1 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.

1.1 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_1984", SPHEROID["WGS 84",6378137,298.2572235630016], TOWGS84[0,0,0,0,0,0,0]], PRIMEM["Greenwich",0], UNIT["degree",0.0174532925199433], AUTHORITY["EPSG","4326"]], PROJECTION["Transverse_Mercator"], PARAMETER["latitude_of_origin",0], PARAMETER["central_meridian",-111], PARAMETER["scale_factor",0.999 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

1.2 Récupération et attribution d'un système de coordonnées

1.2.1 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 2

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 !

1.2.2 Avec des géométries

Cette fonction peut également être appelée sur une géométrie.

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

1.3 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**.

1.4 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()
```

1.5 Exercice

Exercice 1 *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 2 *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`.

2 Exercices bilans

2.1 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 **??**). 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 !).

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.

2.1.1 Compréhension du code

Exercice 3 (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.

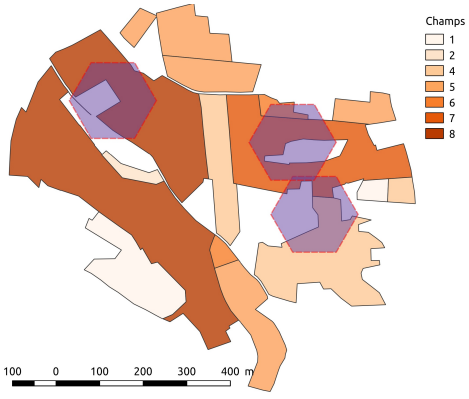
FIGURE 1 – Carte représentant les données du sujet (vue extraite avec QGis)

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 4 (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.

2.1.2 Généralisation du code

Exercice 5 (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.

2.2 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 **??**). 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.

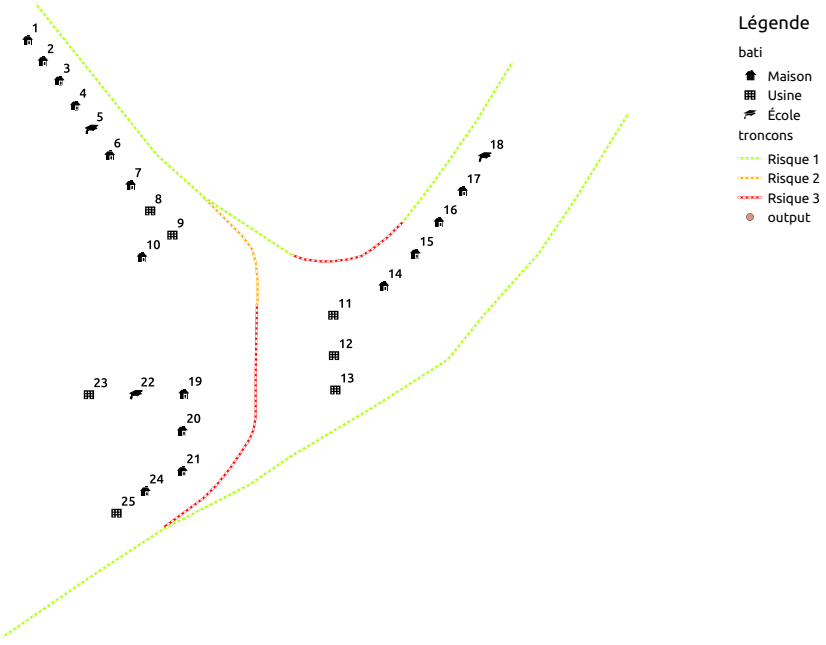


FIGURE 2 – Carte représentant les données du sujet (vue extraite avec QGis)

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

2.2.1 Compréhension du code

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

Exercice 6 (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 ? ($\star$) Pourquoi est-elle nécessaire dans cette fonction ?

Exercice 7 (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) ($\star$) Expliquer ce que fait le programme.

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

2.2.2 Généralisation du code

Exercice 8 (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** ?