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)
```

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

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.

— **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 2

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.

Remarque 3 - 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 1).

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

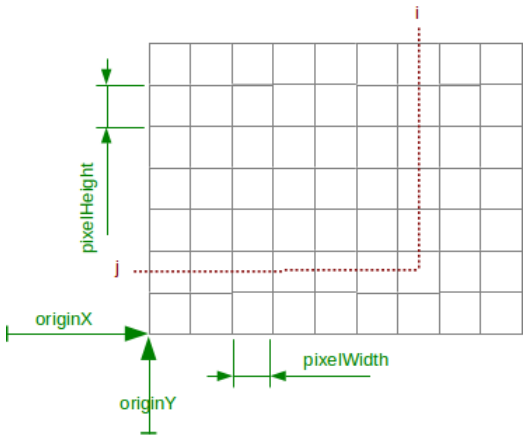

FIGURE 1 – Illustration de la correspondance entre coordonnées physiques (en vert) et index des pixels (en rouge).

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 4

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 5 - 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
```

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 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 :

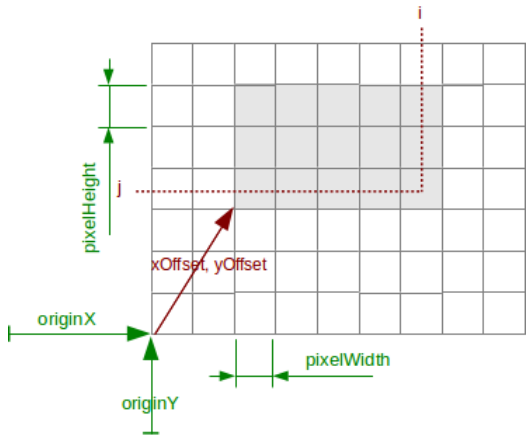

FIGURE 2 – Illustration de la correspondance entre coordonnées physiques (en vert) et index des pixels (en rouge) dans le cas d'une extraction de la zone grisée.

```
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 6 - 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

— 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.

4. La notion de ligne et colonne est conventionnelle

```
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

Reprenons l'exemple du calcul de l'indice NDVI

Le programme peut être téléchargé ici.

```
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 1 (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 2 (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 3 (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 4 (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 5 (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 6 (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 7 (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).*