classique. Il vise à construire des **classifieurs** (resp. des **regresseurs**), c'est-à-dire des objets qui peuvent associer des données d'entrées à une étiquette, la classe (resp. à une valeur prédite).
Pour préciser les concepts de cette librairie, il est donc utile de présenter rapidement la structure générale des données qui peuvent être traitées, puis les concepts autours des classifieurs (et leur méthodes d'apprentissages associés).

1.1.1 Les données

Les données traitées par la librairie **sklearn** sont des données tabulaires. Ces données sont représentées par des tableaux à 2 dimensions :
— Les lignes représentent les enregistrements,
— Les colonnes les attributs qui caractérisent chacun des enregistrements

Une donnée (un enregistrement) est un vecteur de caractéristiques (valeurs des attributs), généralement des réels, mais des entiers, booléens et des valeurs discrètes sont autorisées dans certains cas [2]. Dans le cas de l'apprentissage supervisé, un attribut spécifique correspond au label d'un enregistrement (son *gold-standard*). Les labels peuvent être de différents types, généralement ce sont des valeurs discrètes (des entiers ou chaînes de caractères). Chaque valeur correspond alors à l'identifiant d'une classe. Dans le cas de problème de régressions, il est possible d'avoir des valeurs continue. Les labels sont contenus dans un tableau à une dimension, sauf rares cas où ils peuvent être dans le vecteur de paramètres (classification multi-objective).

Remarque 16 - Est-ce une limitation ?

Les données tabulaires peuvent répondre à beaucoup de problèmes pratiques, mais cette forme de représentation des données ne sont pas adéquates dans tous les problèmes. Par exemple, on peut avoir des données sous forme structurées (des séquences, des graphes) qui ne vont pas facilement se transformer sous forme tabulaire.
Des méthodes spécifiques sont parfois disponibles pour ce type de données, mais ce n'est pas l'objet de la librairie **sklearn**.

Pour **sklearn** les données peuvent être "mangées" sous deux formes usuelles : des tableaux **Numpy** ou **Pandas**. D'un point de vue de la programmation, le travail principal consiste souvent à transformer les données brutes en entrée sous la forme d'un tableau qui sera exploitable par les algorithmes de **sklearn**.

En pratique, pour des images satellites, les tableaux de données pourrait assez naturellement être construit sous une forme tabulaire en prenant chaque pixel comme enregistrement avec les valeurs des bandes comme attributs. Le travail principal à réaliser est donc de transformer une source de données sous une forme tableau de données **pandas** (par exemple).
L'exemple ci-dessous illustre l'utilisation de la GDAL pour lire une image contenant 5 bandes, et pour laquelle on constitue un jeu de données avec uniquement 3 de ses 5 bandes. Dans le cas de cet exemple, on commence par créer le jeu de données sous la forme d'un liste de liste (structure de données Python), puis cette structure de données est transformée sous la forme d'un **DataFrame** de la librairie **pandas**. Ce jeu de données est prêt pour être utilisé par les algorithmes d'apprentissage de **sklearn**.

```
import gdal
from gdalconst import *
import pandas as pd

dataset = gdal.Open('zone1.tif', GA_ReadOnly)

#Recuperation des donnees sous la forme de matrices
bands=[]
```

2. Cela dépend de la capacité d'un classifieur.

```
for i in range(dataset.RasterCount):
    bands.append( dataset.GetRasterBand(i+1).ReadAsArray() )

#Creation d'un dataframe panda : version lente !
cols=['b1','b3','b5'] # nom des bandes pour nommer les colonnes
data=[]
for posi in range(int(dataset.RasterYSize)):
  for posj in range(int(dataset.RasterXSize)):
        data.append( [bands[0][posi][posj], bands[2][posi][posj], bands[4][posi][posj] ] )

df=pd.DataFrame(data, columns=cols)
```

L'exemple ci-dessus illustre le principe de la construction du jeux de données, mais on peut être beaucoup plus efficace en utilisant au mieux les fonctionnalités des modules et de Python. En particulier, le changement d'une liste de matrices à une matrice de vecteur, effectuée avec la double boucle, sera ici horriblement lente alors que **numpy** est faite pour cela. De même, plutôt que de faire la sélection a priori des bandes intéressantes ... le module **panda** gère très bien ce type de sélection de colonnes.
Dans le code ci-dessous, on reprend donc la même fonctionnalité, mais de manière plus élégante et "Pythonistique". Les deux codes construisent exactement le même tableau de données. Le second s'exécute en 0.01s, tandis que le premier met 1.48s ... il est donc à peu près 150 fois plus lent [3] ! De plus, le second code est plus flexible : il va fonctionner de manière identique pour quasiment toutes vos images, et vous n'aurez qu'à sélectionner les bandes qui vous intéressent.

```
1  import gdal
2  from gdalconst import *
3  import numpy as np
4  import pandas as pd
5
6  dataset = gdal.Open('zone1.tif', GA_ReadOnly)
7
8  #Recuperation des donnees sous la forme de matrices
9  bands=[]
10 for i in range(dataset.RasterCount):
11     bands.append( dataset.GetRasterBand(i+1).ReadAsArray() )
12
13 # nom des bandes pour nommer les colonnes
14 cols=['b'+str(i) for i in range(dataset.RasterCount)]
15
16 #constitution du jeux de données
17 data = np.reshape(np.dstack(bands), (dataset.RasterXSize*dataset.RasterYSize,dataset.RasterCount))
18
19 df=pd.DataFrame(data, columns=cols)
20
21 df_sel = df[["b0","b2","b4"]]
```

Quelques explications du code :

- ligne 14, j'utilise une construction de vecteur avec une boucle à l'intérieur. En gros, je construits un vecteurs avec des éléments de la forme `'b'+str(i)` avec `i` qui va de 0 au nombre de bande !
- ligne 17 ... je fais deux opérations importante en 1 ligne
  - **np.dstack** : cette fonction permet de transformer le vecteur de matrice **bands** en une matrice de vecteurs : pour chaque pixel (i, j), j'ai maintenant un vecteur de dimension 5 (le nombre de bande de cette image)
  - **np.reshape** : cette fonction réorganise les dimensions de mon tableau de données ! Plutôt que de le voir comme une matrice de pixel (donc, à trois dimensions), j'aplatis pour n'avoir plus qu'une liste de pixels. Cette nouvelle matrice aura donc plus que deux dimensions : **dataset.RasterCount** colonnes et **dataset.RasterXSize*dataset.RasterYSize** lignes !
- ligne 19 : je créé un *dataframe* avec toutes les données de l'image ! Attention néanmoins ... si vous avez vraiment des grosses image, réifier toute l'image peu poser problème.
- ligne 21 : je sélectionne uniquement les bandes qui m'intéressent

1.1.2 Deux phases : apprentissage et prédiction

En apprentissage automatique, il faut clairement identifier deux tâches :

- l'apprentissage : qui consiste à construire un modèle à partir de données (jeu de données d'entraînement $\mathcal{D}$)
- la prédiction : qui consiste à utiliser un modèle sur de nouvelles données.

3. Il faut noter ici, qu'il ne s'agit que du choix approprié de l'utilisation d'un langage de programmation qui permet d'améliorer l'efficacité u code, et aucunement un changement de complexité dans la manière d'aborder le problème.

L'objet au centre de ces deux tâches est le "modèle". On appelle ces modèles plus généralement des classifieurs (ou parfois prédicteurs, quand la notion temporelle rentre en jeu). Un classifieur est un objet complexe, parce qu'on peut le voir comme la composition :

- d'une représentation (interne) des "règles" qui régissent les données,
- une méthode (ou algorithme) de prédiction (ou de décision) qui déterminer comment utiler la représentation interne sur de nouvelles données pour déterminer la classe de cette données
- une méthode (ou algorithme) d'apprentissage qui déterminer comment construire la représentation interne à partir des données.

Chaque type de classifieur correspond à un choix pour ces différents éléments. Ces choix donnent des caractéristiques différentes à ces classifieurs :

- ils sont plus ou moins faciles à apprendre (nécessitent plus de temps ou plus d'exemples)
- ils ont des capacités de séparabilité plus ou moins riche (voir Section 1.2.1 pour l'apprentissage supervisé et 1.5.2 pour l'apprentissage non-supervisé)
- ils sont plus où moins faciles à paramétrer / sensibles aux paramétrages

Voici trois exemples d'algorithmes classiques :

- dans le cas d'un algorithm k-NN,
  - la représentation est une copie du jeu de données d'entraînement $\mathcal{D}$.
  - la méthode de prédiction d'un nouvel exemple X consiste à faire une moyenne des k valeurs les plus proches dans $\mathcal{D}$
  - la méthode d'apprentissage ... elle ne fait rien (on parle d'apprentissage *Lazy*)
  - $\Rightarrow$ ce classifieur est très simple à construire, mais risque d'être peu performants pour des données de grande dimension
- dans le cas des arbres de décision
  - la représentation est un arbre de décision,
  - la méthode d'apprentissage est un algorithme, tel que CN2, J48 ou Ripper qui se basent sur des critères statistiques pour construire de manière heuristique un arbre
  - la méthode de décision consiste à "parcourir" l'arbre avec le nouvel exemple
  - $\Rightarrow$ l'avantage de ce classifieur est d'être facilement interprétable, mais ses capacités de séparation sont limitées
- dans le cas des réseaux de neurones (non-profond)
  - la représentation est une fonction qui est une composition de combinaisons linéaires,
  - l'apprentissage est une procédure de descente de gradient (algorithme d'optimisation) permettant d'apprendre les coefficients des combinaisons linéaires
  - la décision est faite par l'application de la fonction sur de nouvelles données
  - $\Rightarrow$ l'avantage de ce classifieur est de théoriquement approximer n'importe quel fonction, mais cela nécessite beaucoup de données et peut être très long

Dans les exemples, on constate que les classifieurs sont également définis par des paramètres qui précisent une partie de leur structure. On parle alors d'**hyper-paramètres**. La valeur de k et le choix de la distance pour le k-NN, le nombre de couche d'un réseau de neurones, le critère statistique (gini, entropie) pour l'arbre de décision, sont des exemples d'hyper-paramètres.

Dans la librairie **sklearn**, ces modèles correspondent à des classes Python. Chaque classifieur dispose alors de deux fonctions :

- **fit(X,y)** (ou **fit(X)** dans le cas non-supervisé) est la fonction qui apprend la représentation à partir de données d'entrée (**X** est la matrice de données et **y** est le vecteur des classes, dans le cas d'un apprentissage supervisé)
- **predict(X)** (parfois appelée **transform**) est la fonction qui donner une décision de classe pour des données dans le vecteur **X**

On retrouve, par exemple, la classe **KMeans** qui va représenter un classifieur (non-supervisé) correspondant à la méthode des K-Means.
L'exemple ci-dessous poursuit le listing précédent qui a permis de construire le jeu de données **df** (sous la forme d'un data frame)

```
# import de la methode KMeans de sklearn
from sklearn.cluster import KMeans

# creation d'un objet KMeans, avec ici 4 classes
kmeans = KMeans( n_clusters=4 )

# apprentissage du KMeans a partir des donnees
kmeans.fit( df )
```

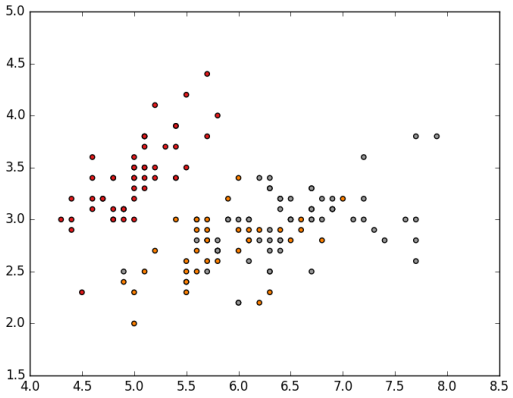


FIGURE V.1 – Illustration des données Iris

```
# utilisation du classifieur sur les memes donnees (!)
decision_array = kmeans.predict( df )

# affichage du resultat comme une image
from matplotlib import pyplot as plt
decision_image = decision_array.reshape( (dataset.RasterYSize,dataset.RasterXSize) )
plt.imshow(decision_image)
```

Remarque 17 - Sauvegarde de l'image résultante

Dans l'exemple ci-dessus, l'image de la décision, **decision_image**, est affiché à l'écran mais elle peut bien évidemment être sauvegardée comme une nouvelle image raster spatialisée. Un exemple complet d'une telle transformation est illustré dans la Section 1.2.2

1.2 Apprentissage supervisé

L'objectif de cette section est d'introduire quelques principes sur l'apprentissage automatique supervisé : les différents classifieurs, leurs limites et les bonnes pratiques (évaluation, validation croisée, etc).

1.2.1 Classification des Iris : un problème de séparabilité des classes

L'objectif de cette section est d'illustrer le problème d'apprentissage supervisé comme un problème de séparation de classes. On utilise pour cela un exemple jouet très classique qui vise à discriminer des catégories iris (fleurs) en fonction de leurs caractéristiques "géométriques" (4 caractéristiques, 3 classes). La librairie **sklearn** dispose en interne de ce jeux de données, il suffit donc d'effectuer les lignes suivantes pour charger et afficher le jeu de données

```
## Chargement et affichage des données
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data[:, [0,1]] # deux premiere caracteristiques
y = iris.target # les classes

plt.figure(2, figsize=(8, 6))
plt.clf()
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1, edgecolor='k')
```

Le graphique V.1 illustre les deux premières composantes du jeu de données iris.
L'objectif d'un classifieur est de tracer des frontières entre les classes (points de même couleurs). Mais, d'une part, on est contraint sur la forme des frontières qui peuvent être tracées (uniquement des droites, ou des cercles, par exemple), de plus, on est confronter à des problèmes de généralisation des exemples. Si on trace des frontières individuellement autour de chacun des points, cela ne représentera aucun intérêt (on parle de sur-apprentissage) dans la mesure où ces frontières seront très peu utiles lorsqu'un nouvel

exemple se présentera ! Le problème de l'apprentissage automatique supervisée vise donc à identifier des frontières qui soient "apprenables" et qui soient "régularisées" pour éviter le sur-apprentissage.
Sur l'image, on se rend compte qu'il va être relativement simple de tracer une ligne droite pour séparer les fleurs de la classe rouge, mais les deux autres classes sont très mixée ! Il faut penser que ce n'est ici qu'une représentation partielle des données (qui sont réellement en dimension 4), néanmoins, ce mélange des deux classes est bien connu pour ce jeu de données.

Construire un classifieur ... Commençons par construire un classifieur dans le but de concrétiser la notion de frontière de séparation. La construction du classifieur se fait très simplement en utilisant la librairie **sklearn** Nous allons commencer par construire un arbre de décision par l'ajout du code suivant :

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(min_samples_leaf=7)
clf.fit(X, y)
```

On complète ensuite ce code par la visualisation des frontières entre les classes. Ce code est fourni ci-dessous. Il consiste à construire une grille (**mesh**) qui recouvre toute la zone de l'image et de faire une prédiction (avec **clf.predict**) pour chacun des points de cette grille. Des surfaces sont ensuite induites par le calcul des contours.

```
## Affichage des bordures de classe
import numpy as np
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))
plt.tight_layout(h_pad=0.5, w_pad=0.5, pad=2.5)

Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)

# Affichage des exemples
for i, color in zip(range(3), "ryb"):
    idx = np.where(y == i)
    plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i], cmap=plt.cm.RdYlBu, edgecolor='black', s
        =15)
```

Le programme peut être téléchargé ici.

Ce qui est intéressant dans cet exemple, c'est de voir que l'arbre de décision a une caractéristique très typique pour ces séparations : seules des droites orthogonales aux axes sont effectivement possibles.

Exercice 29 (À vous de jouer) *Vous allez maintenant expérimenter un peu les différentes possibilités :*

- *Commencer par exécuter le même code en changeant les 2 caractéristiques qui servent à l'apprentissage (changer [0,1] par [0,3], [2,3])*
- *Tester ensuite d'autres algorithmes d'apprentissage (et observer la forme des séparations)*

1. *Changer votre classifieur à base d'arbre de décision par un classifieur de type k-NN (il suffit de l'importer et de changer la ligne correspondant à la définition de la variable* **clf** *:*

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier()
```

2. *Faire de même pour les classifieurs de type RandomForest, SVM (noyau RBF ou linéaire), GaussianNB dont le détail des imports est le suivant :*

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC, LinearSVC
```

Question a) Comparer les résultats des différents classifieurs. Lesquels vous semblent les plus appropriés pour ces données ?

Mesurer les performances ... À l'issue de cette première expérimentation, il est intéressant d'être en mesure de quantifier la qualité des différents classifieur pour être en mesure de choisir celui qui sera le plus approprié à nos données.
Le premier concept important est qu'il faut toujours évaluer les classifieurs en séparant les jeux de données d'apprentissage en utilisant un *train set* et un *test set* : le premier est utiliser pour construire le modèle et le second est utilisé pour tester.

Dans l'exemple ci-dessous, on illustre l'utilisation de la fonction **train_test_split** qui réalise la séparation aléatoire (uniforme) entre les deux jeux d'exemples. La séparation peut être assez difficile à mettre en oeuvre lorsque les classes sont déséquilibrées. On parle alors de séparation stratifiée (ce qui n'est pas le cas ici).

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn. cross_validation import train_test_split

iris = load_iris ()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split ( iris.data, iris.target, test_size =0.4, random_state=0 )

clf = DecisionTreeClassifier ()
clf.fit (X_train, y_train )
score = clf.score(X_test, y_test ) #calcul de la précision de la prédiction (accuracy)
print ("Accuracy: "+str(score))
```

Un programme "augmenté" peut être téléchargé ici.

Dans la source à télécharger, vous trouverez également le code pour utiliser d'autres mesures que la précision (*p.ex.* rappel, F1) et illustre également comment automatiser des traitements sur une **liste de classifieurs**. Cette dernière manipulation est rendu très simple par la librairie **sklearn**

Répéter les tests ... Bon, séparer le jeux de test et d'apprentissage est une bonne chose, mais on doit faire mieux ! Le classifieur peut être bon "par hasard" grâce à un choix adéquat des exemples. Pour donner plus de robustesses à nos mesures, il est alors indispensable de passer par de la validation croisée. Ce schéma d'évaluation permet de calculer un score moyen (**et sa variance**) sur plusieurs jeux de données tirés aléatoirement à partir du jeux de données d'apprentissage.
Le code ci-dessous illustre comment mener une validation croisée :

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn. cross_validation import cross_val_score

iris = load_iris ()
X = iris.data
y = iris.target

clf = DecisionTreeClassifier ()
scores = cross_val_score ( clf, X, y, cv=5, scoring='f1_macro')

print ( "Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

Le programme peut être téléchargé ici.

> **Exercice 30 (Meilleur modèle)**
>
> *Question a)* En reprenant les deux exemples précédents, indiquer quel classifieur est le meilleur pour les Iris selon le critère de précision (*accuracy*) ?

Paramétrer les classifieurs ... Bien ! ! Mais jusque là, on n'a pas vraiment "optimisé" vos classifieurs ! ! On a utilisé les (hyper-)paramètres par défaut de ceux-ci ! Mais il est évident que le changement de leurs (hyper-)paramètres donnent des résultats très différents. Il faut alors être en mesure de choisir ces (hyper-)paramètres pour votre jeu de données.
La méthode pousse l'idée de la validation croisée un peu plus loin, puisqu'il est carrément possible de demander à **sklearn** de choisir le meilleur modèle (classifieur + hyper-paramètres) automatiquement à partir du jeu d'apprentissage.

Le principe consiste à définir une grille de paramètres que l'on souhaite explorer, ensuite, **sklearn** va se charger d'automatiser la réalisation des validations croisées pour chacune des configurations de la grille et décider quel est le meilleur selon la mesure de performance. Ceci se fait simplement au moyen d'un **GridSearchCV** :

```
from sklearn.datasets import load_iris
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
```

```
import numpy as np

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=DeprecationWarning)

iris = load_iris()
X = iris.data
y = iris.target

X, X_test, y, y_test = train_test_split( iris.data, iris.target, test_size=0.2, random_state=0 )

Cs = np.logspace(-4, 2, 10)
kernels = ['rbf','linear', 'sigmoid']
clf = GridSearchCV(SVC(), dict(C=Cs,kernel=kernels))

clf.fit(X, y)
print( "Meilleure accuracy: %0.2f"%clf.best_score_ )
print( "Meilleurs paramètres %0.2f, %s"% (clf.best_estimator_.C,clf.best_estimator_.kernel) )
print( "Accuracry finale %0.2f"%clf.score(X_test, y_test) )
```

Le programme peut être téléchargé ici.

La logique de ce programme est de commencer par mettre de côté un jeu d'évaluation final (10% des données), puis de lancer le recherche des paramètres optimaux (ici, en faisant varier le paramètre `C` et le paramètre `kernel`). Une fois ce choix réalisé, alors on peut tester, dans la dernière ligne, sur des données totalement neutres.

Remarque 18 - Pas toujours des paramètres communs

Lorsque vous souhaitez choisir le meilleur modèle en "optimisant" le choix du modèle et celui de ces hyper-paramètres, il est difficile d'automatiser cette tâche. En effet, chaque type de classifieur à ses propres hyper-paramètres. Il faut alors procéder un peu à la main ...

1.2.2 Une première chaîne de traitements de classification d'images

L'exemple ci-dessous illustre la construction d'un modèle à partir d'une image. Dans cet exemple, on utilise un ensemble réduit de **pixels tirés aléatoirement** dans l'image pour créer le jeu d'apprentissage.

```
import sys
import gdal
from gdalconst import GA_ReadOnly
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.cross_validation import cross_val_score

dataset = gdal.Open('zone1.tif', GA_ReadOnly)
if dataset is None:
    print('Could not open image')
    sys.exit(1)

datasetcl = gdal.Open('zone1_cl.tif', GA_ReadOnly)
if datasetcl is None:
    print('Could not open image')
    sys.exit(1)

#recuperation des donnees sous la forme de matrices
bands=[]
for i in range(5):
    bands.append( dataset.GetRasterBand(i+1).ReadAsArray() )
#idem pour l'image classée
bands.append( datasetcl.GetRasterBand(1).ReadAsArray() )

#creation d'un dataframe panda par tirage aléatoire d'échantillons
cols=['b1','b2','b3','b4','b5','cl']
data = np.reshape(np.dstack(bands), (dataset.RasterXSize * dataset.RasterYSize, dataset.RasterCount+1))

nbpixels = np.shape(data)[0]

nb_ech=1000 #nombre d'échantillons
pos_rand=np.random.randint(0,nbpixels, nb_ech)
```

```
sampled_data=data[pos_rand]

df=pd.DataFrame(sampled_data, columns=cols)

clf = DecisionTreeClassifier()
acc = cross_val_score(clf, df[list(['b1','b2','b3','b4','b5'])].values, df['cl'].values)
print("mean accuracy: %0.2f (+/- %0.2f)"%(acc.mean(), acc.var()*2))

#construction du model
clf.fit(df[list(['b1','b2','b3','b4','b5'])].values, df['cl'].values)

########## Construction d'une image classée ##########
ldata=[]
#attention, ici, il faut prendre uniquement les données de l'image satellite, pas la vérité terrain !!
ldata = np.reshape(np.dstack(bands[:-1]), (dataset.RasterXSize * dataset.RasterYSize, dataset.RasterCount))
df = pd.DataFrame(ldata, columns=['b1','b2','b3','b4','b5'])

array = clf.predict(df)
array = array.reshape((dataset.RasterYSize,dataset.RasterXSize))
plt.imshow(array)
```

Listing V.1 – Apprentissage supervisé

Le programme peut être téléchargé ici.

Exercice 31 (Identification d'un bon classifieur)

Question a) Tester le code du listing V.1 ci-dessus pour le comprendre

Question b) Modifier le code précédent pour tester l'influence du nombre d'exemples sur les performances du classifieur *Pour cela, vous pourrez répéter la construction du jeu de données et la validation croisée pour des différentes valeurs de nombre d'exemple (utiliser* **np.logspace***(1,5,10) qui génère un vecteur de 10 valeurs entre* 10^1 *et* 10^5*). Vous pourrez retenir les résultats dans une liste et les afficher ensuite.* correction.
Par la suite, vous fixerez une taille de jeu de données qui soit raisonnable en temps de calcul et qui donne des résultats "satisfaisants" (par exemple 5000 points).

Question c) Adapter le code précédent pour tester d'autres classifieurs. Lequel donne de meilleurs résultats pour cette image ?

Question d) Adapter le code pour faire enregistrer la matrice de l'image classée sous la forme d'un nouveau fichier au format GeoTiff.

- *vous considérerez que les valeurs de l'image classée sont au format* **gdal.GDT_Int32**
- *vous recopierez les informations relatives à la spatialisation des données (c.-à-d. SRS et* **geotransform***)*

correction

Question e) Modifier le code pour que l'image classée ne soit plus l'image originale, mais une nouvelle image (`zone2.tif`) par exemple.

Question f) (⋆)Créer une fonction **predict(filename, clf)** qui prend en paramètre le nom d'un fichier image et un classifieur et qui construise une nouvelle image GeoTiff avec le nom **filename**[:−4]+"_"+**type(clf).__name__**+"_cl.tif" qui sera l'image classée correction

Remarque 19 - Tirage aléatoire des exemples

La manière de tirer aléatoirement des exemples est une question difficile lorsqu'on construit un jeu d'exemples. L'échantillonnage peut introduire un biais dans le modèle appris. L'utilisation de validation croisée permet de limiter les biais d'apprentissage, ou au moins de les mesurer pour identifier la sensibilité de la méthode d'apprentissage à cette sélection.
Dans le cas de l'exemple ci-dessus, l'objectif était d'avoir moins de données pour limiter les temps de calculs et on n'a pas pris de soin particulier à cette construction du jeux d'exemples.

1.2.3 Utilisation de ROI vectorielle : version vectorielle

Dans de nombreuses situations, l'information terrain est enregistrée dans un format vectoriel. Chaque polygone (resp. point) correspond à une région (resp. localisation) pour laquelle une information terrain est disponible.

Dans cet exemple, on propose d'utiliser une image LANDSAT8 comportant 7 couches à une résolution de 30m (Aerosol, VIS, NIR, SWIR). On dispose également de fichiers qui correspondent à des relevés terrain : 1 fichier par classe. Les fichiers comportantent des multi-points.

```
1
2  # recuperation des donnees de l'image sous la forme de matrices
3  # image landsat8 avec 7 bandes
4  ds = gdal.Open('image/2298119ene2016recorteTT.tif', GA_ReadOnly)
5  bands=[]
6  for i in range(7):
7      bands.append( ds.GetRasterBand(i+1).ReadAsArray() )
8
9  geotransform = ds.GetGeoTransform()
10 originX = geotransform[0]
11 originY = geotransform[3]
12 pixelWidth = geotransform[1]
13 pixelHeight = geotransform[5]
14
15 X=[]
16 y=[]
17 driver = ogr.GetDriverByName('ESRI Shapefile')
18 for cl in ['A','B','C','D','E']:
19     inDS = driver.Open('train/'+cl+'.shp', 0)
20     inLayer = inDS.GetLayer()
21
22     for inFeature in inLayer:
23         #parcours d'un feature de type "multipoint"
24         for i in range(0, inFeature.GetGeometryRef().GetGeometryCount()):
25             geom = inFeature.GetGeometryRef().GetGeometryRef(i)
26             i = int((geom.GetX() - originX) / pixelWidth)
27             j = int((geom.GetY() - originY) / pixelHeight)
28
29             X.append( [bands[b][j][i] for b in range(7)] )
30             y.append( cl )
31
32 #transform label names into integers (required for saving images)
33 le = preprocessing.LabelEncoder()
34 le.fit(y)
35 y= le.transform(y)
36
37 ############### Construction de modele ####################
38 clf = SVC()
39 clf.fit(X, y)
40
41 ########### Construction d'une image classée ###########
42 ldata = np.reshape( np.dstack(bands), (ds.RasterYSize*ds.RasterXSize,7) )
43 array = clf.predict( ldata )
44 array = array.reshape( (ds.RasterYSize,ds.RasterXSize) )
45
46 # Sauvegarde en fichier
47 driver = gdal.GetDriverByName( "GTiff" )
48 dst_ds = driver.Create( "output.tif", ds.RasterXSize, ds.RasterYSize, 1, gdal.GDT_Int32 )
49 dst_ds.SetProjection( ds.GetGCPProjection() )
50 dst_ds.SetGeoTransform( ds.GetGeoTransform() )
51
52 dst_ds.GetRasterBand(1).WriteArray( array )
53
54 dst_ds = None
```

Le programme peut être téléchargé ici.

Ce programme est organisé de la sorte :

- Les lignes 3 à 12 chargent l'image avec ses caractéristiques géométriques,
- Les lignes 14 et 15 définissent les variables qui vont servir à faire l'apprentissage aux lignes 37-38.
- Les lignes 16 à 29 construisent le jeu de données d'apprentissage à partir des 5 classes (`A`, `B`, `C`, `D` et `E`). Pour chaque classe, un fichier au format `shp` est chargé (contenant des localisations des points d'exemple). Pour un fichier, toutes les points sont parcourus pour, lignes 24-29, aller chercher le pixel correspondant à la localisation du point du *shapefile*, et ajouter les caractéristiques des 7 bandes au jeu d'exemples avec la classe correspondante. La particularité de ces données vectorielles et de comporter des multi-points : les deux boucles l.21 et l.23 permettent de parcours tous les points de tous les multi-points.
- Les lignes 32-34 transforment le vecteurs de labels, qui contient des chaînes de caractères. Par habitude, je transforme ici ces valeurs en un ensemble de valeurs numériques (entiers). Ce type de données est plus facilement utilisé par les algorithmes d'apprentissage et permettra aussi d'enre-

gistrer les données dans un *GeoTiff* avec des données entières.
— Les lignes 37 et 38 apprennent un classifieur SVM (sans configuration particulière, ni sans évaluation de sa qualité). La fin du programme utilise ce classifieur pour classer l'intégralité de leur image.
— Les lignes 41 à 46 construisent la matrice des 7 bandes de l'image (pour tous les pixels). Attention, ici il faut construire la matrice les pixels dans le bon ordre pour que la reconstruction de la ligne 49 se fasse bien.
— La ligne 48 utilise le classifieur appris sur les données d'exemple pour classer chacun des pixels.
— Les lignes 51-59 enregistrent la matrice de l'image prédite sous la forme d'une image *GeoTiff* avec la recopie des caractéristiques géométrique de l'image d'origine.

! Attention ! - Temps d'exécution

L'image est assez grande et l'exécution de ce code peut être relativement long. La partie la plus longue du code est celle de la prédiction (sur tous les pixels de l'image). Le temps d'apprentissage est finalement relativement négligeable vu le nombre d'exemples à disposition.

1.2.4 Utilisation de ROI vectoriels : rasterization d'une couche

On reprend ici le même contexte que la section précédente. Cette fois-ci on dispose d'une couche vectorielle de polygones pour les ROI (*Regions of Interest*). Une première approche pourrait consister à regarder si un pixel de l'image est à l'intérieur d'un polygone, mais ceci pourrait être inutilement très long à réaliser (plein de pixels n'appartiennent à aucun ROI).
La solution usuelle, illustré dans le code ci-dessous, consiste à effectuer une rasterisation de l'image vectorielle, c'est à dire à projeter les informations d'une couche vectorielle sur une grille correspondant à cette de l'image satellite.

```
1  # recuperation des donnees de l'image sous la forme de matrices
2  # image landsat8 avec 7 bandes
3  ds = gdal.Open('image/2298119ene2016recorteTT.tif', GA_ReadOnly)
4  bands=[]
5  for i in range(7):
6      bands.append( ds.GetRasterBand(i+1).ReadAsArray() )
7  data = np.dstack(bands)
8
9  ################ Definition des ROI ########################
10 driver = ogr.GetDriverByName('ESRI Shapefile')
11 inDS = driver.Open('train/polygons.shp', 0)
12 layer = inDS.GetLayer()
13
14 projection = ds.GetGCPProjection()
15 geotransform = ds.GetGeoTransform()
16
17 driver = gdal.GetDriverByName('MEM') # In memory dataset
18 target_ds = driver.Create('', ds.RasterXSize, ds.RasterYSize, 1, gdal.GDT_Float32)
19 target_ds.SetGeoTransform(geotransform)
20 target_ds.SetProjection( projection )
21 gdal.RasterizeLayer(target_ds, [1], layer, burn_values=[0], options=["ATTRIBUTE=Herbometre"])
22 labeled_pixels = target_ds.GetRasterBand(1).ReadAsArray()
23
24 ################ Construction de modele ######################
25
26 train_pxl = np.nonzero(labeled_pixels)
27 X=data[train_pxl]
28 y=labeled_pixels[ train_pxl ]
29
30 clf = SVC()
31 clf.fit(X, y)
32
33 ############ Construction d'une image classée ############
34 ldata = np.reshape( data, (ds.RasterYSize*ds.RasterXSize,7) )
35 array = clf.predict( ldata )
36 array = array.reshape( (ds.RasterYSize,ds.RasterXSize) )
37
38 # Sauvegarde en fichier
39 driver = gdal.GetDriverByName( "GTiff" )
40 dst_ds = driver.Create( "output.tif", ds.RasterXSize, ds.RasterYSize, 1, gdal.GDT_Int32 )
41 dst_ds.SetProjection( projection )
42 dst_ds.SetGeoTransform( geotransform )
```

```
43
44 dst_ds.GetRasterBand(1).WriteArray( array )
45 dst_ds = None
```

Le programme peut être téléchargé ici.

Quelques commentaires sur ce code :

- les lignes 3 à 6 permettent de charger l'image satellite LANDSAT avec 7 bandes. La fonction de la ligne 7 transforme la pile d'image en une image de vecteurs de dimension 7 (cette structuration est plus simple à utiliser par la suite).
- les lignes 10-12 chargent l'image vectorielle (faite de polygones) : ici, les polygones de toutes les classes sont dans une unique couche et le numéro de la classe est donné par l'attribut `label`.
- les lignes 14 à 22 réalisent la rasterisation de l'image vectorielle. Pour cela, un *raster virtuel* est créé (création avec un driver `MEM`), avec les mêmes caractéristiques géométriques que l'image satellite d'origine. On fait ensuite appel à la fonction de rasterisation de GDAL : le premier argument est le raster virtuel à remplir, `[1]` [4] désigne la liste des bandes à remplir, **layer** est la couche vectorielle à projeter, **burn_values** spécifie le vecteur de valeur qui doit être utilisé si il n'y a pas de données (hors polygone) et **options** permet ici d'indiquer quel attribut de la couche doit être utilisé pour donner une valeur au pixel du raster. Finalement, la ligne 22 récupère les données qui ont été transformées.
- Les lignes 26-28 construisent le jeu d'exemples en ne récupérant les données que pour les coordonnées non-nulles de la matrice **labeled_pixels**
- La fin du programme est similaire à l'exemple précédent : le classifieur est construit, puis utilisé pour faire la prédiction sur l'ensemble des pixels de l'image

1.3 Régression

Dans la section précédente, la tâche d'apprentissage visait à prédire la classe de chacun des pixels. Mais dans certain cas, ce n'est pas une classification qui est nécessaire, mais une régression : la variable à prédire est une variable continue. Par exemple, dans l'exemple ci-dessous, l'objectif est de prédire la hauteur d'herbe à partir d'image satellite.
Les méthodes de régressions fonctionnent selon le même principe que les méthodes de classification supervisée : on fournit un jeu d'exemples qui associe des données à la valeur à prédire (une valeur numérique), l'algorithme d'apprentissage construit un modèle prédictif (fonction **fit**) et une valeur est prédite par ce modèle lorsqu'on donne d'autres exemples (fonction **predict**).
La différence principale réside la manière dont on évalue la qualité d'un modèle prédictif. Contrairement à la classification où on somme des nombres de mauvaise classification, dans le cas de la régression, on somme directement les erreurs numériques. Les indices de performances sont classiquement :

- la RMSE : Root Mean Square Error
- le coeffiscient de détermination ou R^2
- la mesure de variance expliquée

Dans l'exemple ci-dessous, on reprend le principe de l'exemple de la classification d'une image à partir de label dans une couche vectorielle en faisant la rasterisation de cette couche. Ici, lors de la rasterisation, on récupère la valeur (un flottant) qui correspond à la hauteur d'herbe d'une parcelle. Ensuite, tous les pixels qui ont une hauteur d'herbe supérieure à 0 servent d'exemple de classification.

```
1  import numpy as np
2
3  import ogr, gdal
4  from gdalconst import GA_ReadOnly
5
6  from sklearn.cross_validation import train_test_split
7  from sklearn.linear_model import LinearRegression, Lasso, ElasticNet, Ridge, BayesianRidge
8  from sklearn.svm import SVR
9  from sklearn.metrics import mean_squared_error, r2_score
10
11 # recuperation des donnees de l'image sous la forme de matrices
12 # image landsat8 avec 7 bandes
13
14 ds = gdal.Open("20170409_scaling_stack.tif", GA_ReadOnly)
15 bands=[]
16 for i in range(10):
17     bands.append( ds.GetRasterBand(i+1).ReadAsArray() )
18 data = np.dstack(bands)
19
20 ############### Definition des ROI ######################
21 driver = ogr.GetDriverByName('ESRI Shapefile')
22 inDS = driver.Open('Parcelles_Thorigne_2017.shp', GA_ReadOnly)
```

4. il est tout à fait possible de récupérer plusieurs information de la couche vectorielle d'un seul coup.

```
layer = inDS.GetLayer()

projection = ds.GetGCPProjection()
geotransform = ds.GetGeoTransform()

driver = gdal.GetDriverByName('MEM') # In memory dataset
target_ds = driver.Create('', ds.RasterXSize, ds.RasterYSize, 1, gdal.GDT_Float32)
target_ds.SetGeoTransform(geotransform)
target_ds.SetProjection( projection )
gdal.RasterizeLayer(target_ds, [1], layer, burn_values=[0], options=["ATTRIBUTE=Herbometre"])
labeled_pixels = target_ds.GetRasterBand(1).ReadAsArray()

############### Jeux de donnees labellise ######################
pxl = np.nonzero(labeled_pixels)
X=data[pxl]
y=labeled_pixels[pxl]

############### Apprentissage ######################
for classifier in LinearRegression, Ridge, Lasso, BayesianRidge, SVR:
    clf = classifier ()
    clf.fit(X, y)
    y_pred= clf.predict(X)

    #calcul de la RMSE
    train_error = mean_squared_error(y, y_pred)
    print("Erreur en apprentissage de "+str(classifier)+": "+str(train_error))
```

Le programme peut être téléchargé ici.

Exercice 32 *À vous de jouer Le programme calcule actuellement l'erreur en prédiction, il est néanmoins intéressant de procéder de la même manière que pour la classification, à savoir de séparer le jeu d'apprentissage du jeu de test pour avoir une évaluation plus juste*

Noter les meilleurs regresseurs

Modifier le script pour séparer les jeux de d'apprentissage et de test

Tester de nouveau pour trouver le meilleur regresseur. Est-ce le même que précédemment ?

Modifier le script pour avoir une validation croisée.

1.4 Deep learning

L'utilisation de méthodes de réseaux de neurones profonds, ou *deep learning*, passe par l'utilisation de librairies annexes. Dans la suite, on présente l'utilisation de la librairie **keras** qui s'interface bien avec **sklearn**. En complément, il faut savoir que **keras** peut interagir avec (au moins) deux librairies de réseaux de neurones profond : *TensorFlow* ou *Theano* (et il en existe d'autres tels que *kaffe* ou *CNTK*).
Le principe de fonctionnement de l'apprentissage profond réside dans la définition d'une structure de réseau et de faire appel à des méthodes d'optimisation pour optimiser les paramètres du réseaux.

- la structure du réseaux en structurée sous la forme de couches : une couche d'entrée permet se connecte aux descripteurs des exemples, puis est connectée à une couche qui réalise des opérations dont le résultat est distribué aux neurones de la couche suivante, etc . jusqu'à avoir une couche de sortie unique.
- la structure du réseau est plus ou moins complexe, mais la notion de "deep learning" sous entend de faire appel à de nombreuses couches !
- il existe également des couches de nature très différentes, avec des possibilités de configuration très très grandes !! Et, à peu près aucune méthodologie pour vous dire quelle couche utiliser dans telle ou telle situation !! Bref, c'est le règne du "essayer et tester" !!
- plus le réseaux est complexe ... et plus il a de paramètres à apprendre et donc ... plus il faut d'exemples pour les apprendre !! Dans le cas de réseaux de neurones complexes, on parle de plusieurs centaines de milliers d'exemples !! Et, comme il s'agit d'une méthode supervisée, il est indispensable d'avoir des exemples annotés !! La question de la constitution du jeux de données devient un enjeu en soit. Deux techniques sont classiquement utilisés : la technique de l'augmentation des données (il s'agit d'utiliser un exemple plusieurs fois en le bruitant légèrement. Pour une image, on donnera la même image, mais avec des recadrage, des rotations, etc. Cela permet d'avoir plus d'exemples et aussi d'avoir un apprentissage plus robuste)
- les temps de calculs sont conséquemment très importants, mais les méthodes d'apprentissage sont très itératives et prennent beaucoup de temps pour "converger". Ce sont donc des méthodes extrêmement coûteuses en temps et en ressources de calcul ! Par contre, tous ces algorithmes d'appren-

tissage peuvent être facilement parallélisés et donc utiliser au maximum vos ressources !

1.4.1 Un premier exemple

Dans cet exemple, nous allons mettre en place un réseaux de neurones pour faire de l'apprentissage à partir d'un jeu de données classique contenant des images de nombres.

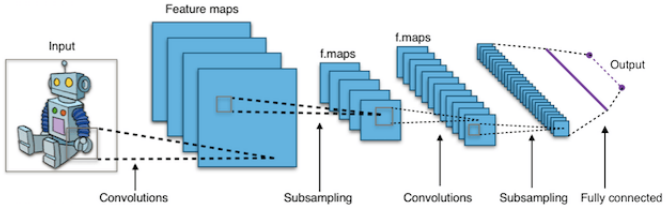
```
1  #Import des modules
2  from keras.models import Sequential
3  from keras.layers import Dense, Dropout, Flatten
4  from keras.layers import Convolution2D, MaxPooling2D
5  from keras.utils import np_utils
6  from keras.datasets import mnist
7  from keras import backend as K
8  K.set_image_dim_ordering('th')
9
10 # Chargement du jeu de données de test et d'apprentissage
11 (X_train, y_train), (X_test, y_test) = mnist.load_data()
12
13 # Prétraitement des données
14 X_train = X_train.reshape(X_train.shape[0], 1, 28, 28)
15 X_test = X_test.reshape(X_test.shape[0], 1, 28, 28)
16 X_train = X_train.astype('float32')
17 X_test = X_test.astype('float32')
18 X_train /= 255
19 X_test /= 255
20
21 # Changement des valeurs numériques en catégoriques
22 Y_train = np_utils.to_categorical(y_train, 10)
23 Y_test = np_utils.to_categorical(y_test, 10)
24
25 # Construction de la structure de réseaux en couche
26 model = Sequential()
27
28 model.add(Convolution2D(32, (3, 3), activation='relu', input_shape=(1,28,28)))
29 model.add(Convolution2D(34, (3, 3), activation='relu'))
30 model.add(MaxPooling2D(pool_size=(2,2)))
31 model.add(Dropout(0.25))
32
33 model.add(Flatten())
34 model.add(Dense(128, activation='relu'))
35 model.add(Dropout(0.5))
36 model.add(Dense(10, activation='softmax'))
37
38 # Compilation du modèle
39 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
40
41 # Apprentissage du modèle (epochs réduits)
42 model.fit(X_train, Y_train, batch_size=32, nb_epoch=1, verbose=1)
```

Le programme peut être téléchargé ici.

— Lignes 2 à 6, on importe les librairies utiles pour nos réseaux de neurones.
— Lignes 7-8, on définit une configuration par défaut de l'ordre des dimensions
— Ligne 11, on récupère les données du jeu de données d'imagette. Vous pouvez consulter une de ces images à l'aide du code ci-dessous

```
import matplotlib.pyplot as plt
plt.imshow(X_train[0])
```

— les lignes 14 à 23 sont des prétraitements sur le jeu de données pour le préparer à l'utilisation du modèle de réseaux de neurone. Il faut en particulier qu'il ait les bonnes dimension : on rajoute pour cela une dimension pour passer des images 2D (sans couleur) comme une image en 3D (avec une seule couche en fait ... mais il faut cette dimension pour la suite)
— Ligne 25 à 36 définissent les couches du réseau de neurones : ici, il s'agit d'un réseau convolutionnel assez classique illustrée par le schéma ci-dessous :



Chaque couche est configurée en donnant son type de couche et un ensemble d'hyper-paramètres.
— Ligne 39, le réseau est "compilé". C'est à dire, c'est le préparer pour l'apprentissage ... et aussi donner encore quelques valeurs à des paramètres essentiels.
— Ligne 42,

Une fois le modèle construit, il peut être utilisé pour prédire la classe de nouvelles imagettes, plus particulièrement celle de la base de test. Ci-dessous, on prédit la classe de la première imagette :

```
p = model.predict(np.reshape(X_test [0],(1,1,28,28) ))
```

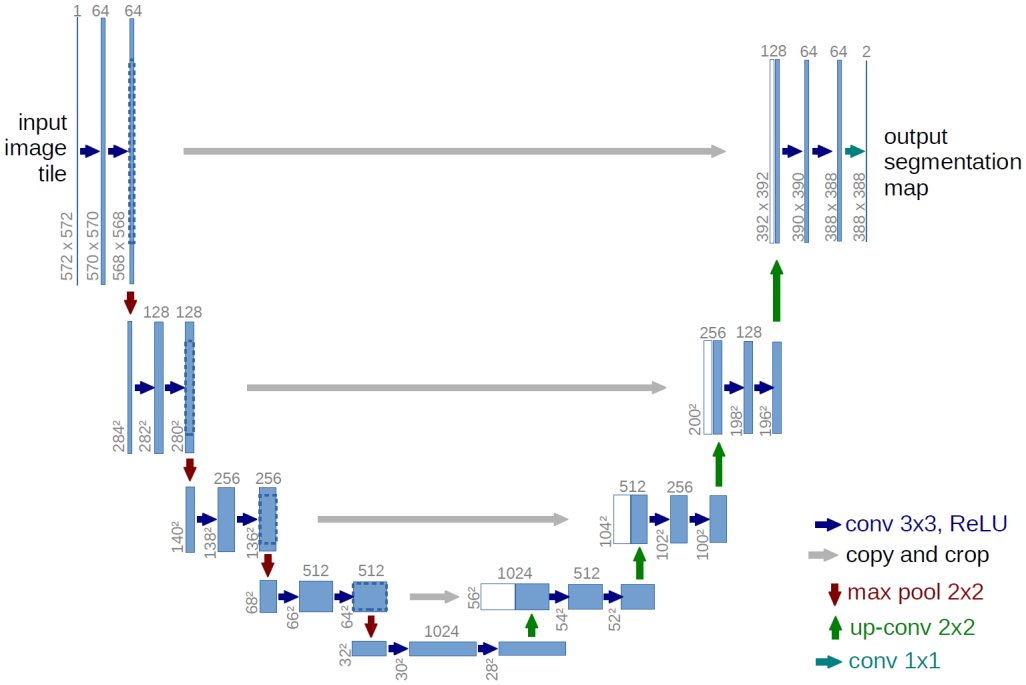
Le résultat de cette prédiction est un vecteur de valeurs :

```
array([[2.0884194e-09, 2.1285624e-07, 2.1206290e-06, 1.3422467e-06,
        4.8926569e-09, 4.1940504e-10, 1.0350737e-13, 9.9999440e-01,
        2.6516824e-09, 1.9325844e-06]], dtype=float32)
```

Dans ce cas, on en déduit une classification nette dans la classe 7.

1.4.2 Apprendre la classe d'un pixel par deep learning : cas des images hyper-spectrale

1.4.3 Segmentation d'une image réseau UNet



1.5 Apprentissage non-supervisé

L'apprentissage non-supervisé, aussi appelé *clustering*, consiste à construire un modèle sans avoir de données terrains. L'idée de l'apprentissage non-superviser est de regrouper les données similaires ensemble dans une classe, en construisant des classes qui sont les plus différentes les unes des autres. L'intérêt de cette méthode est double :

1. identifier une organisation structurelle des données qui révèle que les données ont des spécificités intrinsèques qui peuvent être exploitées
2. simplifier les données en réduisant des informations riches (données multi-bandes) en des données simplifiées (numéro de classe)

La seconde caractéristique utilisée sur des images satellites permet de simplifier une image en en donnant une représentation mono-couche cohérente.

1.5.1 Construction d'un modèle par *clustering*

La construction d'un modèle de *clustering* est finalement plus simple en programmation que le cas supervisé puisqu'il est uniquement utile de construire un vecteur de caractéristiques provenant de l'image, sans avoir à croiser avec des informations provenant d'une autre couche.
L'exemple ci-dessous illustre le clustering de l'image `zone1.tif`.

```python
import gdal
from gdalconst import *
from sklearn.cluster import KMeans

#### Chargement de l'image
dataset = gdal.Open('zone1.tif', GA_ReadOnly)
bands=[]
for i in range(5):
    bands.append( dataset.GetRasterBand(i+1).ReadAsArray() )

X=[]
for i in range(int(dataset.RasterYSize)):
    for j in range(int(dataset.RasterXSize)):
        X.append( [bands[b][i][j] for b in range(5)] )

#### clustering
clf=KMeans(n_clusters=3)
y = clf.fit_predict (X)

#### enregistrement de l'image résultante
array = y.reshape( (dataset.RasterYSize,dataset.RasterXSize) )

driver = gdal.GetDriverByName( "GTiff" )
dst_ds = driver.Create( "output.tif", dataset.RasterXSize, dataset.RasterYSize, 1, gdal.GDT_Int32 )

dst_ds.SetProjection( dataset.GetGCPProjection() )
dst_ds.SetGeoTransform( dataset.GetGeoTransform() )

dst_ds.GetRasterBand(1).WriteArray( array )
dst_ds = None
```

Le programme peut être téléchargé ici.

Dans cet exemple, on illustre l'utilisation de l'algorithme K-Means avec un nombre de classe de 3. La fonction **fit_predict** combine les fonctions **fit** et **predict**. Cette fonction construit donc le modèle associé au KMeans et retourne le vecteur correspondant aux valeurs des classes attribués aux exemples.

Exercice 33 (Comparaison visuelle des clusterings)

Question a) Modifier le code pour afficher l'image à l'aide de la librairie **matplotlib**

Question b) Tester différentes valeurs de k avec l'algorithme KMeans

Question c) Tester d'autres algorithmes, par exemple DBSCAN et MeanShit. Voir `http://scikit-learn.org/stable/modules/clustering.html` pour plus d'information sur les algorithmes.

```python
clf = cluster.DBSCAN(eps=.3)
bandwidth = cluster.estimate_bandwidth(X, quantile=.3)
clf = cluster.MeanShift(bandwidth=bandwidth, bin_seeding=True)
```

Remarque 20 - Choix de la mesure

Un paramètre clé lors de l'utilisation d'un algorithme de clustering est la mesure qui permet d'effectuer le regroupement. Dans le cas des KMeans, il s'agit d'une distance Euclidienne. Le choix de cette mesure influence fortement la notion de voisinage et donc les résultats du clustering. Dans l'idéal, il serait intéressant d'étudier l'influence du choix de la mesure de distance ... mais on sort du contexte de ce cours.

Remarque 21 - Mesure ou distance

L'algorithme KMeans requiert d'avoir une mesure qui a les propriétés d'une distance. En particulier, il est nécessaire d'avoir la propriété d'inégalité triangulaire pour que l'algorithme "fonctionne" et que les résultats soient significatifs. C'est un problème lorsqu'on veut comparer des objets complexes tels que des séries temporelles.
Les algorithmes de clustering agglomératifs requiert d'avoir la propriété d'une ultra-métrique ... un peu plus faible que la notion de distance.
D'autres algorithmes s'affranchissent de toute contrainte, tel que l'algorithme Affinity Propagation, mais leurs performances en temps de calcul ne sont pas aussi bonnes.

1.5.2 Comparaison des classifieurs

De la même manière que pour l'apprentissage supervisé il est intéressant d'évaluer les résultats d'un clustering. Le soucis, c'est qu'on ne dispose pas de *gold standard* pour savoir si la classification est correcte.
Dans le cas de l'exemple précédent, où on dispose effectivement d'une image à laquelle se comparer. On peut alors calculer une matrice de confusion. La matrice de confusion permet d'évaluer, classe par classe, quels sont les erreurs qui sont comises. Cette information est plus riche que simplement une valeur tels qu'une précision de classification, puisqu'elle permet de savoir pour quelles classes le classifieur est bon ou non.
L'exemple ci-dessous illustre le calcul d'une matrice de confusion à la suite de l'exemple précédent.

```
from sklearn.metrics import confusion_matrix

datasetcl = gdal.Open('zone1_cl.tif', GA_ReadOnly)
bandcl = datasetcl.GetRasterBand(1).ReadAsArray()
bandcl = bandcl.reshape( datasetcl.RasterYSize*datasetcl.RasterXSize )

conf = confusion_matrix(array, bandcl)
print(conf)
```

Cette méthode de comparaison peut être utilisée pour comparer n'importe quel résultat de classification, par exemple des résultats obtenus avec différentes valeurs de k.

CHAPITRE

VI Automatisation des traitements

1 Fonctionnalité du "système"

1.1 Modules de Python

Voici quelques modules de python classiques qui nous seront utile pour la réalisation :

- **os** : ce module permet de gérer les fichiers et les répertoires de votre ordinateur (par exemple, listing, création, suppression des répertoires et fichiers), mais également de faire des appels systèmes.
- **glob** : ce module propose une fonction permettant de lister les fichiers d'un répertoire en utilisant des expressions régulières. On peut ainsi facilement lister tous les fichiers d'un type donné.
- **urllib2** : ce module propose une fonction **urlopen** qui ouvre une URL. Cette fonction sera utilisée pour lancer la requête sur le web pour récupérer son contenu (par exemple, une image).

1.2 Manipulation des noms chemins

En Python, les chemins sont tout simplement des objets de type **str** (chaînes de caractères). Il est donc possible de les manipuler avec les opérations classiques des chaînes, bien que cela ne soit pas toujours conseiller (par exemple, pour coller deux chemins, il vaut mieux utiliser **join**() que l'opérateur +). Toutes les fonctions de base pour manipuler les noms de chemin se trouvent dans le module **os.path**.

- **os.path.abspath(path)** : Retourne le chemin absolu du chemin relatif path donné en argument. Il permet aussi de simplifier les éventuelles répétitions de séparateurs.
- **os.path.split(path)** : Sépare **path** en deux parties à l'endroit du dernier séparateur : la deuxième partie contient le fichier ou le dossier final, et la première contient tout le reste (c'est à dire le chemin pour y accéder).
- **os.path.basename(path)** : Retourne le dernier élément du chemin.
- **os.path.dirname(path)** : Retourne le chemin pour accéder au dernier élément.
- **os.chdir(path)** : change le répertoire courant de travail

Exemple 7 - Manipulation de chemins

```
>>> path = '/home/tguyet/ProgSIG'
>>> os.path.split(path)
('/home/tguyet', 'ProgSIG')
>>> path = '/home/tguyet/ProgSIG/' #attention a ces chemins !
>>> os.path.split(path)
('/home/tguyet/ProgSIG', '')
>>> os.path.split(os.path.abspath(path))
('/home/tguyet', 'ProgSIG')
>>> path=os.path.abspath('.')
>>> print(path)
'/home/tguyet/ProgSIG/Enseignements/Python'
>>> os.path.dirname(path)
'/home/tguyet/ProgSIG/Enseignements'
>>> os.path.basename(path)
'Python'
```

1.2.1 Lister les fichiers et les dossiers

Le module **os** contient une fonction qui permet de lister simplement les éléments d'un dossier `path` : la fonction **os.listdir(path)**
Toutefois, pour un usage plus compliqué, on préférera le module **glob**, qui contient deux fonctions uniquement. Les fonctions de ce module permettent de lister tous les fichiers dont les noms correspondent au

motif donné en argument. Le motif peut contenir certains caractères spéciaux :
- * remplace n'importe quelle séquence de caractères.
- ? remplace un caractère.
- [] symbolise n'importe quel caractère indiqué dans les crochets.

Les deux fonctions sont les suivantes :
- **glob.glob(pathname)** : Liste tous les dossiers et fichiers dont le motif du nom correspond à `pathname`.
- **glob.iglob(pathname)** : Fonctionne de même que **glob()** mais retourne un itérateur.

L'exemple qui suit permet de lister tous les fichiers `tif` du répertoire courant.

```
import glob
files = glob.glob('*.tif')
for f in files :
  print( f )
```

1.2.2 Autres manipulations sur les fichiers

Tests sur les fichiers et les dossiers Les tests de base sur les dossiers et fichiers se font tous à partir de fonctions du module **os.path**.
- **os.path.exists(path)** Permet de vérifier si **path** est un chemin réel.
- **os.path.isabs(path)** Teste si **path** est un chemin absolu.
- **os.path.isdir(path)** Teste si **path** pointe vers un dossier.
- **os.path.isfile(path)** Teste si **path** pointe vers un fichier.

Création et suppression de fichiers/répertoires
- **os.mkdir(path)** Crée le dossier à la fin de **path**.
- **os.makedirs(path)** Cette fonction est équivalente à un appel récursif à **mkdir()** : elle crée récursivement tous les dossiers qui n'existe pas jusqu'au dossier final de **path**.
- **os.remove(path)** Supprime le fichier ou le dossier pointé par **path**.

1.3 Exécuter des commandes systèmes

La librairie **os** dispose également d'une fonction **system** pour faire exécuter des traitements. Cette fonctionnalité est très utile lorsqu'on souhaite automatiser des traitements qui ne sont pas directement des fonctions python, mais des outils extérieurs.
Dans l'exemple ci-dessous illustre l'utilisation de l'outil `ogr2ogr` pour fusionner des fichiers shapefile. Nous avons besoin de faire deux types d'opération :
- la recopie d'un fichier
  `$ ogr2ogr <inputfile> <outputfile>`
- la fusion d'un fichier `file2` à la suite d'un fichier `file1`
  `$ ogr2ogr -append <file1> <file2>`

Pour ces deux commandes[1], les fichiers de formes doivent tous être du même type (point, polyligne ou polygone) et contenir les mêmes attributs.

```
import glob
import os

def mergeSHPfiles( filePath, newSHPfile):
    files = glob.glob( filePath + '/*.shp')

    # Une variable pour savoir si on est au debut
    first = True

    #variable qui va contenir la commande
    command=''

    # pour tous les fichiers :
    for file in files :
        if first :
            # Pour le premier fichier , on fait une copie pour creer le fichier de sortie
            command = 'ogr2ogr ' + newSHPfile + ' ' + file
            first = False
        else :
            # sinon, on ajoute des elements au fichier de sortie
```

1. Cette commande est disponible avec l'installation de GDAL sur les systèmes de type Linux, mais dans le cas des Windows, il faut voir comment y accéder par des commandes systèmes ...

```
            command = 'ogr2ogr -append ' + newSHPfile + ' ' + file

        print command
        # Execution de la commande
        os.system(command)

#lancement de la fonction
mergeSHPfiles('.../TreeCrowns', 'testmerge.shp')
```

Remarque 22

Une étape utile consiste à d'abord exécuter la commande à partir de la ligne de commande manuellement pour s'assurer que vous comprenez comment cette commande fonctionne, et pour être sûr que tout ce passe correctement. Ensuite, vous pouvez l'intégrer au programme.
Vous pouvez utiliser un copier-coller de la commande testée, puis de modifier cette commande par l'utilisation des variables à la place des noms spécifiques testés.

2 Exemple du traitement des fichiers par lot

La bonne solution pour faire un traitement par lot d'images est de décomposer votre programme en deux :

1. une fonction qui va prendre en paramètre le nom d'une image et qui fera le traitement désiré
2. une programme principal qui va utiliser les fonctionnalités vues dans la section précédente, et qui va appliquer la fonction de traitement sur chacun des fichiers.

```
import ogr, os, sys, osr, glob
import numpy
import gdal
from osgeo.gdalconst import *

'''
- filein : nom du fichier entrant
- fileout : nom du fichier sortant (image NDVI)
'''
def computeNDVI( filein, fileout ):
  # ouvrir l'image
  ds = gdal.Open(filein, GA_ReadOnly)
  if ds is None:
    print 'Could not open image'
    return(0)

  if ds.RasterCount<3:
    print 'Not enough bands'
    return(0)

  #Creation d'une nouvelle image
  driver = gdal.GetDriverByName( "GTiff" )
  dst_ds = driver.Create( fileout , ds.RasterXSize, ds.RasterYSize, 1, gdal.GDT_Float32 )

  dst_ds.SetProjection( ds.GetGCPProjection() ) # meme projections
  geotransform = ds.GetGeoTransform() # meme transformations geometriques
  if not geotransform is None:
    dst_ds.SetGeoTransform( geotransform )

  # creation d'une liste de bandes
  bandList = []

  # lecture des bandes uniquement pour la region d'interet
  for i in range(ds.RasterCount):
    band = ds.GetRasterBand(i+1)
    data = band.ReadAsArray()
    bandList.append(data)

  rasterlayer = numpy.zeros( (ds.RasterYSize, ds.RasterXSize), dtype=numpy.float32 )
  #Parcours des donnees
  for i in range(0, ds.RasterXSize):
    for j in range(0, ds.RasterYSize):
      if ( float(bandList[1][j][i]) + float(bandList[2][j][i]) )!=0:
        #calcul du NDVI
        ndvi = (float(bandList[1][j][i]) - float(bandList[2][j][i])) / ( float(bandList[1][j][i]) + float(bandList[2][j
            ][i]) )
```

```
        else :
          ndvi = 0
        rasterlayer [j][i]=ndvi

    #ecriture dans l'image
    dst_ds.GetRasterBand(1).WriteArray( rasterlayer )


'''
Programme principal : traitement par lot
'''

files = glob.glob('*.tif')
for fin in files :
    print ("process" + fin)
    fout = os.path.dirname(fin) + "/out_" + os.path.basename(fin)
    computeNDVI(fin, fout)
```

3 Introduire un peu de parallélisation dans les traitements

Sachant que la plupart des ordinateurs usuels modernes sont multi-cœurs, le lancement de traitements séquentiels sur plusieurs fichiers par un programme Python n'utilise pas de manière optimal toutes les capacités d'un ordinateur.
L'exemple ci-dessous permet d'effectuer un lancement de traitements (défini dans une fonction) sur plusieurs fichiers en parallèle.

```
from multiprocessing import Pool

def process_file (filename):
    #... traiter ici le fichier filename
    return filename


pool = Pool()
files = ['file1', 'file2', 'file3', 'file4', 'file5', 'file6', 'file7']

results = pool.imap( process_file , files )

for result in results :
    print result
```

4 Exercices

Exercice 34 (Simplification d'une image de classification) *Le résultat d'une classification automatique d'une image raster est une image dont les pixels contiennent des valeurs entières correspondant au numéro de la classe (par exemple, bati, forêt, champs, etc.). Lorsque les classifications sont faites pixel par pixel, il peut arriver que des pixels soient isolés (un seul pixel d'une classe entouré de pixels d'autres classes). Ils introduisent donc un bruit dans l'image que cet exercice propose de supprimer. Ces situations sont fréquemment rencontrées sur les images hautes résolutions*[2].
On vous propose le programme `base_batchSimplification.py` *qui implémente une méthode de traitement du fichier* `zone1_cl.tif` *(dans le répertoire* `batch/DonneesPaytal/` *des ressources du cours). Le principe de cette méthode de contruire une nouvelle image pour laquelle chaque pixel correspond au pixel majoritaire de tous ces voisins. Le paramètre* **n** *permet de définir la taille du voisinage.*
Dans cet exercice, il n'est pas nécessaire de comprendre tous les détails de l'algorithme. L'objectif est de transformer ce programme pour automatiser le traitement sur tous les fichiers du répertoire.

Question a) Créer une fonction **majority(filein , fileout , n)** qui transformera le fichier **filein** en un fichier **fileout** à l'aide de la méthode fournie, en utilisant une taille de voisinage de **n**.

Question b) Créer un script qui utilisera votre fonction pour appliquer la transformation à toutes les images du répertoire `batch/DonneesPaytal/`. Vous prendrez la valeur 2 pour **n**.

Exercice 35 (Apprentissage par lot) *On réutilise ici les images utilisées dans le répertoire données PayPal :* `zones1.tif`*, etc ...*

Question a) Écrire une fonction qui prendra en paramètre un nom de fichier d'entrée et un nom de fichier de sortie et qui réalisera la catégorisation des pixels de l'images. Pour cela, on vous donne le code ci-dessous utilisant la bibliothèque `sklearn` (vous pouvez chercher à le comprendre, mais ce n'est en fait pas nécessaire !).

Question b) Écrire un code qui permettra de catégoriser toutes les zones enregistrées dans un répertoire.

- *vous utiliserez une boucle pour parcourir toutes les images du répertoire contenant les images,*
- *si nécessaire, vous créer un répertoire* `output` *dans le répertoire*
- *vous devrez automatiser la génération d'un nom de fichier de sortie : le résultat du traitement du fichier* `zones1.tif` *devra s'appeler* `output/out_zones1.tif` *(c.-à-d. qu'il sera placé dans le répertoire* `output`*).*

```
from osgeo import gdal
from sklearn import cluster
import numpy

'''
 Classification non supervisee d'une image
'''

print ('chargement de l\'image')
dataset = gdal.Open('./DonneesPayTal/zone1.tif')

# lecture des bandes de l'image
bandList = []
for i in range(dataset.RasterCount):
  band = dataset.GetRasterBand(i+1)
  data = band.ReadAsArray()
  bandList.append(data)

print ('construction de la matrice de donnees')
data= numpy.zeros( (dataset.RasterXSize*dataset.RasterYSize,dataset.RasterCount) )
for p in range(0, dataset.RasterCount):
  for i in range(0, dataset.RasterXSize):
    for j in range(0, dataset.RasterYSize):
      data[j*dataset.RasterXSize+i][p] = bandList[p][j][i]

print ('Classification')
k_means = cluster.KMeans(k=4)
k_means.fit (data)

#On recupere les classes sous la forme d'une matrice
clusterlayer = k_means.labels_.reshape(dataset.RasterYSize,dataset.RasterXSize)

fileout ='output.tif'

'''
Creation d'un fichier de sortie
'''

#Creation d'une nouvelle image
driver = gdal.GetDriverByName( "GTiff" )
dst_ds = driver.Create( fileout , dataset.RasterXSize, dataset.RasterYSize, 1, gdal.GDT_Byte )

dst_ds.SetProjection( dataset.GetGCPProjection() ) # meme projections
geotransform = dataset.GetGeoTransform() # meme transformations geometriques
if not geotransform is None:
    dst_ds.SetGeoTransform( geotransform )

#ecriture dans l'image
dst_ds.GetRasterBand(1).WriteArray( clusterlayer )
```

Exercice 36 (Parallélisation) *Reprendre le programme pour le calcul d'images NDVI pour paralléliser son exécution.*
Vous utiliserez les images du répertoire `DonneesPAYTAL` *qui sont des images RapidEye à 5 bandes. Le PIR est a priori la bande 5 et le rouge est la bande 3 (Rappel* $NDVI = \frac{PIR-R}{PIR+R}$*.*

Exercice 37 (Automatisation de la transformation d'images Raster) *Dans cet exercice, on vous invite à utiliser les fonctionnalités de l'outil* `gdal_translate` *pour manipuler des données des images raster.*

Question a) Changement de format d'image ENVI *La commande ci-dessous permet de changer de format d'image :*

```
$ gdal_translate -of <OutputFormat> <inputfile> <OutputFile>
```

On dispose d'images au format ENVI dans le répertoire `ENVI_Image` *et on souhaite les transformer en format GeoTiff (utiliser le format* `GTIFF` *dans la commande).*

Exercice 38 (Extraction des fichiers de cadastre/commandes Windows) *La commande Windows* `xcopy` *est une commande qui peut être utilisée en ligne de commande DOS pour copier et sauvegarder des fichiers et des répertoires.*
La commande qui nous intéresse consiste à copier des fichiers en format compresser `bz2` *dans un nouveau répertoire. En pratique, il s'agit de fichiers de cadastre dont la dénomination est normalisée. On supposera que tous les fichiers se trouvent dans un unique répertoire. Quelques exemples de fichiers avec la destination souhaitée :*

- *le fichier* `edigeoDK011000AB01.tar.bz2` *doit aller dans le répertoire* `C:\Cadastre\29011\`
- *le fichier* `edigeoDK011000AD01.tar.bz2` *doit aller dans le répertoire* `C:\Cadastre\29011\`
- *le fichier* `edigeoDK015000AD01.tar.bz2` *doit aller dans le répertoire* `C:\Cadastre\29015\`

On en déduit que les caractères 9 à 11 du nom de fichier donne le nom du répertoire de destination de la copie.
La commande qui est nécessaire pour extraire le premier fichier dans son répertoire de destination en le décompressant est la commande suivante :

```
xcopy edigeoDK011000AB01.tar.bz2 C:\Cadastre\29011\ /D /C /I /Y
```

Les options[3] *de* `xcopy` *correspondent aux effets suivants :*

- `/D` *: Copie les fichiers modifiés à partir de la date spécifiée. Si aucune date n'est donnée, copie uniquement les fichiers dont l'heure source est plus récente que l'heure de destination.*
- `/C` *: Continue la copie même si des erreurs se produisent.*
- `/I` *: Si la destination n'existe pas et que plus d'un fichier est copié, considère la destination comme devant être un répertoire.*
- `/Y` *: Supprime la demande de confirmation de remplacement de fichiers de destination existants.*

L'objectif de l'exercice est d'automatiser l'application de cette commande à l'ensemble des fichiers qui sont contenus dans un répertoire source. L'exécution d'une commande système est "facile" grâce à l'utilisation de la fonction **os.system** de python. La difficulté de l'exercice se trouve dans la construction automatisé de la commande. Dans un premier temps, on réalise un programme dont l'objectif va être simplement d'afficher les commandes qui nous intéresse. En finalisation, on fera exécuter cette commande au système.

Question a) *Écriture d'une fonction "cadre"* Écrire une fonction **create_cmd(filename)** qui prend en entrée un nom de fichier **filename**. Dans la suite, on modifiera cette fonction pour qu'elle construise la commande `xcopy` attendue. Pour le moment, faite en sorte qu'elle retourne uniquement le nom de fichier. Ajouter également une fonction principale qui parcourt tous les fichiers `.tar.bz2` du répertoire où sont les fichiers, applique la fonction **create_cmd** et affiche la commande générée.

Question b) *Extraction du numéro d'intérêt* Modifier la fonction **create_cmd(filename)** pour que celle-ci ne retourne que uniquement les trois caractères qui caractérise le chemin de destination dans la commande.
Aides

- si la variable **chaine** contient la chaîne de caractères `Voila un test`, l'instruction **chaine**[4:10] permet d'extraire la sous-chaîne de caractères `a un t`.
- vous connaissez la position des caractères qui vous intéressent uniquement par rapport au début du nom du fichier ! Pas de son chemin en entier. Il faudra peut être décomposer le nom du chemin !

Question c) *Ajout du préfixe du répertoire* Modifier la fonction **create_cmd(filename)** en **create_cmd(filename, dstdir, prefix =''')** où **dstdir** sera le répertoire de destination des copies et **prefix** sera le préfixe des noms de répertoires créés (dans l'exemple) il s'agit de la chaîne de caractères `29`. La nouvelle fonction générera le nom du répertoire de destination de fichier.

Question d) *Ajout de la commande* `xcopy` Commencez par définir deux variables globales **cmd** et **options** pour définir la commande à appliquer (`xcopy`) et ses options (`/D /C /I /Y`). Modifiez ensuite la fonction pour qu'elle construise la commande à partir de ces variables et de

Question e) *Modifier le programme principal pour qu'il affiche la commande et fasse exécuter cette dernière.*

Lorsque vous installer l'utilitaire `7z`, celui-ci fonctionne classiquement avec une interface, mais il offre également une fonction utilisable en ligne de commande. Vous pouvez ainsi intégrer la décompression automatique de votre fichier à votre script. Par exemple, `7z x filename.tar.bz2` est une commande pour décompression un fichier `.tar.bz2` avec `7z`.

Question f) *Modifier le programme principal pour ajouter la décompression automatique des fichiers déplacés.* (plusieurs solutions différentes sont possibles : modifier la fonction, créer une nouvelle fonction, modifier le programme principal).

5 Traitements par lot d'images MODIS (tutoriel)

La programmation permet d'automatiser des tâches. C'est une solution très appropriée pour faire des "traitements par lots". Par exemple, lorsque vous devez traiter un ensemble d'images selon une même méthode, il est souvent moins coûteux en temps, plus sûr et plus pérenne de réaliser un programme qui automatise la tâche plutôt que de tout réaliser à la main. L'objectif de ce tutoriel est de vous faire développer un programme Python qui automatisera :

— le téléchargement d'images à partir de données disponibles librement sur le Web
— le traitement en série des images téléchargées

La situation pratique est le calcul d'un indice NDVI sur des images issues du jeu de données NASA-LANCE (Land Atmosphere Near Real-time Capability for EOS) proposant des images capteurs MODIS Terra et Aqua. On utilise plus précisément le système Rapid Response qui offre la possibilité de télécharger gratuitement (et sans login) des images géo-rectifiées, en format GIS, pour certaines zones du globe pour le jour même et tous les jours depuis au moins 1 an. Par chauvinisme, nous nous intéresserons aux images du sous-ensemble "Paris". Différents images sont proposées au téléchargement libre : images TrueColor, 721, 367 et NDVI, à différentes échelles spatiales : 2km, 1km, 500m et 250m. L'image de 367 rend possible le calcul d'un indice NDVI [4]. Ce sera l'objet de la seconde partie de ce tutoriel.

5.1 Chargement des images MODIS par lot

L'objectif de cette partie est de créer permettant de récupérer des images par lot. Le type de lot qui nous intéressera ici sera constitués d'images d'une même zone pour plusieurs jours d'une même année.

5.1.1 Analyse du problème

La première partie du travail consiste à comprendre comme récupérer des images. On dispose d'une "interface web" qui donne la possibilité de télécharger une image au format GeoTIFF en effectuant la requête suivante :

```
http://lance-modis.eosdis.nasa.gov/imagery/subsets/?project=aeronet&subset=Carpentras.
                                    2014076.terra.721.2km.tif
```

D'un point de vue informatique, il est possible d'automatiser la récupération des images à partir du moment où on sait synthétiser une telle requête HTTP. Il faut donc en comprendre la structure. Dans ce cas précis, cette requête permet de récupérer l'image de la zone de "Carpentras" pour la date du 17 mars 2014 (76ème jour de l'année 2014), pour le capteur Terra à une résolution de 2km et contenant les bandes 721. On comprend alors rapidement que si on veut récupérer l'image du jour précédent, il suffira de proposer une requête très similaire ...
Ayant "découvert" cela, l'exécution d'une requête n'est en rien compliquée, surtout en Python !

5.1.2 Construction de la solution

La figure VI.1 illustre la décomposition proposée du problème. On cherche à construction une précédure globale (**load_and_save**) que l'on va décomposer en trois parties : 2 fonctions **compose_url** et **compose_filename**

4. Enfin, c'est ce que j'ai compris ... vous n'hésiterez pas à remettre en cause les trois paragraphes de contextualisation pour lesquels je ne suis que très peu compétent.

qui donneront les éléments nécessaires au module **urllib2** pour faire le chargement de l'image dans un fichier. Ces deux éléments sont, d'une part, l'url de l'image à télécharger et, d'autre part, le nom du fichier à enregistrer.

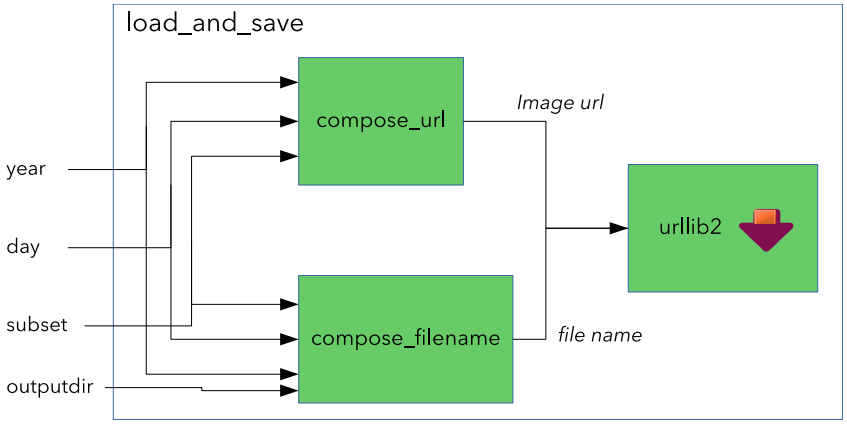


FIGURE VI.1 – Illustration de la décomposition du programme

Cette fonction sera ensuite utile pour automatiser le chargement.

Génération d'une URL

1. Créer une nouveau fichier python nommé `image_loader.py` et faire l'importation des deux modules **os** et **urllib2**
2. Écrire une fonction **compose_url** qui construit une URL pour accéder à une image Terra 721 à 2km. L'URL composée dépendra des membres de la classe et sera retournée par la fonction.

```
def compose_url(year, day, subset="Paris")
    theurl=""
    ...
    return theurl
```

Les paramètres de la fonction sont les suivants :
- **year** : un entier qui donne l'année de l'image désignée par l'URL
- **day** : un entier qui donne le jour dans l'année de l'image désignée par l'URL
- **subset** : désigne le jeu de données, par défaut, ce paramètre à la valeur '`Paris`'

Remarque 23

Vous penserez à utiliser la fonction **str**() pour transformer des nombres en chaînes de caractères.

Vous testerez cette fonction en ajoutant le code suivant en fin de votre fichier :

```
if __name__ == '__main__':
    # ce code sera execute lors de l'appel de ce fichier
    generated_url = compose_url(...)
    print(generated_url)
```

Remarque 24

Ce type de code commençant par **if** `__name__ == '__main__'` est fréquemment rencontré en python. Il désigne des parties du code qui sont exécutés uniquement lorsqu'on exécute le fichier. Dans le cas de l'importation de votre fichier depuis un autre programme Python, ce code ne sera pas exécuté. Ceci est particulièrement approprié pour des parties de code testant les fonctionnalités proposées dans votre fichier.

4. De la même manière, écrire une fonction **compose_filename**(...) qui construit un nom de fichier de la forme : `images/Image_Paris_2014232.tif`

où image correspond au répertoire de sortie (`outputdir`), `Paris` est le subset, et `2014` est la concaténation de l'année et du jour.

```
def compose_filename(year, day, subset, outputdir)
  ...
  return ...
```

Vous compléterez la fonction principale de sorte à tester votre nouvelle fonction.

Remarque 25

Vous ferez attention au cas où `outputdir` est vide ! Je vous laisse voir le problème qui se présentera dans ce cas !

Lancement de la requête et récupération du fichier Ca y est ! On a quasiment tous les éléments pour finaliser la fonction **load_and_save** qui va faire le travail de chargement d'une image. Les éléments d'utilisation de **urllib2** étant un peu techniques, on vous fournit une partie du code.

5. Recopiez la fonction **load_and_save** ci-dessous dans votre module. En consultant la documentation et avec du bon sens, vous la commenterez et l'expliquerez ligne par ligne.
6. Vous modifierez ensuite la fonction principale de votre fichier sorte à tester cette nouvelle fonction.

```
def load_and_save(year, day, subset, outputdir="images"):
  if not os.path.exists(outputdir):
    os.mkdir(outputdir)
  url = compose_url(year, day, subset)

  filehandler =urllib2.urlopen( url )

  filename=compose_filename(year, day, subset, outputdir)

  fout = open(filename, "wb")
  fout.write( filehandler.read() )
  fout.close()
```

Chargement par lot

7. Créer une nouvelle fonction **batch_load** (...) . L'appel de fonction **batch_load**(2014, 15, 18, "images") chargera toutes les images entre les jours 15 et 18 de 2014 dans le répertoire '`images`'.

5.2 Traitements d'images par lots

L'objectif de cette partie est de créer un "programme" pour traiter un lot d'images. On suppose pour cela que les images sont situées dans un même répertoire. On illustre cette tâche en réalisant un traitement simple : celui du calcul d'un indice NDVI.

5.2.1 Analyse du problème

Deux difficultés se présentent dans cette partie :
— le traitement individuel d'une image : comment réaliser le calcul d'une image NDVI en Python ?
— le traitement par lot : comment automatiser un même traitement sur plusieurs images ?

Pour la première difficulté, il s'agit premièrement de récupérer les informations contenues dans un fichier en format "géographique" (GeoTiff, HDR, etc.). Cette étape de chargement doit construire des objets utilisables en python, typiquement, des matrices. Ensuite, une fois qu'on a les matrices, on peut faire tous les traitements que l'on veut ... c'est en fait la partie simple. Pour le calcul d'un indice NDVI, c'est particulièrement simple puisque c'est la combinaison pixel à pixel de deux couches. Par contre, il faut ensuite pouvoir enregistrer les fichiers en un format standard (géolocalisé). Pour la seconde difficulté, une fois qu'on a le traitement d'un fichier, il faut pouvoir lister l'ensemble des fichiers d'un répertoire ...

5.2.2 Calcul de l'indice NDVI

7. Créer une nouveau fichier python nommé **ndvi_processor.py**
8. Ajouter une fonction **compose_filename** qui prend en paramètre le nom d'un fichier (sans le chemin) et qui retourne un nom de fichier (une chaîne de caractères) ajoutant "`_NDVI.tif`" avant l'extension et préfixant par "`output/`". Cette fonction a pour but de construire automatiquement le nom de l'image de sortie à partir du nom de l'image d'entrée.

Par exemple, l'utilisation de la fonction sur "`toto.txt`" donnerait "`output/toto_NDVI.tif`" Pour cela, vous vous intéresserez à la fonction **os.path.splitext** (). Dans cette fonction, on prendra soin de créer le répertoire "`output`" (fonction **os.mkdir**) si jamais il n'existait pas (tester son existence avec **os.path.exists**)

9. Ajouter une fonction qui charge un fichier d'image dont le nom sera passé en paramètre et enregistre un nouveau fichier image obtenu par le calcul des indices NDVI (s'inspirer fortement du code illustrant le chargement d'une image).

```
# inputdir : chemin d'acces au fichier
# filename : nom du fichier d'image (MODIS 721) e traiter
# la fonction enregistre l'image NDVI correspondante dans le repertoire output
def computeNDVI(inputdir, filename):
```

L'image traitée sera une image MODIS contenant les bandes 721 (identique aux images téléchargées par le module de la section précédente). Le calcul de la valeur d'un pixel (x,y) de l'image sera donc obtenu en combinant la valeur du pixel de la bande numéro 2 (bande proche infra-rouge) et de la bande 3 (bande rouge). On a alors quelque chose comme :

```
band = dataset.GetRasterBand(2)
 array_nir =band.ReadAsArray()
band = dataset.GetRasterBand(3)
array_red =band.ReadAsArray()

#(...)
array[x][y]= ( array_nir [x][y] - array_red[x][y]) / ( array_nir [x][y] + array_red[x][y])
```

En faisant les choses ainsi et dans la mesure où les bandes contiennent des entiers, vous faites alors une division entière dont le résultat sera soit 0 soit 1... pas très intéressant. Alors, il faut explicité que les nombres sont, en fait, des nombres à virgule en procédant ainsi :

```
array[x][y]= (float(bande_nir[x][y]) - float(bande_red[x][y])) / (float(bande_nir[x][y]) + float(bande_red[x][y]))
```

10. Une fois votre fonction mise en place, testée là sur l'une des images téléchargées. Si besoin, modifier votre code et tester de nouveau.

5.2.3 Traitements par lot

L'instruction suivante permet de récupérer la liste des fichiers et des dossiers dans le dossier "`toto`' : `repfiles =os.listdir ( "toto" )`.

11. Écrire une fonction **serialNDVI(inputdir)** qui traite toutes les images avec l'extension "`.tif`" contenue dans le répertoire inputdir avec la procédure **computeNDVI**. Attention, vous ne devez pas donner des fichiers qui ne soient pas des "`.tif`" à votre procédure, sinon celle-ci provoquera des erreurs.

5.3 Aller plus loin : exercices

Exercice 39 (Traitement d'une sous-partie de l'image) *Créer une fonction* **computeNDVI_sec(inputdir, filename, box)** *pour laquelle box contient les coordonnées d'une sous-région de l'image à traiter. Cette fonction réalisera la même chose que* **computeNDVI** *mais en ne traitant que la zone définie par la box.*

Exercice 40 (Image stack) *Créer une nouvelle fonction* **createNDVIStack(inputdir)** *qui construit une image unique avec 1 bande par image du répertoire d'image. Pour chaque image, la bande contiendra l'indice NDVI calculé dans cette même fonction.*

! Attention !

Cette fonction risque d'être assez imposante : il est nécessaire de bien réfléchir à votre stratégie avant de vous lancer dans sa construction. D'autre part, je conseille fortement de bien la commenter au fur et à mesure de votre code.

CHAPITRE

VII Automatisation des traitements dans QGis

QGis offre une possibilité de manipuler les données directement à l'aide de Python. Trois mode d'utilisation sont possibles, du plus simple au plus complexe à mettre en place :
- la console : elle vous permet de faire exécuter des commandes simples
- l'utilisation de scripts pré-programmés (notamment grâce au `script runner`)
- les plugins Python

QGis propose une API très proche de celle de OGR/GDAL (parce qu'elle est basée sur cette dernière). Vous ne serez donc pas dépaysés dans les noms de fonctions. Dans l'état actuel de l'API de QGis, les traitements sur les couches raster sont moins étendus qu'avec GDAL, il est alors parfois nécessaire de passer par des traitements GDAL sur des fichiers pour ce type de données !
Dans tous les cas, il faut vous référez à la documentation en ligne pour retrouver les syntaxes pour les opérations élémentaires : `http://docs.qgis.org/1.8/en/docs/pyqgis_developer_cookbook/`.

Dans la suite de cette partie, on s'intéresse à la construction d'une extension pour l'affichage d'information sur la répartition des distances deux à deux entre géométries d'une mêmes couche. Le cas typique d'utilisation est celui du fichier `sites.shp` (qui contient des points avec différentes caractéristiques), et on souhaite connaître les distributions des distances entre toutes les pairs de points ou entre pairs de points d'un certain type.

1 Utilisation de la console

Dans un premier temps, nous allons procéder à quelques manipulations dans la console python.
Commencez par ouvrir la console python. L'interface vous mentionne qu'il existe un objet particulier, appelé **qgis.utils.iface** qui représente l'interface de QGis. C'est au travers de cet objet qu'on peut faire des manipulation.
Commençons par importer des données :

```
>>> vLayer = qgis.utils.iface.addVectorLayer("/home/tguyet/Enseignements/ProgSIG/Tutoriel/data/data_transfogeo/
    ", "sites.shp", "ogr")
```

On applique ici la fonction **addVectorLayer** sur l'objet **qgis.utils.iface** en lui passant trois arguments : le nom du répertoire dans lequel se trouve le fichier, le nom du fichier et l'information `"ogr"` qui spécifie le *driver* pour lire le fichier.
Avec cette commande, tout ce passe ensuite comme si vous aviez ouvert votre couche à l'aide de l'interface. La couche est alors gérée au travers d'un registre de couche. Ce registre permet notamment de fermer toutes les couches ouvertes :

```
>>> QgsMapLayerRegistry.instance().removeAllMapLayers()
```

mapcanvas ! !

Une fois que la couche a été ouverte et qu'on a récupérer l'objet **vLayer** pour désigner notre couche, il est possible de traiter les éléments de cette couche. On va commencer par les afficher 1 à 1 par les commandes ci-dessous :

```
>>> provider = vLayer.dataProvider()
>>> feat = QgsFeature()
>>> while provider.nextFeature(feat):
...
...     # fetch geometry
...     geom = feat.geometry()
...     print "Feature ID %d: " % feat.id()
...
```

Ceci permet de lister toutes les *feature* du fichier ...

Bon voilà ... vous avez compris qu'on peut faire du code dans la console ! Ce peut être utile pour de petites manipulations avec une bonne maîtrise de l'API ou pour faire des tests sur l'existence de telle ou telle fonctionnalité, mais on ne va pas loin ...
Le deuxième message, c'est que l'API diffère très peu de GDAL !
Je passe donc dès maintenant aux scripts !

> **Exercice 41 (Ouvertures de tous les fichiers SHP d'un répertoires)** *Notez que la console python de QGis est une spécialisation de la console python normale. Il est possible d'y faire les mêmes choses que précédemment.*
> *En particulier, il est possible de faire appel à des fonctionnalités extérieurs utiles qu'on a déjà pu voir : le parcours d'un ensemble de fichier.*
>
> *Question a)* En utilisant le module **glob** et **os**, proposer une séquence d'instructions qui va permettre de charger automatiquement tous les fichiers `shp` du répertoire `data/vecteurs/`.

2 Utilisation de scripts

2.1 Création d'un script

2.1.1 Premier script simple

Le principe des scripts est le même que d'habitude ... il regroupe plusieurs lignes de commandes qui vont être exécutées successivement.
Si je reprends ce qu'on a fait précédemment, en l'adaptant aux nécessités du script, nous avons dans le fichier *pairwise_distance.py* :

```
#pour la definition de l'API de qgis
from qgis.core import *

def PairwiseDistance():
    if qgis.utils.iface.activeLayer()==None:
        print "No available active layer"
        return
    vLayer = qgis.utils.iface.activeLayer()
    if vLayer.type()!=0 :
        print "Active layer is not a vector layer"
        exit

    provider = vlayer.dataProvider()
    feat = QgsFeature()
    allAttrs = provider.attributeIndexes()
    while provider.nextFeature(feat):
        geom = feat.geometry()
        print "Feature ID %d: " % feat.id()
```

Tout d'abord, 1) il est nécessaire d'importer l'API QGis pour que python reconnaisse les fonctions qui sont spécifiques à cette API. 2) on a fait une fonction pour rendre l'usage plus facile !
On remarque que si la boucle est la même que précédemment, je n'ai pas ouvert l'image de la même manière. En fait, je n'ai pas inclus l'ouverture de l'image dans ce script (en supposant que l'utilisateur sait fait cela de son côté !).
En revanche, il faut récupérer un objet sur la couche qui m'intéresse, l'objet **vLayer**. Pour cela, j'utilise **qgis.utils.iface.activeLayer()** qui désigne la couche active dans QGis (celle qui est grisée dans la liste des couches). Je vérifie d'abord qu'il y a bien une couche active, et ensuite que cette couche est bien une couche vectorielle. Ces deux vérifications vont éviter les erreurs en cas de mauvaise manipulation.

Retournons maintenant dans QGis et activons la couche qui nous intéresse, et dans la console nous allons utiliser les commandes suivantes :

```
>>> import pairwise_distance
>>> pairwise_distance.PairwiseDistance( qgis.utils.iface )
Feature ID 0:
Feature ID 1:
Feature ID 2:
```

```
...
```

La première ligne sert à importer les fonctions du module précédent, et la seconde ligne lance effectivement la fonction.

! Attention ! - Configuration du *PYTHONPATH*

Pour que QGis reconnaisse votre script, il est nécessaire que le répertoire de localisation de vos scripts soit connu de python, il faut l'ajouter au *PYTHONPATH*.
Sous Windows, il faut configurer ce chemin dans les variables d'environnement. Sous Linux, il faut exécuter la commande ci-dessous :

```
export PYTHONPATH=$PYTHONPATH:/home/tguyet/Enseignements/ProgSIG/Tutoriel/QGis_scripts/
```

Ceci doit être fait avant de lancer QGis.

Remarque 26 - Rechargement d'un script dans la console QGis

Si vous êtes amené à modifier votre script (pour le corriger par exemple), faut le recharger à chaque fois dans la console Qgis. Pour cela, utiliser la commande ci-dessous :

```
>>>reload(pairwise_distance)
```

Exercice 42 (Ouverture sélective de fichiers) *On créé ici un script* `load_images.py` *que vous prendrez soin de placer dans un répertoire accessible par le PYTHONPATH.*

Question a) Commencez par transformer les commandes de l'exercice précédent en un script dans lequel vous définirez une fonction **open_shp(rep)** qui prendra en paramètre le répertoire duquel ouvrir les fichiers `shp`.
Tester votre fonction dans QGis.

Question b) Faire une nouvelle fonction **open_tif(rep)** qui aura la fonctionnalité que vous devinez ...
Tester votre fonction dans QGis par exemple avec les données du répertoire `DonneesPaytal`.

On souhaite maintenant ouvrir de manière sélective les fichiers issues du répertoire de travail. Plus particulièrement, on ne veux ouvrir uniquement les fichiers qui ont un point commun avec la zone actuellement visible dans l'interface.
Pour accéder aux informations de la fenêtre actuelle visualisée, il faut utiliser le **canvas** *de l'interface. Tester les quelques lignes suivantes dans la console QGis :*

```
>>> canvas = qgis.utils.iface.mapCanvas()
>>> extent = canvas.extent()
>>> print extent.xMaximum(), extent.xMinimum(), extent.yMaximum(), extent.yMinimum()
```

Pour savoir si il faut ouvrir un fichier, on va se servir des fonctionnalités de GDAL pour pré-ouvrir un fichier et vérifier si son étendue correspond à notre fenêtre actuelle ou non !

Question c) Écrire des fonctions **sopen_shp(rep)** et **sopen_tif(rep)** pour l'ouverture sélective d'images.
Vous pourrez tester votre fonction **sopen_tif(rep)** *en utilisant les données du répertoire* `DonneesPaytal` *: ouvrez les une fois et zoomez sur une seule partie des images, fermez les toutes et testez votre script. Attention, dans GDAL, vous n'accédez pas directement à une extent d'une image raster, il faut passer par le vecteur de* **GeoTransfom** *pour avoir les informations utiles pour déterminer si l'image correspond au canvas. Dans le cas de l'image raster, le test de vérification n'est pas simple ... faites un dessin !*

2.1.2 Complétons le script

Modifier la fonction de sorte qu'elle intègre les éléments suivants :

```
#construction de la liste de features
allg=[]
while provider.nextFeature(feat):
    #on recupere que les centroids
    geom = feat.geometry().centroid()
    allg.append(geom)

#construction de toutes les distances 2 a 2
dists=[]
for i in range(len(allg)):
    geom_i = allg[i]
    for j in range(i+1, len(allg)-1):
        geom_j = allg[j]
        #calcul de la distance
        d = geom_j.distance( geom_i )
        dists.append( d )

# affichage de l'histogramme
num_bins = 10
n, bins, patches = plt.hist(dists, num_bins, normed=1, facecolor='green', alpha=0.5)
plt.xlabel('Pairwise distances')
plt.ylabel('Probability')
plt.title(r'Histogram of distances')
plt.show()
```

Pour les fonctionnalités de dessin, vous aurez à importer les fonctionnalités de **matplotlib** :

```
import matplotlib.pyplot as plt
```

Recharger votre script et exécuter depuis QGis.

2.2 Utilisation du `Script Runner`

Le `Script Runner` est une extension de QGis qui permet d'exécuter des scripts python facilement. En particulier, il évite d'avoir à faire la configuration du PYTHONPATH (*cf.* warning 2.1.1).
la figure VII.1 illustre l'interface très intuitive du plugin : on ajoute ses plugins dans la liste et pour les exécuter, il suffit de double-cliquer dessus !

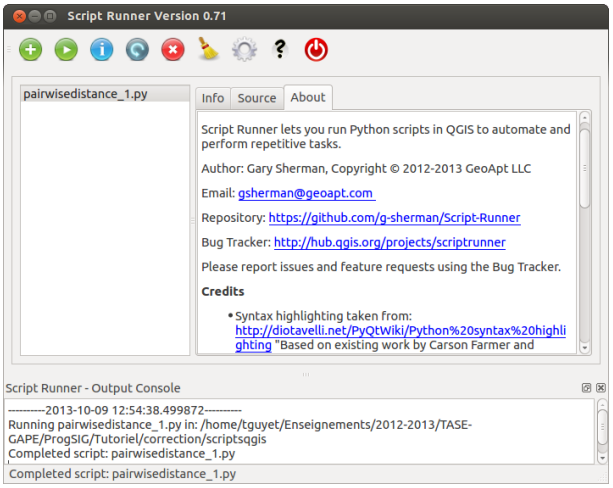


FIGURE VII.1 – Interface du `script runner`

Les seules contraintes à respecter sur le plugin sont les suivantes :

- le script doit importer **qgis.core** et **qgis.gui**
- le script doit comporter une fonction **run_script(iface)** : cette fonction sera le point d'entrée du plugin (*c.-à-d.* ce qui sera effectué). Le paramètre **iface** de la fonction est le même objet que **qgis.utiles.iface** dans la console.

Dans le cas de notre plugins, il est très facile de modifier le script pour le faire correspondre à ce besoin. Seul le début change :

```
from qgis.core import *
```

```
from qgis.gui import *
import matplotlib.pyplot as plt

def run_script(iface):
    if iface.activeLayer()==None:
        print "No available active layer"
        return

    vLayer = iface.activeLayer()
    if vLayer.type()!=0 :
        print "Active layer is not a vector layer"
        return
```

3 Programmation d'un plugin en Python

L'objectif de cette partie est de rendre notre calcul d'histogramme de distances pair à pair encore plus intuitif. Pour cela ; on va construire une extension (ou plugin) qui aura une interface graphique dans laquelle on laissera l'utilisateur choisir la couche à traiter et le nombre de barres dans l'histogramme.

3.1 Étape 1 : créer une coquille vide !

Créer la base de développement d'un plugins Qgis a peu d'intérêt, et est très systématique. Il est possible d'utiliser des outils qui vous aiderons à être efficace dans cette étape.

1. Installation du plugin builder : Dans Qgis, vous pouvez installer l'extension "Plugin builder" à partir du menu `Extension>Installateur d'extensions`.
2. Création d'une structure de plugin : Lancer le plugin, et remplir tous les champs nécessaires puis faire `ok`. Dans la suite, je nommerai mon plugins *PairwiseDistancesPlugin* qui a pour but de séparer les couches d'un Raster avec plusieurs couches.

Lorsque vous faites `ok`, il vous demande d'enregistrer les fichiers que le plugin va générer. Choisissez d'enregistrer votre plugin dans le répertoire des plugins de QGis. Sous Linux, il s'agit du répertoire `/home/tguyet/.qgis/python/plugins/`.

1. Dans un terminal, placez-vous dans le répertoire de votre plugins (`cd /home/tguyet/.qgis/python/plugi`
2. Utiliser la commande "`make`" dans le terminal pour compiler les interfaces graphiques,
3. Retourner dans Qgis puis dans `Extension>Gestionnaire d'extension`, activer le nouveau module *Pairwise Distances*

Maintenant, votre module doit apparaître dans la liste des extensions, et vous pouvez même le lancer... mais pour le moment, il ne fait pas grand chose.

3.2 Étape 2 : Modification de l'interface avec QtCreator

L'interface graphique, c'est-à-dire l'agencement des éléments graphiques (boutons, listes, labels, etc.) est définie dans le fichier nommé `ui_layerseparator.ui`. Ce fichier peut être édité grâce à l'outil QtCreator. Depuis le terminal, lancer QtCreator en utilisant la commande suivante :

```
qtcreator ui_layerseparator.ui
```

Modifier l'interface graphique en ajoutant :
- un *Label* contenant le texte "sélectionner les couches à traiter"
- une *ListWidget* qui contiendra la liste des couches actives
- un second *label* avec marqué "bins" (nombre de colonnes dans l'histogramme)
- un *spin-box*
- les boutons *Cancel* et *Ok* déjà en place.

Très grossièrement, cela ressemble à la figure VII.2.
Enregistrez le fichier et quittez QtCreator.

Remarque 27 - Prise en compte des modifications du plugin dans QGis

Si vous relancer de nouveau votre plugin, les modifications n'ont pas été prises en compte, il faut tout d'abord 'compiler' le fichier `ui`. Pour cela, faites de nouveau la commande '`make`' dans le terminal.
Dans QGis, désactiver le plugins puis réactivez le pour le réinitialiser. Ensuite, relancer votre plugin, les modifications de l'interface graphique ont été prises en compte. C'est une procédure très embêtante ... mieux vaut bien éviter les erreurs !

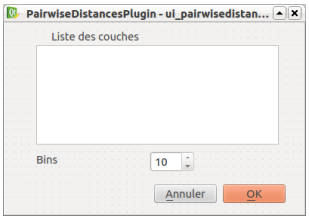


FIGURE VII.2 – Illustration de notre plugin QGis

En l'état, l'interface graphique ne fait rien, il faut maintenant passer à un peu de programmation pour :

1. remplir la liste des couches actives à partir de la liste des couches de QGis,
2. décrire ce qu'il doit se passer quand on clique sur `Ok`

On a fait les choses techniques, on va maintenant faire un peu de python.

3.3 Intégration du calcul des distributions de distances 2 à 2

Commençons simple avec pour objectif de réaliser un plugin qui fasse le calcul des distances pair à pair. Laissons les premières méthodes (**__init__** , **initGui** et **unload**) et intéressons-nous à **run**(). C'est au sein de celle-ci que se passera l'essentiel des actions.
Ce fichier contient ces quelques lignes :

```
# show the dialog
self.dlg.show()
# Run the dialog event loop
result = self.dlg.exec_()
# See if OK was pressed
if result == 1:
    # do something useful (delete the line containing pass and
    # substitute with your code)
    pass
```

Pour l'initialisation de l'interface graphique, il faut mettre du code entre la création de **dlg** et son affichage (avec **show**()). Et comme indiqué dans les commentaires, l'action a mener par le plugin est à mettre à la place de **pass**. Le **result**==1 sert à savoir si l'utilisateur a cliqué sur le bouton `Cancel` (0) ou sur le bouton `ok` (1). L'action n'est à faire que si l'utilisateur à taper sur le bouton `ok`.

3.3.1 Remplissage de la liste des couches

Intéressons nous ici à mettre en place l'interface graphique. Le code sera inséré avant **self.dlg.show**().
Rappelez-vous, nous souhaitons afficher dans notre zone de texte la liste des couches actives. Pour cela nous avons besoin de savoir ce qu'il y a dans la carte de Qgis. Le contenu de la carte est dans l'objet **mapcanvas** qui peut être récupéré ainsi auprès de l'interface Qgis (**self.iface** est l'objet qui représente l'interface générale de Qgis)

```
mapCanvas = self.iface.mapCanvas()
```

Ensuite, il me suffit de faire une boucle sur les différentes couches de la carte ce qui permet de créer une ligne pour chaque couche et de l'insérer dans la liste en construisant une chaîne de caractères appropriée :

```
#on parcourt les couches de la derniere a la premiere
for i in range(mapCanvas.layerCount()-1,-1,-1):
    # on recupere la i-eme couche de l'image
    layer = mapCanvas.layer(i)
    # insertion d'un nouvel element a la liste
    lvi = QListWidgetItem( self.dlg.ui.listWidget )
    # modification du texte du nouvel element en prenant le nom de la couche
    lvi.setText( layer.name() )
```

3.3.2 Implémentation de la fonctionnalité

Intéressons nous maintenant à la partir d'exécution de la fonctionnalité dans la partie où il faut *do something unseful...*

Là, on ne va pas tenter le diable et on va réutiliser ce qu'on a déjà ... une belle fonction bien mise en forme. Je vous propose donc de recopier intégralement la fonction de la section précédente (celle du `script runner` par exemple).
On va donc avoir maintenant quelque chose comme :

```
if result == 1:
    # do something useful (delete the line containing pass and
    # substitute with your code)
    run_script( ...)
    pass
```

On va ensuite adapter cette fonction pour faire en sorte qu'elle soit utilisable facilement. On va faire en sorte d'avoir une fonction avec le profil ci-dessous :

```
def run_script(self, vLayer, bins):
```

- on ajoute le premier paramètre **self** pour une fonction définie dans une classe (!!)
- on supprime le paramètre **iface** pour le remplacer par le paramètre **vLayer** puisque le choix de la couche ne se fait plus avec la couche active, mais par celle qui sera sélectionnée par l'utilisateur
- on ajoute le paramètre **num_bins** pour mettre avoir le choix de l'histogramme

En conséquent, il faut apporter des modifications à la fonction :

- tout le début de la fonction sur l'utilisation de la couche active doit être supprimé,
- on peut conserver le test sur le type de couche,
- la ligne **num_bins** = 10 servant à l'initialisation de la variable **num_bins** peut être supprimée. Sa valeur est passée en paramètre de la fonction.

Revenons à l'appel de fonction. D'après notre fonction, il faut passer en paramètre la couche à traiter et le **num_bins**. Ces éléments là vont être récupérés sur les *widgets* de l'interface.

```
if result == 1:
    # do something useful (delete the line containing pass and
    # substitute with your code)

    ItemSelect = list(self.dlg.ui.listWidget.selectedItems())
    ok=False
    #on cherche la couche avec le meme nom !
    for i in range(mapCanvas.layerCount()-1,-1,-1):
            if mapCanvas.layer(i).name()==ItemSelect[0].text():
              layer = mapCanvas.layer(i)
              ok=True
              break

    nbins=self.dlg.ui.spinBox.value()

    if(ok):
        self.run_script(layer, nbins)
    else:
        QMessageBox.information(self.dlg, 'Info Message', 'Invalid layer name selected', QMessageBox.Ok)

    run_script(layer, nbins)
```

Bibliographie

— `http://fr.openclassrooms.com/informatique/cours/langage-python` : tutoriel sur le langage Python
— `http://www.gis.usu.edu/~chrisg/python/2009/` : tutoriel sur l'utilisation de python et des libraires GDAL
— `https://www.e-education.psu.edu/geog485/node/91` : cours en ligne en licence *creative common*. Ce cours utilise principalement **arcpy** (python pour ArcGIS). Reprend les bases de python, comprend des exercices avec corrections.
— `http://courses.washington.edu/geog465/Programming_Resources.html` : liste de ressources sur la programmation SIG, avec principalement du python, (qqs ressources non accessibles !)
— `http://courses.washington.edu/geog465/465s09.html` : cours orienté sur l'utilisation de bases de données géographiques dans des programmes python