Utilisation de données raster avec Python

Clément Delgrange
03/2018

Table des matières

1 Objectifs

— Apprendre à manipuler des raster avec Python
— Combiner des données raster et vecteur

2 L'information géographique raster

Un raster est une image (photographie, plan, modèle numérique de terrain, etc.) géoréférencée. L'information représentée dans le raster est stockée sous forme de matrices, éventuellement à plusieurs dimensions, où chacune des cellules, aussi appelée **pixel**, est associée à une valeur chiffrée représentant la valeur de l'information. Cette valeur peut être de différentes natures : bruit, température, altitude, chiffre représentant une classe d'occupation du sol, valeur spectrale représentant une couleur, etc. Une valeur *no_ data* peut indiquer qu'aucune information n'est connue pour un pixel.

Un raster se caractérise notamment par :

— une taille (nombre de lignes et de colonnes) ;
— un nombre de canaux ;
— une taille terrain d'un pixel ou résolution ;
— un type pour les valeurs de pixels (entier positif/relatif, nombre à virgule) ;
— une valeur de *no_ data* ;
— un format.

Parmi la nombreuse liste de formats, propriétaires ou ouverts, nous les formats images classiques et quelques formats spécifiques à la géomatique : le tif, l'ecw, le gif, le bmp, l'hdf, etc.

Le traitement et l'analyse des rasters est souvent plus complexe que celle des vecteurs. Notons par exemple que la récupération des coordonnées spatiales d'un pixel n'est pas automatique. Celui-ci est référencé à l'intérieur du raster en coordonnées image (position en terme de numéros de ligne et de colonne). Seules les coordonnées spatiales d'un coin du raster sont connées. Les coordonnées spatiales du pixel sont ensuite recalculées à partir des coordonnées du coin, de coordonnées images du pixel, de la résolution et de l'orientation du raster (angle avec le nord).

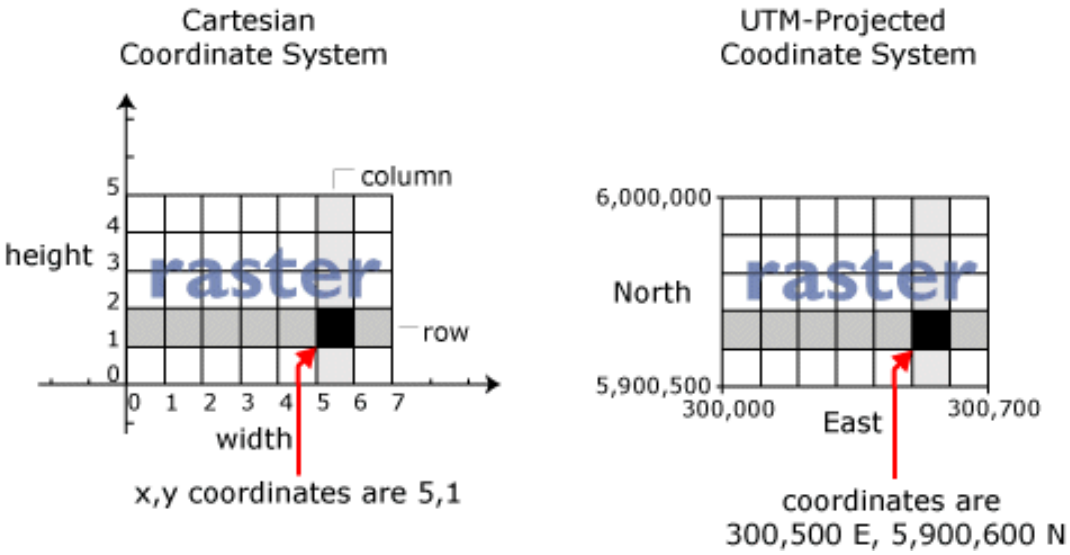

FIGURE 1 – Coordonnées images et spatiale d'un raster (source Esri)

GDAL est la librairie fondamentale de gestion des rasters en géomatique. Son développement est supportée par l'OSGeo.

Avec PROJ.4, pour les reprojections, GEOS, pour la manipulation de géométries vecteur, GDAL est la troisième librairie fondamentale de la géomatique.

Si GDAL est écrite en C, un binding Python existe : `osgeo.gdal`. L'exemple suivant permet d'ouvrir un raster `osgeo.gdal` et d'afficher ses métadonnées :

```
from osgeo import gdal
gtif = gdal.Open("raster.tif")
print(gtif.GetMetadata())
```

Malheureusement, la syntaxe de cette librairie n'est pas très intuitive et son utilisation se révèle souvent assez complexe [1]. Aussi, d'autres outils on vu le jour et proposent des interfaces plus pythoniques pour manipules de raster géoréférencés. La librairie la plus utilisée est certainement `rasterio`.

Le contenu des différentes bandes du raster sont directement stockées dans des tableau `numpy`. Dans l'exemple ci-dessous nous récupérons la valeur de chacune des bandes RGB d'un raster pour ensuite les afficher individuellement.

```
import rasterio
import matplotlib.pyplot as plt

with rasterio.open('RGB.tif') as src:
    r, g, b = src.read()

plt.imshow(r, cmap='gray')
plt.show()
```

Pour accéder au métadonnées d'un raster :

1. voir le livre d'exemples : https://pcjericks.github.io/py-gdalogr-cookbook/.

```
with rasterio.open('RGB.tif') as src:
    print(src.width, src.height)
    print(src.crs)
    print(src.transform)
    print(src.count)
    print(src.indexes)
```

Ou avec un appel plus synthétique :

```
with rasterio.open('RGB.tif') as src:
    print(src.profile)
```

2.1 Exercices

1. Créer un masque de l'orthographie `ortho.tif` où les pixels ne peuvent prendre que deux valeurs :

— 255 si la somme des trois bandes (rouge, vert et bleu) est supérieure à 200 ;
— 0 sinon.

Sauvegarder le résultat dans un nouveau raster.

2. Permettre à l'utilisateur de choisir la projection du raster en sortie.

3 Combiner données raster et vecteur

Superposer des données raster et vecteur pour effectuer des analyses uniquement à l'aide de la programmation est souvent un exercice périlleux. Cela peut l'être encore plus si les données sont exprimées dans des systèmes de coordonnées différents. En effet un raster est géoréférencé globalement (classiquement on connaît les coordonnées spatiales de l'emprise du raster) mais les coordonnées des pixels sont exprimées en coordonnées images (ligne-colonne). Aussi le processus de recalage d'un vecteur sur un raster, qui consiste à déterminer les coordonnées images des points de la géométrie, nécessite d'effectuer quelques calculs mathématiques. Ces opérations qui permettant d'effectuer la transformation coordonnées spatiales vers coordonnées images, si elles ne sont pas d'une complexité monstrueuse sont rarement implémentées dans les libraires que nous utilisons.

Il est possible par exemple de s'en sortir par exemple à l'aide de `fiona` pour ouvrir des fichiers de formes et `rasterio` pour les rasters. Ces librairies, ayant été développées par une même personne, présentent l'avantage d'offrir une API similaire facilitant le travail du développeur. Des sous-modules de `rasterio` apportent par ailleurs des fonctionnalités intéressantes pour combiner différentes données.

Dans l'exemple suivant nous creons un mask dans le raster à partir des géométries dans le fichier de formes :

```
import fiona
import rasterio
from rasterio.tools.mask import mask
```

```
with fiona.open("polygons.shp", "r") as shapefile:
    geoms = [feature["geometry"] for feature in shapefile]

with rasterio.open("image.tif") as src:
    out_image, out_transform = mask(src, geoms, crop=True)
    out_meta = src.meta.copy()

out_meta.update({"driver": "GTiff",
                 "height": out_image.shape[1],
                 "width": out_image.shape[2],
                 "transform": out_transform})

with rasterio.open("masked.tif", "w", **out_meta) as dest:
    dest.write(out_image)
```

Une librairie comme `buzzard` (https://github.com/airware/buzzard/) vise également à faciliter la gestion des données géospatiales de différents type et dans différentes projections tout en offrant un interface pythonique à l'utilisateur.

L'exemple suivant calcule les coordonnées images d'une ligne :

```
import buzzard as buzz

ds = buzz.DataSource()
ds.open_raster("ortho", "ortho.tif")
ds.open_vector("line", "linestring.shp")

spatial_coords = ds.line.get_data(0, None, geom_type="coordinates")
img_coords = ds.ortho.spatial_to_raster(spatial_coords)
```

4 Exercice final

Consignes :

— Le fichier de formes `polygons.shp` représentent des polygones ;
— Le raster `dsm.tif` est un MNT ;
— Pour chacun des polygones, il s'agit de calculer l'altitude minimale du MNT sous son emprise.
— Le résultat est inscrit dans la données d'entrée.

5 Références

— Introduction à `numpy` : http://www.courspython.com/apprendre-numpy.html
— Documentation de `rasterio` : https://mapbox.s3.amazonaws.com/playground/perrygeo/rasterio-docs/python_manual.html